目录

[1 课程设计题目 3](#_Toc76244277)

[2 课程设计目标和要求 3](#_Toc76244278)

[3 开发环境 4](#_Toc76244279)

[4 需求分析 4](#_Toc76244280)

[5 设计 4](#_Toc76244281)

[5.1 设计方案 4](#_Toc76244282)

[5.2 部署方案 5](#_Toc76244283)

[5.3 服务器数量估算 8](#_Toc76244284)

[6 实现 8](#_Toc76244285)

[6.1 搭建服务注册中心（eureka） 9](#_Toc76244286)

[6.2 注册服务提供者（client） 12](#_Toc76244287)

[6.3 统一配置中心（config） 15](#_Toc76244288)

[6.4 网关配置（gateway） 20](#_Toc76244289)

[6.5 公共代码块（common） 21](#_Toc76244290)

[6.6 服务提供者（account、demand、order） 21](#_Toc76244291)

[6.7 数据库表结构（onlineTaxi.sql） 22](#_Toc76244292)

[7 测试报告 26](#_Toc76244293)

[7.1 测试环境 26](#_Toc76244294)

[7.2 功能测试情况 26](#_Toc76244295)

[7.3 性能测试情况 32](#_Toc76244296)

[8 课程设计总结 34](#_Toc76244297)

[8.1 设计过程中遇到或存在的主要问题及解决方案 34](#_Toc76244298)

[8.2 改进建议 35](#_Toc76244299)

[8.3 体会/收获 35](#_Toc76244300)

# 课程设计题目

分布式架构网约车平台（DD打车）后端原型系统设计与实现

# 课程设计目标和要求

**设计目标：**

某互联网网约车平台（DD打车）业务快速发展，预计一年内注册用户数可达5000万，日均订单800万，高峰时段每小时200万订单。根据这一需求设计一套分布式可伸缩的网约车平台后端原型系统。

**主要功能要求：**

* 用户系统：包括普通用户和司机用户的注册、登录、退出等。
* 会员积分：支持按打车里程进行积分，并升级为不同的会员等级。
* 约车服务：司机可以设置开始/结束接单，用户可以发起/取消约车请求。
* 派单服务：系统向距离用户较近的若干司机发送约车信息，司机自行选择接单。（如果实现不了消息推送的话可以简化为前端轮询请求状态更新）
* 订单服务：用户上车后形成订单，用户到达目的地，司机提交费用，用户支付后订单结束，订单信息可查询。
* 评价服务：用户可以查看接单司机的评价，在订单完成后可以对接单司机进行评价。

**性能要求：**

* 可以通过扩展部署多台服务器的方式达到预期容量需求
* 95%的用户请求响应时间不超过1秒钟

**扩展要求：**

* 优化高峰时段车辆不足时的派单策略
* 支持消息推送服务

**其它要求：**

* 根据设计方案估算出达到预期性能时需要部署的服务器数量及类型，给出测算过程
* 测试环境：不少于两节点（也可以是虚拟机或docker）部署，模拟用户请求，进行性能测试。

# 开发环境

开发平台：Windows 10

IDE：IDEA 2020.1.1

数据库：MySQL 5.7.11

# 需求分析

本项目包含用户登录界面（完成乘客与司机的登录与注册功能），按照的登陆身份的不同，再分为乘客端和司机端两个用户界面。

**乘客端：**

乘客端包含用户退出登录；查询积分和会员等级；发起约车需求、取消约车、查询正在申请的约车；查询订单和订单处理；查看接单司机的评价，对接单司机进行评价。

**司机端：**

司机端包含用户退出登录；查询可接单项目、接单或者取消接单。

为了保证该系统是可伸缩的、易部署和易维护的分布式系统，可以采用Spring Cloud系列框架。Spring Cloud利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，还可以将Spring Boot的开发风格做到一键启动和部署。

# 设计

## 设计方案

需求了解完之后，接下来设计系统架构，首先分配出3个服务提供者，account、demand、order。

account 提供账户服务：乘客和司机登陆。

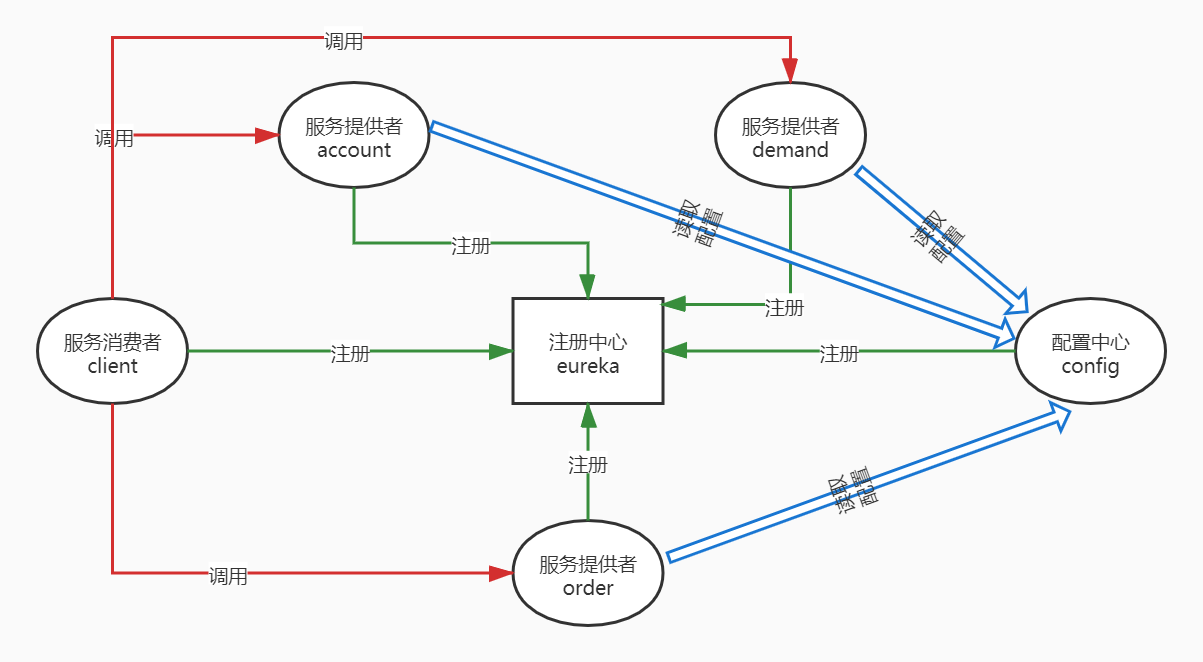
demand 提供下单服务：用户选择地点、终点形成订单，并按打车里程生成相应的积分

order 提供订单服务：查询订单、删除订单、处理订单（当司机接单之后demand就变成了order）。

接下来分配出1个服务消费者client，包括乘客端的前端页面和后台接口、司机端的前端页面和后台接口，乘客/司机直接访问的资源都保存在服务消费者中，然后服务消费者client调用3个服务提供者account、demand、order对应的接口完成业务逻辑，并通过 Feign 完成负载均衡，通过Hystrix实现服务降级和限速，达到微服务的自我保护能力。

3个服务提供者和1个服务消费者都需要在注册中心eureka完成注册，同时注册配置中心，服务提供者和服务消费者的配置信息保存在配置中心config。

关系如下图所示：



## 部署方案

微服务数量众多且相互之间存在复杂的依赖关系，为了使微服务架构能高效、稳定正确运行，可以提炼出⼀些基础组件：

* 服务注册与发现：当有大量的微服务时，需要⼀个独立的组件来管理服务实例
* 服务调⽤：OpenFeign声明式REST调⽤
* 负载均衡与路由网关：请求到来时,确定有哪⼀个节点进行请求响应，还具备访问控制、日志记录、 服务适配、请求管理等
* 功能服务保护：断路器，服务降级和限速，微服务的自我保护能力

本次实验中整体采用Spring Cloud的微服务框架搭建，为了部署此Spring Cloud生态，我们采取了以下组件：

* Spring Cloud Eureka：Spring Cloud Eureka是 Spring Cloud Netflix 微服务套件中的一个组件，它基于Netflix Eureka 做了二次封装，主要负责完成微服务架构中的服务治理功能。Eureka由多个instance(服务实例)组成，这些服务实例可以分为两种：Eureka Server和Eureka Client。为了便于理解，我们将Eureka client再分为Service Provider和Service Consumer。

Eureka Server 提供服务注册和发现

Service Provider 服务提供方，将自身服务注册到Eureka，从而使服务消费方能够找到Service Consumer服务消费方，从Eureka获取注册服务列表，从而能够消费服务

