

# 运输系统设计报告 v4.0

## 目录

1 引言 .....	1
1.1 编写目的 .....	1
1.2 项目背景 .....	2
1.3 项目概述 .....	2
1.4 参考资料 .....	3
2 设计概述 .....	4
2.1 需求概述 .....	4
2.2 条件与限制 .....	5
3 系统详细需求分析 .....	7
3.1 详细功能需求分析 .....	7
3.2 详细性能需求分析 .....	8
3.3 详细资源需求分析 .....	9
4 Restful 设计 .....	11
4.1 POST 用户登录 .....	11
4.2 POST 用户注册 .....	11
4.3 GET 获得公司名下所有车辆和司机 .....	11
4.4 POST 新增一位司机 .....	12
4.5 POST 新增一辆车 .....	12
4.6 POST 承运商接单创建订单 .....	12
4.7 POST 更新订单状态和所在地址 .....	13
4.8 POST 更新订单状态为已签收 .....	13
4.9 POST 商家新建商品 .....	13
5 数据库系统设计 .....	15
5.1 设计要求 .....	15
5.2 信息模型设计 .....	15
5.3 数据库设计 .....	15
6 系统设计 .....	18
6.1 关键技术 .....	18
6.2 Spring Boot 框架开发 .....	19
6.3 安全认证 .....	20
7 微服务架构、Spring Cloud 和 Kafka 实现 .....	23
7.1 微服务架构概述 .....	23
7.2 架构设计 .....	23
7.3 Spring Cloud 集成 .....	24

7.4 网关实现 .....	24
7.5 Kafka 实现 .....	25
7.6 配置与集成 .....	27
8 总结 .....	29

# 1 引言

本文档旨在详细介绍智能公路运输调度管理系统的微服务架构和 Spring Cloud 实现。本系统的重构目标在于实现一个高效、智能、可靠的运输调度管理系统，提供给货主、货代和承运商等相关方便利的服务。本系统的开发基于 Spring Boot 框架和 Spring Cloud 微服务架构，使用 Eureka 进行服务发现，Resilience4j 或 Hystrix 实现断路器，集成 OAuth2 授权服务器，使用网关向外部用户暴露 API，使用 Spring Cloud Config Server 和 Sleuth 进行集中配置和跟踪。本文档将详细介绍每个微服务的功能和如何使用 Spring Cloud 进行服务发现、断路器、集中配置和跟踪，以便读者更好地理解系统的设计和实现。

## 1.1 编写目的

本系统的重构目标在于实现一个高效、智能、可靠的运输调度管理系统，提供给货主、货代和承运商等相关方便利的服务。本系统的开发基于 Spring Boot 框架和 Spring Cloud 微服务架构，使用 Eureka 进行服务发现，Resilience4j 或 Hystrix 实现断路器，集成 OAuth2 授权服务器，使用网关向外部用户暴露 API，使用 Spring Cloud Config Server 和 Sleuth 进行集中配置和跟踪。通过本份详细设计报告，详细说明了系统的各个微服务、各个组成部分的设计和实现细节，以确保系统能够达到其预期的功能和性能要求。

同时，本系统的开发基于以下技术栈和框架：

- Spring MVC：用于实现 Web 应用程序的开发框架，提供了灵活的 MVC 架构支持。
- MyBatis：用于实现数据访问层的框架，提供了便捷的数据库操作和查询功能。
- Thymeleaf：用于实现 Web 页面的模板引擎，支持动态数据绑定和页面渲染。
- Spring Boot：用于快速构建基于 Spring 的应用程序的框架。
- Spring Cloud：用于构建分布式系统的微服务框架，提供了服务发现、断路器、负载均衡、配置管理、API 网关等功能。
- Eureka：用于服务发现和注册的组件，提供了高可用性和可扩展性。
- Resilience4j 或 Hystrix：用于实现断路器的组件，提供了容错和故障恢复的功能。
- OAuth2：用于身份验证和授权的协议，提供了安全的 API 访问控制。

- **Zuul 或 Spring Cloud Gateway:** 用于 API 网关的组件, 提供了路由、负载均衡和安全性等功能。
- **Spring Cloud Config Server:** 用于集中配置管理的组件, 提供了动态配置和版本控制的功能。
- **Sleuth:** 用于分布式跟踪的组件, 提供了请求链路追踪和日志聚合的功能。

此外, 本系统也重视身份验证和授权的必要性, 以确保只有经过授权的用户可以访问系统的功能和数据。我们将采用 **Spring Security** 来实现身份验证和授权机制, 以确保系统的安全性和可靠性。

测试也是开发过程中的重要环节。为了确保系统的质量和稳定性, 我们将进行单元测试和集成测试, 其中单元测试用于测试存储库层面的功能, 而集成测试将验证控制器层的功能和系统整体的交互。。

## 1.2 项目背景

当前, 公路运输行业是我国交通运输领域的重要组成部分, 具有广阔的市场前景和发展空间。然而, 由于公路运输行业存在着信息不对称、运力利用率低、服务质量难以保障等问题, 导致了运输行业的低效率和服务质量不稳定的局面。

因此, 开发一种智能公路运输调度管理方法成为了当前行业的需求之一。本软件系统旨在解决公路运输行业中存在的问题, 提高运输效率和服务质量, 为用户提供更好的使用体验和服务。

本系统将利用现有的承运商运力资源信息和车联网运力资源信息, 加入运单调度管理系统中的运力池匹配管理系统中, 实现运单管理匹配和分配调度运单给承运商系统执行。同时, 智能监控中心将通过接口连接承运商系统和车联网系统, 对运单进行在途跟踪、服务质量管理及预警与事件管理, 以确保运输过程的安全和可靠。通过本软件系统, 运输行业可以实现智能化调度和管理, 提高行业的效率和服务质量, 为行业的发展注入新的动力。

## 1.3 项目概述

本系统的设计和实现基于 **Spring Cloud** 微服务架构, 使用 **Eureka** 进行服务发现, **Resilience4j** 或 **Hystrix** 实现断路器, 集成 **Oauth2** 授权服务器, 使用网关向外部用户暴露 API, 使用 **Spring Cloud Config Server** 和 **Sleuth** 进行集中配置和跟踪。本系统包括以下微服务:

- **authorization\_service:** 授权服务微服务负责身份验证和授权功能。它集成了 **Oauth2** 授权服务器, 用于对用户进行身份验证和颁发访问令牌。

- **car\_driver\_service:** 车辆和司机服务微服务负责管理公司名下的车辆和司机信息。它提供了 API 来获取公司名下所有车辆和司机的信息，以及新增司机和车辆的功能。
- **gateway\_service:** 网关服务微服务作为系统的入口，负责接收外部用户的请求并将其路由到相应的微服务。它提供了 API 的外部访问接口，并处理路由、负载均衡和安全性等方面的功能。
- **goods\_service:** 商品服务微服务负责商家的商品管理。它提供了 API 来新增商家的商品，并可能包括其他商品相关的功能，如查询、修改和删除商品等。
- **order\_service:** 订单服务微服务负责处理订单相关的操作。它提供了 API 来创建订单、更新订单状态和所在地址，并包括其他订单管理功能，如查询订单、取消订单等。
- **register\_service:** 注册服务微服务负责用户注册功能。它提供了 API 来注册新用户，并可能包括验证注册信息、发送确认邮件等相关功能。
- **user\_service:** 用户服务微服务负责用户管理功能。它提供了 API 来进行用户登录、获取用户信息等操作，并包括用户资料修改、密码重置等功能。

