

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](#)。

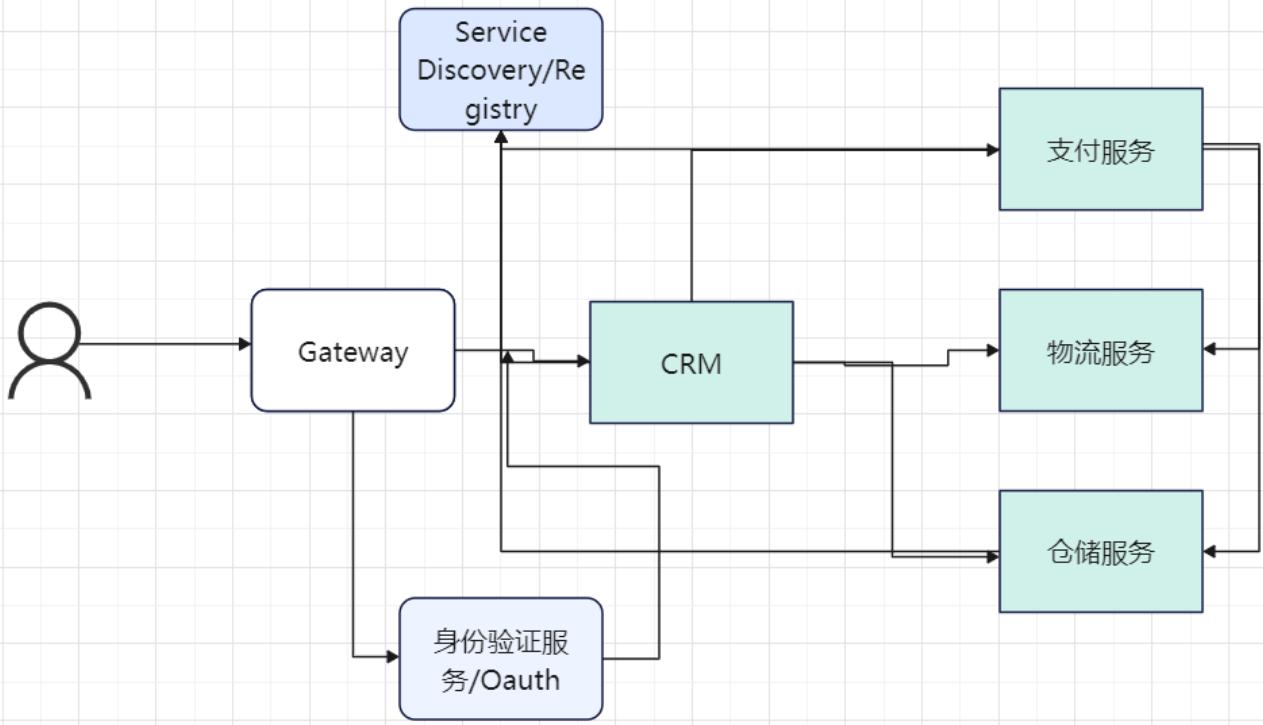
在本阶段中，我们开发了一个微服务系统，该系统集成了多个服务，例如CRM服务、物流服务、自动仓库服务和支付服务。例如，当客户下订单将一批货物存储到仓库时，CRM服务将调用相应的物流服务、仓库服务和支付服务来完成服务交易。为了开发这个系统，我们还需要添加微服务基础设施，如服务发现、网关和OAuth服务器。

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。

The screenshot shows the Nacos service management interface. On the left, there's a sidebar with sections like 'NACOS 2.3.2', '模式 standalone', '配置管理', '服务管理', '服务列表' (which is selected), '订阅者列表', '命名空间', '集群管理', and '设置中心'. The main area is titled '服务列表' and shows a table of services. The table has columns: 服务名 (Service Name), 分组名称 (Group Name), 集群数目 (Cluster Count), 实例数 (Instance Count), 健康实例数 (Healthy Instance Count), 触发保护阀值 (Protection Valve Triggered), and 操作 (Operations). Services listed include auth-service, config-server, warehouse-service, discovery-service, gateway-service, logistic-service, payment-service, and crm-service, all grouped under 'DEFAULT_GROUP'. At the bottom, there are pagination controls for '每页显示' (Items per page) set to 10, and navigation buttons for '上一页' (Previous page), '下一页' (Next page).