* Spring Cloud OpenFeign：Spring Cloud OpenFeign作为Spring Cloud的子项目之一，Spring Cloud OpenFeign以将OpenFeign集成到Spring Boot应用中的方式，为微服务架构下服务之间的调用提供了解决方案。首先，利用了OpenFeign的声明式方式定义Web服务客户端；其次还更进一步，通过集成Ribbon或Eureka实现负载均衡的HTTP客户端。
* Spring Cloud Gateway：网关是系统的唯一对外的入口，介于客户端和服务器端之间的中间层，处理非业务功能提供路由请求、鉴权、监控、缓存、限流等功能。它将"1对N"问题转换成了"1对1”问题。通过服务路由的功能，可以在对外提供服务时，只暴露网关中配置的调用地址，而调用方就不需要了解后端具体的微服务主机。

**gateway三大核心概念：**

1. Route（路由）：路由是构建网关的基本模块，它由ID，目标URI，一系列的断言和过滤器组成，如果断言为true则匹配该路由
2. Predicate（断言）：参考的是java8的java.util.function.Predicate开发人员可以匹配HTTP请求中的所有内容（例如请求头或请求参数），如果请求与断言相匹配则进行路由
3. Filter（过滤）：指的是Spring框架中GatewayFilter的实例，使用过滤器，可以在请求被路由前或者之后对请求进行修改。

* Spring Cloud Config将配置信息中央化保存
* Spring Cloud Netflix-Hystrix：Hystrix是Netflix开源的一款容错框架，具有自我保护能力。

**Hystrix设计目标：**

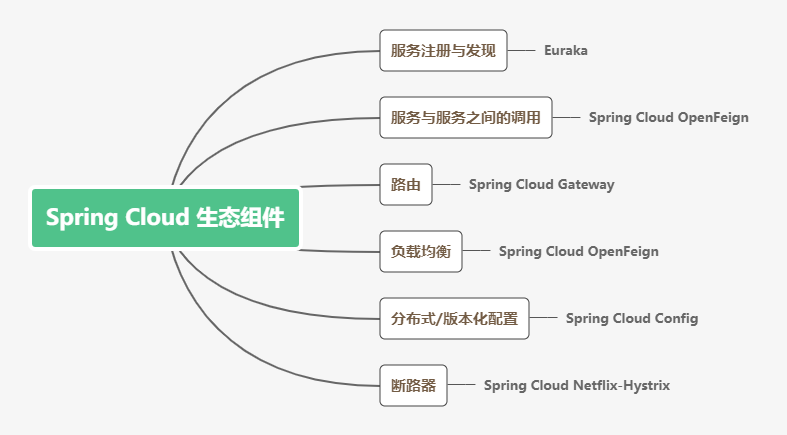
* 对来自依赖的延迟和故障进行防护和控制——这些依赖通常都是通过网络访问的
* 阻止故障的连锁反应
* 快速失败并迅速恢复
* 回退并优雅降级
* 提供近实时的监控与告警

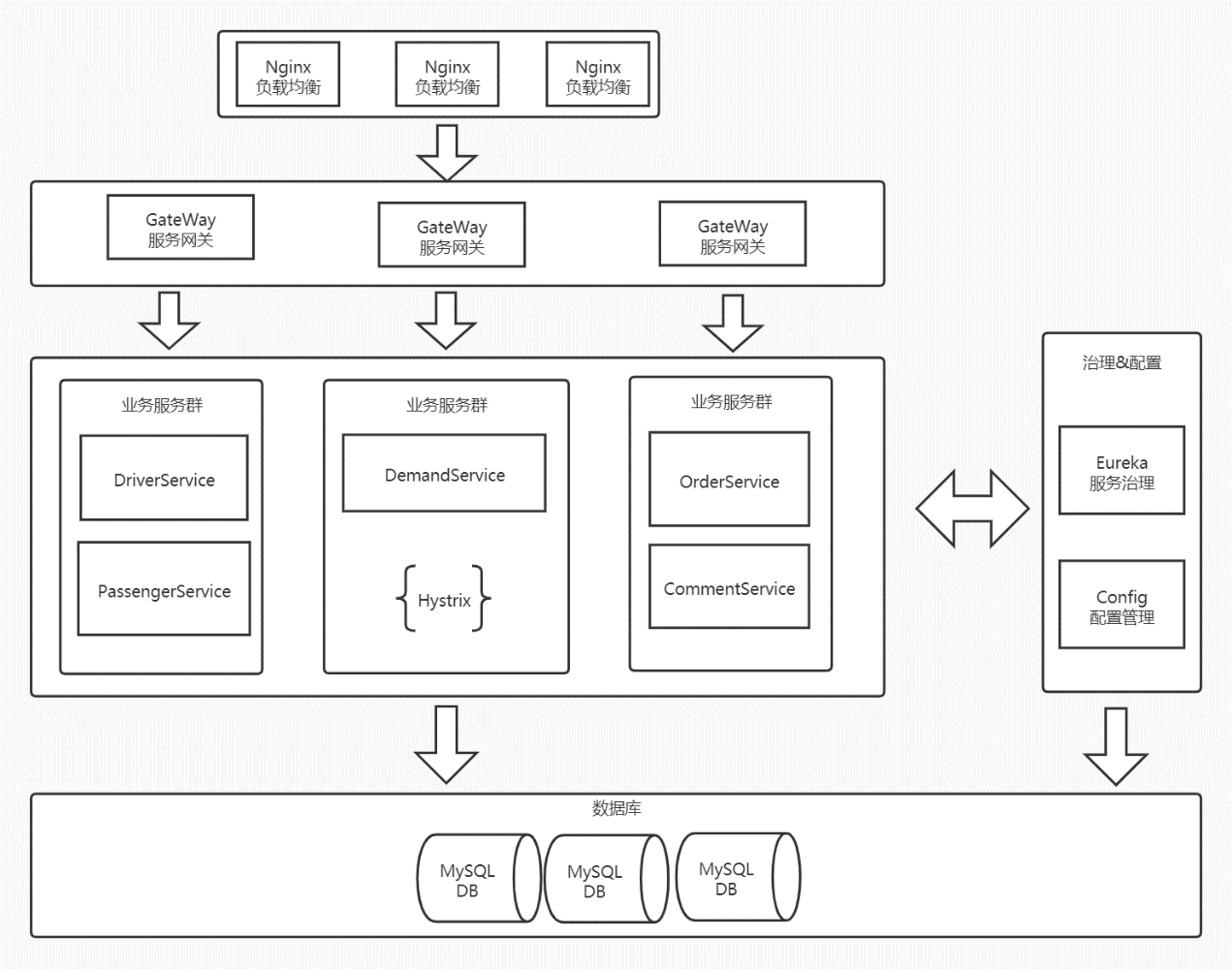
**Hystrix遵循的设计原则：**

* 防止任何单独的依赖耗尽资源（线程）
* 过载立即切断并快速失败，防止排队
* 尽可能提供回退以保护用户免受故障
* 使用隔离技术（例如隔板，泳道和断路器模式）来限制任何一个依赖的影响
* 通过近实时的指标，监控和告警，确保故障被及时发现
* 通过动态修改配置属性，确保故障及时恢复
* 防止整个依赖客户端执行失败，而不仅仅是网络通信

**Hystrix如何实现这些设计目标？**

* 使用命令模式将所有对外部服务（或依赖关系）的调用包装在HystrixCommand或HystrixObservableCommand对象中，并将该对象放在单独的线程中执行。
* 每个依赖都维护着一个线程池（或信号量），线程池被耗尽则拒绝请求（而不是让请求排队）。
* 记录请求成功，失败，超时和线程拒绝。
* 服务错误百分比超过了阈值，熔断器开关自动打开，一段时间内停止对该服务的所有请求。
* 请求失败，被拒绝，超时或熔断时执行降级逻辑。
* 近实时地监控指标和配置的修改。



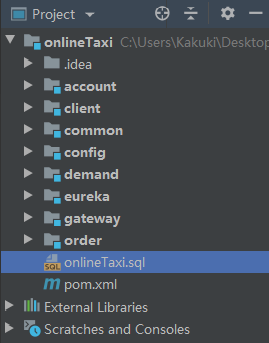


## 服务器数量估算

Tomcat默认配置最大请求数150  
Spring Boot内置Tomcat服务器  
要达到QpS555，那就需要4个Tomcat，至少4个微服务

# 实现

项目模块主要有：account（乘客/司机账户信息）、client（客户端）、common（存放各个实体类）、config（配置中心）、demand（用户需求模块）、erurka（服务注册与发现中心）、gateway（网关）、order（订单模块）