## 1.4 参考资料

智能公路运输调度管理方法说明书

## 2 设计概述

本系统的设计和实现基于 Spring Cloud 微服务架构，使用 Eureka 进行服务发现，Resilience4j 或 Hystrix 实现断路器，集成 OAuth2 授权服务器，使用网关向外部用户暴露 API，使用 Spring Cloud Config Server 和 Sleuth 进行集中配置和跟踪。

### 2.1 需求概述

所开发的智能公路运输调度管理系统主要用于货主或货代对货物进行发货下运输订单，并实现对运单的调度管理和监控服务。下面是系统的概要描述：

#### 1.主要业务需求：

(1)实现货主或货代对货物进行发货下运输订单，根据需要预先选择运输产品和运输要求。

(2)对运输产品、运输要求、货物运输运单和承运商运力资源信息进行运单管理匹配，分配调度运单给承运商系统进行运力执行。

(3)对运单进行在途跟踪、服务质量管理和预警与事件管理，保障运输服务质量。

#### 2.输入和输出：

(1)输入：货物运输运单、运输产品和运输要求、承运商运力资源信息、车联网运力资源信息等。

(2)输出：运单管理匹配状态、在途跟踪、服务质量管理和预警与事件管理状态

#### 3.主要功能：

(1)发货系统：支持货主或货代对货物进行发货下运输订单，并预先选择运输产品和运输要求，生成统一标准的货物运输运单。

(2)运单调度管理系统：管理和匹配承运商运力资源信息、车联网运力资源信息、运输产品信息和货物运输运单信息，分配调度运单给承运商系统进行运力执行。

(3)智能监控中心：对运单进行在途跟踪、服务质量管理和预警与事件管理。

#### 4.性能要求：

(1)系统需要支持大规模的运单管理和监控服务，能够处理高并发的业务请求。

(2)系统需要实现快速响应和高效处理，保证服务的实时性和可靠性。

(3)系统需要保证数据的准确性和完整性，避免数据丢失或错误。

(4)系统需要具备一定的安全性和稳定性，避免系统被攻击或崩溃。

综上所述，该系统的性能要求主要集中在系统的响应速度、处理能力、数据准确性和系统稳定性等方面。需要具备高效、稳定、安全的性能特点，以满足用户的实际业务需求。

## 2.2 条件与限制

该智能公路运输调度管理方法受到以下内部和外部条件的约束和限制：

### 1.业务条件与限制：

(1)承运商系统和车联网系统的接口标准需与运单调度管理系统和智能监控中心进行协商，以实现信息互通和交互。

(2)发货系统需要支持运输产品和运输要求的预先选择，以生成统一标准的货物运输运单。

(3)运单调度管理系统需要实现对承运商运力资源信息、车联网运力资源信息、运输产品信息和货物运输运单信息的管理和匹配分配，以实现运单的调度管理。

(4)智能监控中心需要实现对运单的在途跟踪、服务质量管理和预警与事件管理，以保障运输服务质量。

### 2.技术条件与限制：

(1)运单调度管理系统和智能监控中心需要具备一定的计算能力和存储能力，以支持大规模的运单管理和监控服务。

(2)所有系统之间需要建立稳定的网络连接，以支持数据传输和信息交互。

(3)所有系统需要保证信息安全和数据隐私，需要具备一定的安全管理和数据隐私保护能力。

(4)车联网设备需要覆盖运输车辆所在的区域，以实现对运输车辆和运输过程的监控和管理。

### 3.进度与管理方面的限制：

(1)该系统需要经过设计、开发、测试和部署等多个阶段，需要有一定的开发和运维人员支持。

(2)系统的实现需要根据实际业务需求进行调整和优化，需要一定的业务分析和需求管理能力。

(3)系统的部署和运行需要进行维护和监控，需要具备一定的系统管理和运维能力。

(4)系统需要满足政府和行业的相关法规和标准要求，需要具备相关的合规管理和认证。



综上所述，该智能公路运输调度管理方法受到多方面的约束和限制，需要根据实际情况进行评估和调整。同时需要具备一定的业务和技术能力，以保证系统的正常运行和维护。

## 3 系统详细需求分析

### 3.1 详细功能需求分析

#### 1. 发货系统功能需求：

- 用户认证功能：使用 Spring Security 进行用户身份验证和授权，包括注册和登录功能。
- 发货功能：用户可以输入货物基本信息、运输要求、承运商信息和车辆信息等，生成货物运输运单。
- 查看订单功能：用户可以查询和查看已经下单的货物运输运单信息。
- 取消订单功能：用户可以对未被承运商接单的订单进行取消操作。

#### 2. 运单调度管理系统功能需求：

- 运单匹配功能：根据货物基本信息、运输要求、承运商运力资源信息和车联网运力资源信息，对运单进行匹配。
- 运单调度功能：根据运单匹配结果，将运单分配给合适的承运商进行运力执行，并进行运单调度管理。
- 运单状态监控功能：对承运商运力执行情况进行监控，及时反馈运单状态。

#### 3. 智能监控中心功能需求：

- 在途跟踪功能：通过车联网系统实现对运输车辆的实时定位，实现对运单的在途跟踪。
- 服务质量管理功能：对承运商的服务质量进行监管和管理，及时发现和处理服务质量问题。
- 预警与事件管理功能：通过对运单数据进行分析和处理，实现预警和事件管理。

#### 4. 系统管理功能需求：

- 用户管理功能：对用户信息进行管理和维护，包括用户注册、登录、权限管理等。
- 数据管理功能：对系统的数据进行管理和维护，包括货物信息、运单信息、承运商信息等。

- 系统配置功能：对系统的配置信息进行管理和维护，包括运输产品信息、运输要求信息等。

#### 5. 系统性能需求：

- 高并发处理：系统需要支持高并发的业务请求，保证快速响应和高效处理。
- 数据准确性和完整性：系统需要保证数据的准确性和完整性，避免数据丢失或错误。
- 安全性和稳定性：系统需要具备一定的安全性和稳定性，使用 Spring Security 进行身份验证和授权，使用 Cookies 和会话管理确保安全性，使用拦截器/过滤器等技术来改进系统的功能。

综上所述，智能公路运输调度管理系统的功能需求主要包括发货系统、运单调度管理系统、智能监控中心和系统管理等四个部分。通过对这些功能进行详细的分析和设计，可以确保系统能够满足用户的实际业务需求，提高系统的使用价值和用户满意度。同时，系统需要具备高效、稳定、安全等性能特点，以保证系统的正常运行和服务质量。

## 3.2 详细性能需求分析

### 1. 登录

- 至少支持一百个左右用户同时并发登录，登录的响应时间不能超过 5 秒。

### 2. 业务

- 货主或货代：货主或货代管理模块包括两个业务过程，进入货主或货代管理界面和新增货主或货代并提交。进入货主或货代管理界面的响应时间不能超过 5 秒，提交新增货主或货代的响应时间不能超过 8 秒。
- 承运商：承运商管理模块包括两个业务过程，进入承运商管理界面和新增承运商并提交。进入承运商管理界面的响应时间不能超过 5 秒，提交新增承运商的响应时间不能超过 8 秒。
- 运输订单：运输订单管理模块包括两个业务过程，进入运输订单管理界面和生成统一标准的货物运输单。进入运输订单管理界面的响应时间不能超过 5 秒，生成统一标准的货物运输单的响应时间不能超过 10 秒。
- 货物管理：货物管理模块包括三个业务过程，获取货物运单，根据货物运单分配货物，更新仓库货物余量。获得货物运单的响应时间不得超过

