JavaEE2024-WMS-stage4

文档摘要（强烈建议使用飞书链接查看文档JavaEE2024-WMS-stage4）

> 💡 我们完成了当前阶段（第四阶段）中的所有basic和credit要求（详见要求文档）。由于功能需求再次产生了一些变化，因此各层的设计均进行了适当的调整（相较于前几次）。此文档说明了我们在这个阶段中完成的各项内容。
>
> 我们在最开始配置Spring Cloud Sleuth时发现其版本与 Spring Cloud 版本不兼容。Sleuth 3.1 版要求迁移到 Micrometer Tracing 而不是 Sleuth，我们对不同版本的提供了两种方案，老版本和新版本。

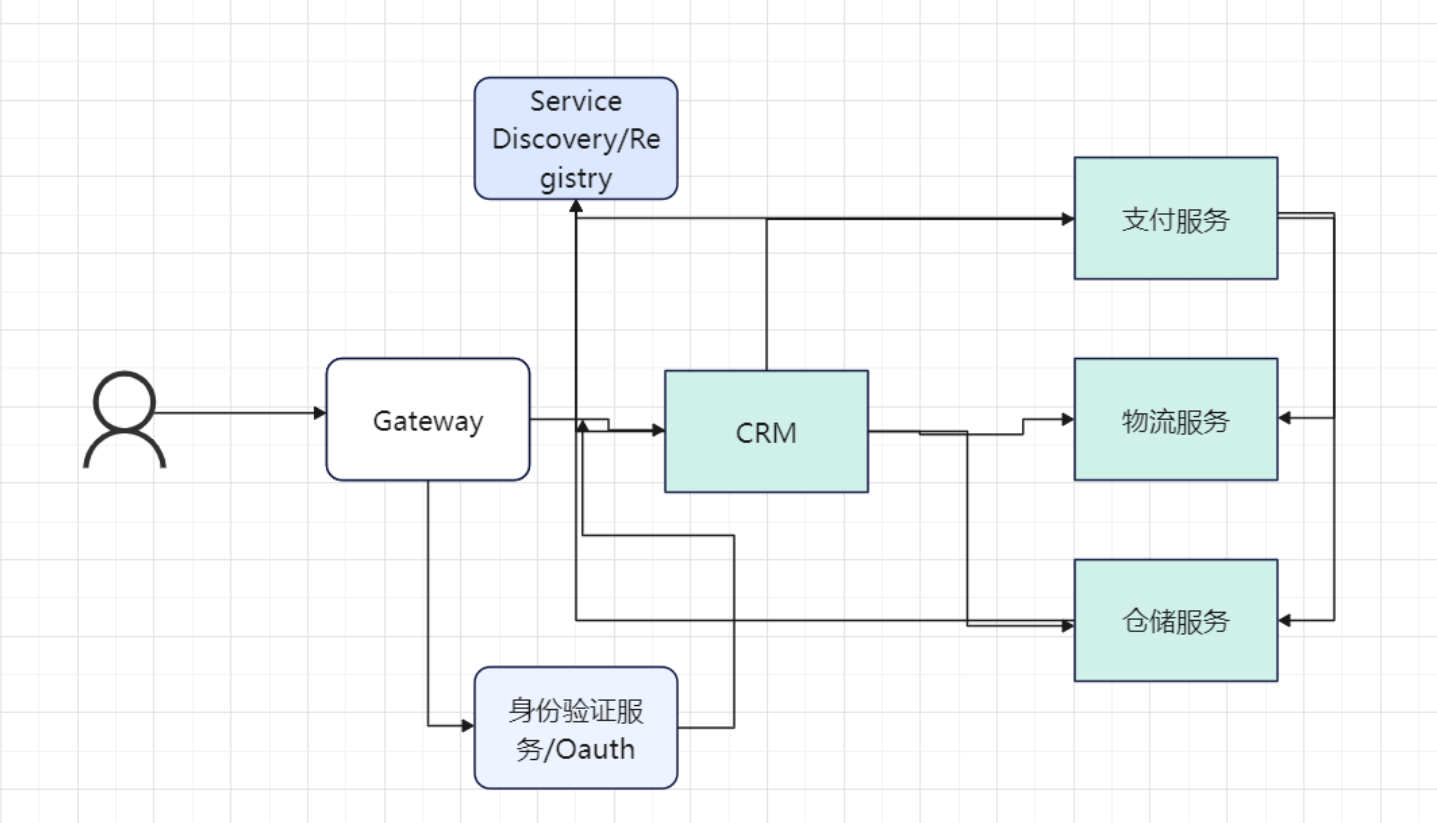
在本阶段中，我们开发了一个微服务系统，该系统集成了多个服务，例如CRM服务、物流服务、自动仓库服务和支付服务。例如，当客户下订单将一批货物存储到仓库时，CRM服务将调用相应的物流服务、仓库服务和支付服务来完成服务交易。为了开发这个系统，我们还需要添加微服务基础设施，如服务发现、网关和OAuth服务器。

下面的文档将说明我们的系统结构，逻辑结构，其他设计等，并列举关键部分代码，以及每部分的实现结果截图（包括但不限于swagger api文档、postman测试截图、nacos服务运行时截图、service-tracking截图等）

一、要求完成说明

我们完成了当前阶段（第四阶段）中的所有basic和credit要求（详见要求文档），详细说明如下。

1.1 Develop a toy microservices system using Spring Cloud infrastructures.

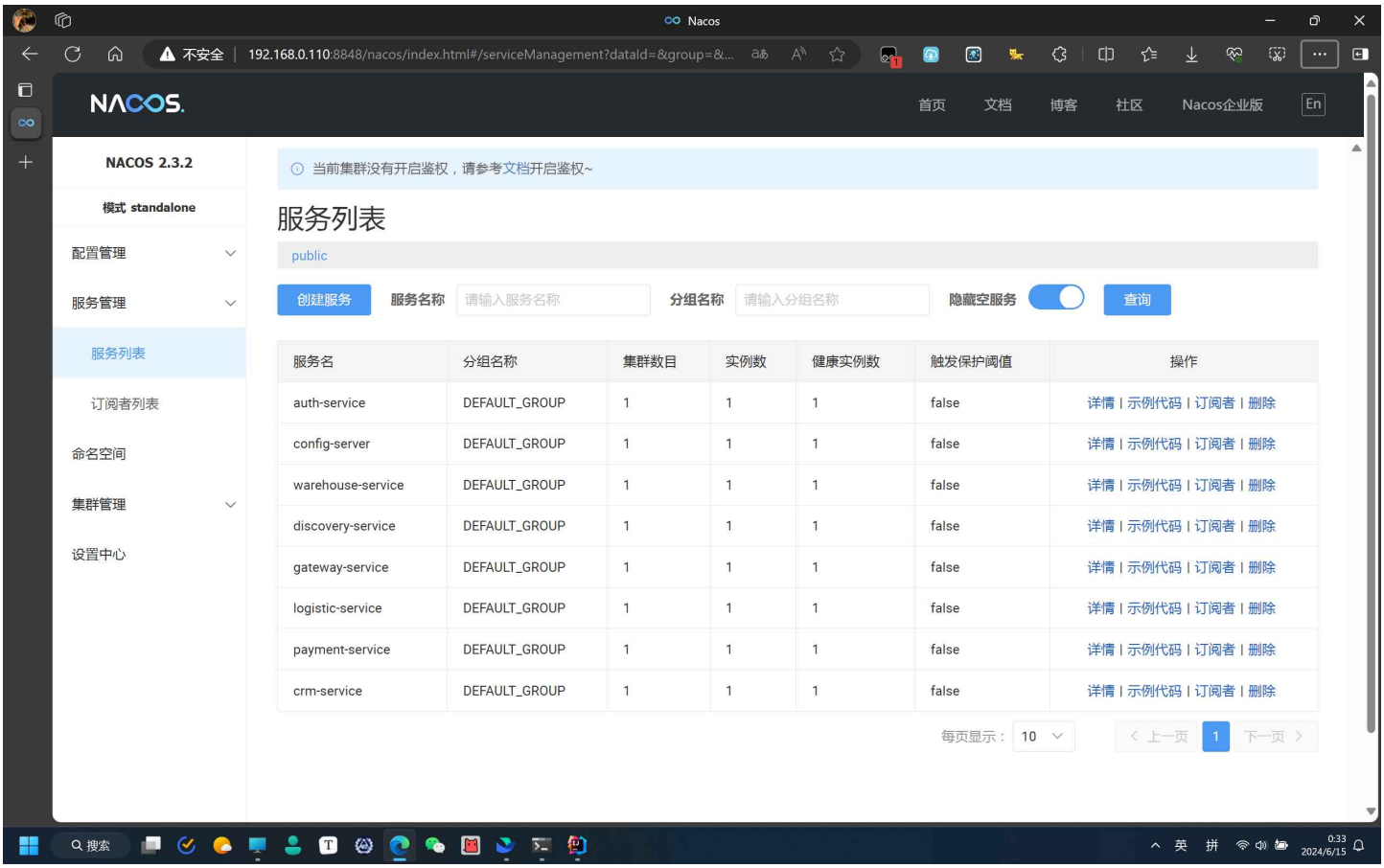


我们使用Spring Cloud基础设施开发了一个玩具微服务系统。系统包含以下主要服务：

- `crm-service`：客户关系管理服务
- `logistic-service`：物流服务
- `warehouse-service`：仓库服务
- `payment-service`：支付服务
- `auth-service`：OAuth2身份验证服务
- `gateway-service`：API网关
- `config-server`：配置服务器
- `discovery-service`：服务发现和注册中心，使用Nacos

1.2 Service discovery with Eureka/Alibaba Nacos is necessary.

我们使用Nacos作为服务发现和注册中心。所有服务都已成功注册到Nacos中，以便进行服务发现和调用。

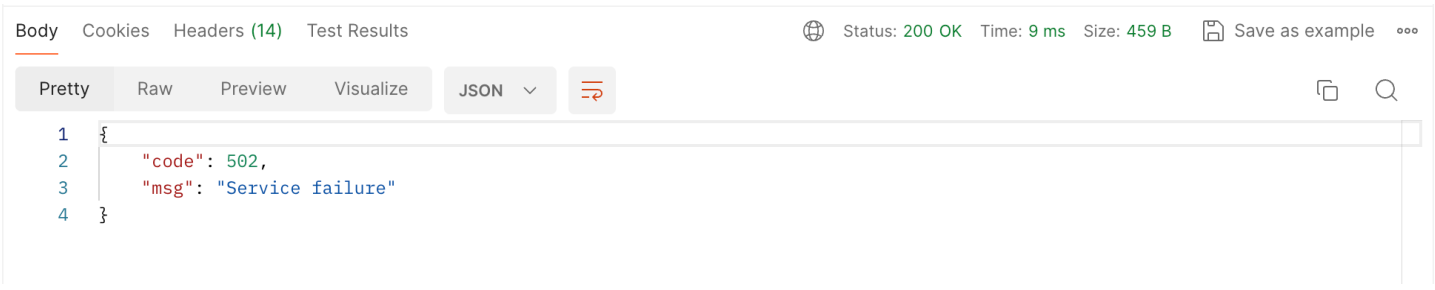


1.3 (Optional) Circuit breaker implementation with Resilience4j or Hystrix.

我们在每个微服务中实现了Resilience4j断路器，以提高系统的可靠性和容错性。

1.3.1 实现效果

我们触发了failureRateThreshold的上限，得到如下返回结果，表示我们成功配置了Resilience4j断路器。



1.3.2 实现细节

配置和代码示例如下：

以Logistic Service为例

在 `logistic-service` 的 `pom.xml` 中添加依赖：

```
<dependency>
```

```
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-spring-boot2</artifactId>
    <version>1.7.1</version>
</dependency>
```

在 `application.properties` 中配置：

```
resilience4j.circuitbreaker.instances.default.slidingWindowSize=100
resilience4j.circuitbreaker.instances.default.failureRateThreshold=50
resilience4j.circuitbreaker.instances.default.waitDurationInOpenState=10000
```

在服务类中使用断路器注解：

```
package com.example.logisticservice.service;

import com.example.logisticservice.model.Logistic;
import com.example.logisticservice.repository.LogisticRepository;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Servicepublic class LogisticService {@Autowiredprivate LogisticRepository repository;
@CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public List<Logistic> findAll() {return repository.findAll();
    }
public Logistic save(Logistic logistic) {return repository.save(logistic);
    }
public void delete(Long id) {
        repository.deleteById(id);
    }
public Logistic update(Long id, Logistic logistic) {Logistic existingLogistic = repository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Logistic not found"));
        existingLogistic.setDescription(logistic.getDescription());
        existingLogistic.setStatus(logistic.getStatus());return repository.save(existingLogistic);
    }
public List<Logistic> fallbackMethod(Throwable throwable) {// 自定义的fallback逻辑
```

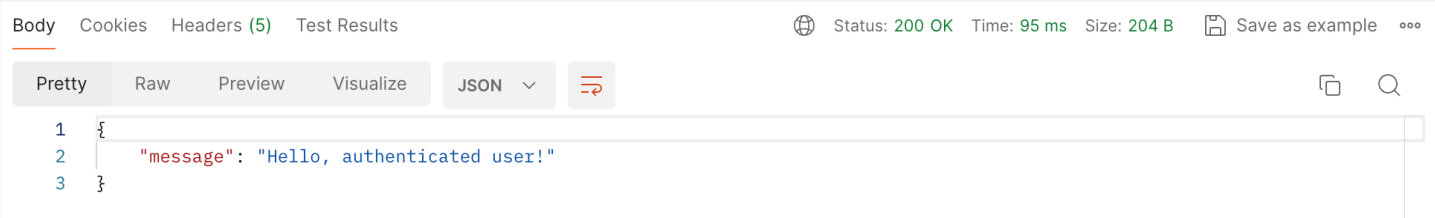
```
        System.out.println("Fallback method called due to: " +
    throwable.getMessage());return List.of();
    }
}
```

1.4 (Optional) Oauth2 authorization server integrated.

我们集成了OAuth2授权服务器，使用Spring Security和Spring Authorization Server实现。

1.4.1 实现效果

Oauth2 authorization 验证通过后postman返回如下图所示：



1.4.2 实现细节

配置和代码示例如下：

`auth-service` 的 `pom.xml`

```
<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.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-oauth2-authorization-server</artifactId>
        <version>0.4.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

```
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version></dependency>
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.2.3.Final</version>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>
</dependencies>
```

`application.properties`

```
server.port=8085
spring.application.name=auth-service
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

`AuthServiceApplication.java`

```
package com.example.authservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication@EnableDiscoveryClientpublic class AuthServiceApplication
 {public static void main(String[] args) {
        SpringApplication.run(AuthServiceApplication.class, args);
    }
}
```

AuthorizationServerConfig.java

```
1  package com.example.authservice.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.authentication.AuthenticationManager;
6  import
   org.springframework.security.config.annotation.web.configuration.EnableAuthoriz
   ationServer;
7  import
   org.springframework.security.oauth2.config.annotation.web.configuration.Authori
   zationServerConfigurerAdapter;
8  import
   org.springframework.security.oauth2.config.annotation.web.configurers.Authoriza
   tionServerEndpointsConfigurer;
9  import
   org.springframework.security.oauth2.config.annotation.web.configurers.Authoriza
   tionServerSecurityConfigurer;
10 import
   org.springframework.security.oauth2.config.annotation.web.configurers.ClientDet
   ailsServiceConfigurer;
11 import org.springframework.security.core.userdetails.User;
12 import org.springframework.security.core.userdetails.UserDetails;
13 import org.springframework.security.core.userdetails.UserDetailsManager;
14 import
   org.springframework.security.core.userdetails.memory.InMemoryUserDetailsManager
   ;
15 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
16
17 @Configuration@EnableAuthorizationServerpublic class AuthorizationServerConfig
   extends AuthorizationServerConfigurerAdapter {
18 @Beanpublic AuthenticationManager authenticationManager() throws Exception
   {return super.authenticationManager();
19     }
20 @Overridepublic void configure(ClientDetailsServiceConfigurer clients) throws
   Exception {
21         clients.inMemory()
22                 .withClient("my-client-id")
23                 .secret(new BCryptPasswordEncoder().encode("my-client-secret"))
24                 .scopes("read", "write")
25                 .authorizedGrantTypes("password", "refresh_token")
26                 .accessTokenValiditySeconds(3600)
27                 .refreshTokenValiditySeconds(7200);
28     }
```

```
@Overridepublic void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints
                .authenticationManager(authenticationManager())
                .userDetailsService(userDetailsService());
    }
@Overridepublic void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
        security
                .tokenKeyAccess("permitAll()")
                .checkTokenAccess("isAuthenticated()");
    }
@Beanpublic UserDetailsService userDetailsService() {UserDetails user = User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build();return new InMemoryUserDetailsManager(user);
    }
}
```

1.5 (Optional) Expose API to external users with Gateway.

我们使用Spring Cloud Gateway实现了API网关，提供统一的API入口。配置和代码示例如下：

`gateway-service` 的 `pom.xml`

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>
</dependencies>
```

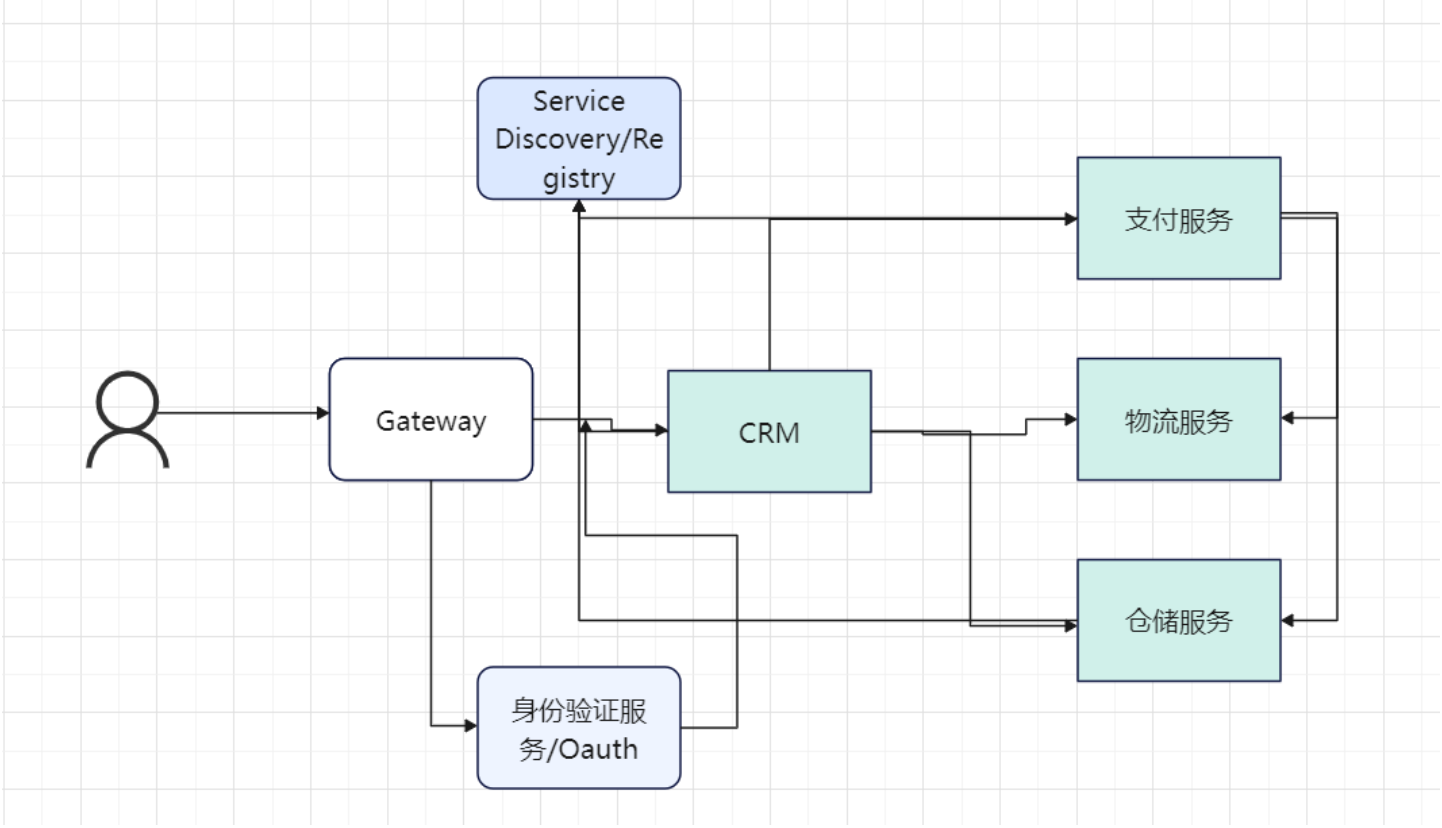
`application.yml`

```yaml
server:
  port: 8081

spring:
  application:
    name: gateway-service
  cloud:
    compatibility-verifier:
      enabled: true
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
    gateway:
      routes:
        - id: crm-service
          uri: lb://crm-service
          predicates:
            - Path=/crm/**
          filters:
            - StripPrefix=1

        - id: logistic-service
          uri: lb://logistic-service
          predicates:
            - Path=/logistic/**
          filters:
            - StripPrefix=1

        - id: warehouse-service
          uri: lb://warehouse-service
          predicates:
            - Path=/warehouse/**
          filters:
            - StripPrefix=1

        - id: gateway-service
          uri: lb://gateway-service
          predicates:
            - Path=/**
          filters:
            - StripPrefix=1

        - id: discovery-service
          uri: lb://discovery-service
          predicates:
            - Path=/discovery/**
          filters:
```