## 搭建服务注册中心（eureka）

创建一个基础的springboot工程，命名为eureka，并在*pom.xml*中引入相关依赖

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.5.2</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.bigbone</groupId>  
 <artifactId>eureka</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>eureka</name>  
 <description>Demo project for Spring Boot</description>  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>2020.0.3</spring-cloud.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

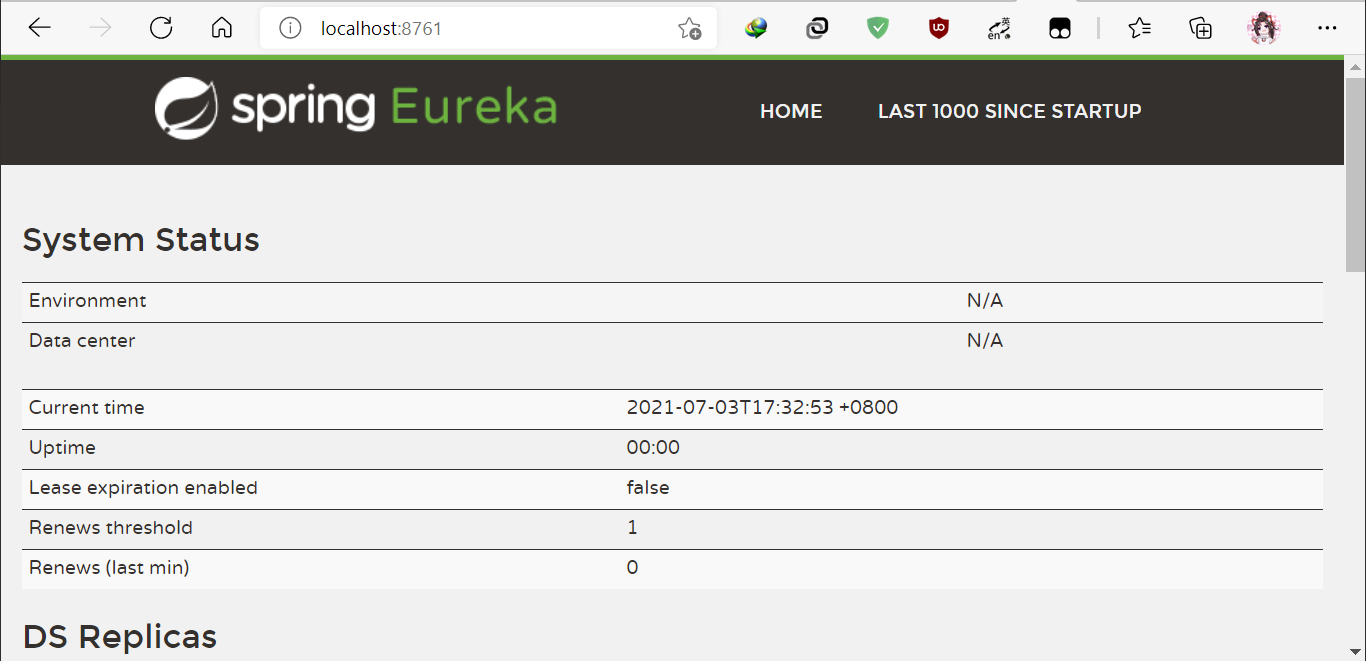
*application.yml*配置如下，在默认配置下，该服务注册中心也会将自己作为客户端来尝试注册它自己，所以我们可以选择禁用它的客户端注册行为，即 eureka.client.register-with-eureka: false 。由于注册中心的职责就是维护服务实例，它并不需要去检索服务，所以eureka.client.fetch-registry也设置为 false。

server:  
 port: 8761  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/ *# 注册中心的访问地址* register-with-eureka: false  
 fetch-registry: false

*EurekaApplication.java* 使用@EnableEurekaServer将项目声明为Spring Cloud中的注册中心

package com.bigbone.eureka;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@SpringBootApplication  
@EnableEurekaServer  
public class EurekaApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaApplication.class, args);  
 }  
}

在完成如上配置后，启动工程，访问 http://localhost:8761/ ,显示eureka注册中心面板如下



## 注册服务提供者（client）

完成了注册中心的搭建，尝试将一个springboot应用注册到eureka的服务治理体系中去，搭建一个springboot应用并加入如下依赖*pom.xml*：

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.5.2</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.bigbone</groupId>  
 <artifactId>client</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>client</name>  
 <description>Demo project for Spring Boot</description>  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>2020.0.3</spring-cloud.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.bigbone</groupId>  
 <artifactId>common</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-bootstrap</artifactId>  
 <version>3.0.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-feign</artifactId>  
 <version>1.4.7.RELEASE</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
 <version>2.0.2.RELEASE</version>  
 </dependency>  
  
 <!--静态模板-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>io.swagger.core.v3</groupId>  
 <artifactId>swagger-annotations</artifactId>  
 <version>2.1.6</version>  
 </dependency>  
 </dependencies>  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

client端配置文件如下：

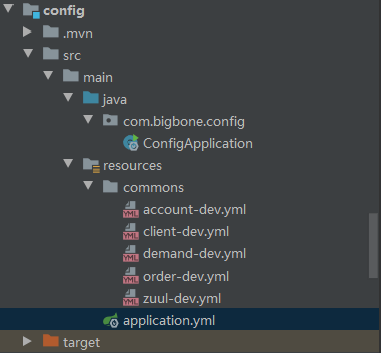
spring:  
 application:  
 name: client  
 profiles:  
 active: dev  
 cloud:  
 config:  
 uri: http://localhost:8762  
 fail-fast: true  
feign:  
 hystrix:  
 enabled: true

## 统一配置中心（config）

创建一个springboot应用，*pom.xml*依赖和client相同，同样将自己作为一个客户端服务，也是要注册到eureka中去的，*application.yml*配置文件如下。

server:  
 port: 8762  
spring:  
 application:  
 name: ConfigServer  
 profiles:  
 active: native  
 cloud:  
 config:  
 server:  
 native:  
 search-locations: classpath:/commons  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/  
 instance:  
 instance-id: config-8762  
 prefer-ip-address: true

配置文件：



*account-dev.yml*

server:  
 port: 9905  
spring:  
 application:  
 name: account  
 datasource:  
 username: root  
 password: 123456  
 url: jdbc:mysql://localhost:3306/onlinetaxi?serverTimezone=Asia/Shanghai&useSSL=true&useUnicode=true&characterEncoding=utf8  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 type: com.alibaba.druid.pool.DruidDataSource  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/  
 instance:  
 instance-id: account-9905  
 prefer-ip-address: true  
*#mybatis:  
# mapper-locations: classpath:/mapping/\*.xml  
# type-aliases-package: com.zcy.entity*

*client-dev.yml*

server:  
 port: 9903  
spring:  
 application:  
 name: client  
 thymeleaf:  
 prefix: classpath:/static/  
 suffix: .html  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/  
 instance:  
 instance-id: client-9903  
 prefer-ip-address: true

*demand-dev.yml*

server:  
 port: 9902  
spring:  
 application:  
 name: demand  
 datasource:  
 username: root  
 password: 123456  
 url: jdbc:mysql://localhost:3306/onlinetaxi?serverTimezone=Asia/Shanghai&useSSL=true&useUnicode=true&characterEncoding=utf8  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 type: com.alibaba.druid.pool.DruidDataSource  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/  
 instance:  
 instance-id: demand-9902  
 prefer-ip-address: true  
  
*#mybatis:  
# mapper-locations: classpath:/mapping/\*.xml  
# type-aliases-package: com.bigbone.common.entity*

*order-dev.yml*

server:  
 port: 9901  
spring:  
 application:  
 name: order  
 datasource:  
 username: root  
 password: 123456  
 url: jdbc:mysql://localhost:3306/onlinetaxi?serverTimezone=Asia/Shanghai&useSSL=true&useUnicode=true&characterEncoding=utf8  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 type: com.alibaba.druid.pool.DruidDataSource  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/  
 instance:  
 instance-id: order-9901  
 prefer-ip-address: true  
  
*#mybatis:  
# mapper-locations: classpath:/mapping/\*.xml  
# type-aliases-package: com.bigbone.common.entity*