8 秒，根据货物运单分配货物的响应时间不得超过 10 秒，更新仓库货物余量的响应时间不得超过 8 秒。

- 运单调度：运单调度管理模块包括四个业务过程，获得货物运单，分配运单给承运商系统执行，更新运单调度匹配状态，并将运单管理匹配状态返回给发货状态系统。获得货物运单的响应时间不得超过 8 秒，分配运单给承运商系统执行的响应时间不得超过 10 秒，更新运单调度匹配状态的时间不得超过 8 秒，将运单管理匹配状态返回给发货状态系统的响应时间不得超过 5 秒。
- 运力调度：运力调度管理模块包括四个业务过程，获得货物运单，根据货物运单分配司机和车辆，更新司机池和车辆池。获得货物运单的响应时间不得超过 8 秒，根据货物运单分配司机和车辆的响应时间不得超过 10 秒，更新司机池和车辆池的响应时间不得超过 10 秒。
- 发货状态：发货状态管理模块包括两个业务过程，进入发货状态界面和获得当前发货状态。进入发货状态界面的响应时间不能超过 5 秒，获得当前发货状态的响应时间不能超过 8 秒。
- 智能监控中心：智能监控中心管理模块包括四个业务过程，获得运单信息，对运单在途跟踪，对运单进行服务质量管理，对运单进行预警与事件管理状态。获得运单信息的响应时间不得超过 8 秒，对运单在途跟踪的响应时间不得超过 15 秒，对运单进行服务质量管理的响应时间不得超过 15 秒，对运单进行预警与事件管理状态的响应时间不得超过 20 秒。

### 3.3 详细资源需求分析

- 计算资源：需要具备一定的计算能力的计算机服务器，以支持运单调度管理系统和智能监控中心的运行。
- 存储资源：需要具备一定的存储能力的计算机服务器，以存储运单调度管理系统和智能监控中心所需的数据，包括承运商运力资源信息、车联网运力资源信息、运输产品信息、货物运输运单信息、运单管理匹配状态信息等。
- 网络资源：需要具备一定的网络带宽和稳定性，以支持承运商系统、车联网系统、发货系统、运单调度管理系统和智能监控中心之间的数据传输和信息交互。
- 接口资源：需要具备一定的接口能力，以实现承运商系统、车联网系统、发货系统、运单调度管理系统和智能监控中心之间的信息互通和交互。

- 人力资源：需要具备一定的技术支持和运维人员，以保证系统的正常运行和维护。
- 安全资源：需要具备一定的安全管理和数据隐私保护能力，以保护系统的信息安全和数据隐私。
- 车联网设备资源：需要具备车联网设备，以实现运输车辆的实时监控和管理。

## 4 Restful 设计

本系统接口设计采用 HTTP 协议进行通信。通过使用 Spring Security 进行身份验证和授权,使用 Cookies 和会话管理来维护用户状态,以及使用拦截器/过滤器等技术来改进系统的功能和安全性。

### 4.1 POST 用户登录

(1)URL:/index

(2)请求参数:

名称	位置	类型	必选	说明
body	body	object	否	none
username	body	string	否	none
password	body	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

### 4.2 POST 用户注册

(1)URL:/user

(2)请求参数:

名称	类型	必选	说明
username	string	否	none
password	string	否	none
email	string	否	none
role_id	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

### 4.3 GET 获得公司名下所有车辆和司机

(1)URL:/company/{id}/cars-drivers

(2)请求参数:

名称	类型	必选	说明
Cookie	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

#### 4.4 POST 新增一位司机

(1)URL: /company/{id}/driver

(2)请求参数:

名称	类型	必选	说明
drivename	string	否	none
phone	string	否	none
age	string	否	none
driving_age	string	否	none
Cookie	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

#### 4.5 POST 新增一辆车

(1)URL:/company/{id}/car

(2)请求参数:

名称	类型	必选	说明
id	string	否	none
car_age	string	否	none
limit_weight	string	否	none
Cookie	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

#### 4.6 POST 承运商接单创建订单

(1)URL:/consigner/{id}/order

(2)请求参数:

名称	类型	必选	说明
Cookie	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

## 4.7 POST 更新订单状态和所在地址

(1)URL:/company/{id}/order/{orderId}/{status}/{now\_addr}

(2)请求参数:

名称	类型	必选	说明
orderId	string	否	none
status	string	否	none
now_addr	string	否	none
Cookie	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

## 4.8 POST 更新订单状态为已签收

(1)URL:/company/{id}/order/{orderId}/statue/signed

(2)请求参数:

名称	类型	必选	说明
orderId	string	否	none
Cookie	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK](https://tools.ietf.org/html/rfc7231#section-6.3.1)	成功

## 4.9 POST 商家新建商品

(1)URL:/consigner/{id}/goods



(2)请求参数:

名称	类型	必选	说明
goods_description	string	否	none
dest_addr	string	否	none
price	string	否	none
dest_time	string	否	none
begin_addr	string	否	none
create_time	string	否	none
demands	string	否	none

(3)返回示例: 200 Response

(4)返回结果:

状态码	状态码含义	说明
200	[OK]( <a href="https://tools.ietf.org/html/rfc7231#section-6.3.1">https://tools.ietf.org/html/rfc7231#section-6.3.1</a> )	成功

## 5 数据库系统设计

### 5.1 设计要求

- 1.数据库需要存储承运运力资源信息和车联网运力资源信息，包括车辆信息、司机信息、车辆状态、车辆位置、车辆能力等。
- 2.数据库需要存储货物运输运单信息，包括发货人信息、收货人信息、货物信息、运输产品信息、运输要求信息、运单状态等。
- 3.数据库需要存储运单调度管理系统中的运力池匹配管理信息，包括运力匹配结果、承运商信息、分配结果等。
- 4.数据库需要存储智能监控中心对运单的在途跟踪、服务质量管理及预警与事件管理信息，包括车辆位置、车辆状态、货物状态、服务质量信息、预警与事件信息等。
- 5.数据库需要支持多用户并发访问和操作，要求具有较高的性能和可靠性。

### 5.2 信息模型设计

- 1.承运商运力资源信息模型：该模型包括承运商信息、车辆信息、司机信息、车辆状态、车辆位置、车辆能力等属性，以及它们之间的关系。
- 2.车联网运力资源信息模型：该模型包括车辆信息、车辆状态、车辆位置、车辆能力等属性，以及它们之间的关系。
- 3.货物运输运单信息模型：该模型包括发货人信息、收货人信息、货物信息、运输产品信息、运输要求信息、运单状态等属性，以及它们之间的关系。
- 4.运单调度管理系统中的运力池匹配管理信息模型：该模型包括运力匹配结果、承运商信息、分配结果等属性，以及它们之间的关系。
- 5.智能监控中心对运单的在途跟踪、服务质量管理及预警与事件管理信息模型：该模型包括车辆位置、车辆状态、货物状态、服务质量信息、预警与事件信息等属性，以及它们之间的关系。