```
48            - StripPrefix=1
49
50        - id: config-service
51          uri: lb://config-service
52          predicates:
53            - Path=/config/**
54          filters:
55            - StripPrefix=1
56
57        - id: payment-service
58          uri: lb://payment-service
59          predicates:
60            - Path=/payment/**
61          filters:
62            - StripPrefix=1
63
64        - id: auth-service
65          uri: lb://auth-service
66          predicates:
67            - Path=/auth/**
68          filters:
69            - StripPrefix=1
70
71  zipkin:
72    base-url: http://localhost:9411
73    discovery-client-enabled: false
74  sleuth:
75    sampler:
76      probability: 1.0
```

1.6 (Optional) Centralized configuration and tracking with Spring Cloud Config Server and Sleuth.

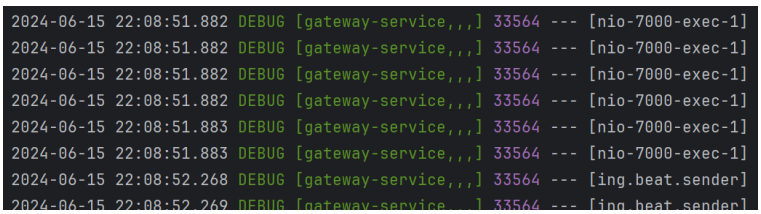
1.6.1 实现效果



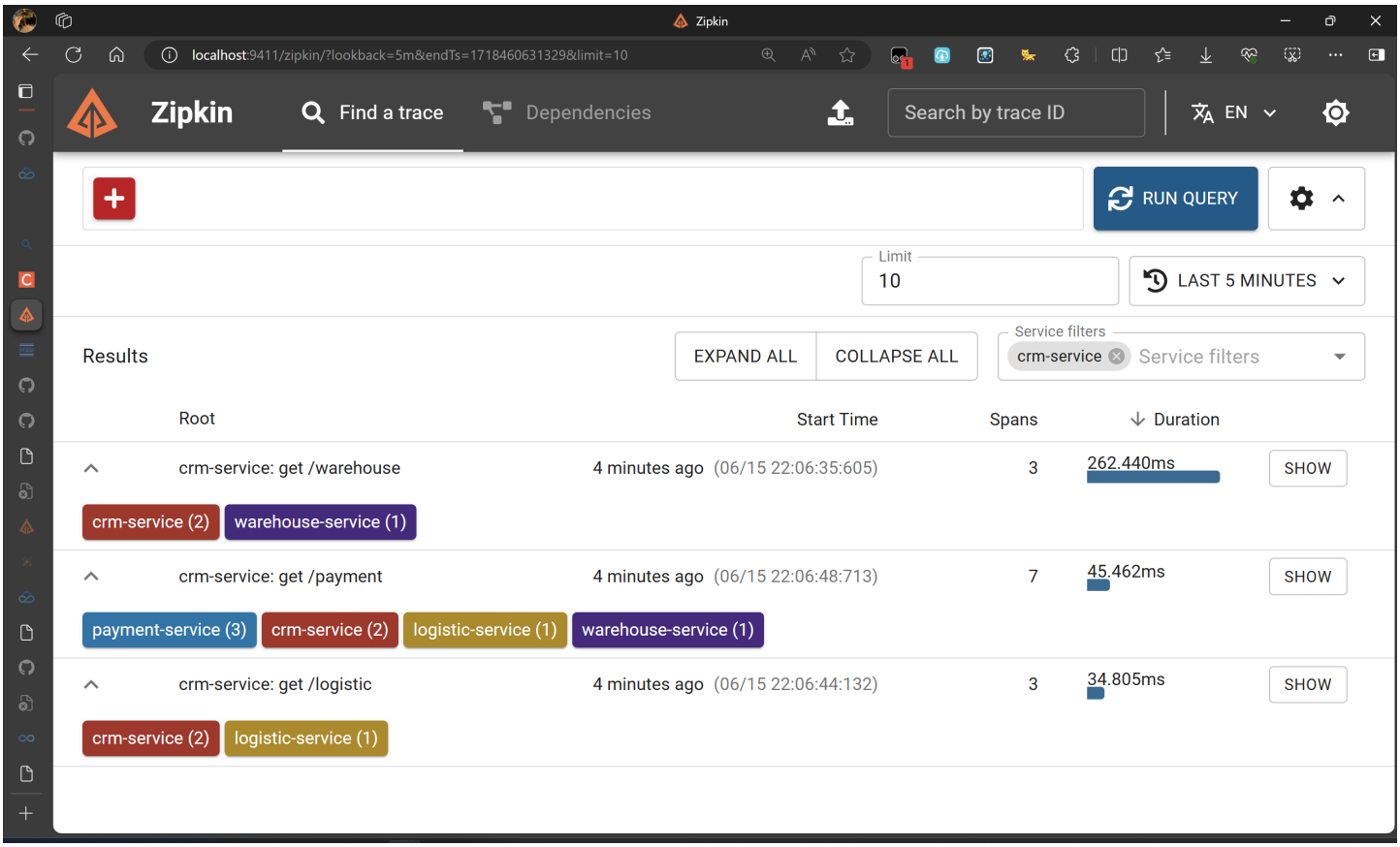
对于我们实现的7个微服务（此处未显示配置服务）`discovery-service`、`gateway-service`、`crm-service`、`logistic-service`、`warehouse-service`、`payment-service`和`auth-service`。

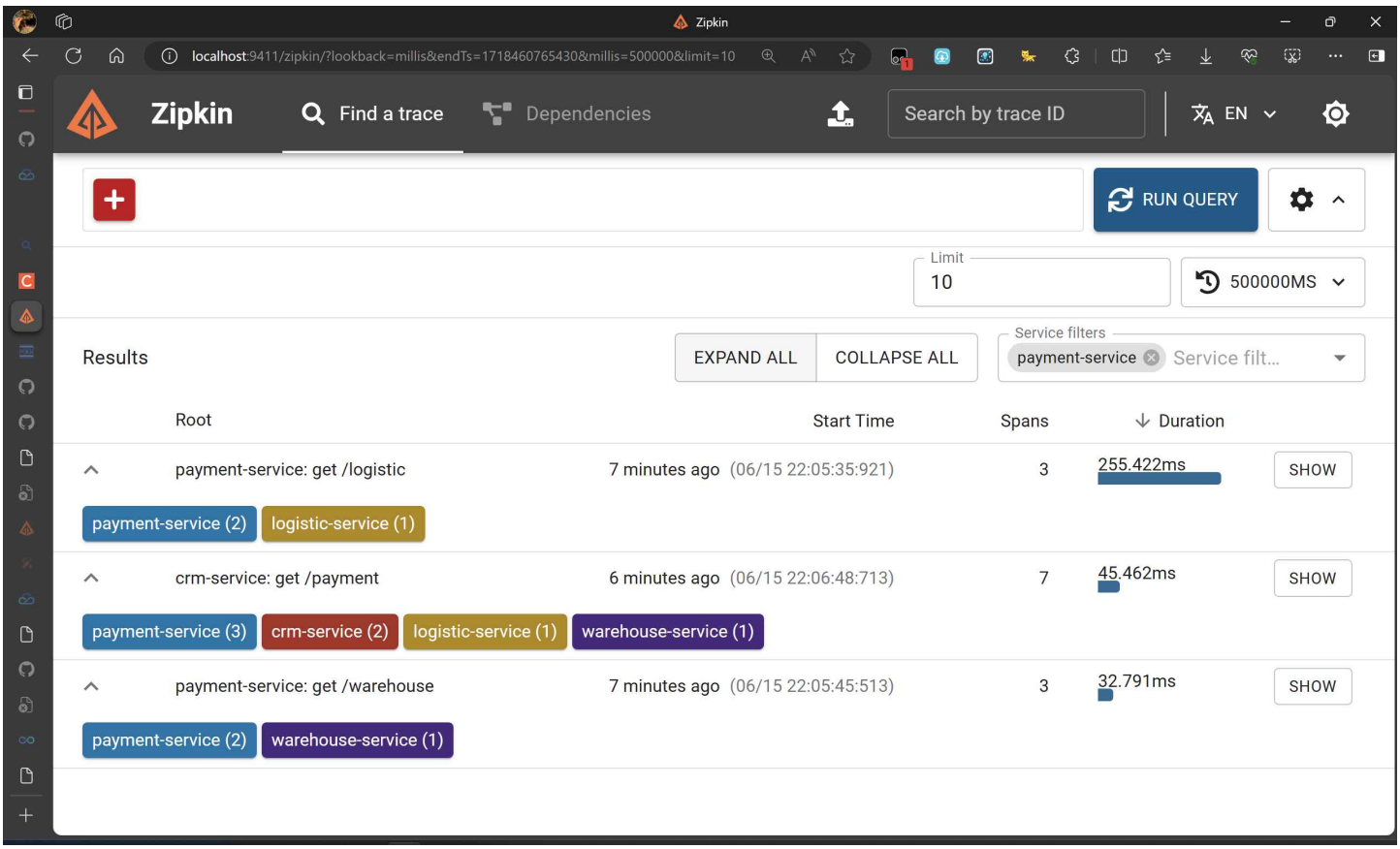
我们以如下6个调用为例，体现**应用了Sleuth与Zipkin的微服务链路追踪：**

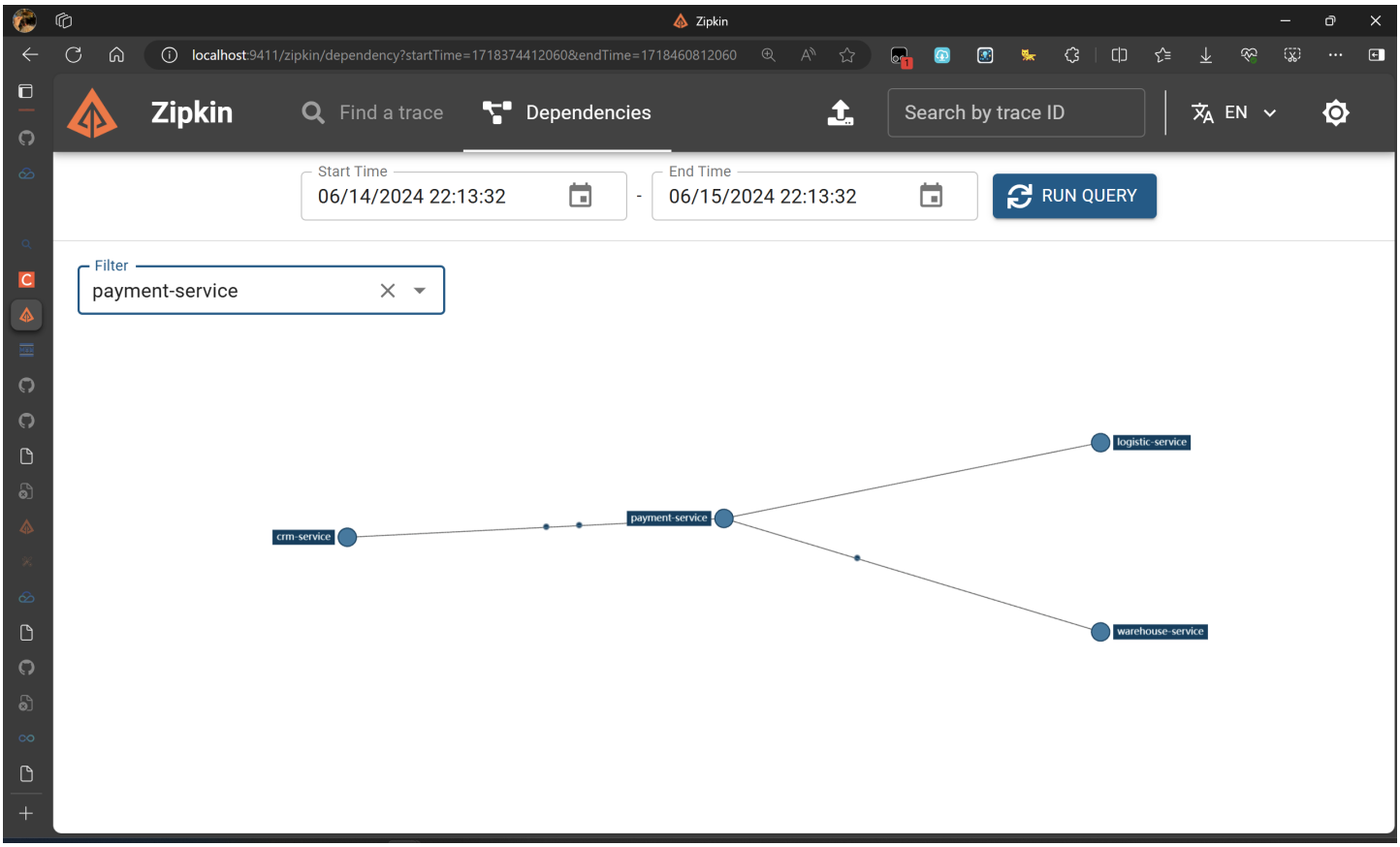
Sleuth的引入将会使得下面的左图变为右图（即会出现Span ID/Trace ID和Parent ID）：

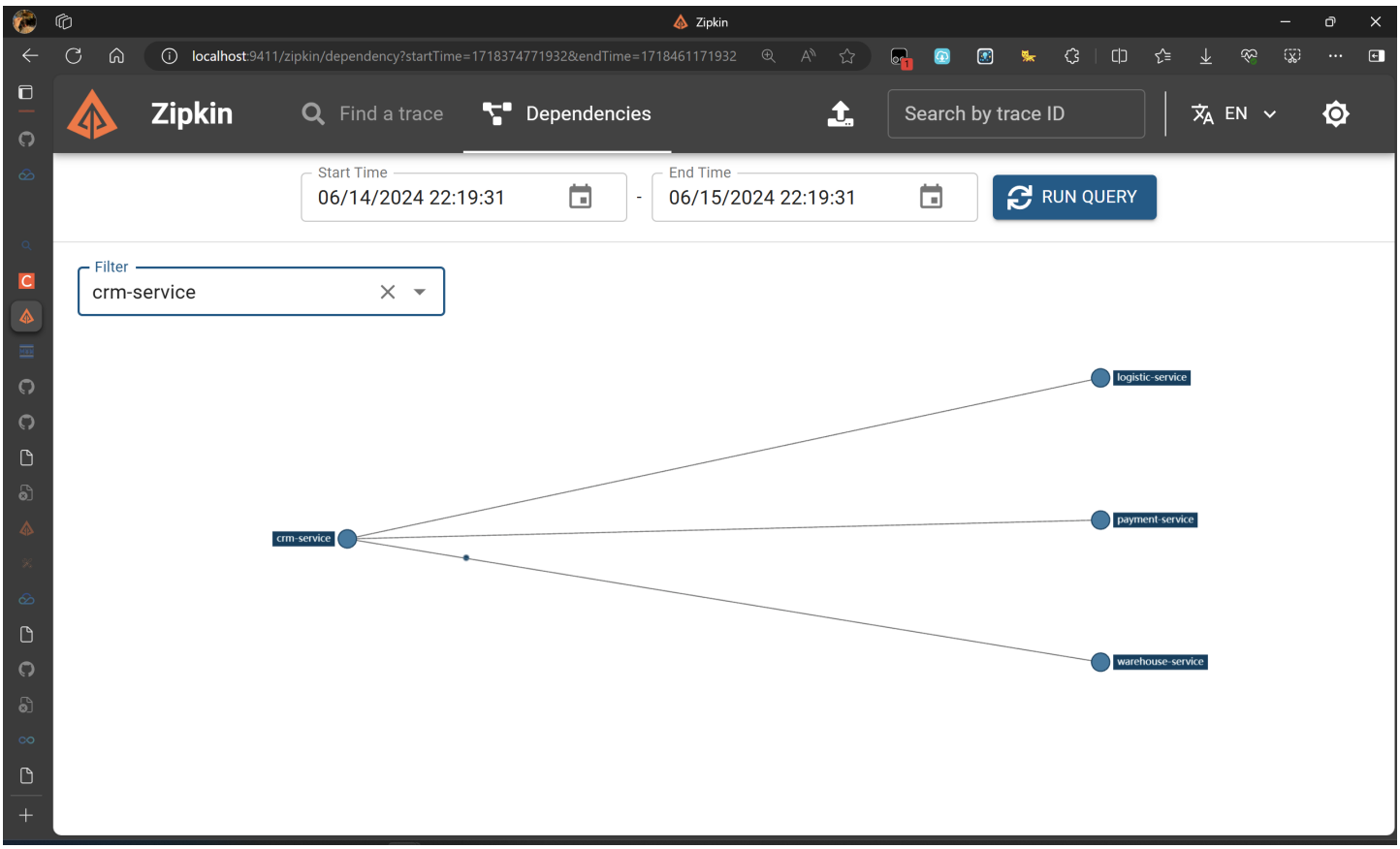


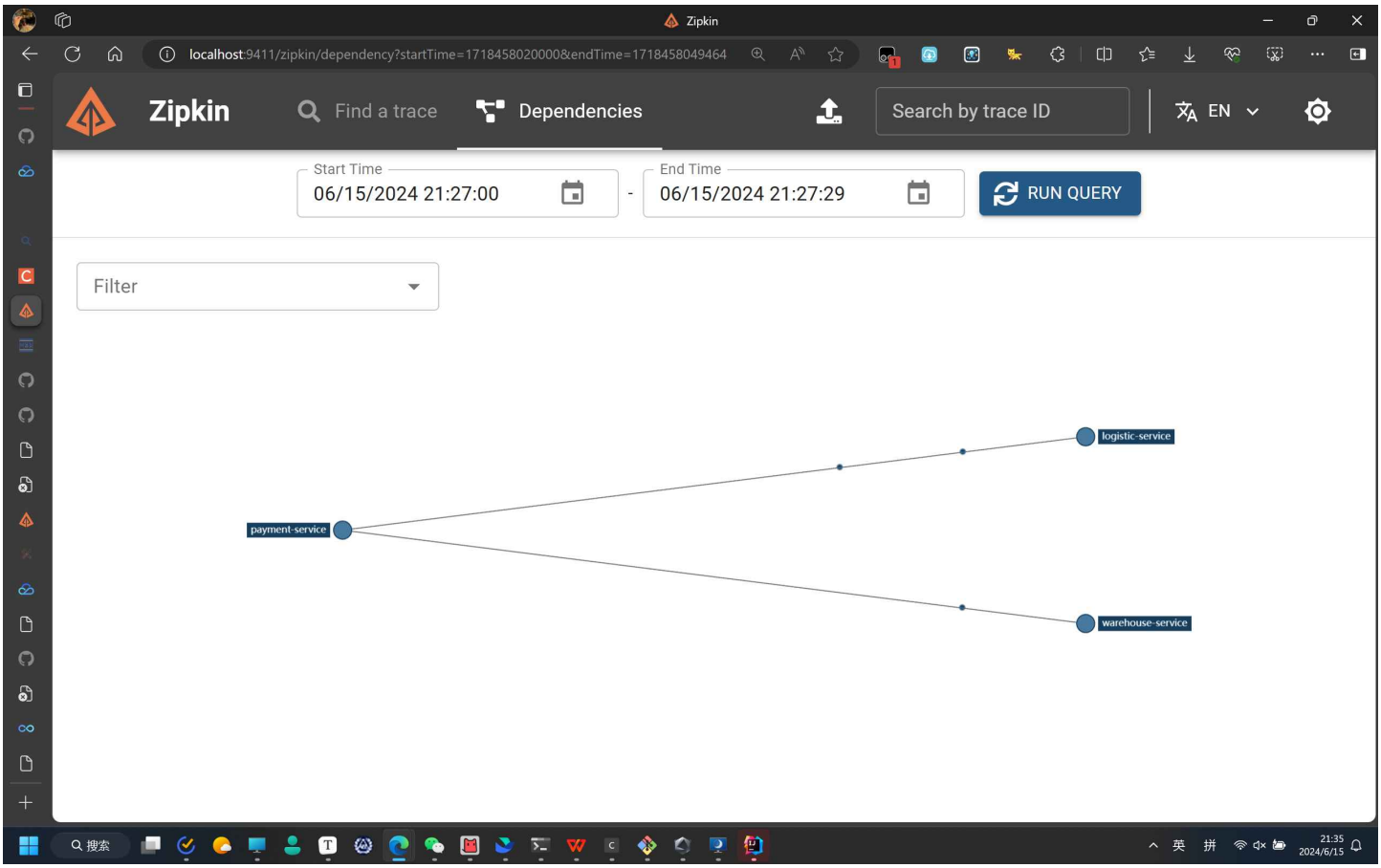
总的追踪图如下：


Zipkin
localhost:9411/zipkin/?lookback=5m&endTs=1718460631329&limit=10
Zipkin
Find a trace
Dependencies
Search by trace ID
EN
RUN QUERY
Limit
10
LAST 5 MINUTES
Results
EXPAND ALL
COLLAPSE ALL
Service filters
crm-service
Service filters
Root
Start Time
Spans
Duration
crm-service: get /warehouse
4 minutes ago (06/15 22:06:35:605)
3
262.440ms
SHOW
crm-service (2)
warehouse-service (1)
crm-service: get /payment
4 minutes ago (06/15 22:06:48:713)
7
45.462ms
SHOW
payment-service (3)
crm-service (2)
logistic-service (1)
warehouse-service (1)
crm-service: get /logistic
4 minutes ago (06/15 22:06:44:132)
3
34.805ms
SHOW
crm-service (2)
logistic-service (1)

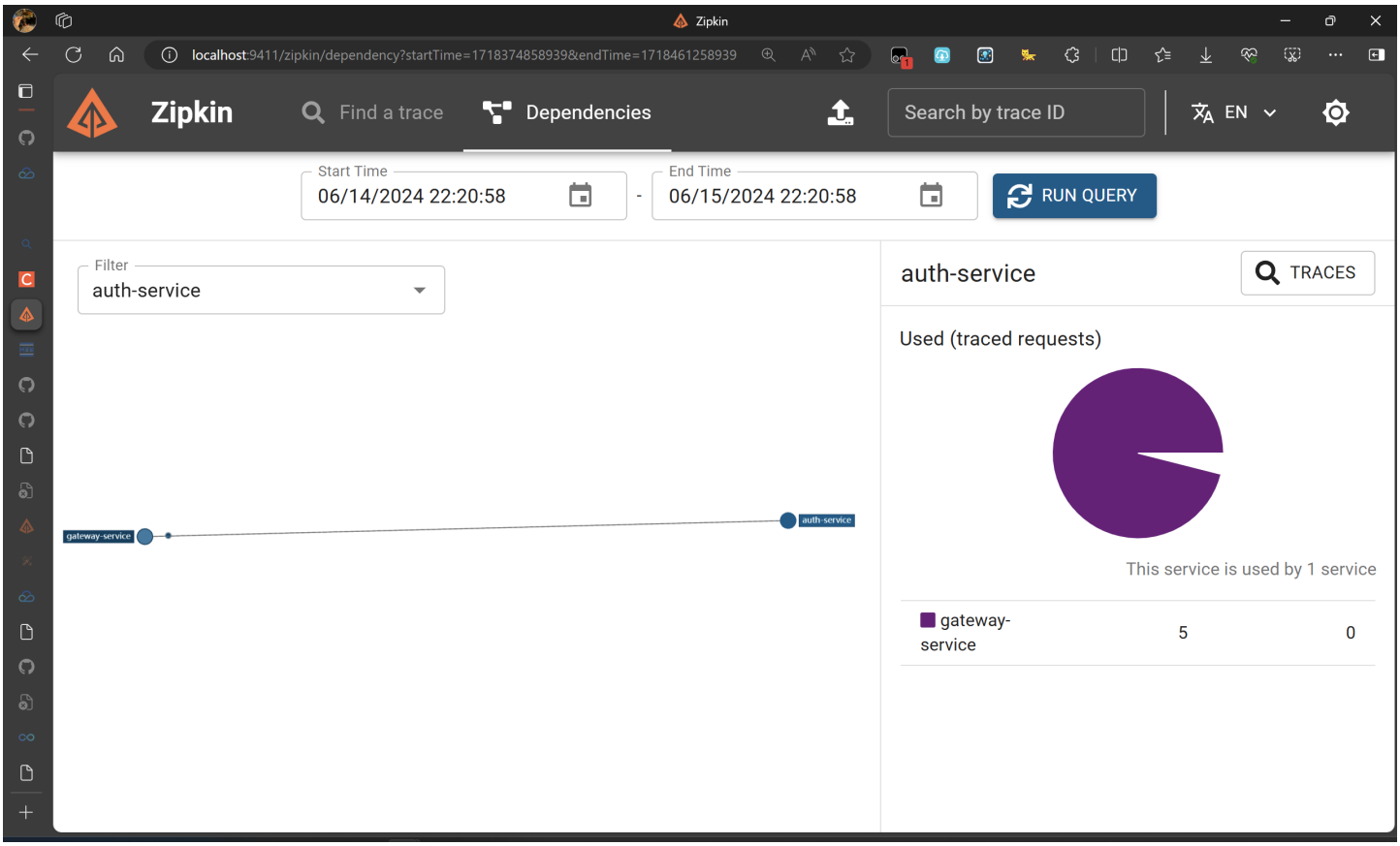

Zipkin
localhost:9411/zipkin/?lookback=millis&endTs=1718460765430&millis=500000&limit=10
Zipkin
Find a trace
Dependencies
Search by trace ID
EN
RUN QUERY
Limit
10
500000MS
Results
EXPAND ALL
COLLAPSE ALL
Service filters
payment-service
Service filt...
Root
Start Time
Spans
Duration
payment-service: get /logistic
7 minutes ago (06/15 22:05:35:921)
3
255.422ms
SHOW
payment-service (2)
logistic-service (1)
crm-service: get /payment
6 minutes ago (06/15 22:06:48:713)
7
45.462ms
SHOW
payment-service (3)
crm-service (2)
logistic-service (1)
warehouse-service (1)
payment-service: get /warehouse
7 minutes ago (06/15 22:05:45:513)
3
32.791ms
SHOW
payment-service (2)
warehouse-service (1)


Zipkin
localhost:9411/zipkin/dependency?startTime=1718374412060&endTime=1718460812060
Zipkin
Find a trace
Dependencies
Search by trace ID
EN
Start Time
06/14/2024 22:13:32
End Time
06/15/2024 22:13:32
RUN QUERY
Filter
payment-service
logistic-service
crm-service
payment-service
warehouse-service

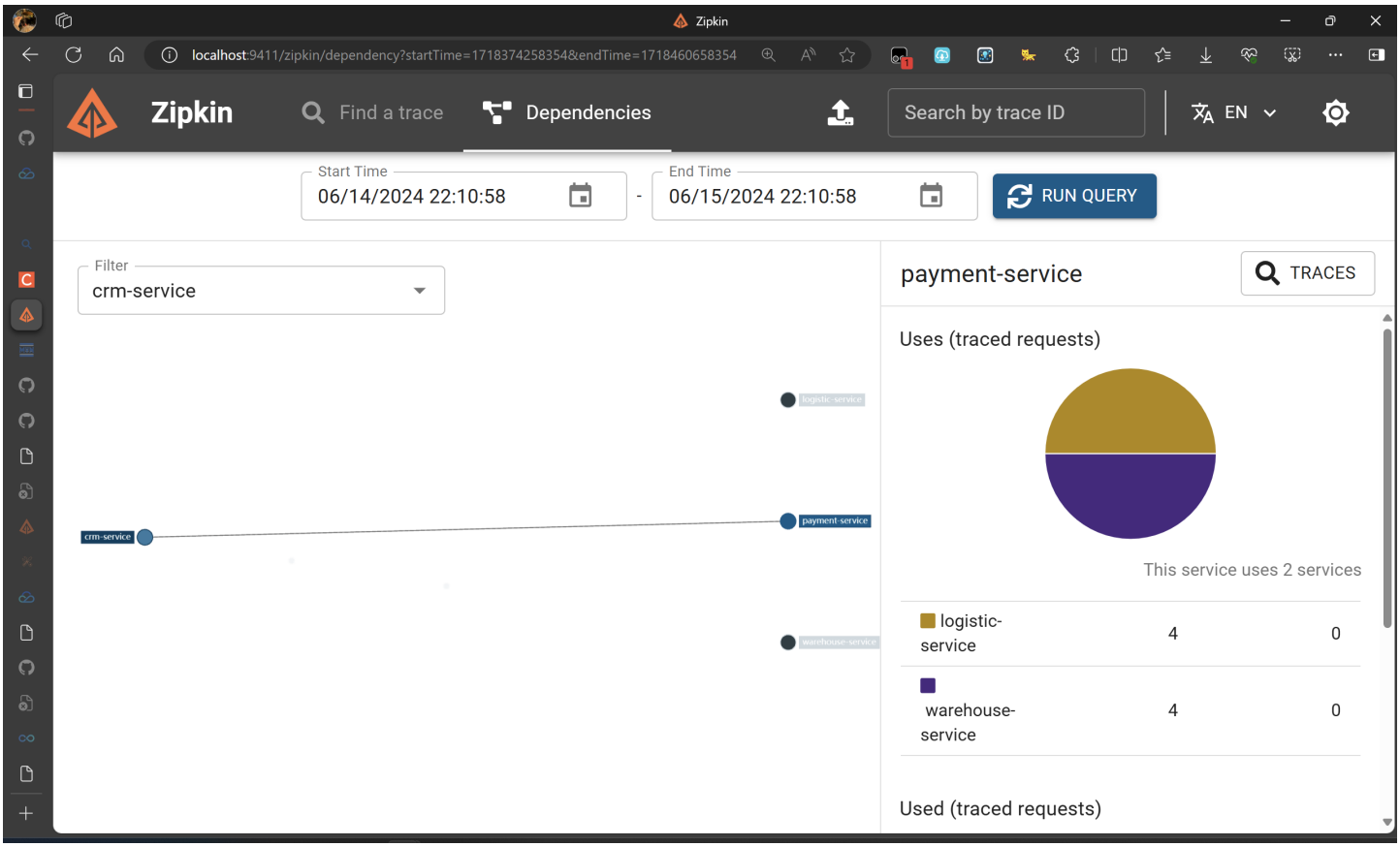

Zipkin
localhost:9411/zipkin/dependency?startTime=1718374771932&endTime=1718461171932
Zipkin
Find a trace
Dependencies
Search by trace ID
EN
Start Time
06/14/2024 22:19:31
End Time
06/15/2024 22:19:31
RUN QUERY
Filter
crm-service
logistic-service
crm-service
payment-service
warehouse-service



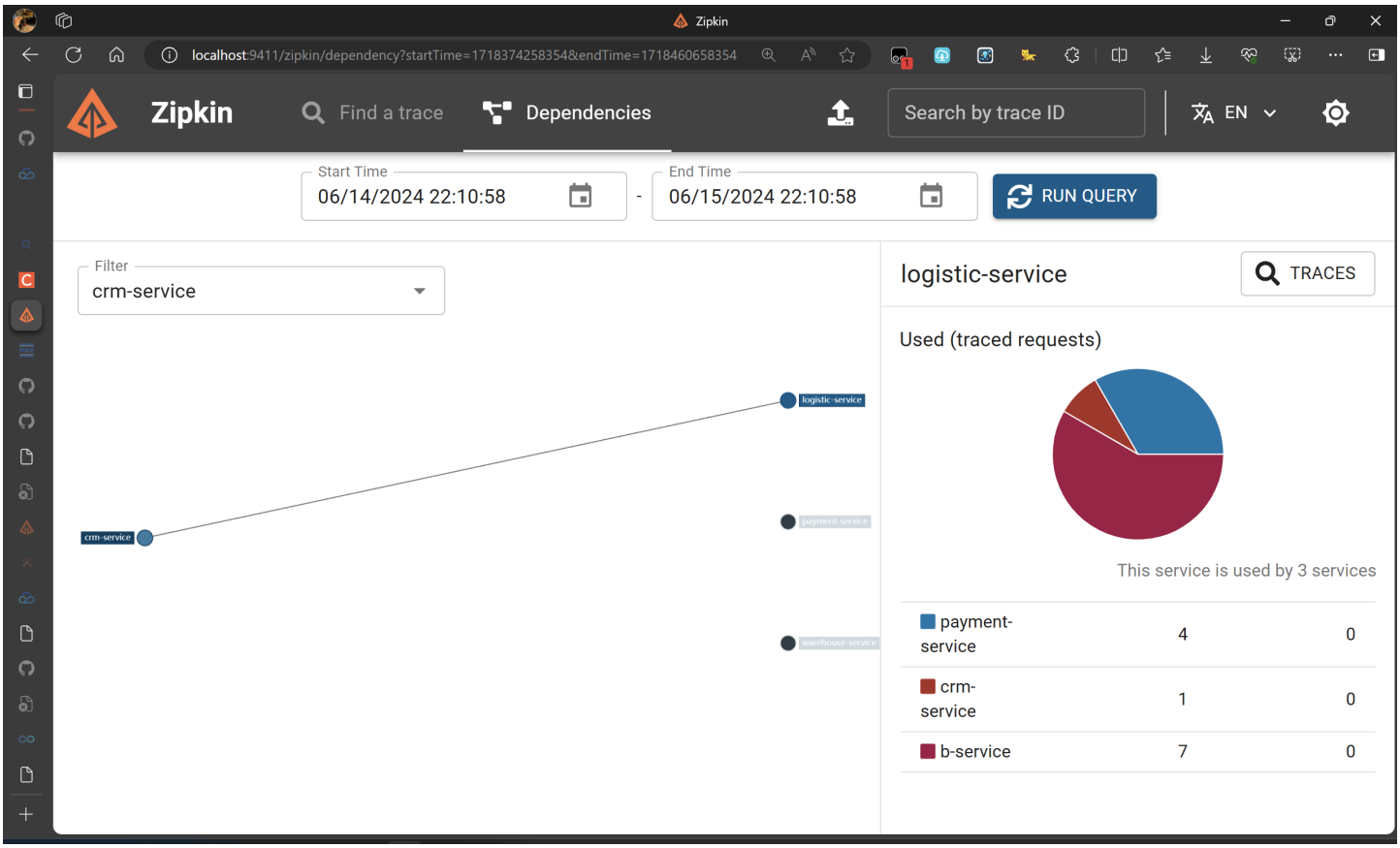
1. `gateway-service` 调用 `auth-service`



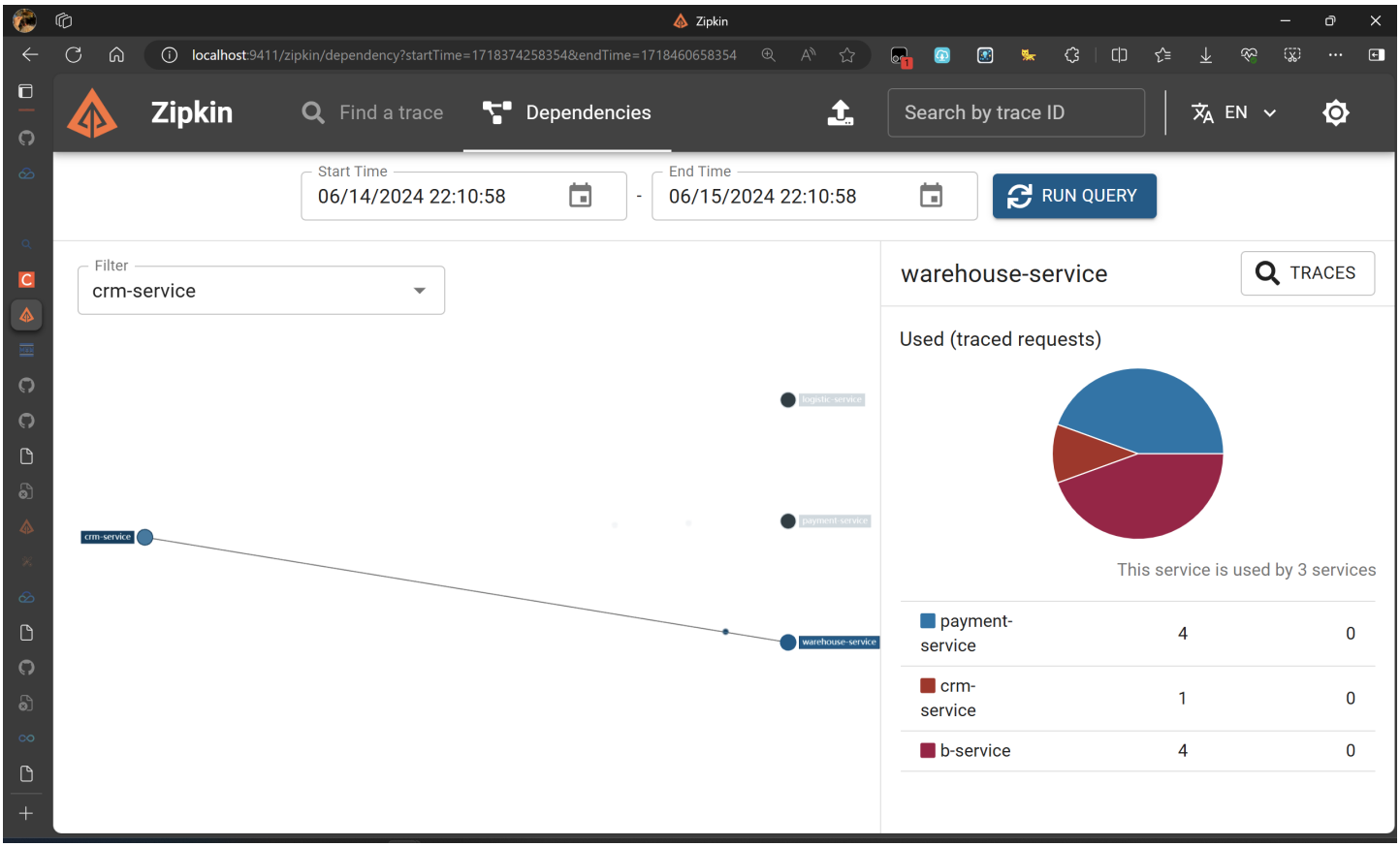
2. `crm-service` 调用 `payment-service`



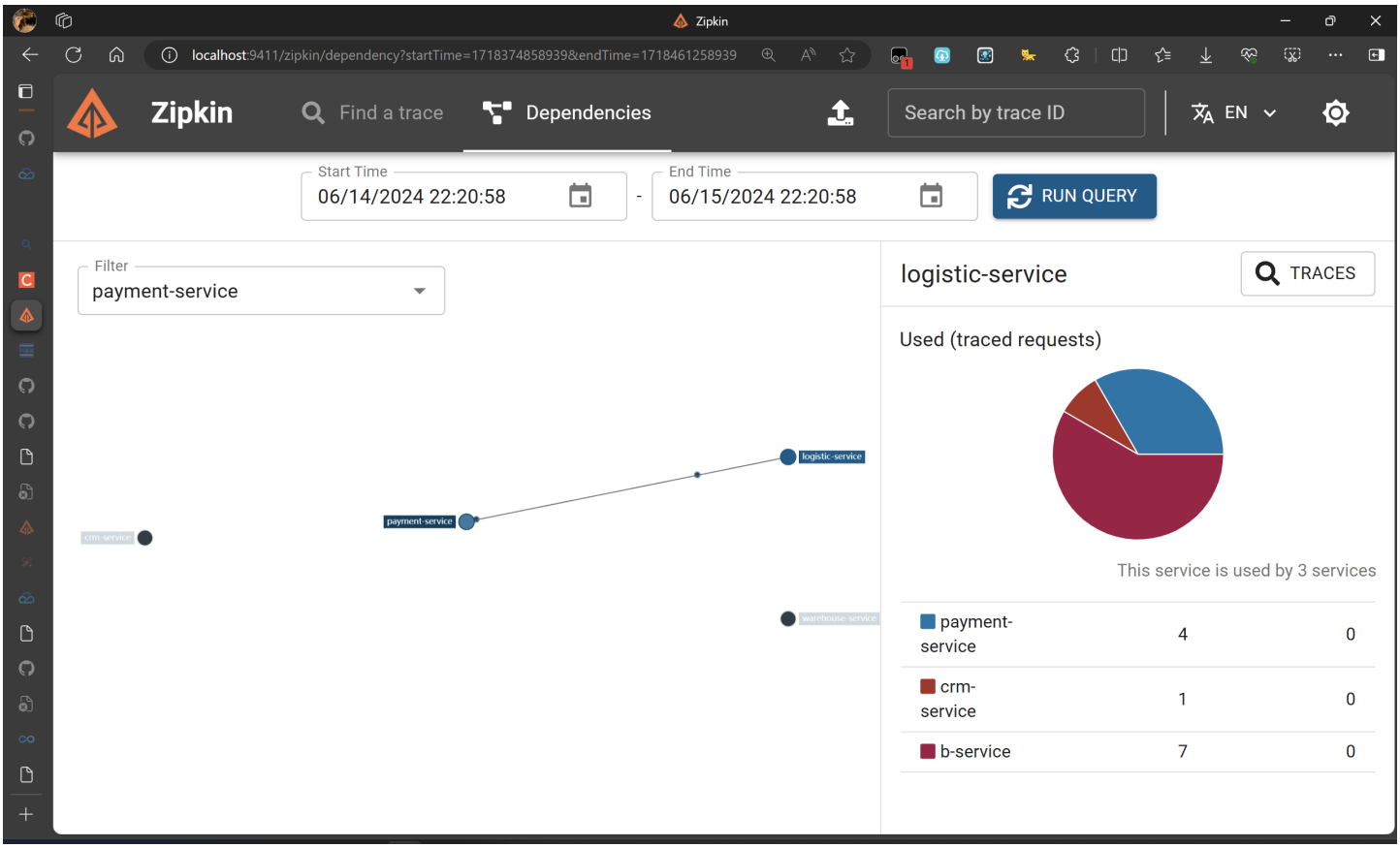
3. `crm-service` 调用 `logistic-service`



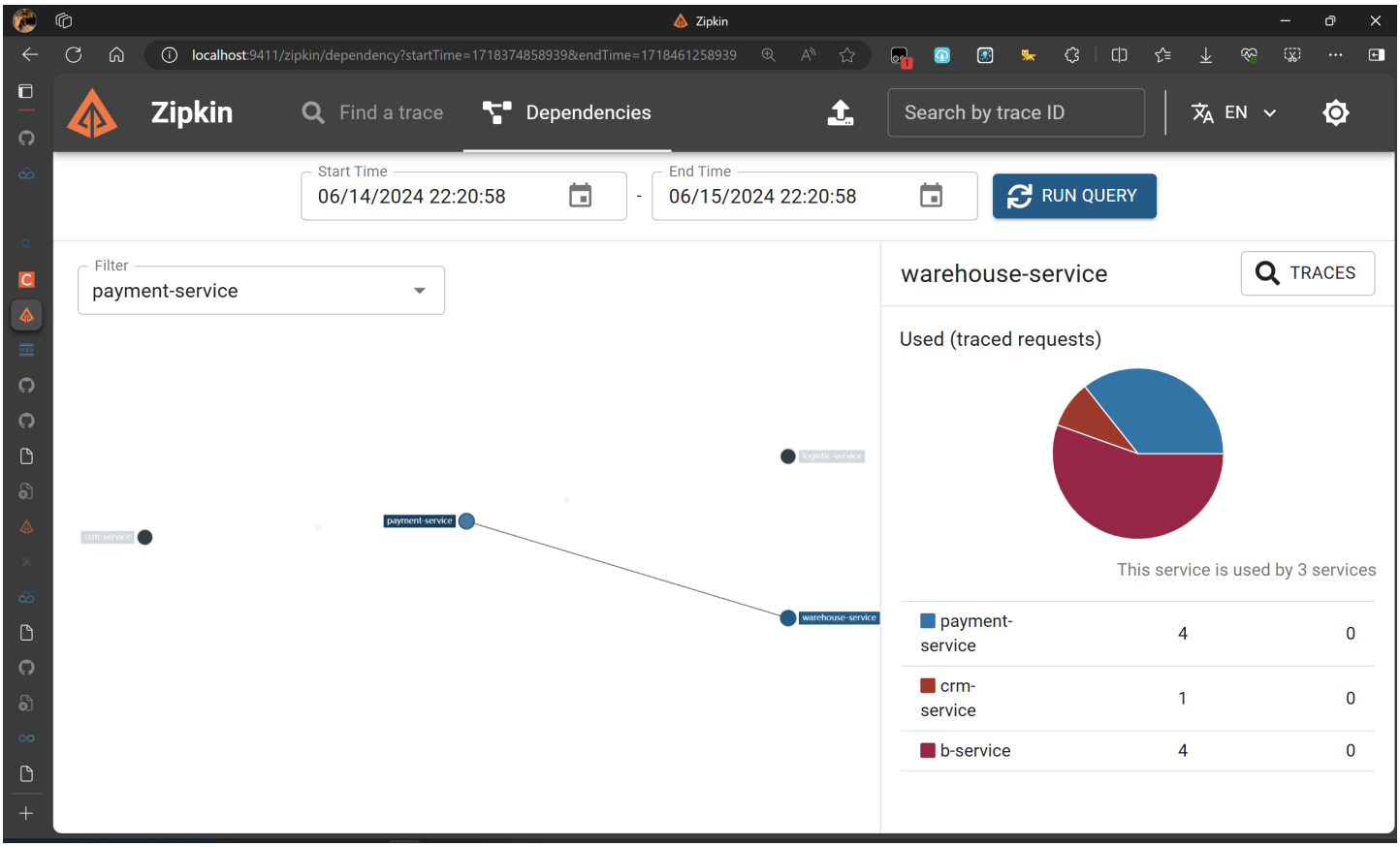
4. `crm-service` 调用 `warehouse-service`



5. `payment-service` 调用 `logistic-service`



6. `payment-service` 调用 `warehouse-service`



1.6.2 下面是原始实现（老版本）：

我们添加了Spring Cloud Config Server进行集中配置管理，并使用Spring Cloud Sleuth和Zipkin进行分布式跟踪。配置和代码示例如下：

`config-server` 的 `pom.xml`

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>
</dependencies>
```