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。

A screenshot of a JSON response in a browser developer tools or similar interface. The response is:

```
1 {
2   "code": 502,
3   "msg": "Service failure"
4 }
```

Details: Status: 200 OK Time: 9 ms Size: 459 B Save as example

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：

```
1 <dependency>
```

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

在 `application.properties` 中配置：

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

在服务类中使用断路器注解：

```
1 package com.example.logisticservice.service;
2
3 import com.example.logisticservice.model.Logistic;
4 import com.example.logisticservice.repository.LogisticRepository;
5 import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import java.util.List;
10
11 @Servicepublic class LogisticService {@Autowiredprivate LogisticRepository
12   repository;
13   @CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public
14   List<Logistic> findAll() {return repository.findAll();
15   }
16   public Logistic save(Logistic logistic) {return repository.save(logistic);
17   }
18   public void delete(Long id) {
19     repository.deleteById(id);
20   }
21   public Logistic update(Long id, Logistic logistic) {Logistic existingLogistic =
22     repository.findById(id).orElseThrow(() -> new
23     ResourceNotFoundException("Logistic not found"));
24     existingLogistic.setDescription(logistic.getDescription());
25     existingLogistic.setStatus(logistic.getStatus());return
26     repository.save(existingLogistic);
27   }
28   public List<Logistic> fallbackMethod(Throwable throwable) {// 自定义的fallback逻辑
29 }
```

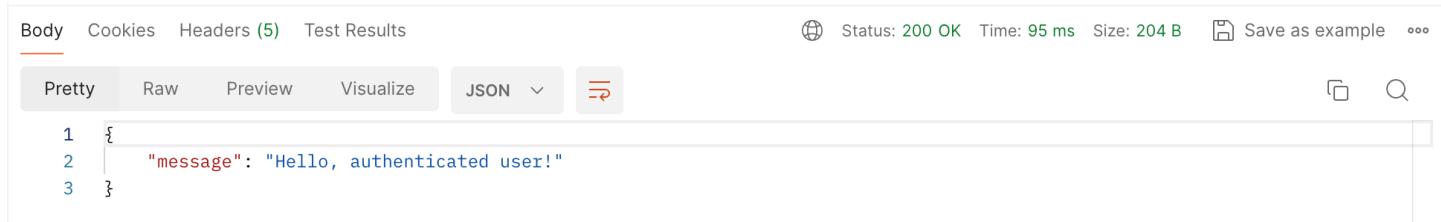
```
24         System.out.println("Fallback method called due to: " +
25             throwable.getMessage());return List.of();
26 }
```

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

```
1 <dependencies>
2     <dependency>
3         <groupId>org.springframework.boot</groupId>
4         <artifactId>spring-boot-starter-web</artifactId>
5     </dependency>
6     <dependency>
7         <groupId>org.springframework.cloud</groupId>
8         <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
9     </dependency>
10    <dependency>
11        <groupId>org.springframework.boot</groupId>
12        <artifactId>spring-boot-starter-security</artifactId>
13    </dependency>
14    <dependency>
15        <groupId>org.springframework.security</groupId>
16        <artifactId>spring-security-oauth2-authorization-server</artifactId>
17        <version>0.4.0</version>
18    </dependency>
19    <dependency>
20        <groupId>org.springframework.boot</groupId>
21        <artifactId>spring-boot-starter-data-jpa</artifactId>
22    </dependency>
```

```
23 <dependency>
24     <groupId>jakarta.persistence</groupId>
25     <artifactId>jakarta.persistence-api</artifactId>
26     <version>3.1.0</version></dependency>
27 <dependency>
28     <groupId>org.hibernate.orm</groupId>
29     <artifactId>hibernate-core</artifactId>
30     <version>6.2.3.Final</version>
31 </dependency>
32 <dependency>
33     <groupId>com.h2database</groupId>
34     <artifactId>h2</artifactId>
35     <scope>runtime</scope>
36 </dependency>
37 <dependency>
38     <groupId>com.alibaba.cloud</groupId>
39     <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
40 </dependency>
41 </dependencies>
```

application.properties

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

AuthServiceApplication.java

```
1 package com.example.authservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication@EnableDiscoveryClientpublic class AuthServiceApplication
8     {public static void main(String[] args) {
9         SpringApplication.run(AuthServiceApplication.class, args);
10    }}
```