### 5.3 数据库设计

1.cars 表

Column	Type	Comment	PK	Nullable	Default
id	varchar(255)	Primary Key	YES	NO	
company_id	int			NO	
driver_id	int			YES	

car_age	int	车龄		YES	
limit_weight	int	载重		YES	

2.driver 表

Column	Type	Comment	PK	Nullable	Default
id	int	Primary Key	YES	NO	
drivername	varchar(255)	司机名字		YES	
phone	varchar(12)	司机手机号		YES	
company_id	int	所属承运商		YES	
sex	char(50)			YES	
statue	int	司机的状态		NO	0
age	int	年龄		YES	18
driving_age	int	驾龄		YES	

3.goods 表

Column	Type	Comment	PK	Nullable	Default
id	int	Primary Key	YES	NO	
create_time	datetime	Create Time		YES	
goods_description	varchar(255)	货物描述信息		NO	
begin_addr	varchar(255)	出发地址		NO	
dest_addr	varchar(255)	目的地		NO	
price	float	运输价格		NO	
consigner_id	int	发货人		NO	
dest_time	datetime	货物要求达到时间		NO	
demands	varchar(255)	运输要求		YES	

4.order 表

Column	Type	Comment	PK	Nullable	Default
id	int	Primary Key	YES	NO	
create_time	datetime	Create Time		NO	
price	float	order price		NO	
status	varchar(20)	order status		NO	
begin_addr	varchar(255)	出发地		NO	
dest_addr	varchar(255)	目的地		NO	
now_addr	varchar(255)	当前所在		YES	
goodsId	int	货品		NO	
companyId	int	承运商		NO	

driverId	int	司机		NO	
dest_time	datetime	货物到达时间		YES	
car_id	varchar(255)			NO	

5.role 表

Column	Type	Comment	PK	Nullable	Default
role_id	int	Primary Key	YES	NO	
role_key	varchar(20)	用户角色		YES	

6.user 表

Column	Type	Comment	PK	Nullable	Default
id	int	Primary Key	YES	NO	
username	varchar(255)	Username		NO	
password	varchar(255)	Password		NO	
email	varchar(30)	User email		NO	
sex	varchar(5)	User's sex		YES	
role_id	int	用户对应的角色		NO	

## 6 系统设计

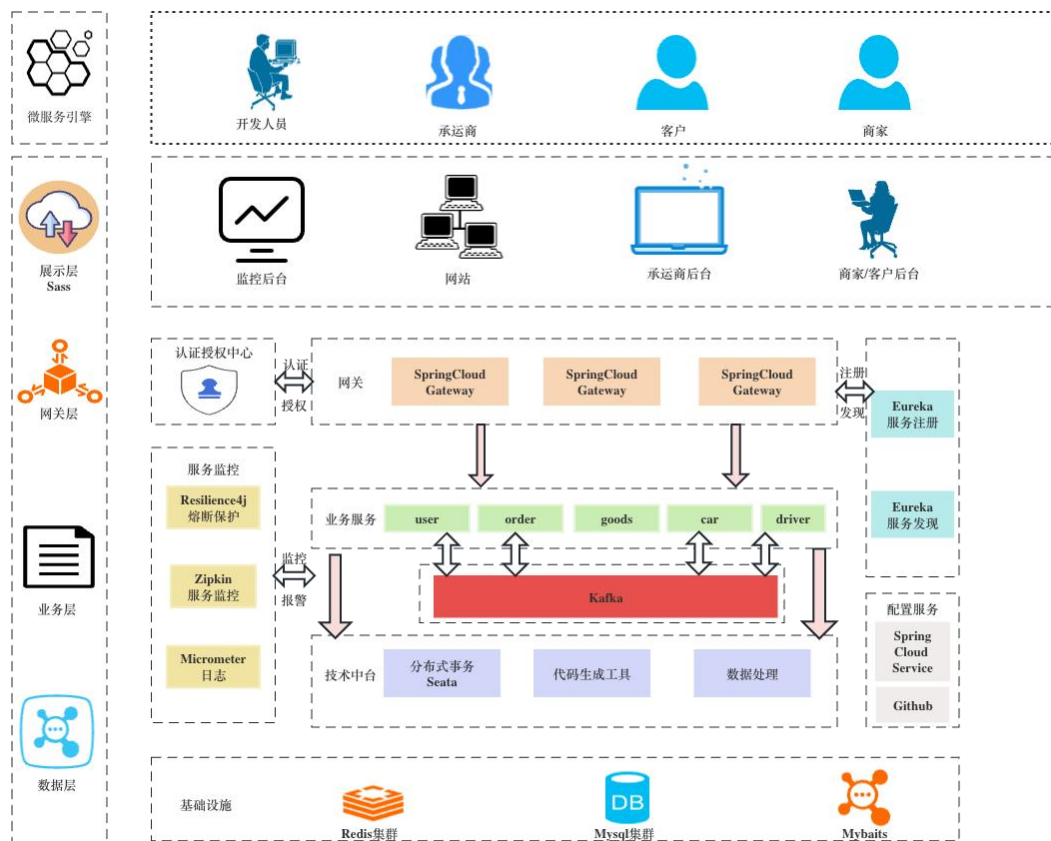


图 6.1 架构图

### 6.1 关键技术

1. Spring Boot: 使用 Spring Boot 作为后端开发框架，提供了快速构建和部署的能力。

2. Spring Data REST: 使用 Spring Data REST 简化 REST API 的创建和管理，可以基于领域模型自动生成 RESTful 服务。

3. Spring Security: 使用 Spring Security 实现 API 的认证和授权功能，确保 API 的安全性。

4. FastJson: 使用 FastJson 作为 JSON 序列化和反序列化库，处理 API 请求和响应的数据转换。

5. JWT (JSON Web Token): 使用 JWT 进行身份验证和令牌管理，实现无状态的身份认证。

6. Thymeleaf: 使用 Thymeleaf 作为前端模板引擎，生成动态 HTML 页面。

7. MyBatis: 使用 MyBatis 作为持久层框架，简化数据库操作和数据访问。

8. **Spring Session:** 使用 Spring Session 实现分布式会话管理，支持在集群环境中进行会话共享。
9. **Lombok:** 使用 Lombok 库简化 Java 代码的编写，通过注解自动生成常用的代码，如 Getter、Setter 等。
10. **MySQL Connector/J:** 使用 MySQL Connector/J 驱动程序连接 MySQL 数据库。
11. **Spring Boot DevTools:** 使用 Spring Boot DevTools 提供的开发工具，支持热部署和自动重启。
12. **Spring Boot Test:** 使用 Spring Boot Test 进行单元测试和集成测试。
13. **Spring Cloud:** 使用 Spring Cloud 对航运和运输管理服务器端进行重构，将航运和运输服务重构为微服务。
14. **Eureka:** 使用 Eureka 进行服务发现，以便微服务之间可以相互发现和调用。
15. **Resilience4j 或 Hystrix:** 使用 Resilience4j 或 Hystrix 实现断路器，以确保系统的可靠性和稳定性。
16. **Oauth2 授权服务器:** 集成 Oauth2 授权服务器，用于对用户进行身份验证和颁发访问令牌。
17. **网关:** 使用网关向外部用户暴露 API，以便外部用户可以访问系统的 API。
18. **Spring Cloud Config Server:** 使用 Spring Cloud Config Server 进行集中配置，以便可以集中管理系统的配置信息。
19. **Micrometer:** Micrometer 是一个用于应用程序度量和监控的度量库。它提供了通用的度量 API，并支持将度量数据发送到各种监控系统，例如 Prometheus、Graphite 等。
20. **Brave:** Brave 是一个分布式追踪解决方案，用于跟踪和监控跨多个服务的请求。它集成了 Zipkin 和 Micrometer，使开发人员能够进行分布式追踪和性能监控。
21. **Spring Kafka:** Spring Kafka 是 Spring Framework 对 Apache Kafka 的集成库，用于实现与 Kafka 消息队列的交互。
22. **Kafka Streams:** Kafka Streams 是 Apache Kafka 的一个客户端库，用于构建基于流处理的应用程序。它提供了对消息流的处理和转换的功能。