`application.properties`

```
server.port=8888
spring.application.name=config-server
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
spring.cloud.config.server.git.uri=https://github.com/Brony01/WarehouseManagementSystem04.git
```

`ConfigServerApplication.java`

```
package com.example.configserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication@EnableConfigServerpublic class ConfigServerApplication{public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

在每个微服务的 `application.properties` 文件中配置：

```
spring.cloud.config.uri=http://localhost:8888
spring.zipkin.base-url=http://localhost:9411
spring.sleuth.sampler.probability=1.0
```

1.6.3 下面是Sleuth 3.1版的实现：

我们在配置Spring Cloud Sleuth时发现其版本与 Spring Cloud 版本不兼容。Sleuth 3.1 版要求迁移到 Micrometer Tracing 而不是 Sleuth，于是我们更新依赖项和配置以使用 Micrometer Tracing。

1. 更新 `pom.xml` 依赖项

移除 Sleuth 和 Zipkin 依赖，添加 Micrometer Tracing 和 Zipkin 依赖：

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>
<dependency>
    <groupId>io.zipkin.brave</groupId>
    <artifactId>brave-instrumentation-http</artifactId>
```

```
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
```

2. 配置 `application.yml`

确保在 `application.yml` 中正确配置了 Zipkin：

```
spring:application:name: logistic-servicecloud:nacos:discovery:server-addr: 127.0.0.1:8848compatibility-verifier:enabled: falsesleuth:sampler:probability: 1.0zipkin:base-url: http://localhost:9411discovery-client-enabled: falselogging:level:io.micrometer: INFOorg.springframework.web.client.RestTemplate: INFO
```

3. 配置 Micrometer Tracing

确保你有一个配置类来配置 Micrometer Tracing：

```
package com.example.config;

import io.micrometer.tracing.Tracer;
import io.micrometer.tracing.brave.bridge.BraveTracer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import zipkin2.reporter.AsyncReporter;
import zipkin2.reporter.okhttp3.OkHttpSender;

@Configurationpublic class TracingConfig {
@Beanpublic Tracer tracer() {OkHttpSender sender = OkHttpSender.create("http://localhost:9411/api/v2/spans");
        AsyncReporter<zipkin2.Span> spanReporter = AsyncReporter.create(sender);
        brave.Tracing braveTracing = brave.Tracing.newBuilder()
            .localServiceName("logistic-service")
            .spanReporter(spanReporter)
            .build();return new BraveTracer(braveTracing);
    }
}
```

4. 配置 `RestTemplate`

确保 `RestTemplate` 配置正确：

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configurationpublic class AppConfig {
@Beanpublic RestTemplate restTemplate() {return new RestTemplate();
    }
}
```

5. 控制器类

确保你的控制器类正确注入并使用 `RestTemplate`：

```
package com.example.springboota;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestControllerpublic class AController {
@Autowiredprivate RestTemplate restTemplate;
private static final String SERVICE_PROVIDER_ADDRESS = "http://logistic-service";
@GetMapping("/test")public String callServiceB() {return restTemplate.getForObject(SERVICE_PROVIDER_ADDRESS + "/test-service-b", String.class);
    }
}
```

二、API接口说明

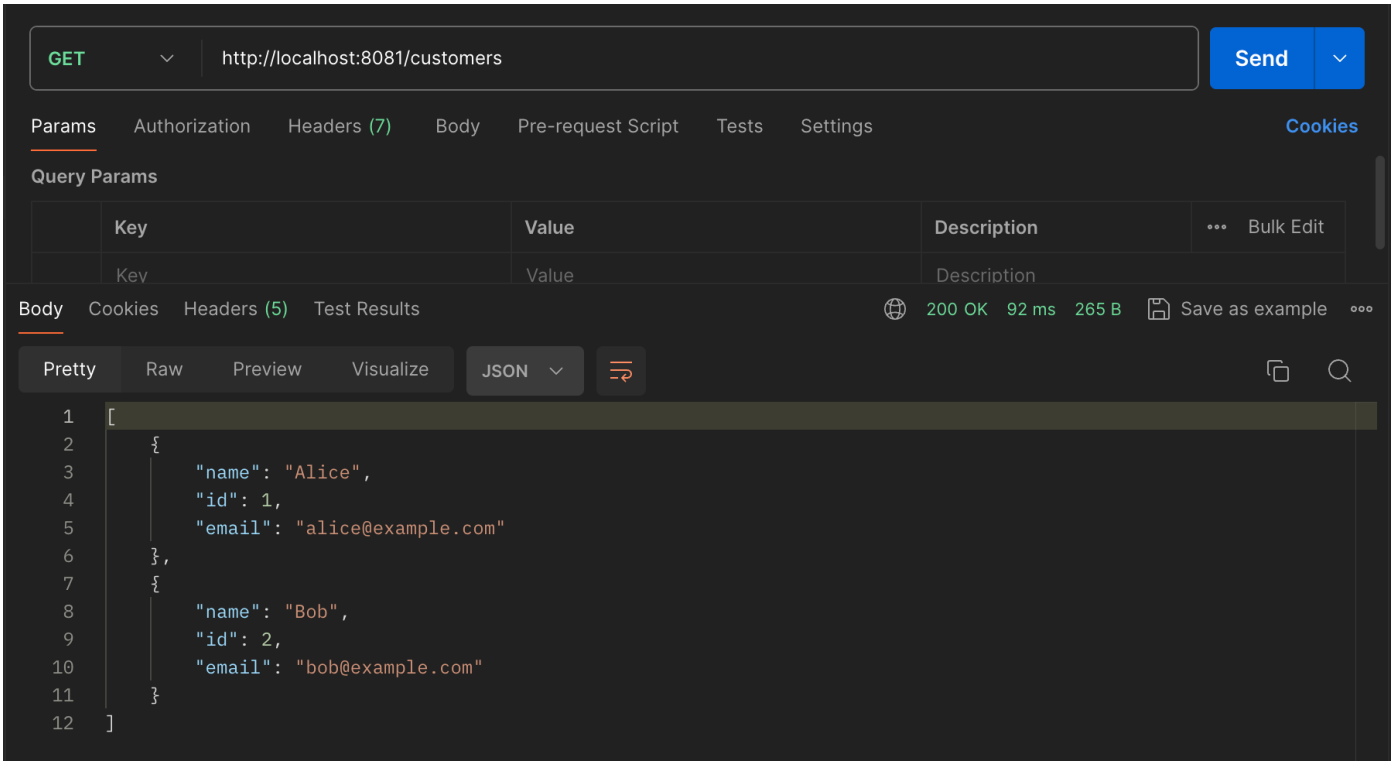
2.1 CRM Service

2.1.1 GET /customers

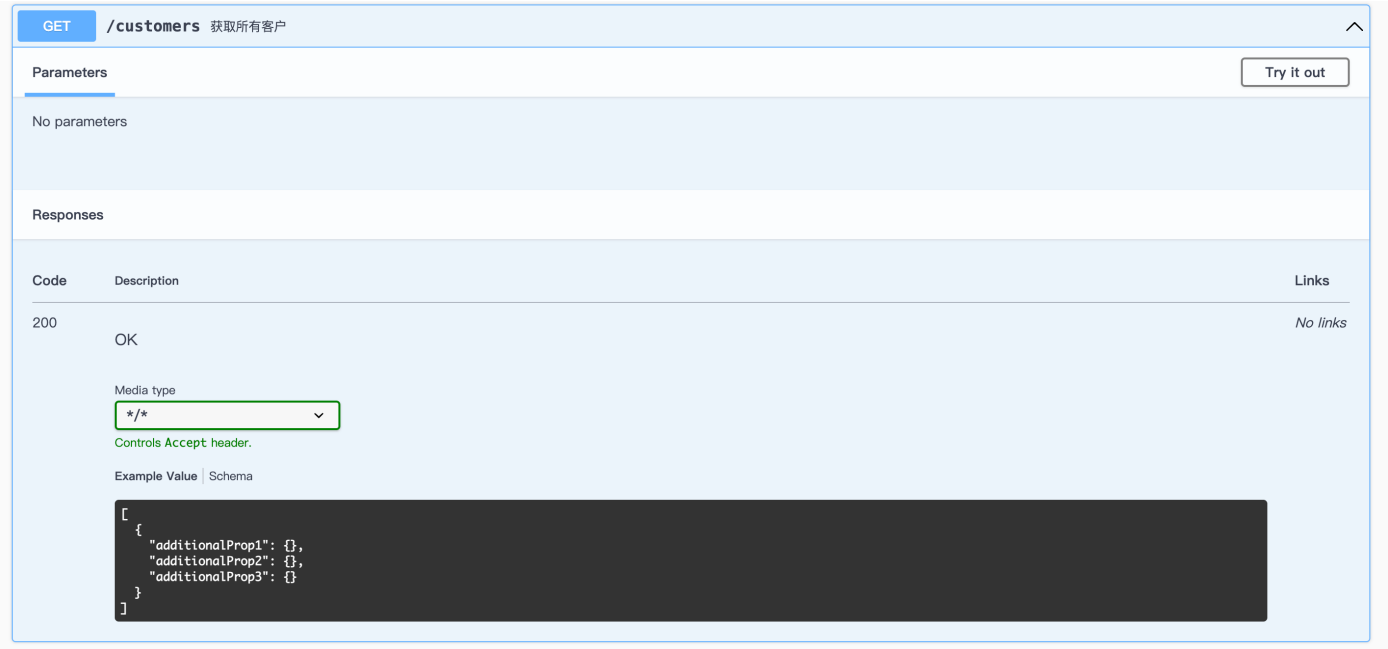
- 描述: 获取所有客户
- 参数: 无
- 返回值:

```
[
    {"id": 1,"name": "Alice","email": "alice@example.com"},
    {"id": 2,"name": "Bob","email": "bob@example.com"}
]
```

- 测试:



- swagger:



2.1.2 POST /customers

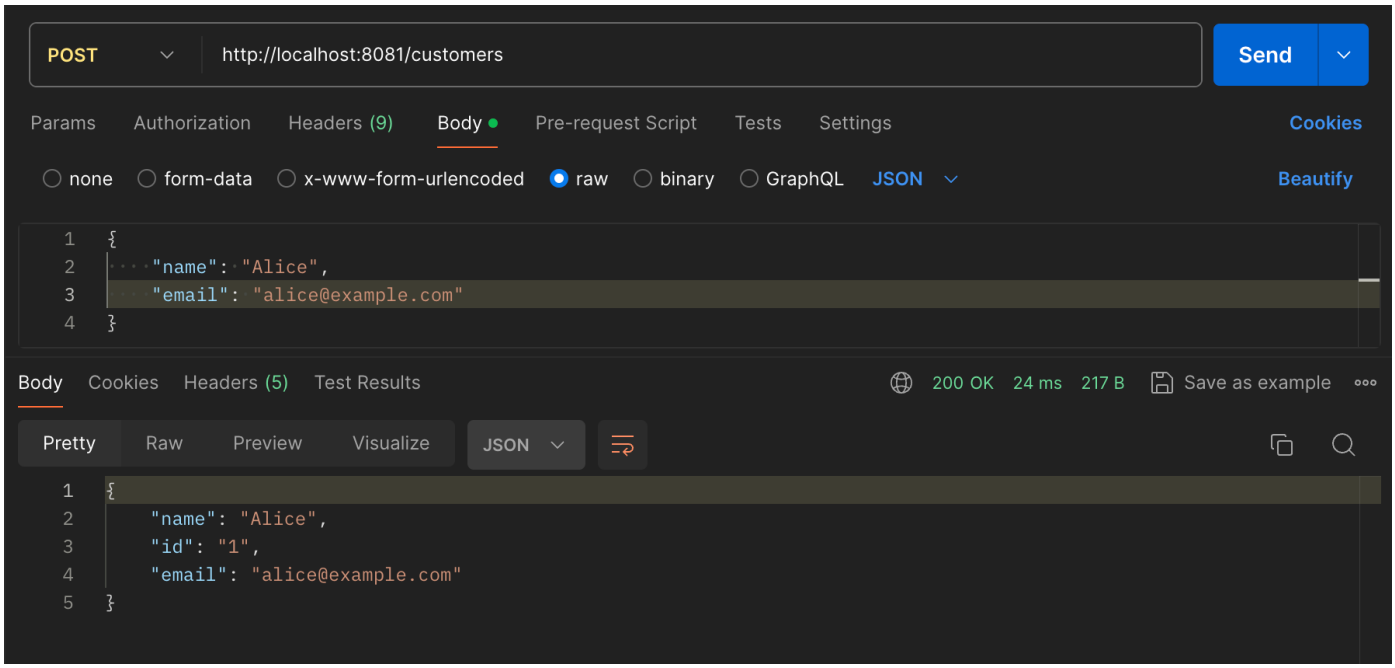
- 描述: 创建一个新的客户
- 参数:

```
1 {"name": "Alice","email": "alice@example.com"}
```

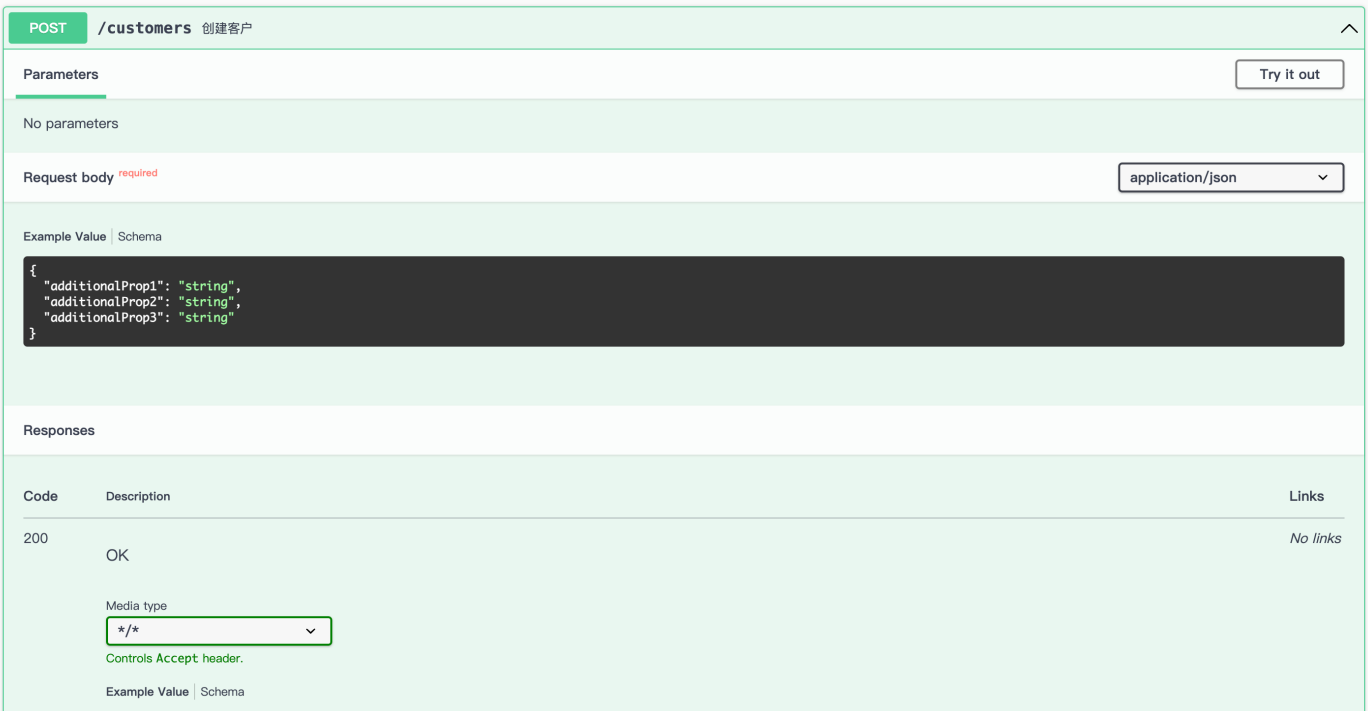
- 返回值:

```
1 {"id": 1,"name": "Alice","email": "alice@example.com"}
```

- 测试:



- swagger:



2.1.3 PUT /customers/{id}

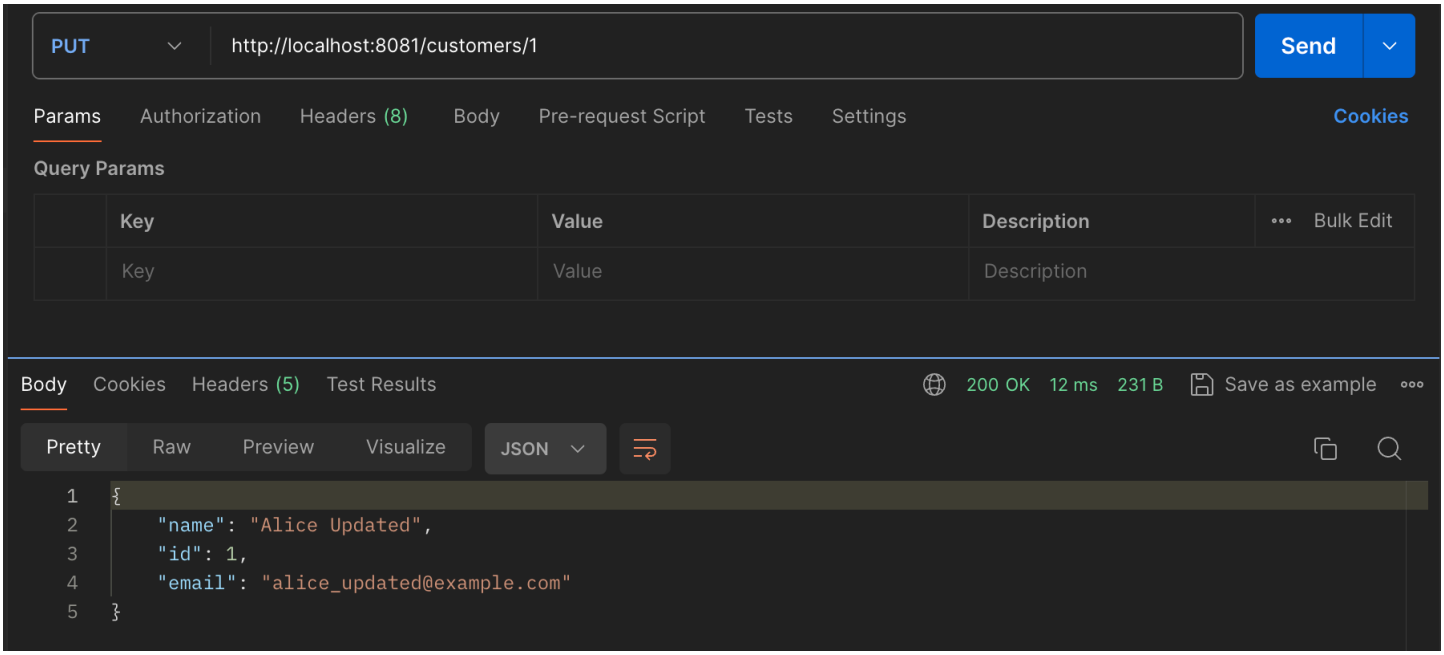
- 描述: 更新一个客户
- 参数:

```
1 {"name": "Alice Updated","email": "alice_updated@example.com"}
```

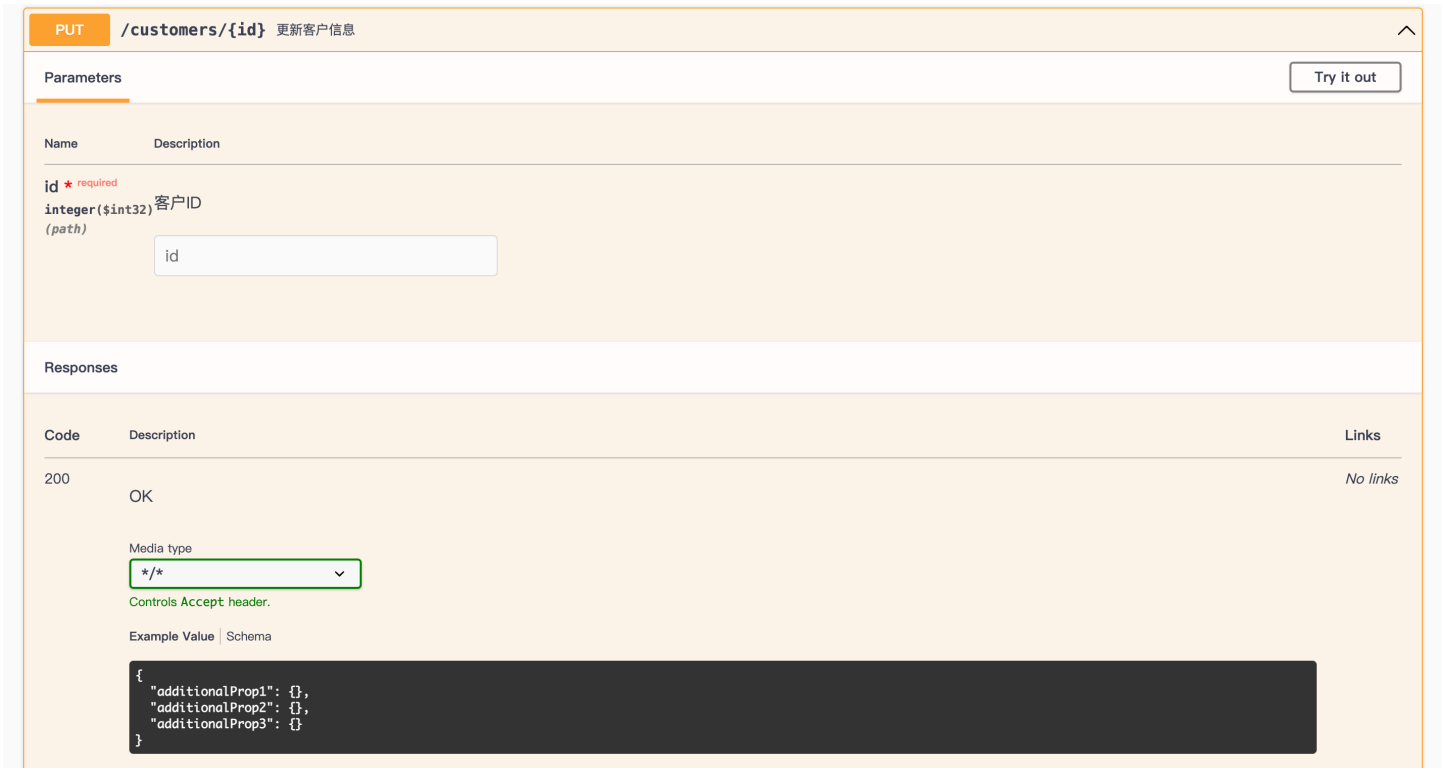
- 返回值:

```
1 {"id": 1,"name": "Alice Updated","email": "alice_updated@example.com"}
```

- 测试:



- swagger:



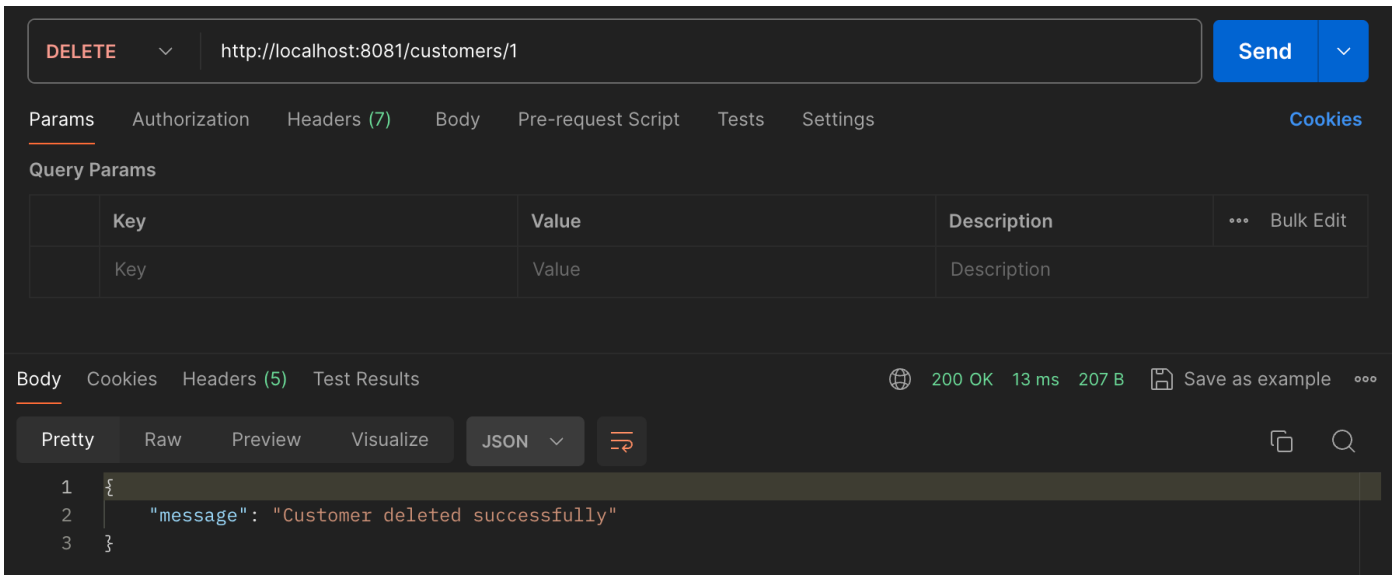
2.1.4 DELETE /customers/{id}

- 描述: 删除一个客户
- 参数: 无

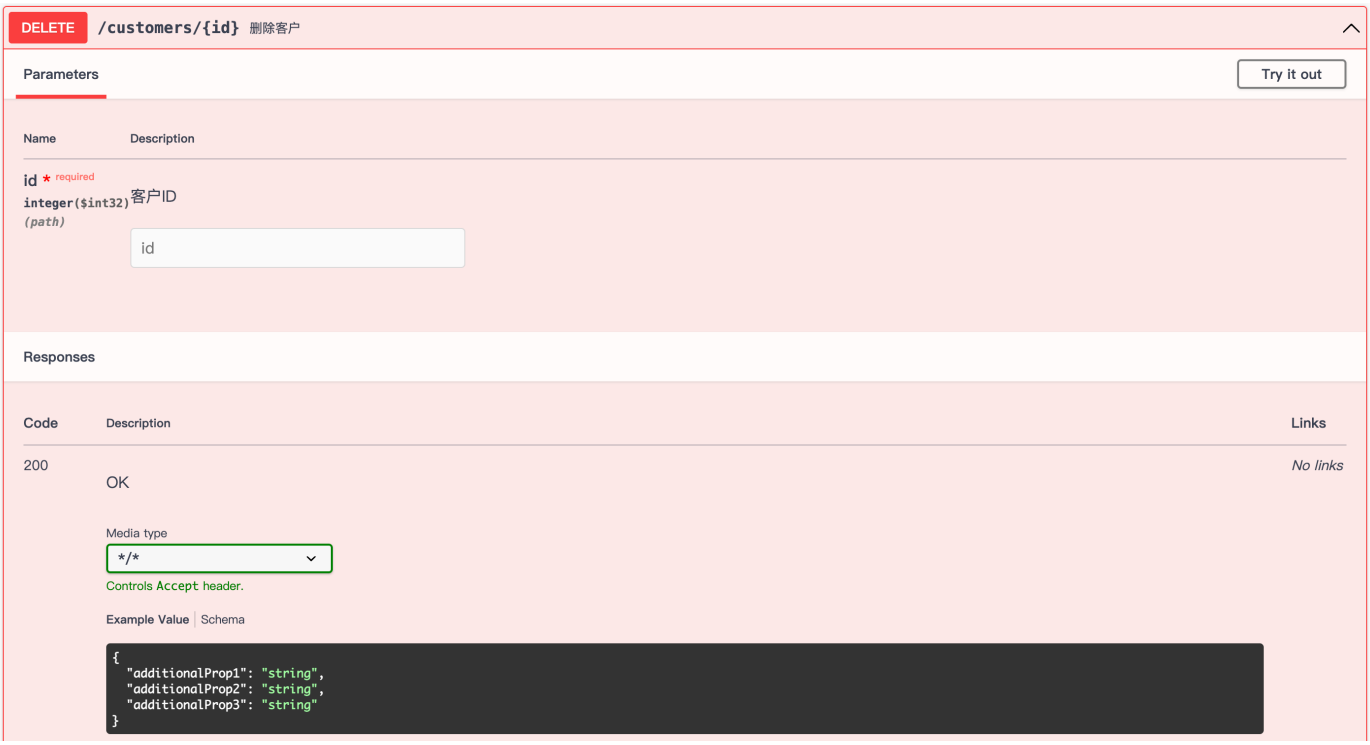
- 返回值:

```
{"message": "Customer deleted successfully"}
```

- 测试:



- swagger:



2.2 Logistic Service

2.2.1 GET /logistics

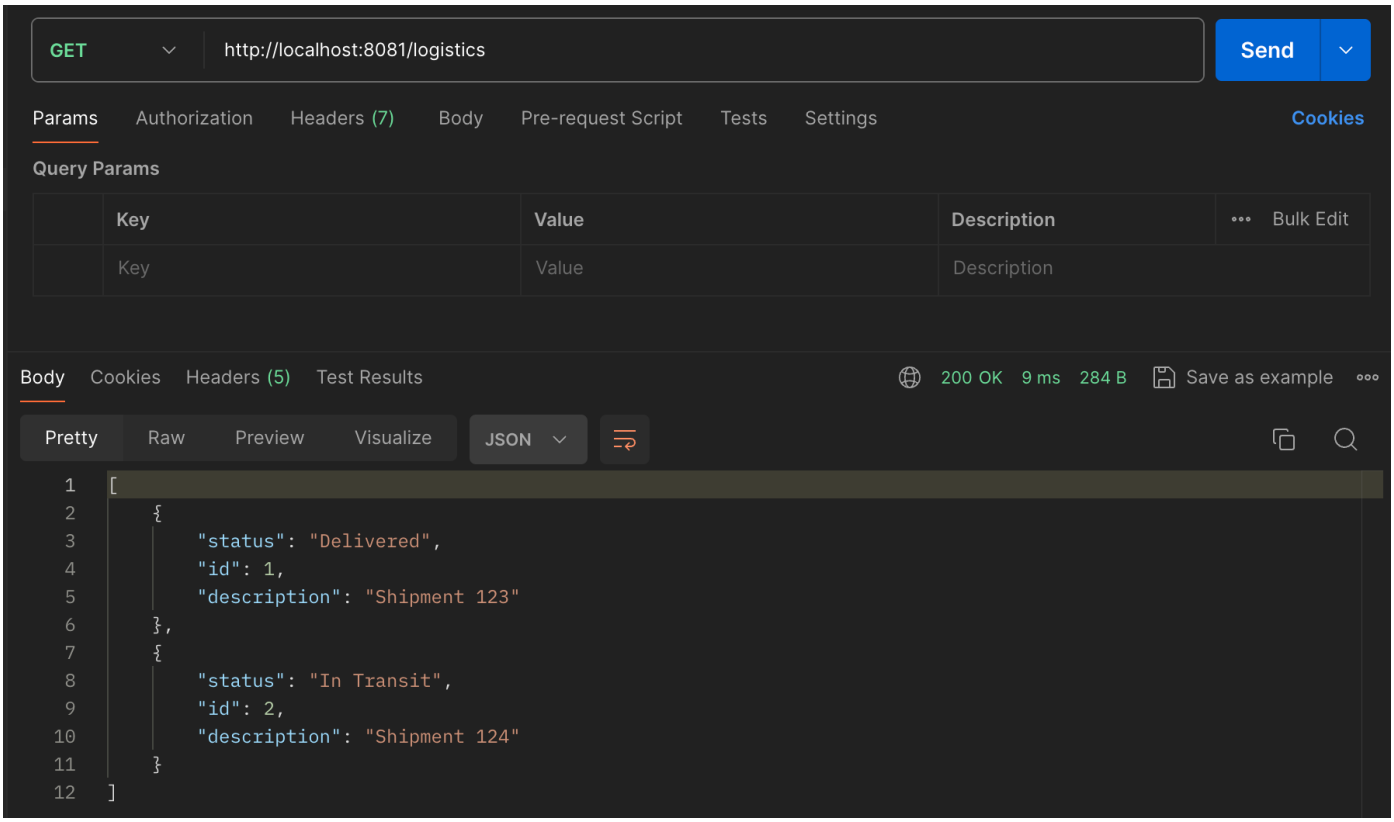
- 描述: 获取所有物流信息

- 参数: 无

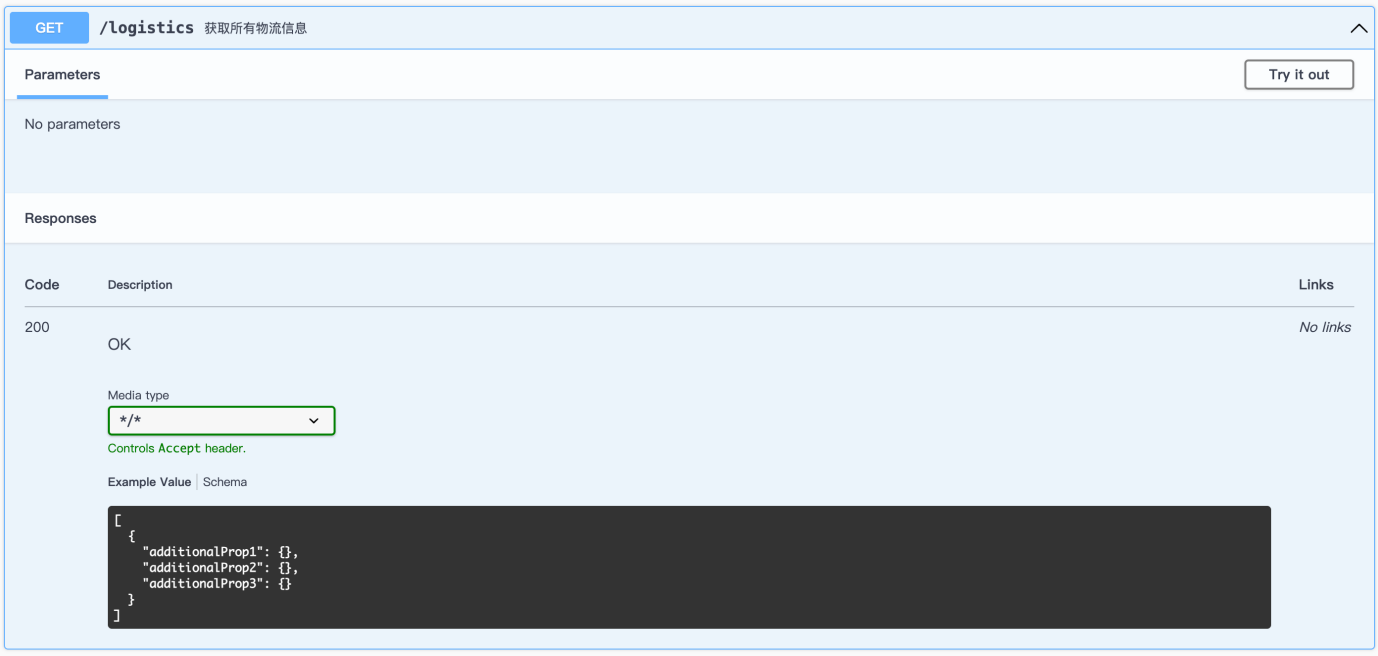
- 返回值:

```
[
    {"id": 1,"description": "Shipment 123","status": "Delivered"},
    {"id": 2,"description": "Shipment 456","status": "In Transit"}
]
```

- 测试:



- swagger:



2.2.2 POST /logistics

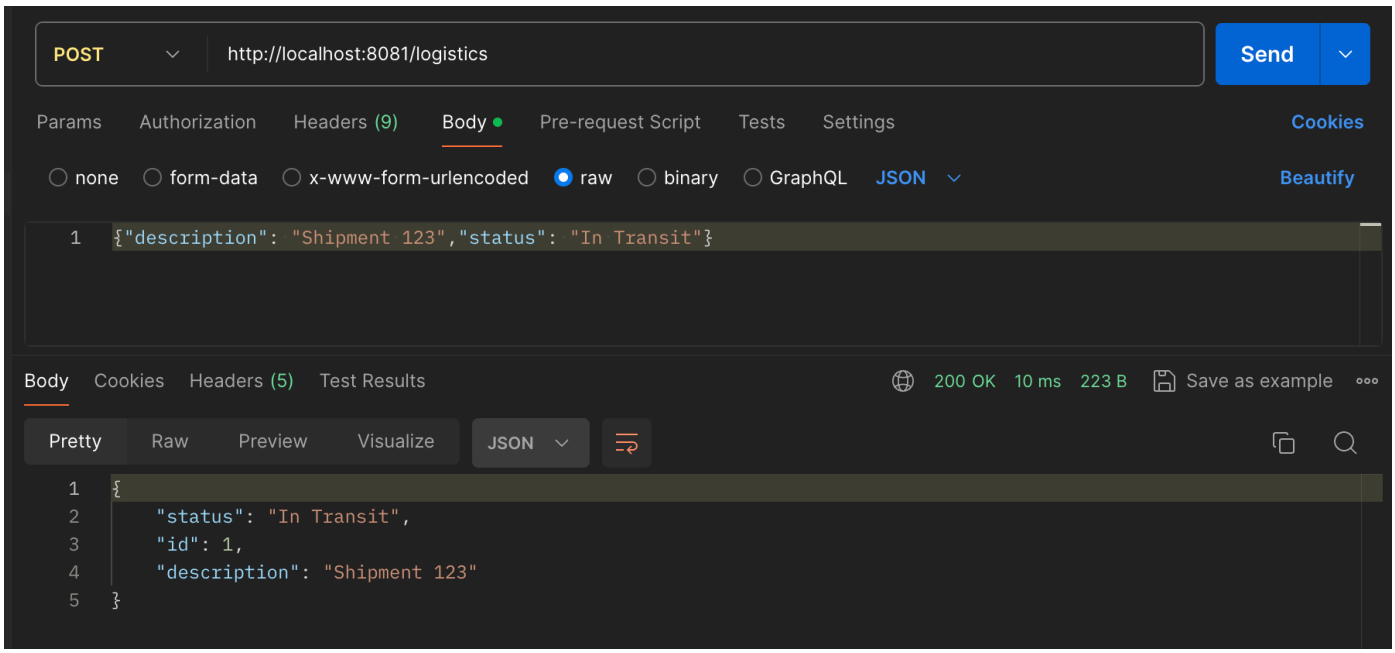
- 描述: 创建新的物流信息
- 参数:

```
1 {"description": "Shipment 123","status": "In Transit"}
```

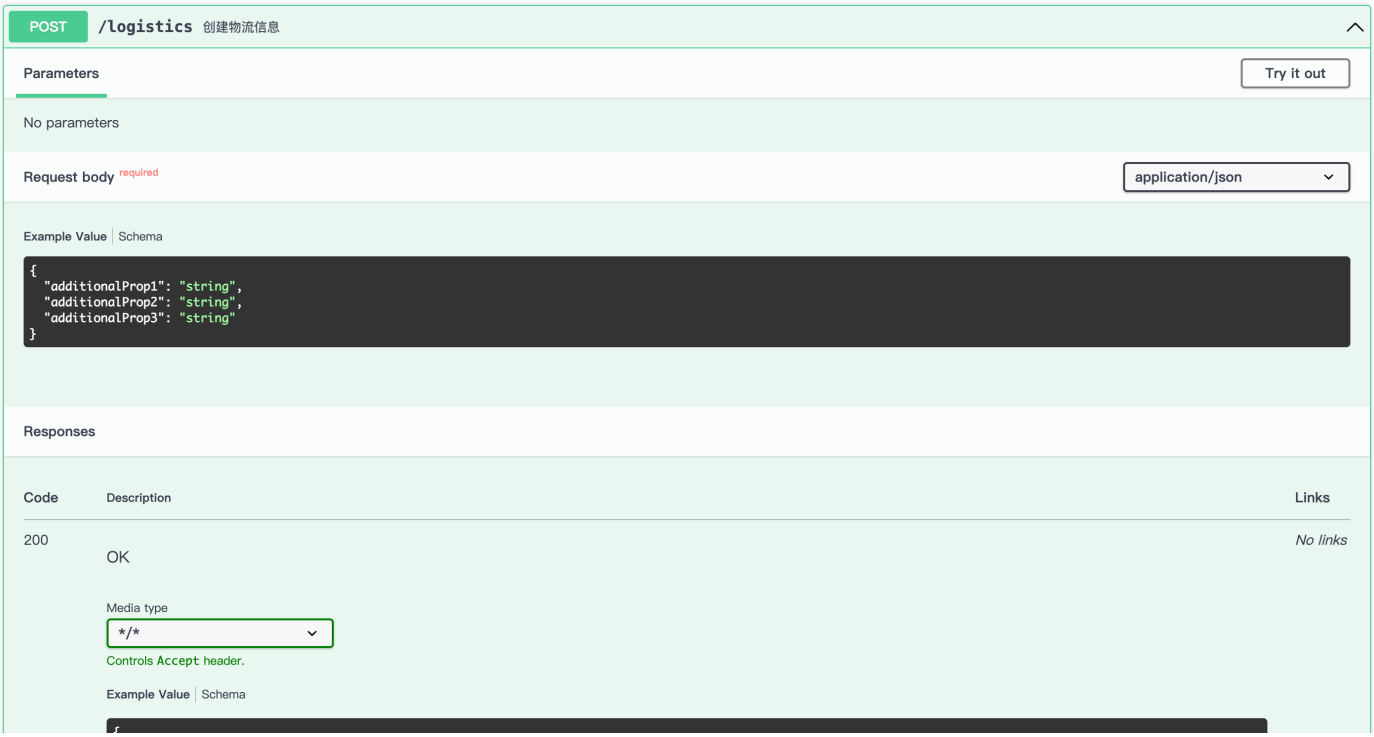
- 返回值:

```
1 {"id": 1,"description": "Shipment 123","status": "In Transit"}
```

- 测试:



- swagger:



2.2.3 PUT /logistics/{id}

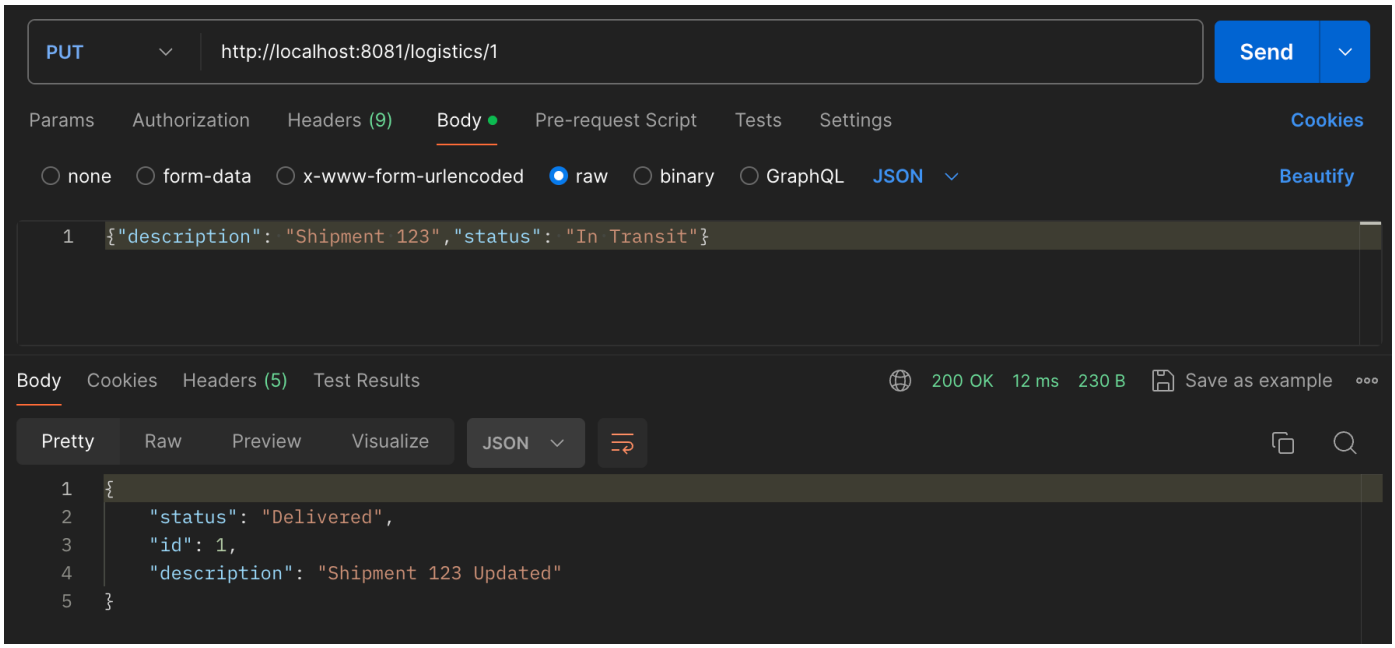
- 描述: 更新物流信息
- 参数:

```
1 {"description": "Shipment 123 Updated","status": "Delivered"}
```

- 返回值:

```
1 {"id": 1,"description": "Shipment 123 Updated","status": "Delivered"}
```

- 测试:



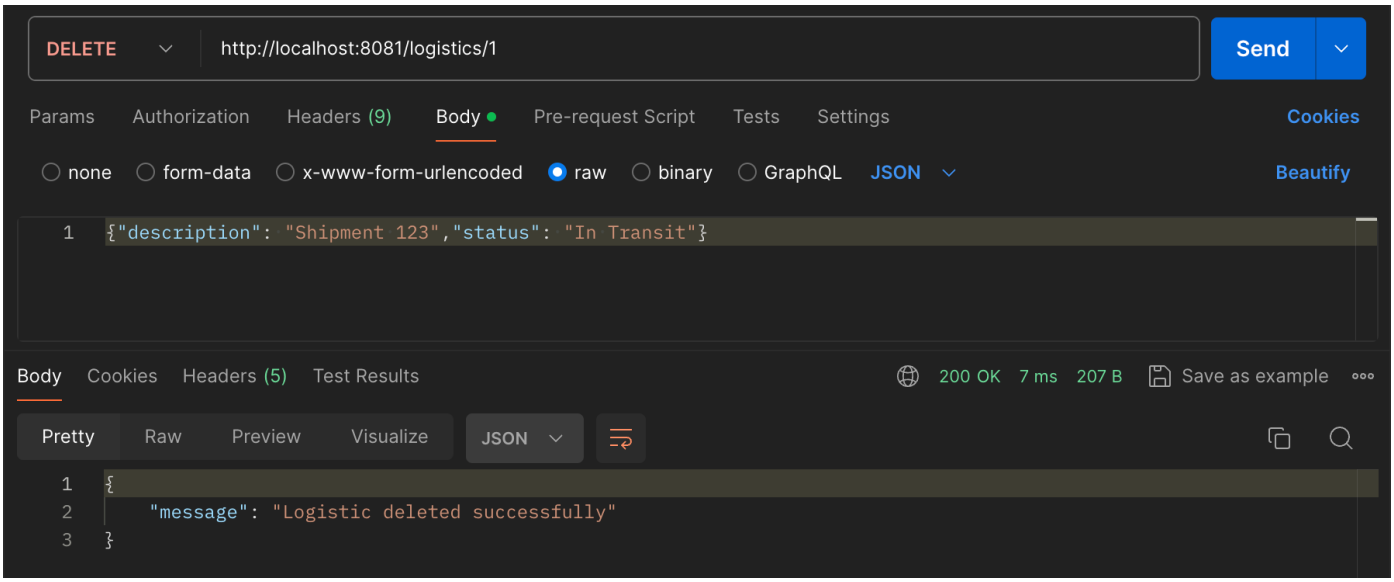
- swagger:

2.2.4 DELETE /logistics/{id}

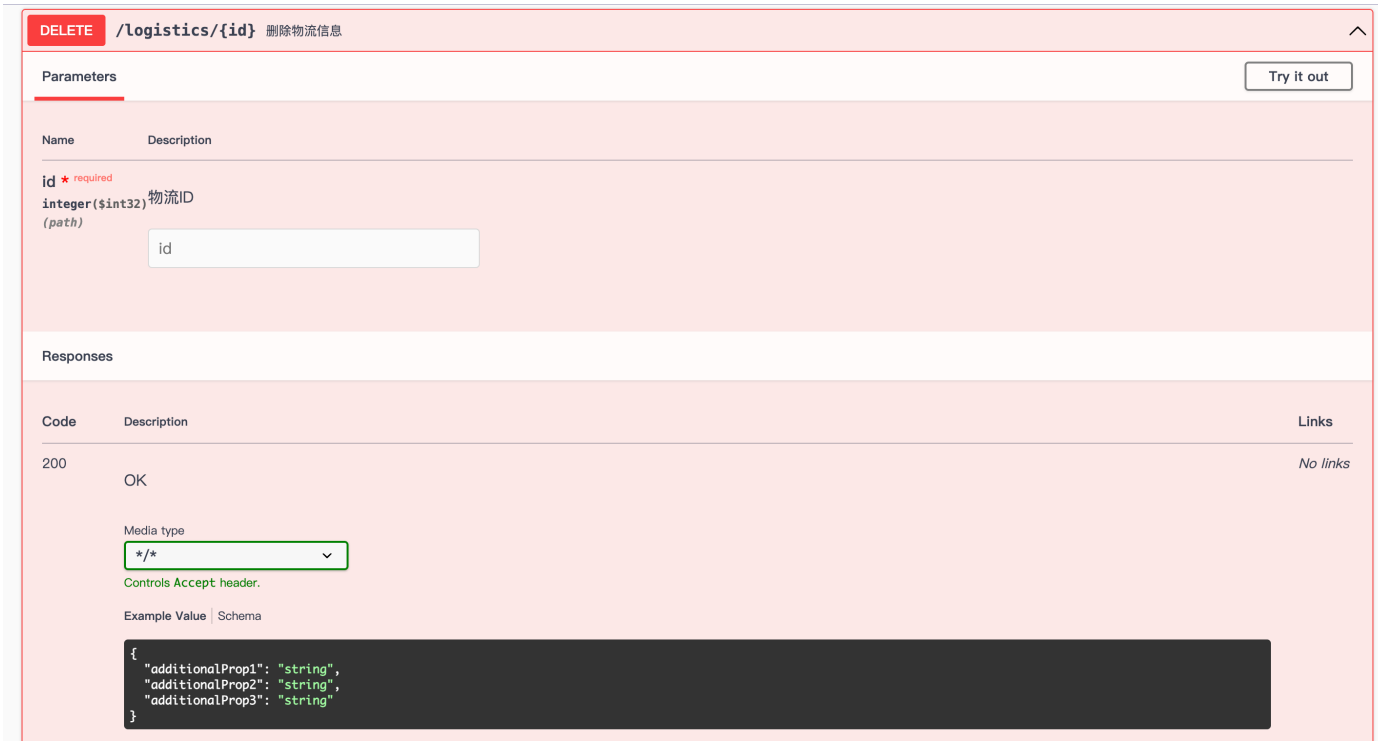
- 描述: 删除物流信息
- 参数: 无
- 返回值:

```
1 {"message": "Logistic deleted successfully"}
```

- 测试:



- swagger:



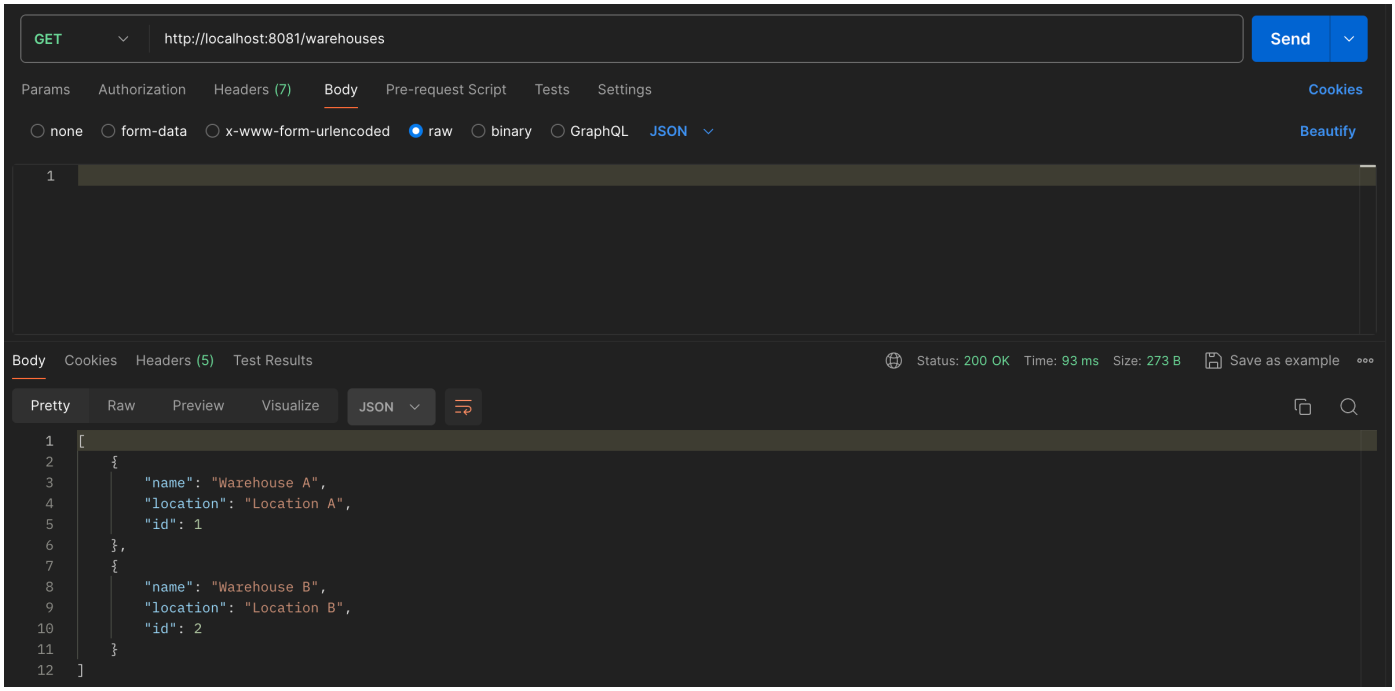
2.3 Warehouse Service

2.3.1 GET /warehouses

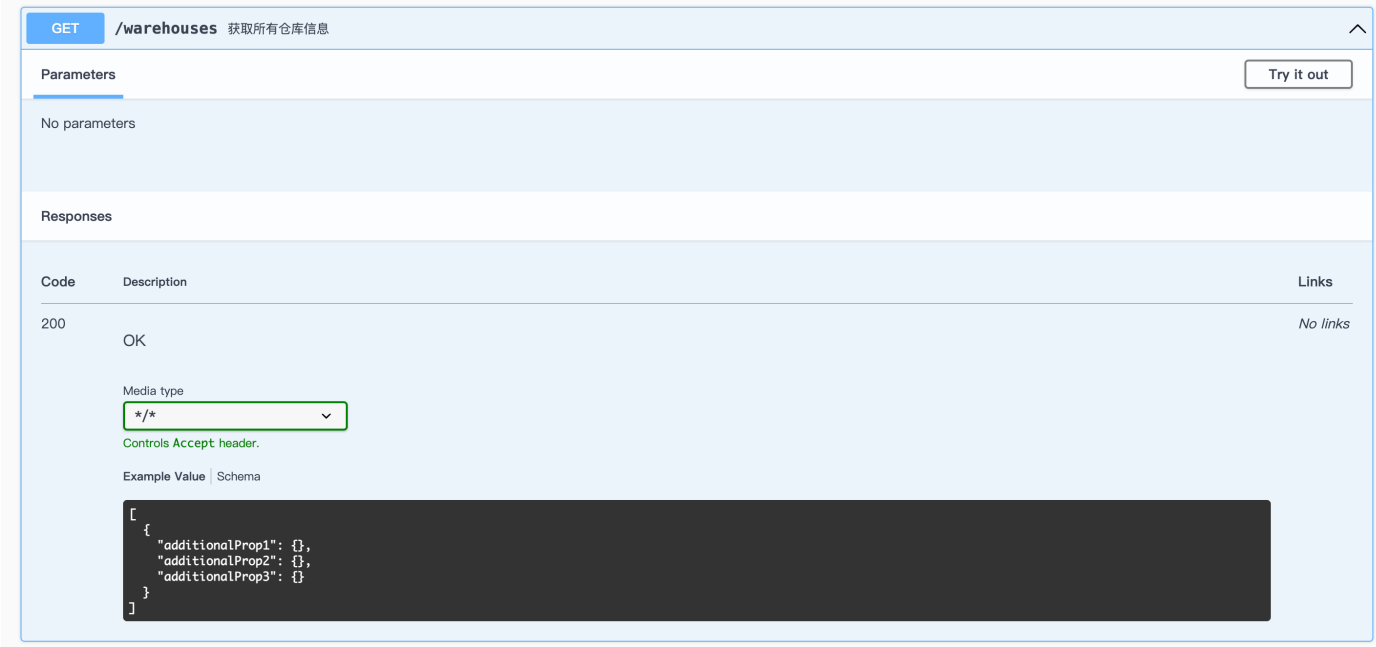
- 描述: 获取所有仓库信息
- 参数: 无
- 返回值:

```
[
    {"id": 1,"name": "Warehouse A","location": "Location A"},
    {"id": 2,"name": "Warehouse B","location": "Location B"}
]
```

- 测试:



- swagger:



2.3.2 POST /warehouses

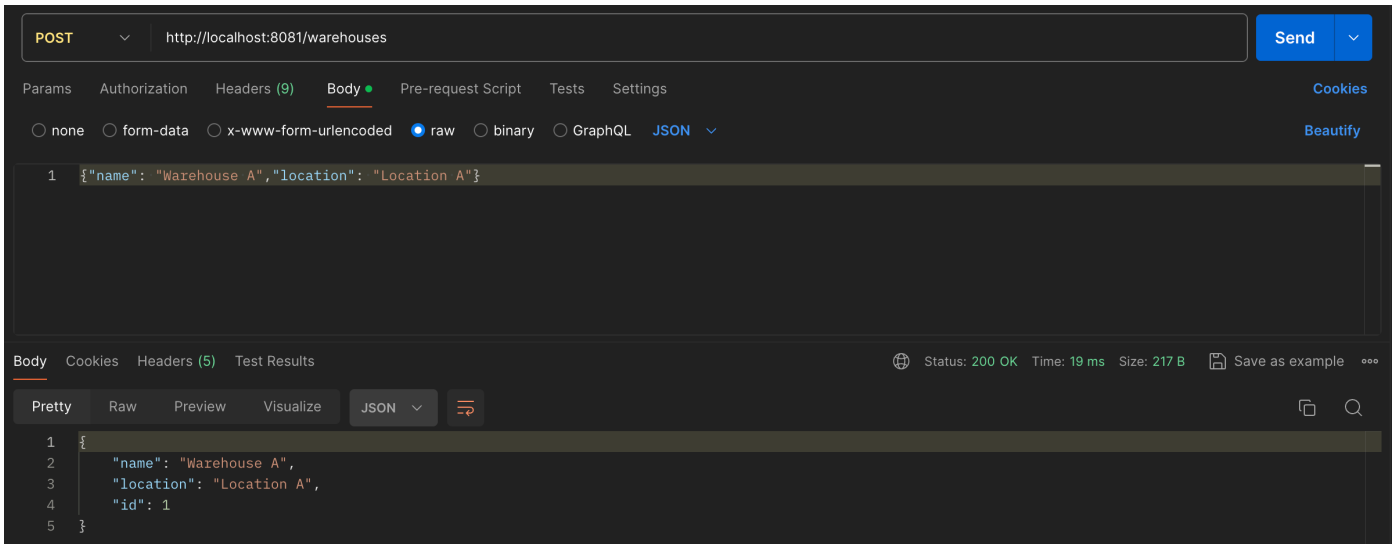
- 描述: 创建新的仓库信息
- 参数:

```
{"name": "Warehouse A","location": "Location A"}
```

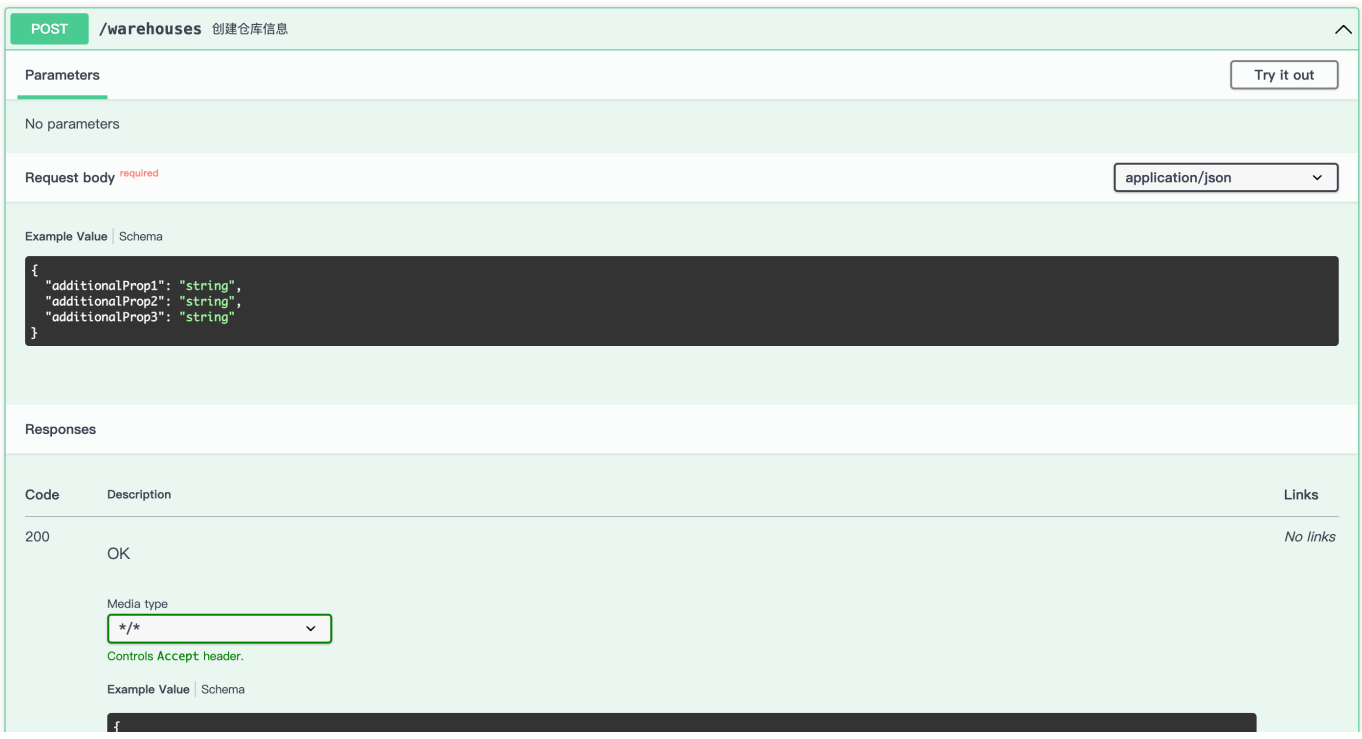
- 返回值:

```
1 {"id": 1,"name": "Warehouse A","location": "Location A"}
```

- 测试:



- swagger:



2.3.3 PUT /warehouses/{id}

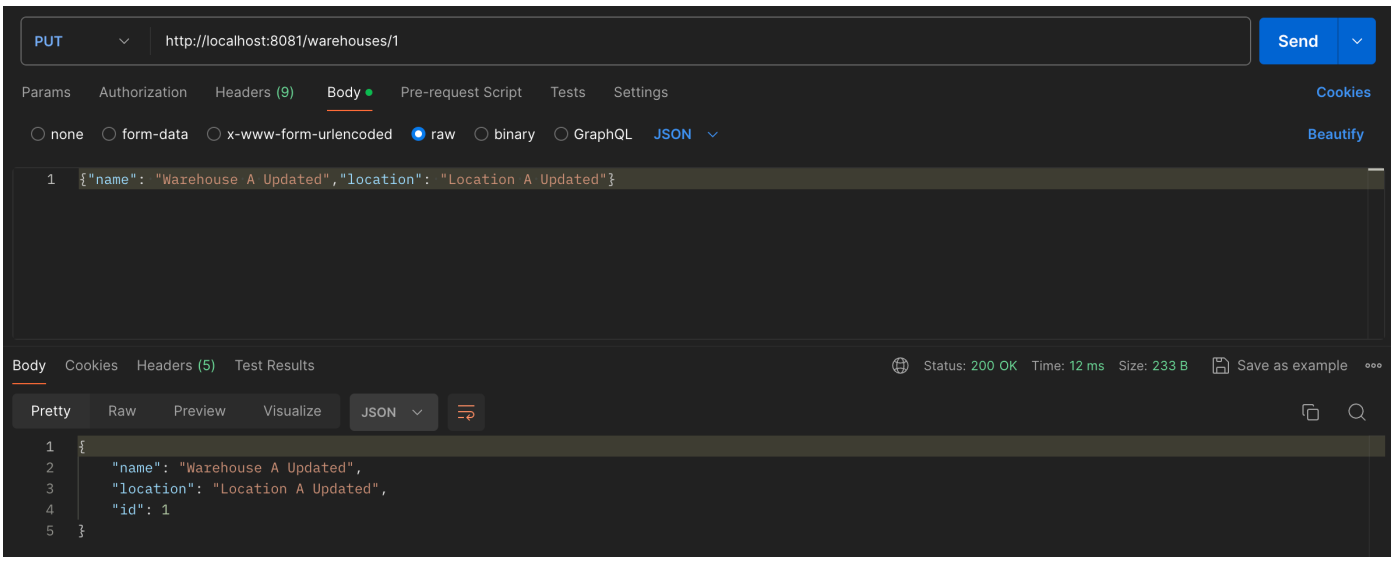
- 描述: 更新仓库信息
- 参数:

```
1 {"name": "Warehouse A Updated","location": "Location A Updated"}
```

- 返回值:

```
1 {"id": 1,"name": "Warehouse A Updated","location": "Location A Updated"}
```

- 测试:



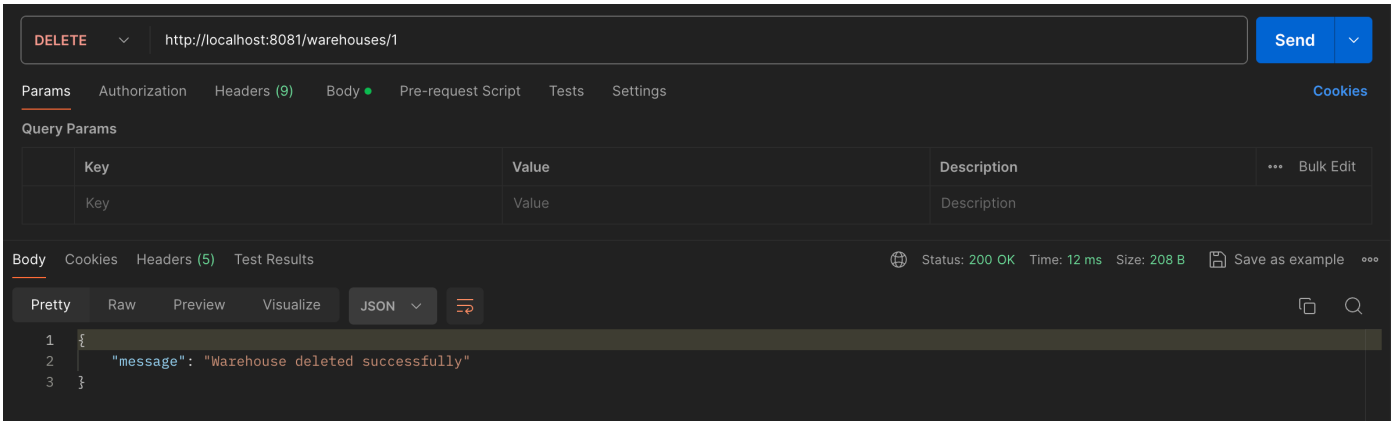
- swagger:

2.3.4 DELETE /warehouses/{id}

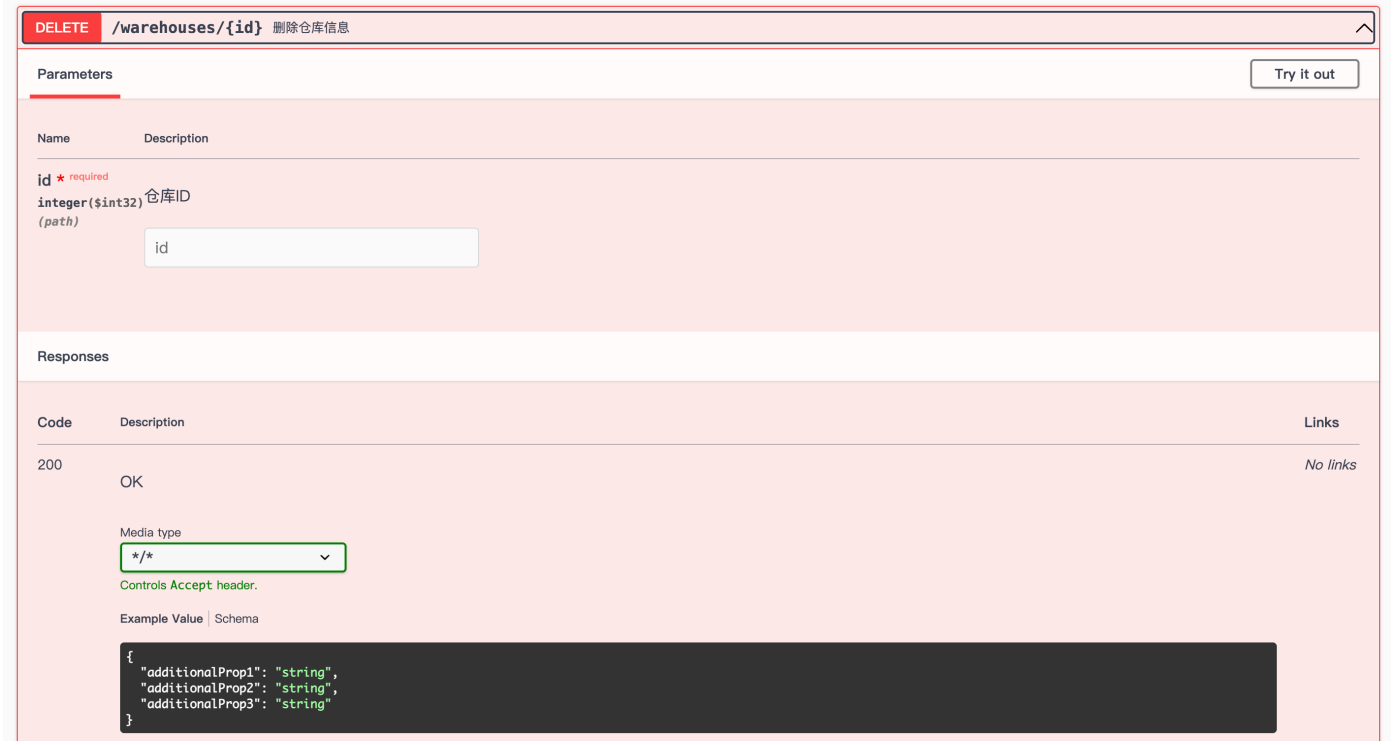
- 描述: 删除仓库信息
- 参数: 无
- 返回值:

```
{"message": "Warehouse deleted successfully"}
```

- 测试:



- swagger:



2.4 Payment Service

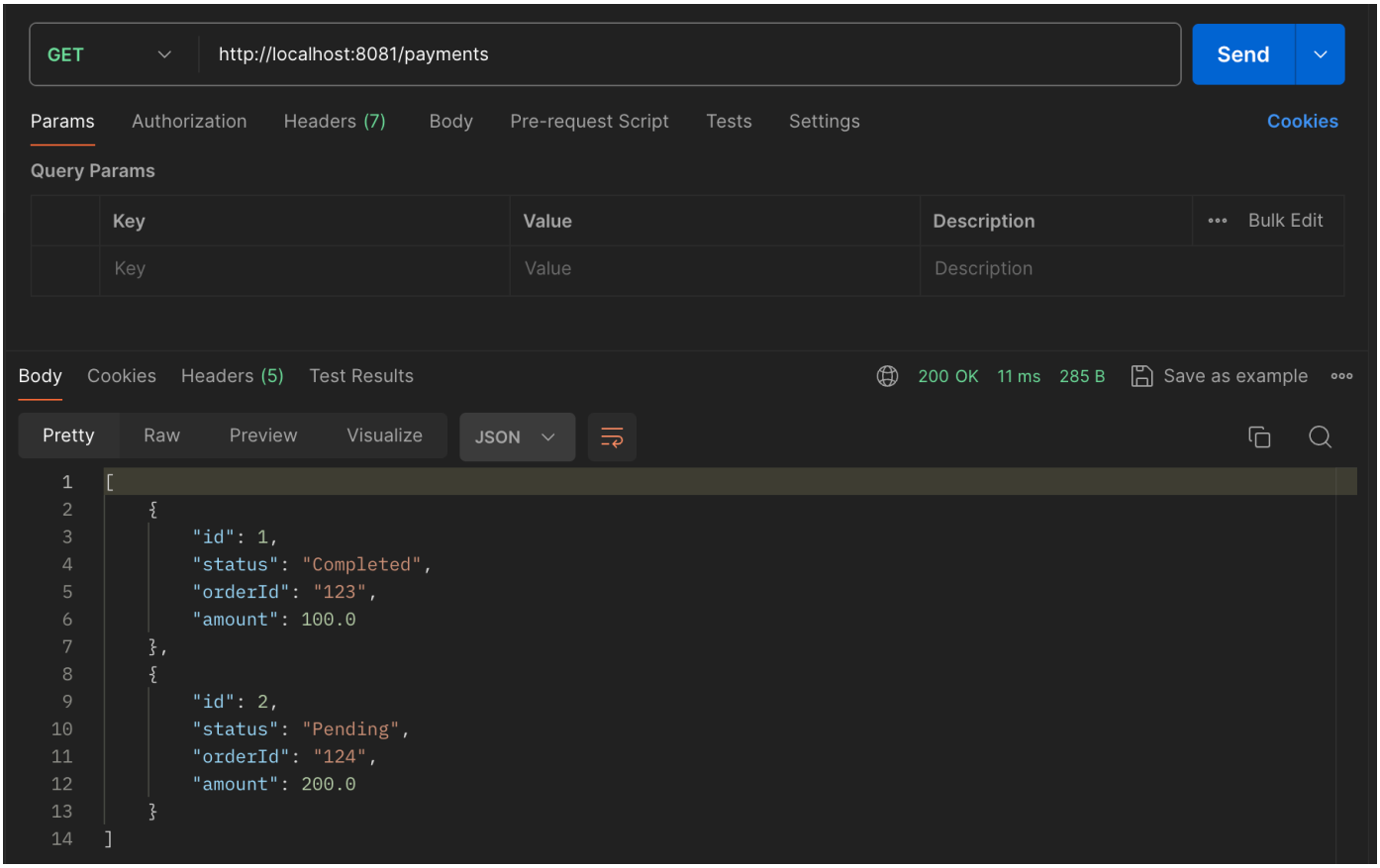
2.4.1 GET /payments

- 描述: 获取所有支付信息
- 参数: 无
- 返回值:

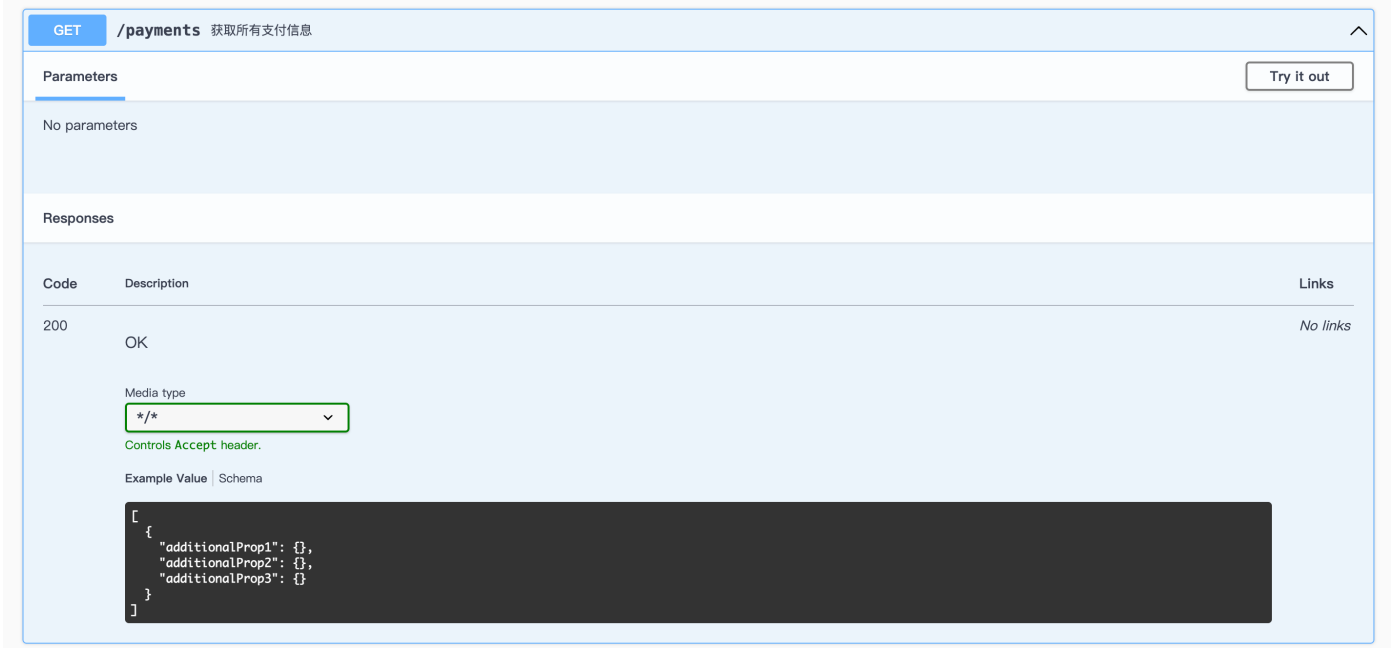
```
[
    {"id": 1,"orderId": "123","amount": 100.0,"status": "Completed"},
    {"id": 2,"orderId": "456","amount": 200.0,"status": "Pending"}
```

```
4 ]
```

- 测试:



- swagger:



2.4.2 POST /payments

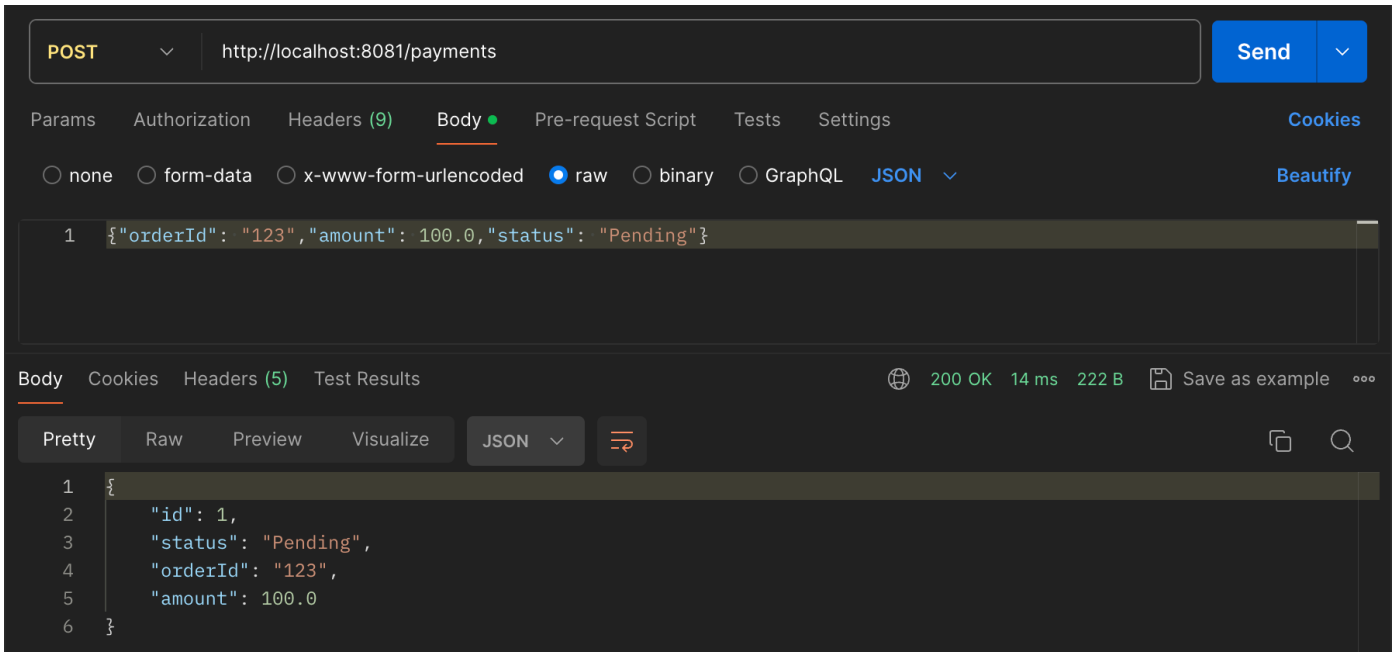
- 描述: 创建新的支付信息
- 参数:

```
1 {"orderId": "123","amount": 100.0,"status": "Pending"}
```

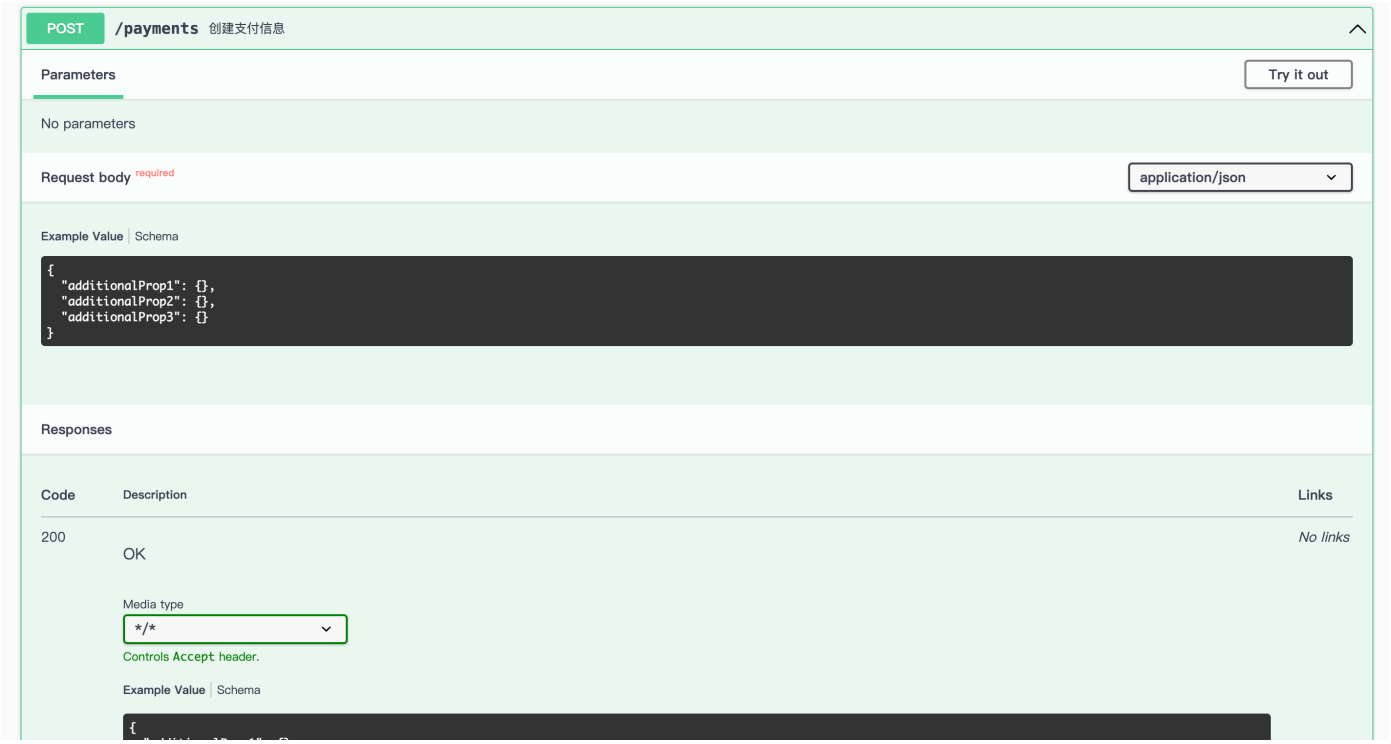
- 返回值:

```
1 {"id": 1,"orderId": "123","amount": 100.0,"status": "Pending"}
```

- 测试:



- swagger:



2.4.3 PUT /payments/{id}

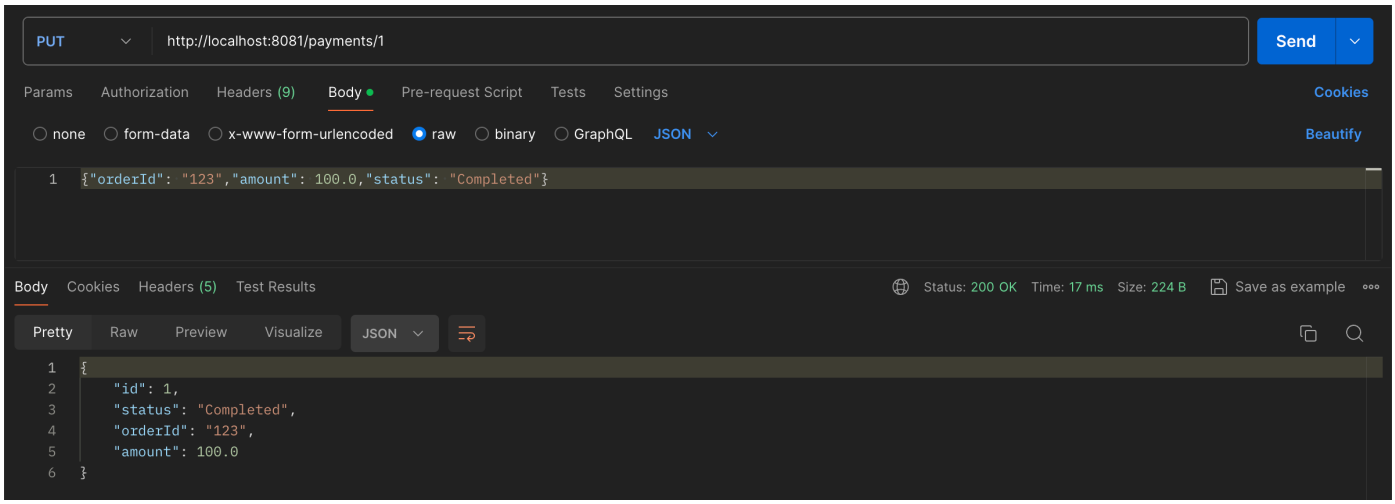
- 描述: 更新支付信息
- 参数:

```
1 {"orderId": "123","amount": 100.0,"status": "Completed"}
```

- 返回值:

```
1 {"id": 1,"orderId": "123","amount": 100.0,"status": "Completed"}
```

- 测试:



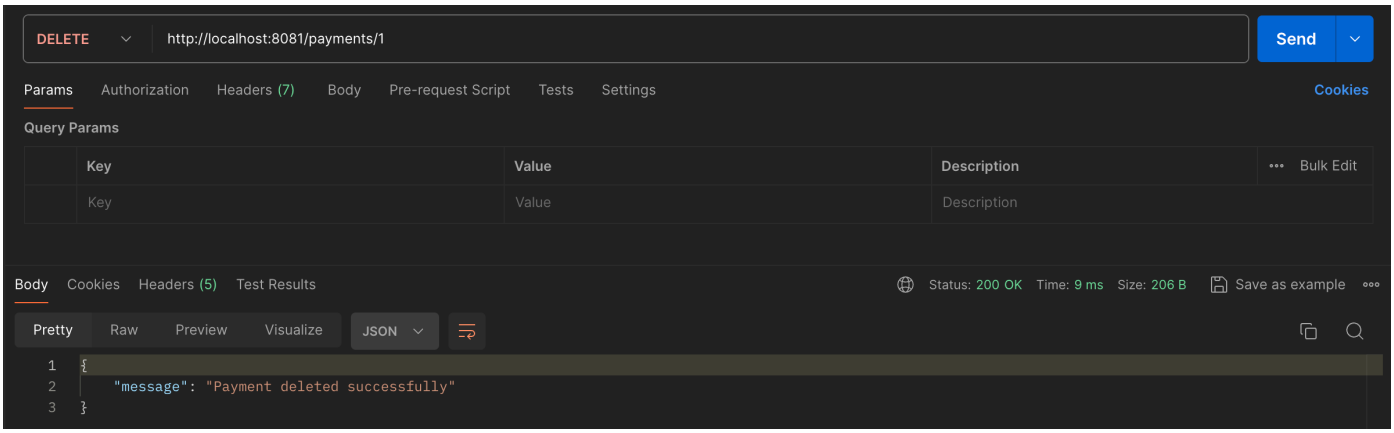
- swagger:

2.4.4 DELETE /payments/{id}

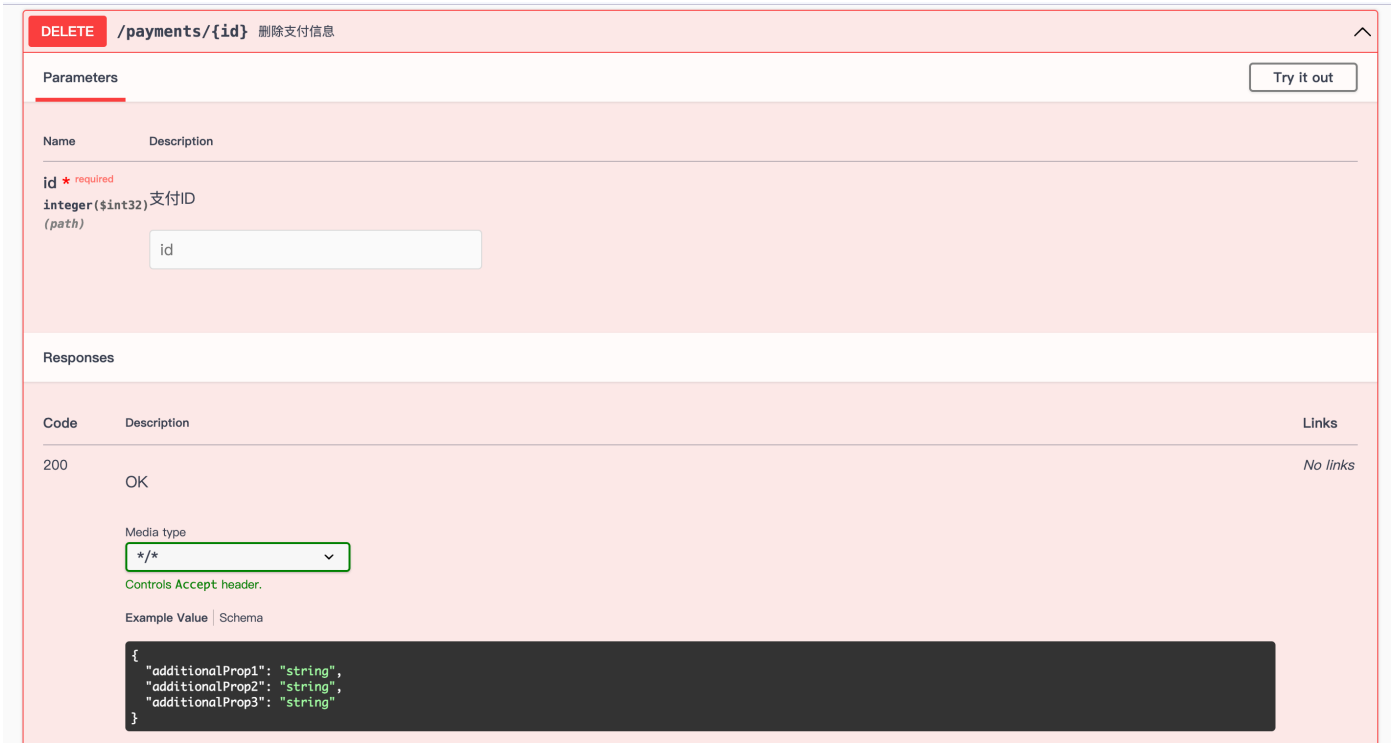
- 描述: 删除支付信息
- 参数: 无
- 返回值:

```
1 {"message": "Payment deleted successfully"}
```

- 测试:



- swagger:



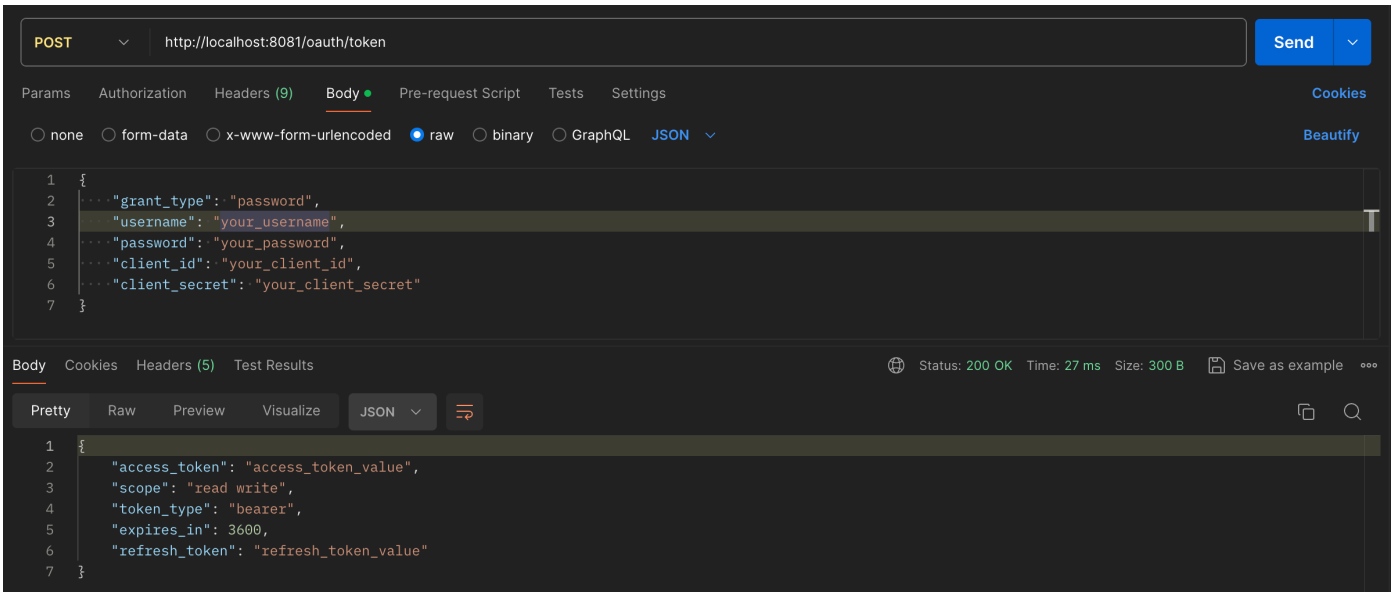
2.5 Auth Service

2.5.1 POST /oauth/token

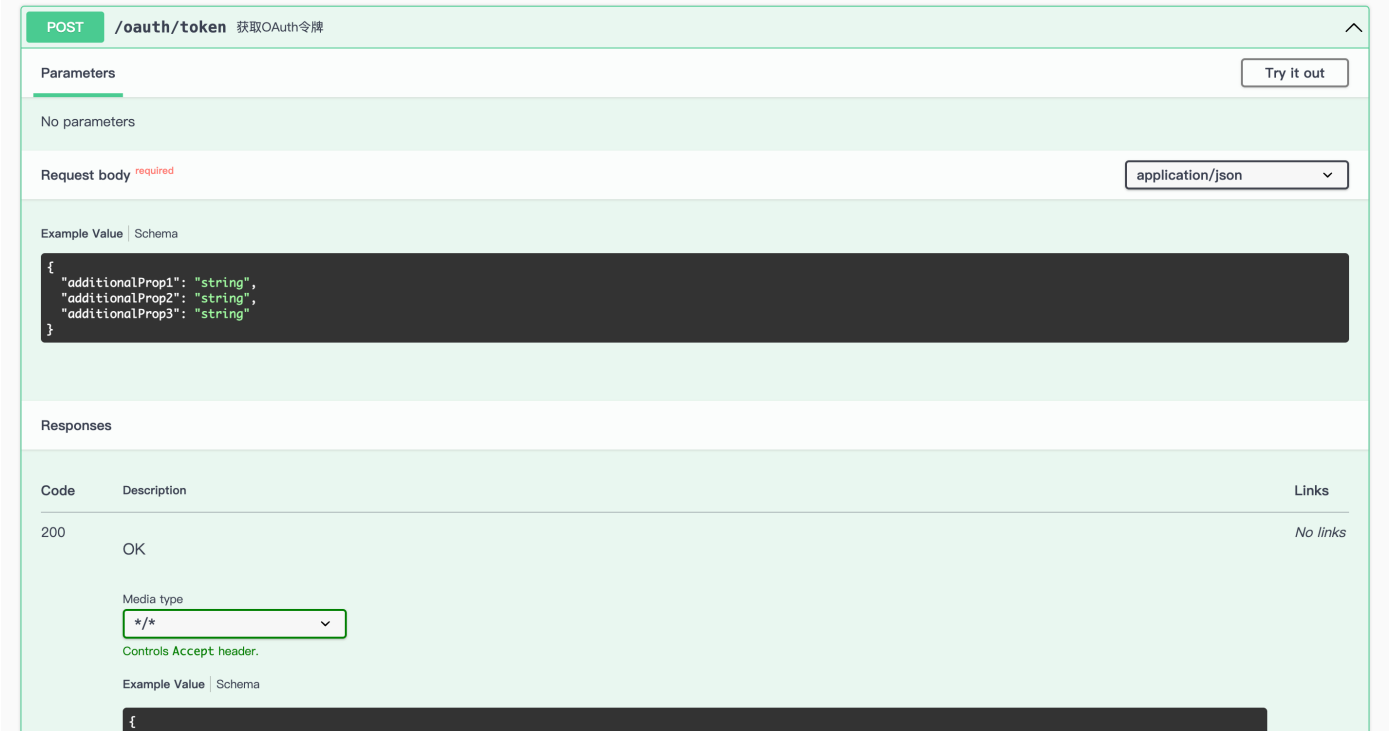
- 描述: 获取OAuth2令牌
- 参数:
  - `grant_type`: `password`
  - `username`: 用户名
  - `password`: 密码
  - `client_id`: 客户端ID
  - `client_secret`: 客户端密钥
- 返回值:

```
{
    "access_token": "access_token_value",
    "token_type": "bearer",
    "expires_in": 3600,
    "refresh_token": "refresh_token_value",
    "scope": "read write"
}
```

- 测试:



- swagger:



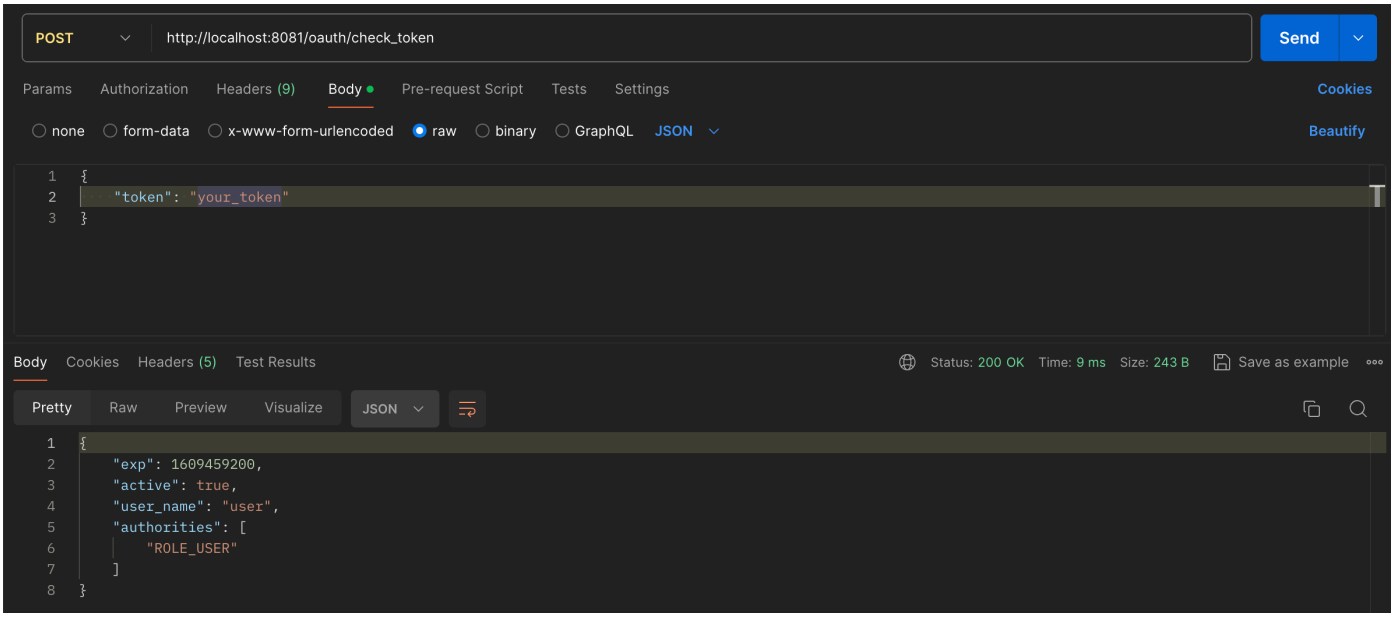
2.5.2 POST /oauth/check_token

- 描述: 检查令牌有效性
- 参数:
  - `token`: 要检查的令牌
- 返回值:

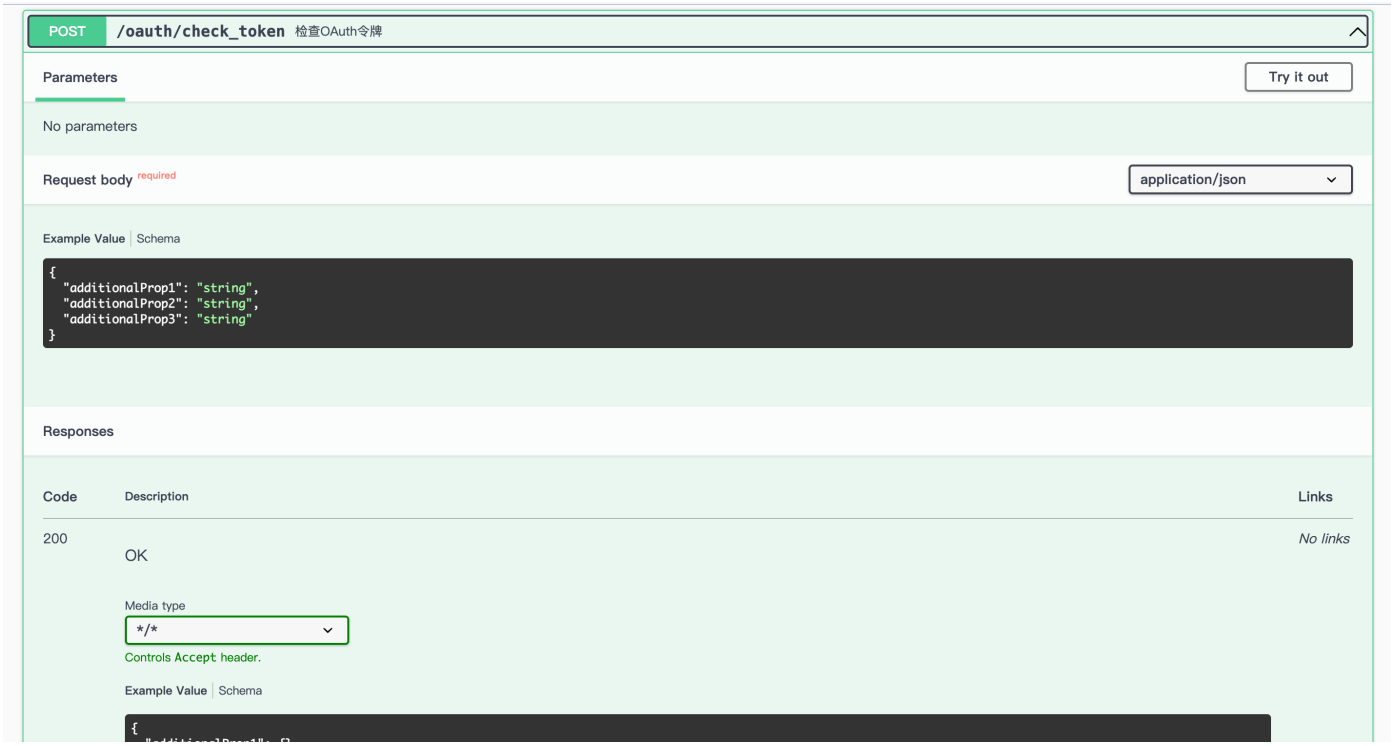
```
{
    "active": true,
    "exp": 1609459200,
    "user_name": "user",
    "authorities": ["ROLE_USER"]
```