*zuul-dev.yml*

server:  
 port: 9527  
spring:  
 application:  
 name: zuul  
  
eureka:  
 client:  
 service-url:  
 *# 表示eureka client发送心跳给server端的频率。如果在leaseExpirationDurationInSeconds后server端没收到client的心跳，则摘除该instance。默认30s* lease-renewal-interval-in-seconds: 5  
 *# 表示eureka server至上一次收到client的心跳之后，等待下一次心跳的超时时间，在这个时间内若没收到下一次心跳，则将移除该instance。默认90s* lease-expiration-duration-in-seconds: 10  
 defaultZone: http://localhost:9900/eureka/  
 instance:  
 instance-id: zuul-9527  
 prefer-ip-address: true

## 网关配置（gateway）

*pom.xml* 引入spring-cloud-starter-gateway依赖，并声明eurek-client

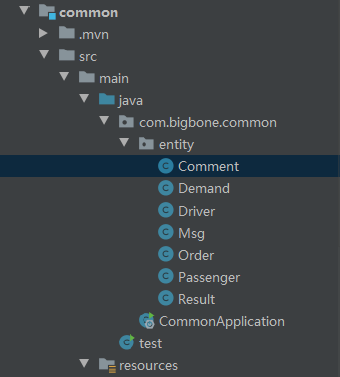
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-gateway</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>

*application.yml* 配置文件

server:  
 port: 9999  
spring:  
 application:  
 name: gateway  
 cloud:  
 gateway:  
 routes:  
 - id: demand\_route  
 uri: http://localhost:9902/  
 predicates:  
 - Path=/demand/\*\*  
  
 - id: account\_route  
 uri: http://localhost:9905/  
 predicates:  
 - Path=/account/\*\*  
  
 - id: order\_route  
 uri: http://localhost:9901/  
 predicates:  
 - Path=/order/\*\*  
  
 - id: client\_route  
 uri: http://localhost:9903/  
 predicates: *# 断言* - Path=/client/\*\*  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/  
 instance:  
 instance-id: gateway-9999  
 prefer-ip-address: true

## 公共代码块（common）

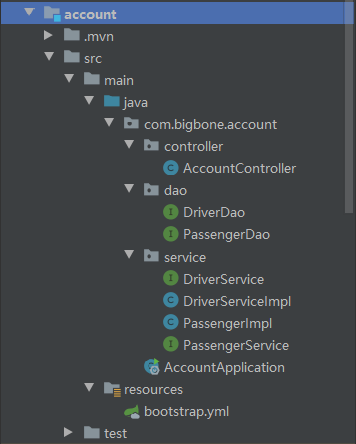
声明各种实体类，供其他模块调用



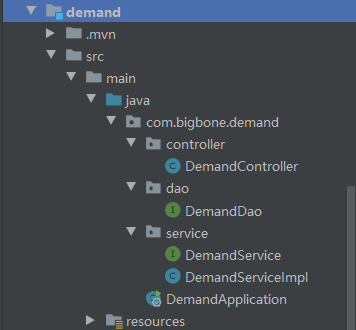
## 服务提供者（account、demand、order）

dao层主要连接数据库，封装增删改查的数据库语句，daoimpl是实现dao层方法的接口，供服务消费者client调用。

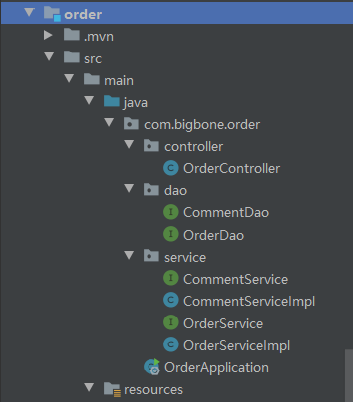
1. account（乘客与司机）



1. demand（需求）



1. order（订单与评论）



## 数据库表结构（onlineTaxi.sql）

/\*  
SQLyog Enterprise v12.09 (64 bit)  
MySQL - 5.7.11 : Database - onlinetaxi  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*/  
  
/\*!40101 SET NAMES utf8 \*/;  
  
/\*!40101 SET SQL\_MODE=''\*/;  
  
/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;  
/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;  
/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;  
/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;  
CREATE DATABASE /\*!32312 IF NOT EXISTS\*/`onlinetaxi` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;  
  
USE `onlinetaxi`;  
  
/\*Table structure for table `t\_comment` \*/  
  
DROP TABLE IF EXISTS `t\_comment`;  
  
CREATE TABLE `t\_comment` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `name` varchar(20) DEFAULT NULL,  
 `content` varchar(100) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;  
  
/\*Data for the table `t\_comment` \*/  
  
insert into `t\_comment`(`id`,`name`,`content`) values (1,'driver','nice'),(2,'driver','a little expensive'),(3,'driver','very nice!'),(4,'driver','good!'),(5,'driver','cheap');  
  
/\*Table structure for table `t\_demand` \*/  
  
DROP TABLE IF EXISTS `t\_demand`;  
  
CREATE TABLE `t\_demand` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `departure` varchar(20) DEFAULT NULL,  
 `destination` varchar(20) DEFAULT NULL,  
 `name` varchar(20) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=17 DEFAULT CHARSET=utf8;  
  
/\*Data for the table `t\_demand` \*/  
  
insert into `t\_demand`(`id`,`departure`,`destination`,`name`) values (16,'A2','A8','passenger1');  
  
/\*Table structure for table `t\_driver` \*/  
  
DROP TABLE IF EXISTS `t\_driver`;  
  
CREATE TABLE `t\_driver` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `username` varchar(11) DEFAULT NULL,  
 `password` varchar(11) DEFAULT NULL,  
 `address` varchar(20) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;  
  
/\*Data for the table `t\_driver` \*/  
  
insert into `t\_driver`(`id`,`username`,`password`,`address`) values (3,'driver','123456','A1'),(4,'driver1','123456','A2');  
  
/\*Table structure for table `t\_order` \*/  
  
DROP TABLE IF EXISTS `t\_order`;  
  
CREATE TABLE `t\_order` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `departure` varchar(20) DEFAULT NULL,  
 `destination` varchar(20) DEFAULT NULL,  
 `state` int(11) DEFAULT NULL,  
 `passenger` varchar(20) DEFAULT NULL,  
 `driver` varchar(20) DEFAULT NULL,  
 `price` int(5) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=12 DEFAULT CHARSET=dec8;  
  
/\*Data for the table `t\_order` \*/  
  
insert into `t\_order`(`id`,`departure`,`destination`,`state`,`passenger`,`driver`,`price`) values (6,'A1','B1',1,'passenger','driver',16),(8,'A1','F1',1,'passenger','driver',80),(10,'A1','A9',1,'passenger','driver',8),(11,'A1','A3',1,'passenger','driver',2);  
  
/\*Table structure for table `t\_passenger` \*/  
  
DROP TABLE IF EXISTS `t\_passenger`;  
  
CREATE TABLE `t\_passenger` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `username` varchar(11) DEFAULT NULL,  
 `password` varchar(11) DEFAULT NULL,  
 `address` varchar(20) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;  
  
/\*Data for the table `t\_passenger` \*/  
  
insert into `t\_passenger`(`id`,`username`,`password`,`address`) values (1,'passenger','123456','A1'),(2,'passenger1','123456','A2');  
  
/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;  
/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;  
/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;  
/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