## 6.2 Spring Boot 框架开发

此代码实现了一个基于 Spring Boot 的运输和运输服务应用程序，使用了 MVC 架构和 RESTful API 设计风格。它通过 Controller 类处理 HTTP 请求，调用

Service 类处理业务逻辑，通过 Dao 类与 MySQL 数据库交互。同时，它还具备权限控制、统一的 API 响应格式、缓存和身份验证等功能，以提供安全、高效和可扩展的运输和运输服务。

1. 使用了 MVC (Model-View-Controller) 架构模式：代码中的各个 Controller 类负责处理不同的 HTTP 请求，并根据业务逻辑返回响应结果。这种分离了请求处理、业务逻辑和视图渲染的架构模式有助于代码的可维护性和可扩展性。

2. 使用了 RESTful API 设计：代码中的 Controller 类使用了 Spring 的注解(如 '@RestController'、'@RequestMapping') 来定义 RESTful 风格的 API 接口。每个 API 接口对应一个 URL 路径，通过不同的 HTTP 方法 (GET、POST、PUT 等) 实现不同的操作。

3. 使用了权限控制：代码中使用了 '@PreAuthorize' 注解来标识需要进行权限验证的接口，确保只有具有相应权限的用户才能访问。这种权限控制机制可以保护敏感数据和操作，提高系统的安全性。

4. 使用了 Service 层进行业务逻辑处理：代码中的 Controller 类通过注入相应的 Service 类来调用业务逻辑的方法，实现了控制器与业务逻辑的解耦。Service 类负责处理具体的业务逻辑，并与数据访问层 (如 Dao 类) 交互，完成相应的操作。

5. 使用了 ResponseEntity 类封装 API 响应结果：代码中定义了一个通用的 ResponseEntity 类，用于封装 API 接口的响应结果，包括状态码、消息和数据。这种统一的响应结果格式可以提高代码的可读性和一致性，并方便客户端解析。

6. 使用了 Redis 缓存和 JWT (JSON Web Token)：代码中使用了 RedisCache 来缓存用户信息，提高系统性能和访问速度。同时，使用了 JWT 来生成和解析身份验证令牌，实现了无状态的身份验证机制，简化了用户身份验证流程。

7. 使用了 MySQL 数据库：根据代码中的实体类和服务类，可以推断出使用了 MySQL 作为持久化存储数据库。代码中的服务类通过调用相应的 Dao 类来访问和操作数据库。

## 6.3 安全认证

### 6.3.1 SpringSecurity 和 JWT 的认证授权

使用 Spring Security 和 JWT 进行身份验证和授权的基本工作流程：

1. 用户进行身份验证：用户在登录页面输入用户名和密码。
2. 身份验证：应用程序使用 Spring Security 的身份验证机制验证用户提供的凭据。这可能涉及将凭据与数据库中的用户存储进行比较或与外部身份验证服务进行交互。

3. 生成 JWT: 如果用户凭据有效, 则应用程序生成一个 JWT, 并将其返回给客户端作为身份验证成功的响应。JWT 包含了用户的身份信息以及其他相关信息, 如过期时间和权限。
4. 客户端存储 JWT: 客户端 (通常是浏览器) 将收到的 JWT 保存在本地, 通常是在 Cookie 或本地存储中。
5. 后续请求中的授权: 在后续的请求中, 客户端将 JWT 包含在请求的标头或其他适当的位置中。这允许服务器对请求进行授权并验证用户身份。
6. 服务器验证和解析 JWT: 服务器使用与生成 JWT 时相同的密钥对 JWT 进行验证和解析。这确保 JWT 没有被篡改或伪造, 并且仍然有效。
7. 授权和访问控制: 服务器根据 JWT 中的用户身份信息和权限进行授权和访问控制决策。这可能涉及检查用户是否具有请求资源的适当权限。

使用 Spring Security 和 JWT 进行身份验证和授权的好处包括:

1. 无状态: JWT 是无状态的, 服务器不需要在会话或数据库中存储任何用户状态。这使得应用程序可以更容易地水平扩展, 并使负载均衡变得更简单。
2. 松耦合: JWT 允许应用程序和身份验证服务之间的松耦合。应用程序只需验证 JWT 的有效性, 而不需要直接与身份验证服务进行通信。
3. 可扩展性: 通过使用 JWT, 您可以将身份验证和授权机制与其他服务 (如单点登录) 进行集成, 从而提供更强大和可扩展的解决方案。

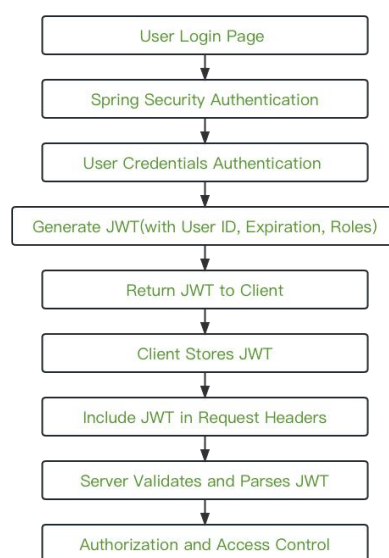


图 6.2 安全认证流程图

### 6.3.2 保护敏感数据和防止安全漏洞 —— 密码加密

`new BCryptPasswordEncoder()` 是使用 Spring Security 提供的 `BCryptPasswordEncoder` 类创建一个密码编码器的示例。 `BCryptPasswordEncoder`