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
7     org.springframework.security.config.annotation.web.configuration.EnableAuthorizationServer;
8 import
9     org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
10 import
11     org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
12 import
13     org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;
14 import
15     org.springframework.security.oauth2.config.annotation.web.configurers.ClientDetailsServiceConfigurer;
16
17 @Configuration@EnableAuthorizationServerpublic class AuthorizationServerConfig
18     extends AuthorizationServerConfigurerAdapter {
19     @Beanpublic AuthenticationManager authenticationManager() throws Exception
20     {return super.authenticationManager();}
21
22     @Overridepublic void configure(ClientDetailsServiceConfigurer clients) throws
23     Exception {
24         clients.inMemory()
25             .withClient("my-client-id")
26             .secret(new BCryptPasswordEncoder().encode("my-client-secret"))
27             .scopes("read", "write")
28             .authorizedGrantTypes("password", "refresh_token")
29             .accessTokenValiditySeconds(3600)
30             .refreshTokenValiditySeconds(7200);
31     }
32 }
```

```

29 @Override public void configure(AuthorizationServerEndpointsConfigurer
30     endpoints) throws Exception {
31     endpoints
32         .authenticationManager(authenticationManager())
33         .userDetailsService(userDetailsService());
34 }
35 @Override public void configure(AuthorizationServerSecurityConfigurer security)
36     throws Exception {
37     security
38         .tokenKeyAccess("permitAll()")
39         .checkTokenAccess(" isAuthenticated()");
40 }
41 @Bean public UserDetailsService userDetailsService() {UserDetails user =
42     User.withDefaultPasswordEncoder()
43         .username("user")
44         .password("password")
45         .roles("USER")
46         .build();return new InMemoryUserDetailsManager(user);
47 }
48 }

```

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`