# 测试报告

## 测试环境

测试平台：Windows 10

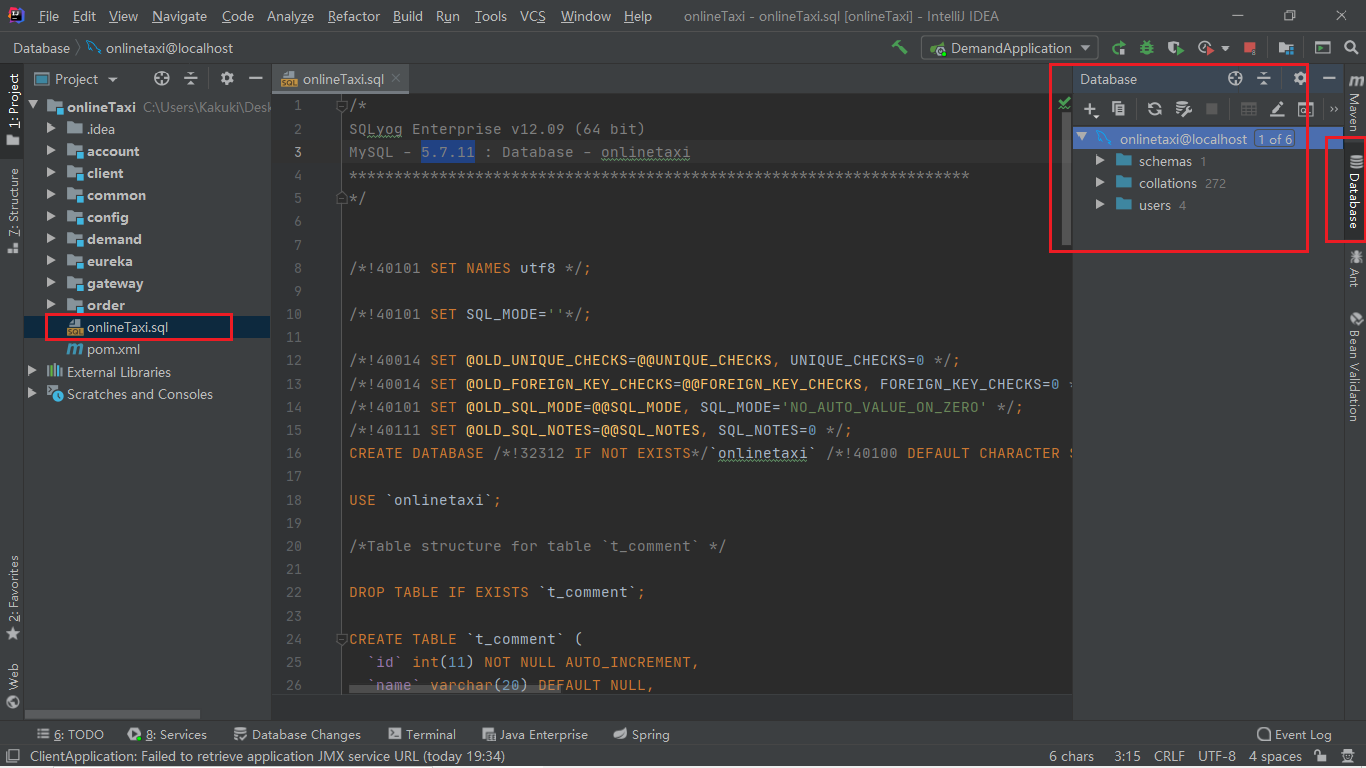
测试IDE：IDEA 2020.1.1

数据库：MySQL 8.0

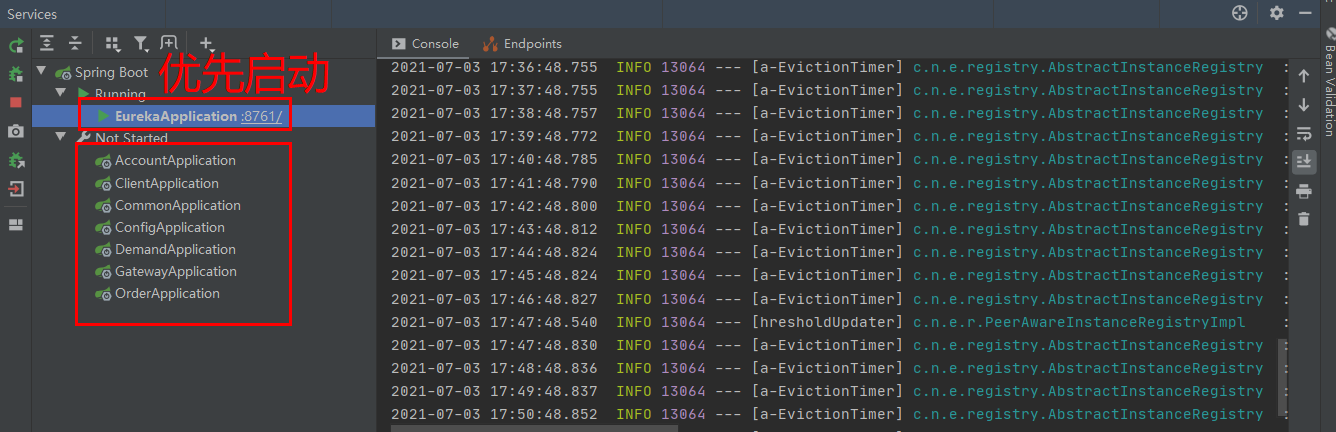
压力测试工具：Apache Jmeter 5.4.1

## 功能测试情况

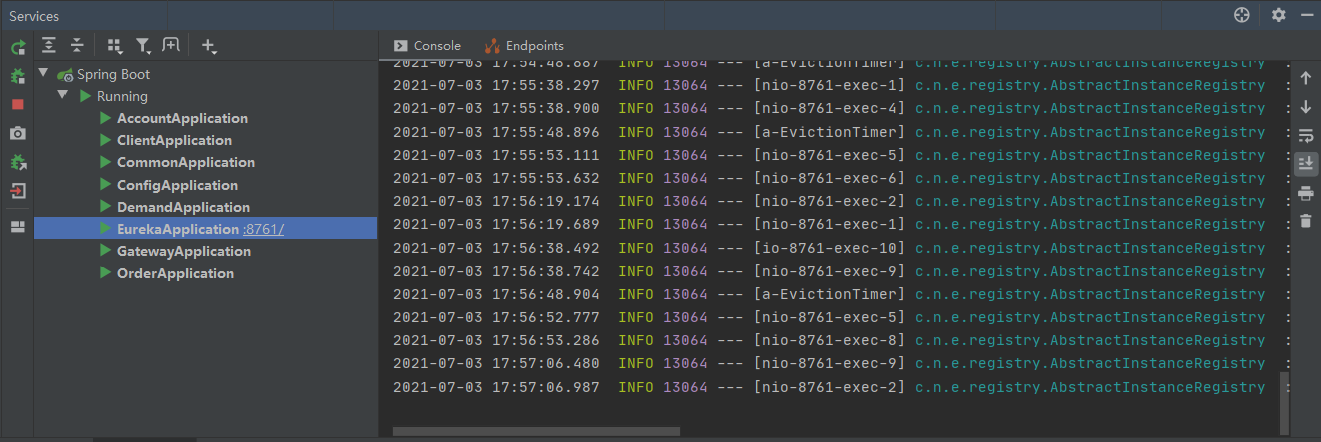
1. 打开项目路径下的onlineTaxi.sql文件，完成数据库的连接与建库建表



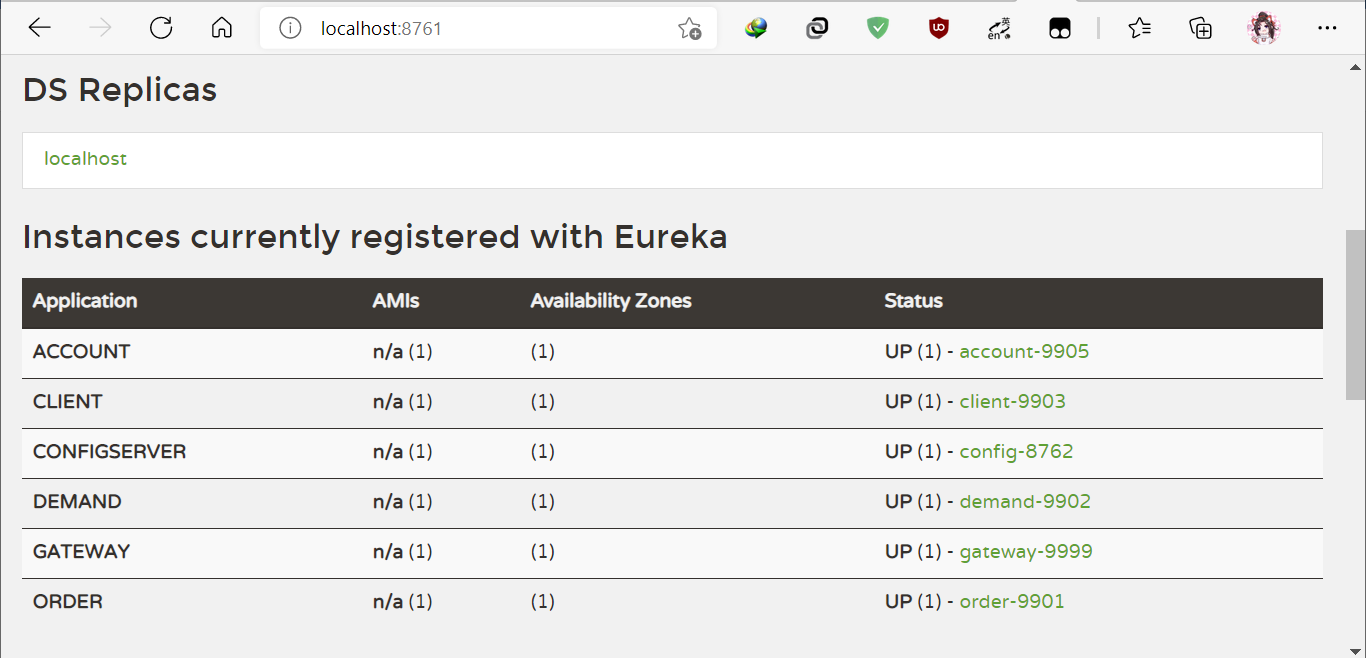
1. 优先启动EurekaApplication，再依次启动完剩余的Application