是一种密码哈希算法，它结合了哈希函数和盐值，提供了安全的密码存储和验证机制。

```
19      * spring security配置
20      */
21      @Configuration
22      @EnableWebSecurity
23      @EnableGlobalMethodSecurity(prePostEnabled = true)
24      public class WebSecurityConfig{
25          @Autowired
26          private AuthenticationConfiguration authenticationConfiguration;
27
28          @Autowired          通过 BCryptPasswordEncoder() 函数来实现加密解密
29          private JwtAuthenticationFilter jwtAuthenticationFilter;
30
31          @Bean
32          public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

图 6.3 密码加密对应代码图

## 7 微服务架构、Spring Cloud 和 Kafka 实现

### 7.1 微服务架构概述

微服务架构是一种将应用程序拆分成一组小型、独立的服务的架构风格。每个服务都运行在自己的进程中，并使用轻量级的通信机制（如 HTTP API）进行通信。这种架构风格的优势在于可以将应用程序拆分成更小的、更易于管理的部分，从而提高应用程序的可伸缩性、可靠性和可维护性。

### 7.2 架构设计

1. 授权服务微服务 (authorization\_service):
  - 负责身份验证和授权功能。
  - 集成了 OAuth2 授权服务器，用于对用户进行身份验证和颁发访问令牌。
  - 提供 API 接口，处理用户认证、访问令牌生成和验证等操作。
2. 车辆和司机服务微服务 (car\_driver\_service):
  - 负责管理公司名下的车辆和司机信息。
  - 提供 API 接口，用于获取公司名下所有车辆和司机的信息。
  - 提供新增司机和车辆的功能，允许用户添加新的司机和车辆信息。
3. 网关服务微服务 (gateway\_service):
  - 作为系统的入口，负责接收外部用户的请求并将其路由到相应的微服务。
  - 提供 API 的外部访问接口，处理路由、负载均衡和安全性等方面的功能。
4. 商品服务微服务 (goods\_service):
  - 负责商家的商品管理。
  - 提供 API 接口，用于新增商家的商品。
  - 可能包括其他商品相关的功能，如查询、修改和删除商品等。
5. 订单服务微服务 (order\_service):
  - 负责处理订单相关的操作。
  - 提供 API 接口，用于创建订单、更新订单状态和所在地址。
  - 包括其他订单管理功能，如查询订单、取消订单等。
6. 注册服务微服务 (register\_service):
  - 负责用户注册功能。
  - 提供 API 接口，用于注册新用户。
  - 可能包括验证注册信息、发送确认邮件等相关功能。
7. 用户服务微服务 (user\_service):
  - 负责用户管理功能。
  - 提供 API 接口，用于用户登录、获取用户信息等操作。

- 包括用户资料修改、密码重置等功能。

以上的微服务模块通过 Spring Cloud 实现了微服务架构。它们相互协作，通过服务注册与发现实现了松耦合的通信方式。网关服务作为整个系统的入口，接收外部用户的请求并将其路由到相应的微服务，同时处理路由、负载均衡和安全性功能。授权服务负责用户身份验证和授权功能，通过集成 OAuth2 授权服务器来验证用户身份并颁发访问令牌。其他微服务负责具体的业务功能，如车辆和司机管理、商品管理、订单管理等，提供相应的 API 接口供其他服务调用。

### 7.3 Spring Cloud 集成

在该系统中，使用了 Spring Cloud 作为微服务架构的支持。下面是具体的 Spring Cloud 集成内容：

- 服务注册与发现：使用 Eureka 作为服务注册与发现组件。每个微服务在启动时会向 Eureka 注册自己的信息，包括服务名称、实例地址等。其他微服务可以通过 Eureka 发现和调用注册的微服务，实现了服务之间的解耦和动态扩展。