```
6 }
```

- 测试:



- swagger:



三、关键代码

3.1 CRM Service

`CrmServiceApplication.java`

```
package com.example.crmservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication@EnableDiscoveryClient@EnableFeignClientspublic class CrmServiceApplication {public static void main(String[] args) {
        SpringApplication.run(CrmServiceApplication.class, args);
    }
}
```

`Customer.java`

```
package com.example.crmservice.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entitypublic class Customer {@Id@GeneratedValue(strategy = GenerationType.IDENTITY)private Long id;private String name;private String email;
// getters and setters
}
```

`CustomerRepository.java`

```
package com.example.crmservice.repository;

import com.example.crmservice.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```

`CustomerService.java`

```
package com.example.crmservice.service;
```

```
2
3  import com.example.crmservice.model.Customer;
4  import com.example.crmservice.repository.CustomerRepository;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  import java.util.List;
9
10 @Servicepublic class CustomerService {@Autowiredprivate CustomerRepository
   repository;
11 public List<Customer> findAll() {return repository.findAll();
12     }
13 public Customer save(Customer customer) {return repository.save(customer);
14     }
15 public Customer update(Long id, Customer customer) {Customer existingCustomer =
    repository.findById(id).orElseThrow(() -> new
   ResourceNotFoundException("Customer not found"));
16         existingCustomer.setName(customer.getName());
17         existingCustomer.setEmail(customer.getEmail());return
   repository.save(existingCustomer);
18     }
19 public void delete(Long id) {
20         repository.deleteById(id);
21     }
22 }
```

`CustomerController.java`

```
1  package com.example.crmservice.controller;
2
3  import com.example.crmservice.model.Customer;
4  import com.example.crmservice.service.CustomerService;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.web.bind.annotation.*;
7
8  import java.util.List;
9
10 @RestController@RequestMapping("/customers")public class CustomerController
   {@Autowiredprivate CustomerService service;
11 @GetMappingpublic List<Customer> getAll() {return service.findAll();
12     }
13 @PostMappingpublic Customer create(@RequestBody Customer customer) {return
   service.save(customer);
14     }
```

```
@PutMapping("/{id}")public Customer update(@PathVariable Long id, @RequestBody Customer customer) {return service.update(id, customer);
    }
@DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
        service.delete(id);return "Customer deleted successfully";
    }
}
```

3.2 Logistic Service

`LogisticServiceApplication.java`

```
package com.example.logisticservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication@EnableDiscoveryClientpublic class LogisticServiceApplication {public static void main(String[] args) {
        SpringApplication.run(LogisticServiceApplication.class, args);
    }
}
```

`Logistic.java`

```
package com.example.logisticservice.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entitypublic class Logistic {@Id@GeneratedValue(strategy = GenerationType.IDENTITY)private Long id;private String description;private String status;
// getters and setters
}
```

`LogisticRepository.java`

```java
package com.example.logisticservice.repository;

import com.example.logisticservice.model.Logistic;
import org.springframework.data.jpa.repository.JpaRepository;

public interface LogisticRepository extends JpaRepository<Logistic, Long> {
}
```

`LogisticService.java`

```java
package com.example.logisticservice.service;

import com.example.logisticservice.model.Logistic;
import com.example.logisticservice.repository.LogisticRepository;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Servicepublic class LogisticService {@Autowiredprivate LogisticRepository repository;
@CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public List<Logistic> findAll() {return repository.findAll();
    }
public Logistic save(Logistic logistic) {return repository.save(logistic);
    }
public Logistic update(Long id, Logistic logistic) {Logistic existingLogistic = repository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Logistic not found"));
        existingLogistic.setDescription(logistic.getDescription());
        existingLogistic.setStatus(logistic.getStatus());return repository.save(existingLogistic);
    }
public void delete(Long id) {
        repository.deleteById(id);
    }
public List<Logistic> fallbackMethod(Throwable throwable) {
        System.out.println("Fallback method called due to: " + throwable.getMessage());return List.of();
    }
}
```

`LogisticController.java`

```
package com.example.logisticservice.controller;

import com.example.logisticservice.model.Logistic;
import com.example.logisticservice.service.LogisticService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController@RequestMapping("/logistics")public class LogisticController
{@Autowiredprivate LogisticService service;
@GetMappingpublic List<Logistic> getAll() {return service.findAll();
    }
@PostMappingpublic Logistic create(@RequestBody Logistic logistic) {return
service.save(logistic);
    }
@PutMapping("/{id}")public Logistic update(@PathVariable Long id, @RequestBody
Logistic logistic) {return service.update(id, logistic);
    }
@DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
        service.delete(id);return "Logistic deleted successfully";
    }
}
```

3.3 Warehouse Service

`WarehouseServiceApplication.java`

```
package com.example.warehouseservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication@EnableDiscoveryClientpublic class
WarehouseServiceApplication {public static void main(String[] args) {
        SpringApplication.run(WarehouseServiceApplication.class, args);
    }
}
```

`Warehouse.java`

```
package com.example.warehouseservice.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entitypublic class Warehouse {@Id@GeneratedValue(strategy =
GenerationType.IDENTITY)private Long id;private String name;private String
location;
// getters and setters
}
```

`WarehouseRepository.java`

```
package com.example.warehouseservice.repository;

import com.example.warehouseservice.model.Warehouse;
import org.springframework.data.jpa.repository.JpaRepository;

public interface WarehouseRepository extends JpaRepository<Warehouse, Long> {
}
```

`WarehouseService.java`

```
package com.example.warehouseservice.service;

import com.example.warehouseservice.model.Warehouse;
import com.example.warehouseservice.repository.WarehouseRepository;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Servicepublic class WarehouseService {@Autowiredprivate WarehouseRepository
repository;
@CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public
List<Warehouse> findAll() {return repository.findAll();
    }
public Warehouse save(Warehouse warehouse) {return repository.save(warehouse);
    }
```

```
16 public Warehouse update(Long id, Warehouse warehouse) {Warehouse
   existingWarehouse = repository.findById(id).orElseThrow(() -> new
   ResourceNotFoundException("Warehouse not found"));
17         existingWarehouse.setName(warehouse.getName());
18         existingWarehouse.setLocation(warehouse.getLocation());return
   repository.save(existingWarehouse);
19     }
20 public void delete(Long id) {
21         repository.deleteById(id);
22     }
23 public List<Warehouse> fallbackMethod(Throwable throwable) {
24         System.out.println("Fallback method called due to: " +
   throwable.getMessage());return List.of();
25     }
26 }
```

`WarehouseController.java`

```
1  package com.example.warehouseservice.controller;
2
3  import com.example.warehouseservice.model.Warehouse;
4  import com.example.warehouseservice.service.WarehouseService;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.web.bind.annotation.*;
7
8  import java.util.List;
9
10 @RestController@RequestMapping("/warehouses")public class WarehouseController
   {@Autowiredprivate WarehouseService service;
11 @GetMappingpublic List<Warehouse> getAll() {return service.findAll();
12     }
13 @PostMappingpublic Warehouse create(@RequestBody Warehouse warehouse) {return
   service.save(warehouse);
14     }
15 @PutMapping("/{id}")public Warehouse update(@PathVariable Long id, @RequestBody
    Warehouse warehouse) {return service.update(id, warehouse);
16     }
17 @DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
18         service.delete(id);return "Warehouse deleted successfully";
19     }
20 }
```

3.4 Payment Service

`PaymentServiceApplication.java`

```
package com.example.paymentservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication@EnableDiscoveryClientpublic class
PaymentServiceApplication {public static void main(String[] args) {
        SpringApplication.run(PaymentServiceApplication.class, args);
    }
}
```

`Payment.java`

```
package com.example.paymentservice.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entitypublic class Payment {@Id@GeneratedValue(strategy =
GenerationType.IDENTITY)private Long id;private String orderId;private double
amount;private String status;
// getters and setters
}
```

`PaymentRepository.java`

```
package com.example.paymentservice.repository;

import com.example.paymentservice.model.Payment;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PaymentRepository extends JpaRepository<Payment, Long> {
}
```

`PaymentService.java`

```
package com.example.paymentservice.service;

import com.example.paymentservice.model.Payment;
import com.example.paymentservice.repository.PaymentRepository;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Servicepublic class PaymentService {@Autowiredprivate PaymentRepository repository;
@CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public List<Payment> findAll() {return repository.findAll();
    }
public Payment save(Payment payment) {return repository.save(payment);
    }
public Payment update(Long id, Payment payment) {Payment existingPayment = repository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Payment not found"));
        existingPayment.setOrderId(payment.getOrderId());
        existingPayment.setAmount(payment.getAmount());
        existingPayment.setStatus(payment.getStatus());return repository.save(existingPayment);
    }
public void delete(Long id) {
        repository.deleteById(id);
    }
public List<Payment> fallbackMethod(Throwable throwable) {
        System.out.println("Fallback method called due to: " + throwable.getMessage());return List.of();
    }
}
```

`PaymentController.java`

```
package com.example.paymentservice.controller;

import com.example.paymentservice.model.Payment;
import com.example.paymentservice.service.PaymentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

```

```
@RestController@RequestMapping("/payments")public class PaymentController {@Autowiredprivate PaymentService service;
@GetMappingpublic List<Payment> getAll() {return service.findAll();
    }
@PostMappingpublic Payment create(@RequestBody Payment payment) {return service.save(payment);
    }
@PutMapping("/{id}")public Payment update(@PathVariable Long id, @RequestBody Payment payment) {return service.update(id, payment);
    }
@DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
        service.delete(id);return "Payment deleted successfully";
    }
}
```

3.5 Auth Service

`AuthServiceApplication.java`

```
package com.example.authservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication@EnableDiscoveryClientpublic class AuthServiceApplication {public static void main(String[] args) {
        SpringApplication.run(AuthServiceApplication.class, args);
    }
}
```

`AuthorizationServerConfig.java`

```
package com.example.authservice.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.web.configuration.EnableAuthorizationServer;
import org.springframework.security.oauth2.config.annotation.web.configuration.Authori
```

```
zationServerConfigurerAdapter;
8 import
org.springframework.security.oauth2.config.annotation.web.configurers.Authoriza
tionServerEndpointsConfigurer;
9 import
org.springframework.security.oauth2.config.annotation.web.configurers.Authoriza
tionServerSecurityConfigurer;
10 import
org.springframework.security.oauth2.config.annotation.web.configurers.ClientDet
ailsServiceConfigurer;
11 import org.springframework.security.core.userdetails.User;
12 import org.springframework.security.core.userdetails.UserDetails;
13 import org.springframework.security.core.userdetails.UserDetailsManager;
14 import
org.springframework.security.core.userdetails.memory.InMemoryUserDetailsManager
;
15 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
16
17 @Configuration@EnableAuthorizationServerpublic class AuthorizationServerConfig
extends AuthorizationServerConfigurerAdapter {
18 @Beanpublic AuthenticationManager authenticationManager() throws Exception
{return super.authenticationManager();
19     }
20 @Overridepublic void configure(ClientDetailsServiceConfigurer clients) throws
Exception {
21         clients.inMemory()
22                 .withClient("my-client-id")
23                 .secret(new BCryptPasswordEncoder().encode("my-client-secret"))
24                 .scopes("read", "write")
25                 .authorizedGrantTypes("password", "refresh_token")
26                 .accessTokenValiditySeconds(3600)
27                 .refreshTokenValiditySeconds(7200);
28     }
29 @Overridepublic void configure(AuthorizationServerEndpointsConfigurer
endpoints) throws Exception {
30         endpoints
31                 .authenticationManager(authenticationManager())
32                 .userDetailsService(userDetailsService());
33     }
34 @Overridepublic void configure(AuthorizationServerSecurityConfigurer security)
throws Exception {
35         security
36                 .tokenKeyAccess("permitAll()")
37                 .checkTokenAccess("isAuthenticated()");
38     }
39 @Beanpublic UserDetailsService userDetailsService() {UserDetails user =
User.withDefaultPasswordEncoder()
```

```
                .username("user")
                .password("password")
                .roles("USER")
                .build();return new InMemoryUserDetailsManager(user);
    }
}
```

四、实现总结

4.1 关键内容

4.1.1 服务发现

所有服务都已成功注册到Nacos，并可以通过Nacos控制台进行管理和监控。

4.1.2 断路器

在每个服务中实现了Resilience4j断路器，确保服务的稳定性和容错能力。

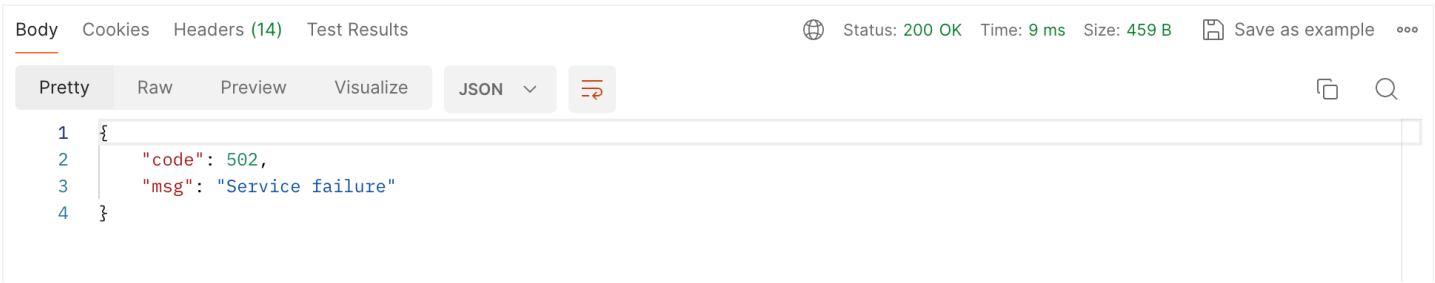
1. 配置

```
import io.github.resilience4j.circuitbreaker.CircuitBreakerConfig;
import io.github.resilience4j.circuitbreaker.CircuitBreakerRegistry;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class Resilience4jConfig {

    @Bean
    public CircuitBreakerRegistry circuitBreakerRegistry() {
        CircuitBreakerConfig config = CircuitBreakerConfig.custom()
                .failureRateThreshold(50)
                .waitDurationInOpenState(java.time.Duration.ofMillis(1000))
                .slidingWindowSize(2)
                .build();
        return CircuitBreakerRegistry.of(config);
    }
}

```

2. 模拟断路控制器

```
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Map;

@RestController
public class SimpleController {

    @CircuitBreaker(name = "simpleCircuitBreaker", fallbackMethod =
"fallbackMethod")
    @GetMapping("/unstable")
    public Map<String, String> unstableEndpoint() {
        // 模拟一个不稳定的服务调用
        if (Math.random() > 0.5) {
            throw new RuntimeException("Service failure");
        }
        return Map.of("message", "Service call successful");
    }

    public Map<String, String> fallbackMethod(Throwable t) {
        return Map.of("message", "Service is currently unavailable, please try
again later");
    }
}

```



4.1.3 OAuth2身份验证

集成了OAuth2授权服务器，提供安全的身份验证和授权功能。

1. 配置类

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
```

```java
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails user = User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build();
        return new InMemoryUserDetailsManager(user);
    }

    @Bean
    public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
throws Exception {
        http
            .authorizeRequests(authorizeRequests ->
                authorizeRequests.anyRequest().authenticated()
            )
            .formLogin().and()
            .oauth2ResourceServer().jwt();
        return http.build();
    }
}

```

2. 配置授权服务器

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
```

```
    public UserDetailsService userDetailsService() {
        UserDetails user = User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build();
        return new InMemoryUserDetailsManager(user);
    }

    @Bean
    public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
throws Exception {
        http
            .authorizeRequests(authorizeRequests ->
                authorizeRequests.anyRequest().authenticated()
            )
            .formLogin().and()
            .oauth2ResourceServer().jwt();
        return http.build();
    }
}

```

3. 配置OAuth2客户端

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.OAuth2ClientCo
nfiguration;
import
org.springframework.security.oauth2.client.InMemoryOAuth2AuthorizedClientServic
e;
import
org.springframework.security.oauth2.client.OAuth2AuthorizedClientService;
import
org.springframework.security.oauth2.client.registration.ClientRegistration;
import
org.springframework.security.oauth2.client.registration.InMemoryClientRegistrat
ionRepository;

@Configuration
public class OAuth2ClientConfig {
```

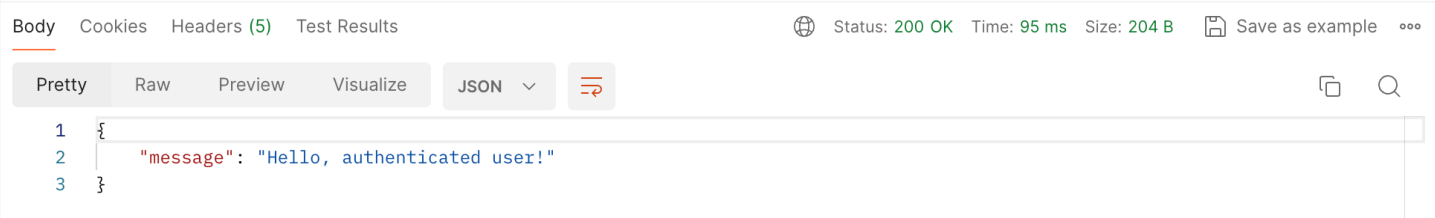
```
12
13      @Bean
14      public InMemoryClientRegistrationRepository clientRegistrationRepository()
    {
15          return new
    InMemoryClientRegistrationRepository(this.googleClientRegistration());
16      }
17
18      @Bean
19      public OAuth2AuthorizedClientService authorizedClientService() {
20          return new
    InMemoryOAuth2AuthorizedClientService(clientRegistrationRepository());
21      }
22
23      private ClientRegistration googleClientRegistration() {
24          return ClientRegistration.withRegistrationId("google")
25                  .clientId("client-id")
26                  .clientSecret("client-secret")
27                  .scope("openid", "profile", "email")
28                  .authorizationUri("https://accounts.google.com/o/oauth2/auth")
29                  .tokenUri("https://oauth2.googleapis.com/token")
30                  .userInfoUri("https://www.googleapis.com/oauth2/v3/userinfo")
31                  .redirectUri("{baseUrl}/login/oauth2/code/google")
32                  .clientName("Google")
33                  .build();
34      }
35  }
36
```

4. 配置授权服务器

```
1  import org.springframework.context.annotation.Bean;
2  import org.springframework.context.annotation.Configuration;
3  import
    org.springframework.security.config.annotation.web.builders.HttpSecurity;
4  import
    org.springframework.security.oauth2.server.authorization.config.annotation.web.
    configuration.OAuth2AuthorizationServerConfiguration;
5  import org.springframework.security.web.SecurityFilterChain;
6
7  @Configuration
8  public class AuthorizationServerConfig {
9
10     @Bean
```

```
    public SecurityFilterChain
authorizationServerSecurityFilterChain(HttpSecurity http) throws Exception {
        OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
        return http.formLogin().and().build();
    }
}

```

5. 验证信息



4.1.4 API网关

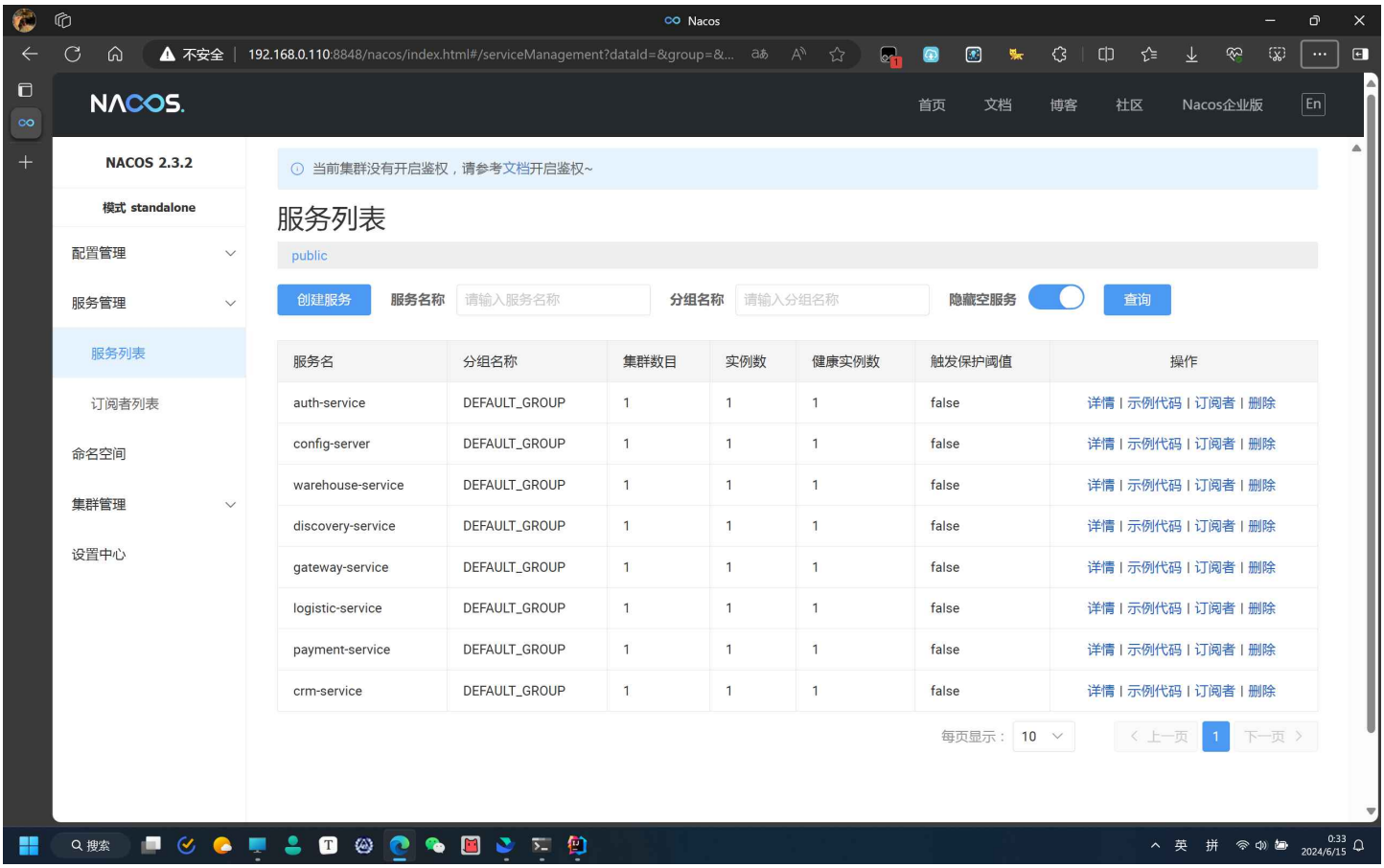
使用Spring Cloud Gateway实现API网关，提供统一的API入口。

4.1.5 集中配置和跟踪

使用Spring Cloud Config Server进行集中配置管理，并使用Spring Cloud Sleuth和Zipkin进行分布式跟踪。

4.2 启动并验证所有服务

1. 启动Nacos Server。
2. 启动 `config-server` 。
3. 启动 `discovery-service` 、 `gateway-service` 、 `crm-service` 、 `logistic-service` 、 `warehouse-service` 、 `payment-service` 和 `auth-service` 。
4. 验证服务是否已成功注册到Nacos。
5. 验证每个服务是否从Config Server加载配置。
6. 使用Postman或其他工具测试服务之间的调用，验证断路器功能。
7. 使用Postman或其他工具测试OAuth2身份验证服务，验证授权和令牌颁发是否正常工作。
8. 访问 `http://localhost:9411` 查看Zipkin的跟踪数据，确保所有请求的跟踪数据能够正确显示。


Nacos
不安全 | 192.168.0.110:8848/nacos/index.html#/serviceManagement?dataId=&group=&...
NACOS.
首页
文档
博客
社区
Nacos企业版
En
NACOS 2.3.2
模式 standalone
配置管理
服务管理
服务列表
订阅者列表
命名空间
集群管理
设置中心
当前集群没有开启鉴权，请参考文档开启鉴权~
服务列表
public
创建服务
服务名称
请输入服务名称
分组名称
请输入分组名称
隐藏空服务
查询
服务名 分组名称 集群数目 实例数 健康实例数 触发保护阈值 操作
auth-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
config-server DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
warehouse-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
discovery-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
gateway-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
logistic-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
payment-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
crm-service DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
每页显示： 10
上一页
1
下一页
搜索
英
拼
0:33
2024/6/15