如图，所有的微服务程序都已启动

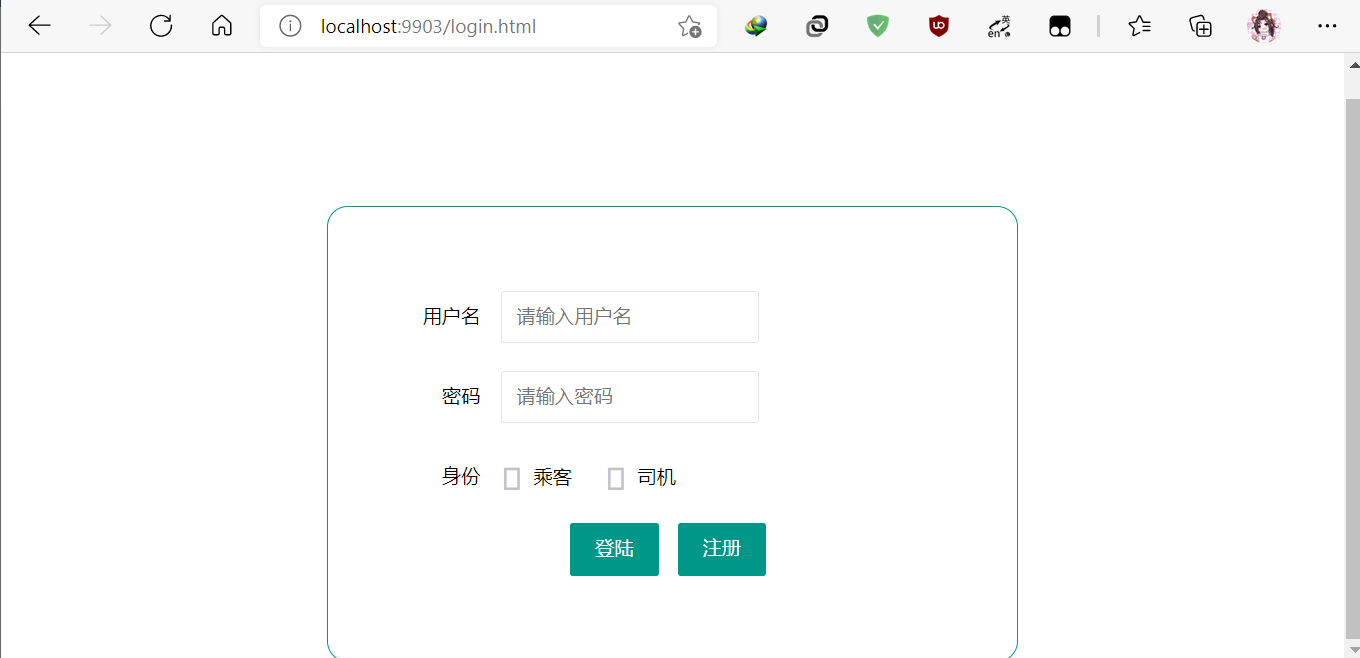


1. 在浏览器访问localhost:8761，即可查看在Eureka上注册并正在运行的实例

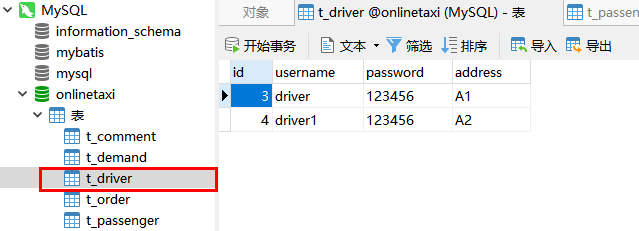


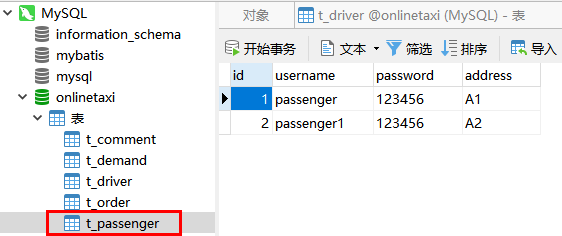
1. 访问localhost:9903即可进入客户端界面

可以选择已乘客身份或者司机身份登录，又或者注册一个账户



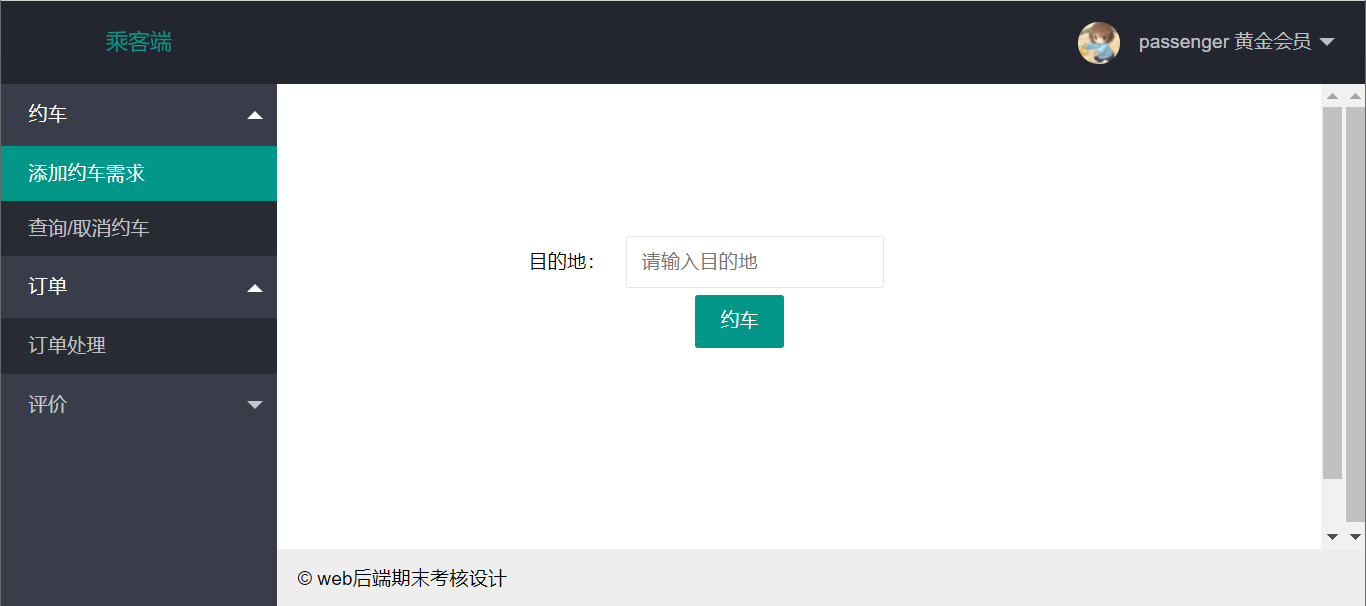
数据库中已经存储有相应的账户，可直接测试使用





1. 乘客端登录





1. 司机端登录





1. 开始测试约车
   1. 乘客端输入目的地开始约车



* 1. 司机端可以收到约车请求，可以进行接单操作。



* 1. 司机确认接单



* 1. 司机接单成功，生成订单信息并按照距离计算费用