- 断路器模式：使用 Resilience4j 或 Hystrix 实现断路器模式。断路器模式可以提高系统的容错能力，在服务故障或不可用的情况下，通过断路器进行降级处理，避免级联故障。通过配置断路器的阈值和熔断策略，可以有效保护系统的稳定性和可靠性。
- 配置中心：使用 Spring Cloud Config Server 实现集中配置管理。配置中心允许将微服务的配置集中存储和管理，提供了动态更新配置的能力。通过集中管理配置，可以实现配置的版本控制、环境隔离和动态刷新等功能，提高了系统的可维护性和灵活性。
- 链路跟踪：使用 Spring Cloud Sleuth 进行链路跟踪。链路跟踪可以追踪和记录微服务间的请求流程和调用关系，提供了全局的请求追踪能力。通过集成 Sleuth，可以对请求进行唯一标识和跟踪，并生成请求链路的日志和监控数据，帮助排查和分析系统的性能问题。

### 7.4 网关实现

在该系统中，使用了网关作为系统的入口，负责接收外部用户的请求并将其路由到相应的微服务。下面是网关实现的具体内容：

- API 路由和负载均衡：网关使用 Spring Cloud Gateway 或 Netflix Zuul 进行 API 路由和负载均衡。通过配置路由规则，网关可以将外部请求路由到相应的微服务实例上，实现请求的转发和负载均衡。网关可以根据请求的路径、方法、头部等条件进行路由规则的匹配和选择。

- **安全认证和授权：**网关可以使用 **Spring Security** 等机制对外部请求进行安全认证和授权。通过配置认证和授权规则，网关可以限制只有经过身份验证和授权的用户才能访问特定的 **API**。这样可以确保系统的安全性，防止未经授权的访问和恶意攻击。
- **请求过滤和处理：**网关可以使用过滤器对请求进行预处理和后处理。通过编写自定义的过滤器，可以对请求进行验证、日志记录、请求转换等操作。这样可以在请求到达微服务之前对请求进行统一的处理，减轻微服务的处理压力，并可以在请求返回时对响应进行统一的处理。
- **服务降级和熔断：**网关可以实现服务降级和熔断机制，以应对微服务的故障或不可用情况。通过配置熔断规则，当微服务出现故障或超时，网关可以返回预定义的响应或转发到备用服务，确保系统的稳定性和可用性。
- **请求聚合和转换：**网关可以聚合多个微服务的请求并返回合并后的响应，以提高系统性能和减少客户端的请求次数。网关可以根据请求的需要，将多个微服务的请求合并为一个请求，并将结果转换为客户端期望的格式返回。

## 7.5 Kafka 实现

在该系统中，使用 **Kafka** 来连接用户、订单、车辆和司机微服务，以增加系统的功能和丰富性：

1. **使用 Kafka 作为消息队列：**可以将用户、订单、车辆和司机微服务之间的通信通过 **Kafka** 进行解耦。当一个微服务产生了重要的事件或数据更新时，可以将该消息发布到 **Kafka** 的相关主题中，而其他微服务则可以通过订阅这些主题来接收并处理这些消息。

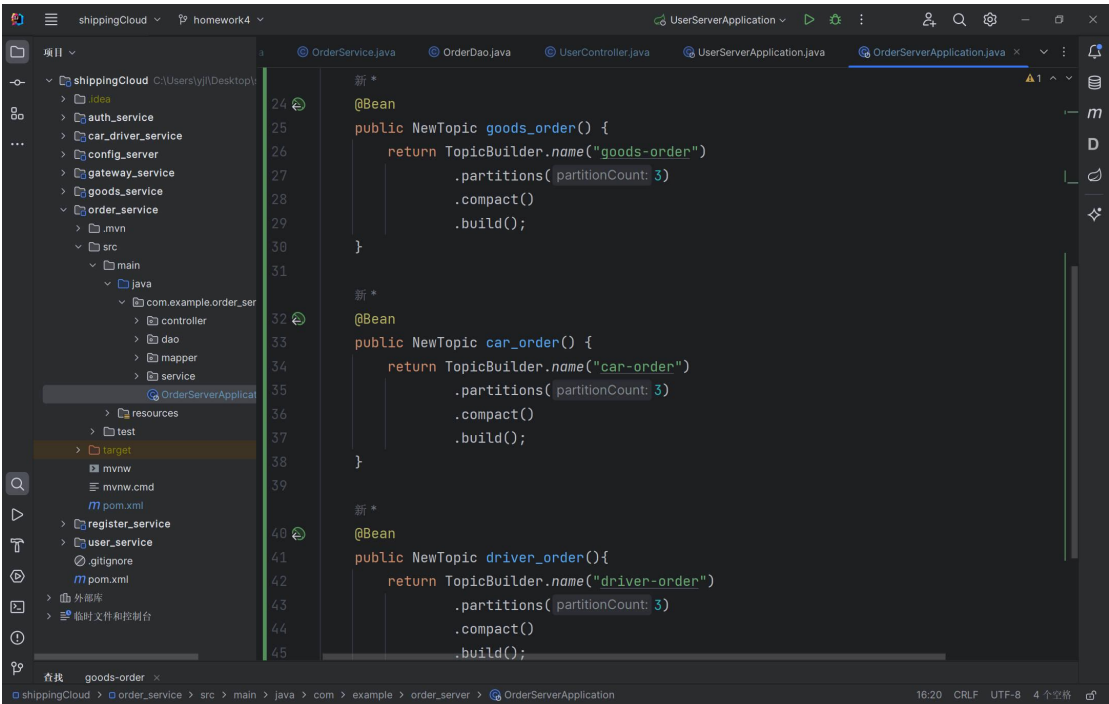
2. **用户注册和订单创建事件：**当用户注册成功时，注册服务微服务可以发布一个用户注册事件到 **Kafka** 中，包含用户的相关信息。订单服务微服务可以通过订阅用户注册事件，自动为新注册用户创建一个默认订单，并进行后续处理。

3. **车辆和司机信息更新：**当车辆和司机服务微服务更新了车辆或司机的信息时，可以将更新后的信息发布到 **Kafka** 中。其他依赖这些信息的微服务，如订单服务微服务，可以通过订阅相应的主题，获取最新的车辆和司机信息。

4. **实时订单状态更新：**订单服务微服务可以将订单的状态更新信息发布到 **Kafka** 中。其他微服务可以通过订阅订单状态更新事件，实时获取订单的最新状态，并进行相应的处理，如通知相关用户或更新相关数据。

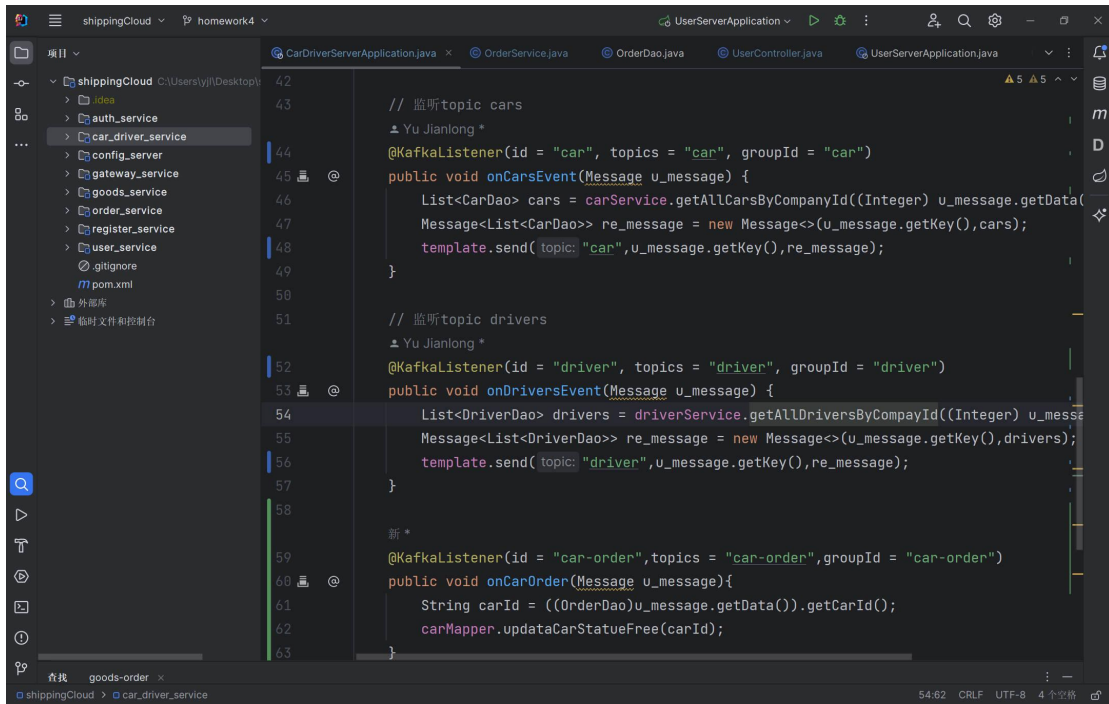
5. 数据分析和统计：通过将关键的业务事件和数据更新发布到 **Kafka** 中，可以构建实时的数据分析和统计系统。使用消费者来订阅相应的主题，对数据进行处理和分析，生成实时的报表、指标和洞察，帮助业务决策和优化。

通过使用 **Kafka** 作为消息队列，实现微服务之间的解耦和异步通信。将关键的事件和数据更新发布到 **Kafka** 中，可以实现实时性和可靠性，并为系统增加更多的功能和丰富性。