```

1 <dependencies>
2     <dependency>
3         <groupId>org.springframework.boot</groupId>
4         <artifactId>spring-boot-starter-webflux</artifactId>
5     </dependency>
6     <dependency>
7         <groupId>org.springframework.cloud</groupId>
8         <artifactId>spring-cloud-starter-gateway</artifactId>
9     </dependency>
10    <dependency>
11        <groupId>com.alibaba.cloud</groupId>
12        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
13    </dependency>
14 </dependencies>

```

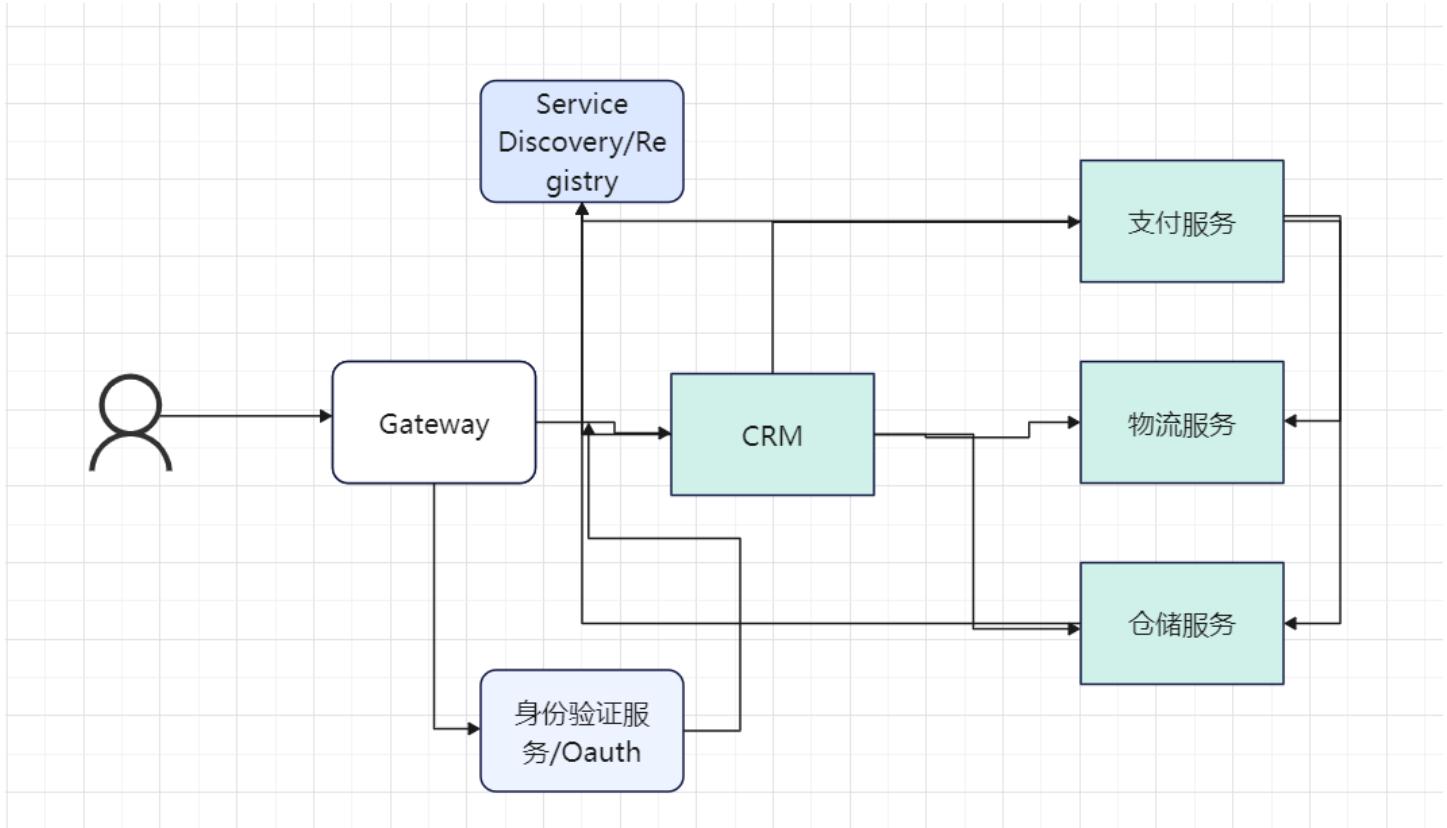
`application.yml`

```
1 server:
2   port: 8081
3
4 spring:
5   application:
6     name: gateway-service
7   cloud:
8     compatibility-verifier:
9       enabled: true
10  nacos:
11    discovery:
12      server-addr: 127.0.0.1:8848
13 gateway:
14   routes:
15     - id: crm-service
16       uri: lb://crm-service
17       predicates:
18         - Path=/crm/**
19       filters:
20         - StripPrefix=1
21
22     - id: logistic-service
23       uri: lb://logistic-service
24       predicates:
25         - Path=/logistic/**
26       filters:
27         - StripPrefix=1
28
29     - id: warehouse-service
30       uri: lb://warehouse-service
31       predicates:
32         - Path=/warehouse/**
33       filters:
34         - StripPrefix=1
35
36     - id: gateway-service
37       uri: lb://gateway-service
38       predicates:
39         - Path=/***
40       filters:
41         - StripPrefix=1
42
43     - id: discovery-service
44       uri: lb://discovery-service
45       predicates:
46         - Path=/discovery/**
47       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）：

```
2024-06-15 22:08:51.882 DEBUG [gateway-service,,] 33564 --- [nio-7000-exec-1]
2024-06-15 22:08:51.883 DEBUG [gateway-service,,] 33564 --- [nio-7000-exec-1]
2024-06-15 22:08:51.883 DEBUG [gateway-service,,] 33564 --- [nio-7000-exec-1]
2024-06-15 22:08:52.268 DEBUG [gateway-service,,] 33564 --- [ing.beat.sender]
2024-06-15 22:08:52.269 DEBUG [gateway-service,,] 33564 --- [ing.beat.sender]
```