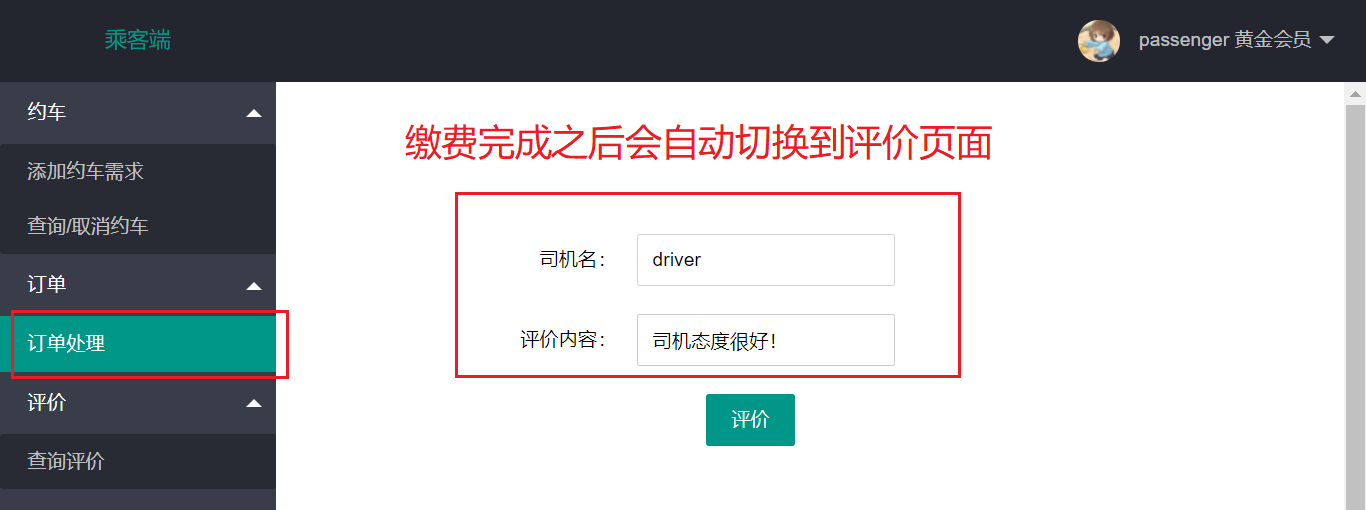


* 1. 乘客可以在订单处理页面查看订单，并缴费





1. 缴费完成之后会自动切换到评价页面，乘客可以输入司机名称和评价内容

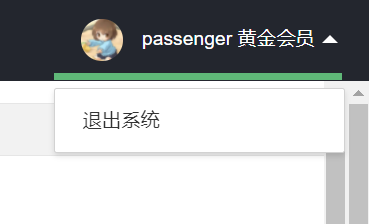


1. 查看评价





1. 用户右上角可以退出登录和查看当前积分等级



## 性能测试情况

**Thread Group(线程组)**

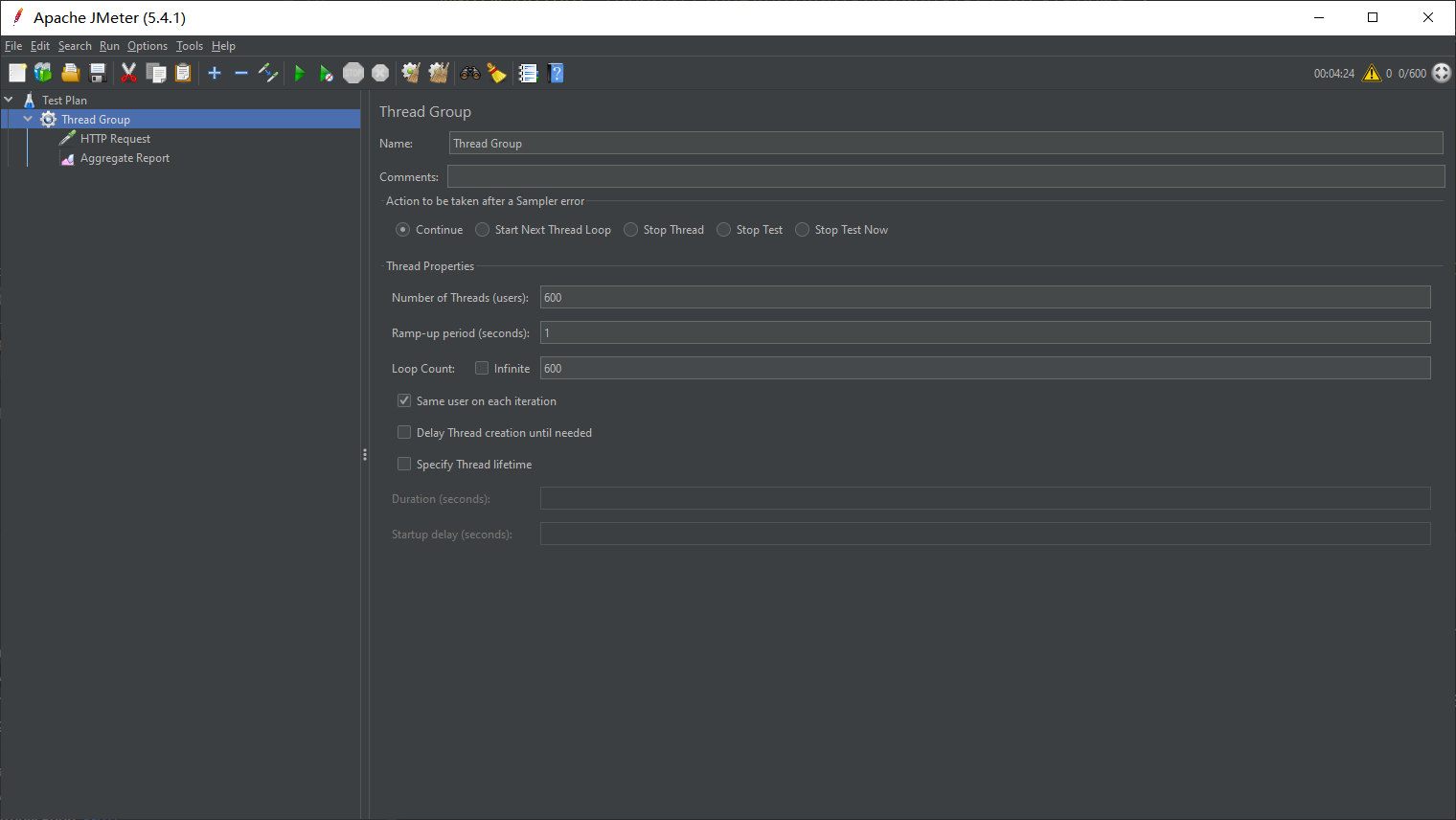
通过线程组添加运行的线程。通俗的讲一个线程组，可以看做一个虚拟用户组，线程组中的每个线程都可以理解为一个虚拟用户。线程组中包含的线程数量在测试执行过程中是不会发生改变的。

线程数：这里选择600

Ramp-Up Period：单位是秒，默认时间是1秒。它指定了启动所有线程所花费的时间，比如，当前的设定表示“在1秒内启动600个线程，每个线程的间隔时间为0.00167秒”。（如果设置为0，就是并发执行）