```
24 @Bean
25 public NewTopic goods_order() {
26     return TopicBuilder.name("goods-order")
27         .partitions(partitionCount: 3)
28         .compact()
29         .build();
30 }
31
32 @Bean
33 public NewTopic car_order() {
34     return TopicBuilder.name("car-order")
35         .partitions(partitionCount: 3)
36         .compact()
37         .build();
38 }
39
40 @Bean
41 public NewTopic driver_order(){
42     return TopicBuilder.name("driver-order")
43         .partitions(partitionCount: 3)
44         .compact()
45         .build();
46 }
```

图 7.1 Kafka 生产者



```
42 // 监听topic cars
43 // Yu Jianlong *
44 @KafkaListener(id = "car", topics = "car", groupId = "car")
45 public void onCarsEvent(Message u_message) {
46     List<CarDao> cars = carService.getAllCarsByCompanyId((Integer) u_message.getData());
47     Message<List<CarDao>> re_message = new Message<>(u_message.getKey(), cars);
48     template.send(topic: "car", u_message.getKey(), re_message);
49 }
50
51 // 监听topic drivers
52 // Yu Jianlong *
53 @KafkaListener(id = "driver", topics = "driver", groupId = "driver")
54 public void onDriversEvent(Message u_message) {
55     List<DriverDao> drivers = driverService.getAllDriversByCompanyId((Integer) u_message.getData());
56     Message<List<DriverDao>> re_message = new Message<>(u_message.getKey(), drivers);
57     template.send(topic: "driver", u_message.getKey(), re_message);
58 }
59
60 // 监听topic car-order
61 // Yu Jianlong *
62 @KafkaListener(id = "car-order", topics = "car-order", groupId = "car-order")
63 public void onCarOrder(Message u_message){
64     String carId = ((OrderDao)u_message.getData()).getCarId();
65     carMapper.updateCarStatueFree(carId);
66 }
```

图 7.2 Kafka 消费者

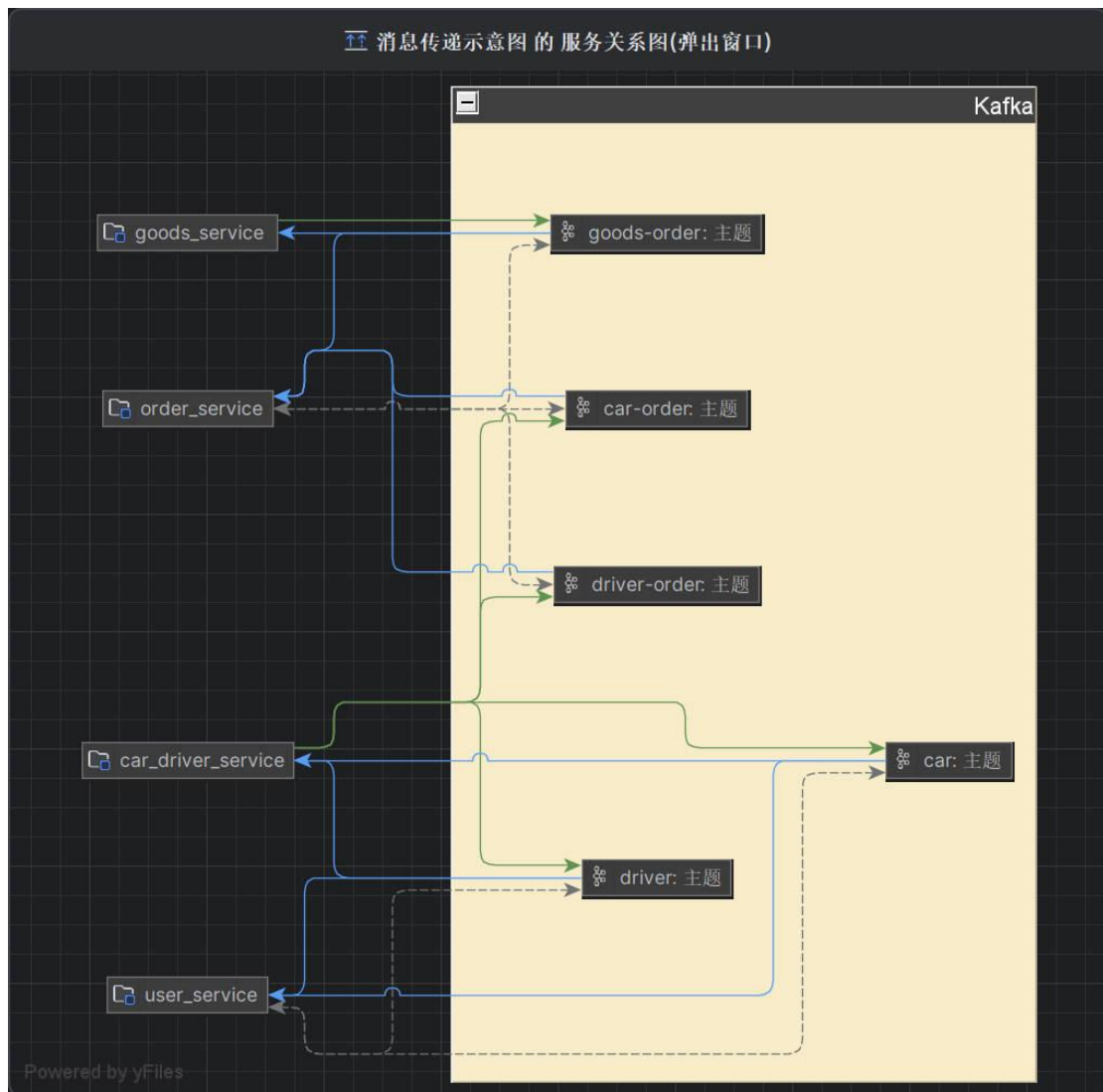


图 7.3 Kafka 消息传递关系示意图

## 7.6 配置与集成

```
@EnableDiscoveryClient
@EnableConfigServer
@SpringBootApplication
public class ConfigServiceApplication {

    Yu Jianlong
    public static void main(String[] args) { SpringApplication.run(ConfigServiceApplicati
}
}
```

图 7.4 Config 服务器

```

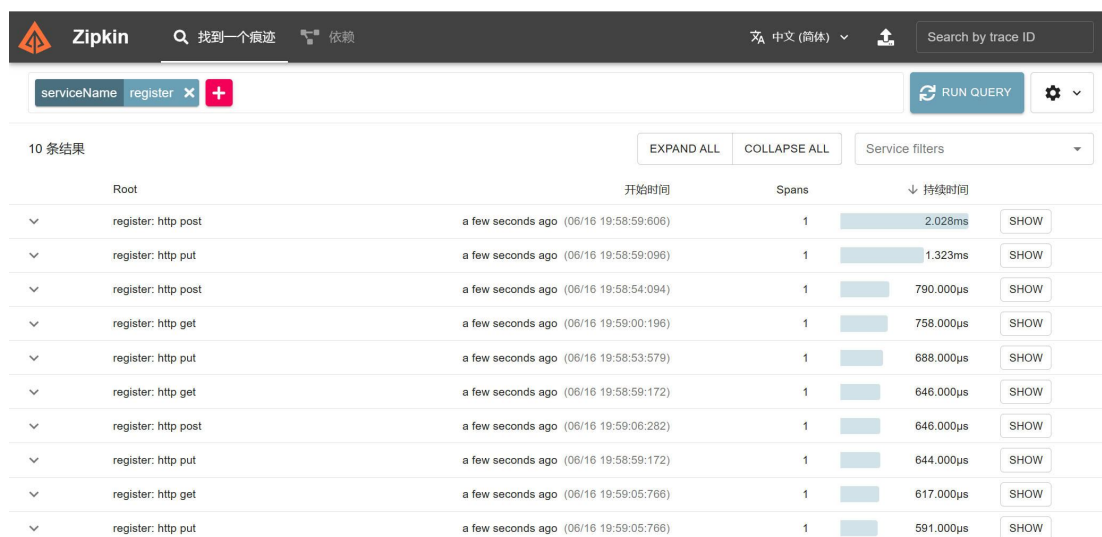
server:
  port: 8888

spring:
  application:
    name: config-server
  profiles:
    active: git
  cloud:
    config:
      server:
        git:
          uri: git@github.com:yjl-001/configuration.git
          default-label: main

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8080/eureka/

```

图 7.5 Config 服务器配置



Root	开始时间	Spans	持续时间
register: http post	a few seconds ago (06/16 19:58:59:606)	1	2.028ms
register: http put	a few seconds ago (06/16 19:58:59:096)	1	1.323ms
register: http post	a few seconds ago (06/16 19:58:54:094)	1	790.000µs
register: http get	a few seconds ago (06/16 19:59:00:196)	1	758.000µs
register: http put	a few seconds ago (06/16 19:58:53:579)	1	688.000µs
register: http get	a few seconds ago (06/16 19:58:59:172)	1	646.000µs
register: http post	a few seconds ago (06/16 19:59:06:282)	1	646.000µs
register: http put	a few seconds ago (06/16 19:58:59:172)	1	644.000µs
register: http get	a few seconds ago (06/16 19:59:05:766)	1	617.000µs
register: http put	a few seconds ago (06/16 19:59:05:766)	1	591.000µs

图 7.6 trace 与 zipkin 集成

## 8 总结

在本次迭代开发中，我们通过引入 **Kafka**，进一步增强了系统的可靠性、实时性和扩展性。它为微服务架构提供了高效的消息传递机制，并使得各个微服务能够以异步和并行的方式进行通信和处理。同时，**Kafka** 的持久化和回放特性也提供了数据的可靠性和灵活性。