总的追踪图如下：

Zipkin

localhost:9411/zipkin/?lookback=5m&endTs=1718460631329&limit=10

Find a trace Dependencies Search by trace ID EN

Results

Root Start Time Spans Duration

crm-service: get /warehouse 4 minutes ago (06/15 22:06:35:605) 3 262.440ms

crm-service (2) warehouse-service (1)

crm-service: get /payment 4 minutes ago (06/15 22:06:48:713) 7 45.462ms

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

crm-service (2) logistic-service (1)

Service filters crm-service Service filters

Zipkin

localhost:9411/zipkin/?lookback=millis&endTs=1718460765430&millis=500000&limit=10

Find a trace Dependencies Search by trace ID EN

Results

Root Start Time Spans Duration

payment-service: get /logistic 7 minutes ago (06/15 22:05:35:921) 3 255.422ms

payment-service (2) logistic-service (1)

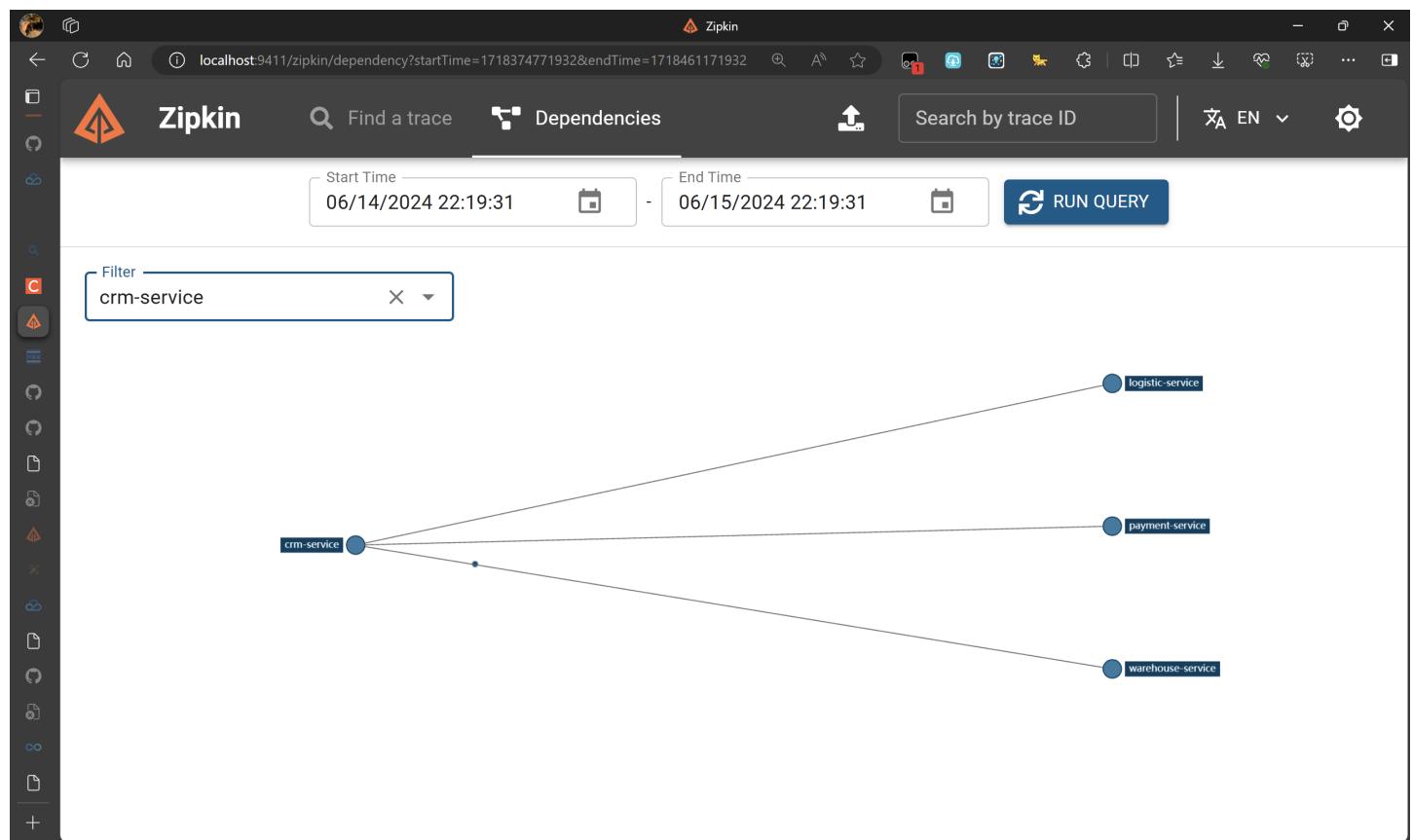
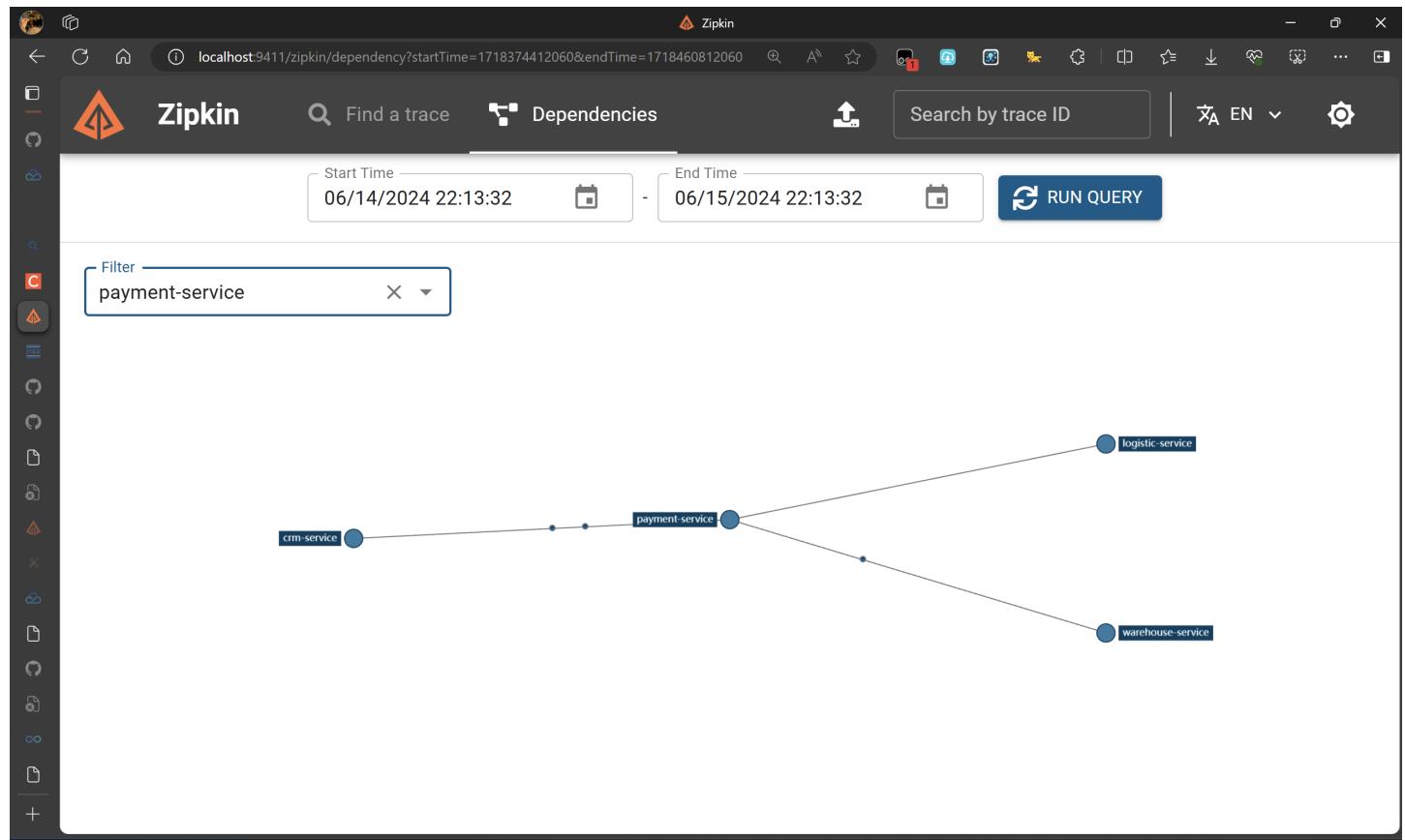
crm-service: get /payment 6 minutes ago (06/15 22:06:48:713) 7 45.462ms