循环次数：选择600，表示每个线程执行600次请求。

本次测试我们设置了600个线程，循环600次，相当于模拟了600个用户访问，总的访问次数达到了360000次。



**结果报告：**

Label：取样器类型，本次取样是HTTP Request

#Samples：取样次数，本次测试共取样370000次

Average：平均响应时间，425ms

Median：响应时间的中位数，431ms

90%Line：90%用户的响应时间，559ms

95%Line：95%用户的响应时间，614ms

99%Line：99%用户的响应时间，792ms

Min：最小响应时间，3ms

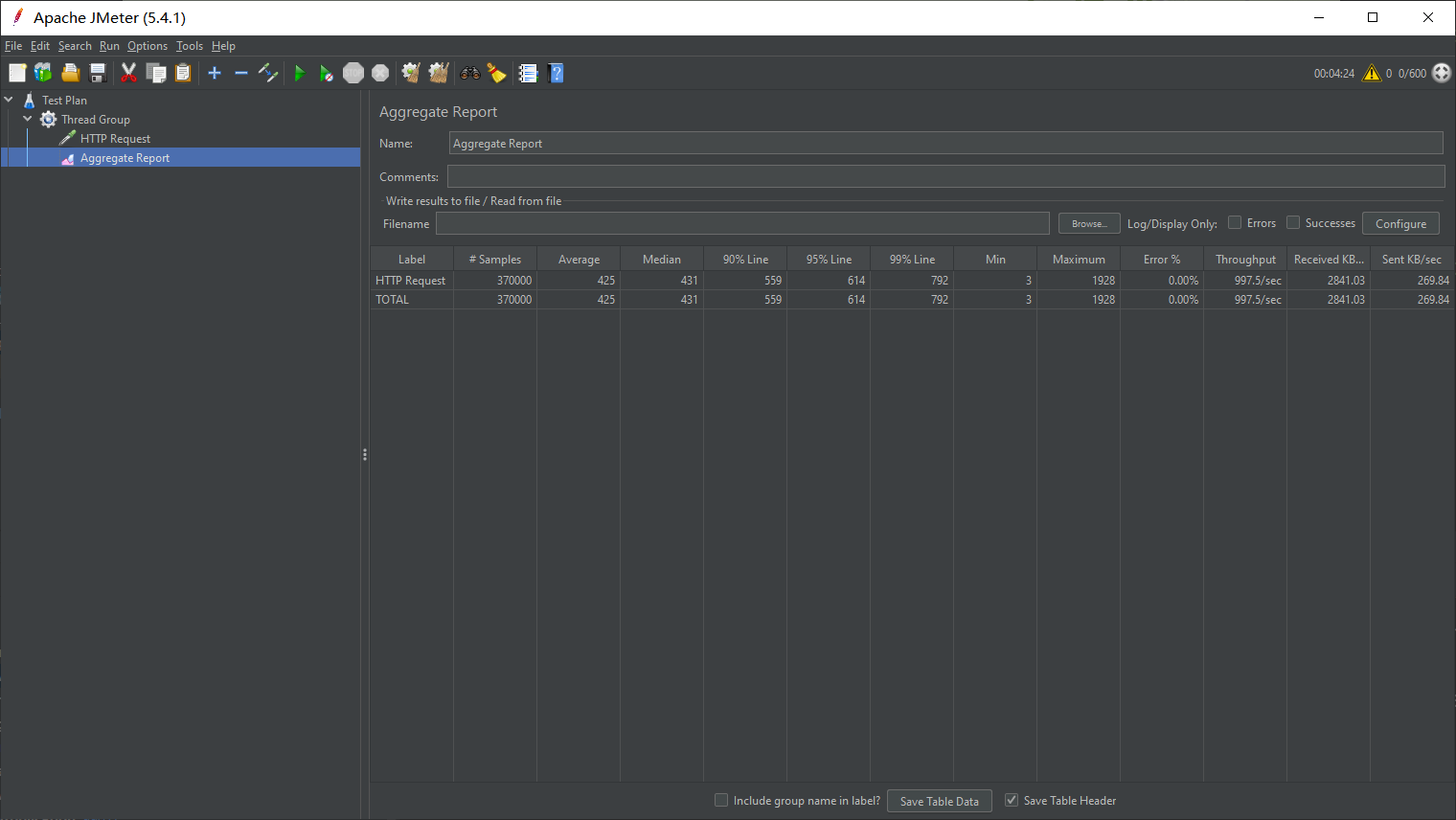
Maximum：最大响应时间，1928ms

Error%：执行错误的统计信息，0.00%

Throughput：吞吐量，997.5/sec

Received：服务器端接收数据的速度，2841.03 KB/sec

Sent：客户端发送数据的速度，268.84 KB/sec

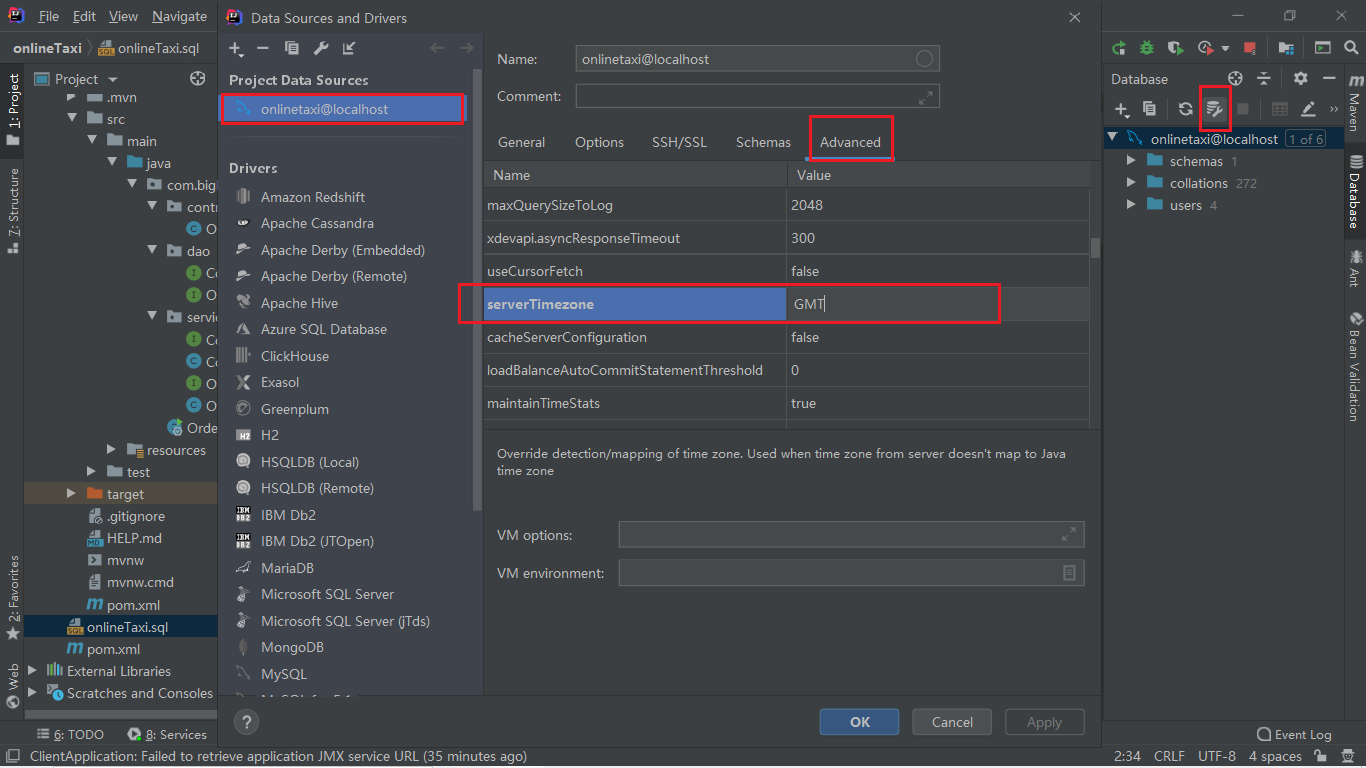


# 课程设计总结

## 设计过程中遇到或存在的主要问题及解决方案

**问题一：**

在MySQL8.0之后的版本，会遇到数据库时区问题，需要将手动将serverTimezone改成GMT，MySQL8.0之前的版本无此问题，开发环境与测试环境不匹配带来的问题。



**问题二：**

bootstrap.yml报错

该配置文件的加载顺序优于application.yml，IDEA报错是因为2020.03的Spring cloud默认配置去掉了bootstrap，因此要导入bootstrap的maven。

## 改进建议

随着时间和业务的发展，数据库中的数据量增长是不可控的，库和表中的数据会越来越大，随之带来的是更高的磁盘、IO、系统开销，甚至性能上的瓶颈，而一台服务的资源终究是有限的，因此需要对数据库和表进行拆分，从而更好的提供数据服务。

本组在数据库设计方面，把所有的表都放到了一个数据库中，这不符合分布式系统的特点，应该进行分库，将表分配到相应的库中，建立外键进行关联，这样每个微服务查询对应的数据库，而不是所有的微服务都查询一个数据库。对于多个司机同时接一个订单的问题考虑不周，本组对于接单问题只是简单的对数据库进行增删改查，当多个司机同时接单时，可能会导致数据库的错误，应该引入事务来保证数据库的正确性，提高系统的安全。

## 体会/收获

通过这次web后端大作业，我对于用spring cloud技术实现微服务架构有了更深的了解。通过自己写代码实现spring cloud技术，本组人员对于Eureka实现服务注册与发现，Hystrix实现请求熔断和服务降级，OpenFeign实现服务调用，Gateway实现负载均衡，配置中心实现对微服务进行集中化配置有了一定的认识，也对于大型服务器如何实现分布式架构有了更深的体会。

本次Web后端的开发，用到了许多新的架构、新的技术，从上学期的web开发到本学期的Spring Cloud，我看到了Web技术的快速发展，就本次Spring Cloud开发而言，Spring Cloud提供的生态组件就十分丰富且多样化，参考资料也及其丰富，这对我们本次实验有巨大的帮助。得益于Spring Cloud的开发便利性，简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，做到了一站式部署服务，我们才得以高效而精准的完成实验。

最重要的是，本组人员通过这次大作业的开发，懂得了如何利用书本资料和博客来解决开发中遇到的问题，增强了自身发现问题解决问题的能力。总之，这次大作业让我们收获良多。