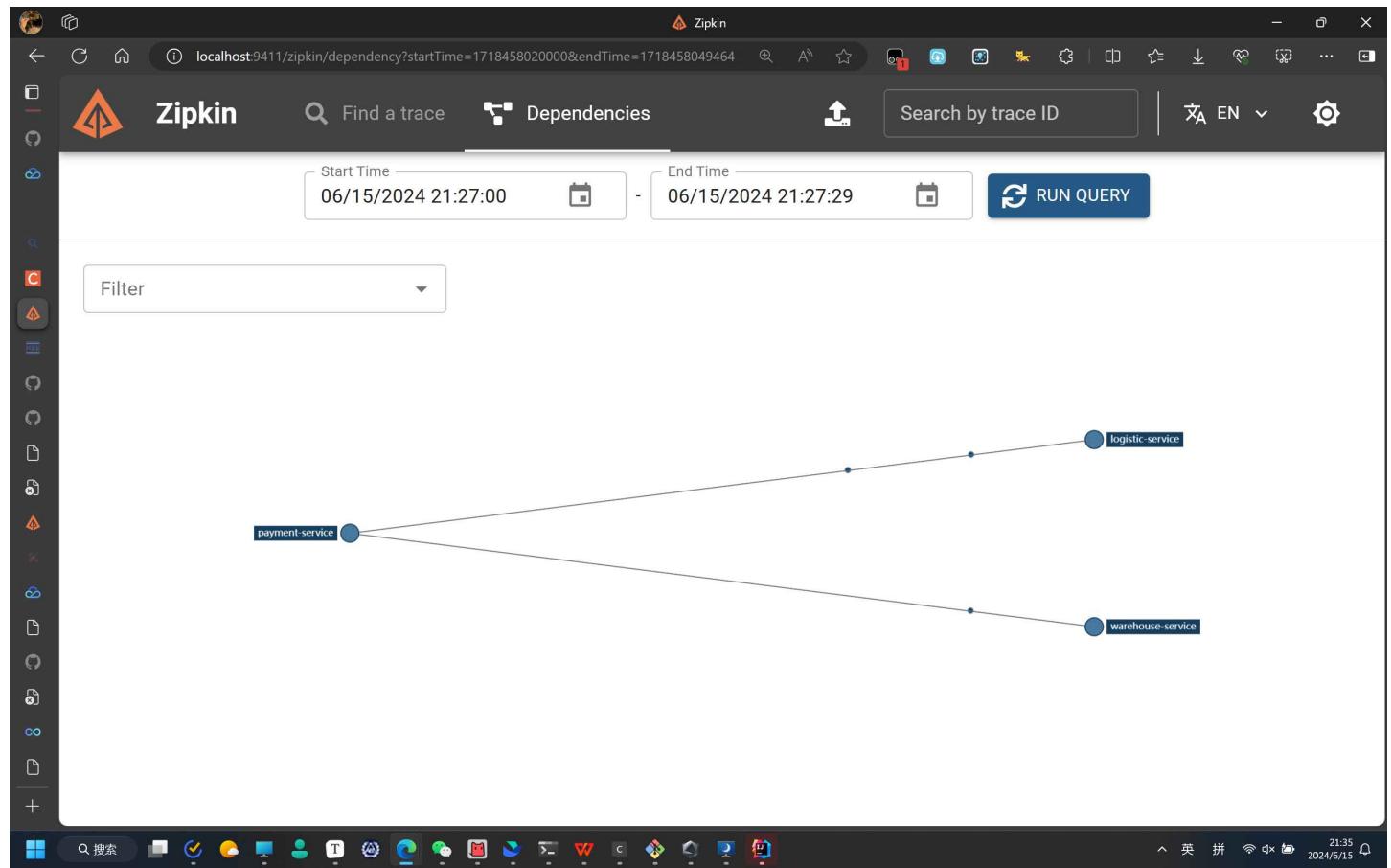
payment-service (3) crm-service (2) logistic-service (1) warehouse-service (1)

payment-service: get /warehouse 7 minutes ago (06/15 22:05:45:513) 3 32.791ms

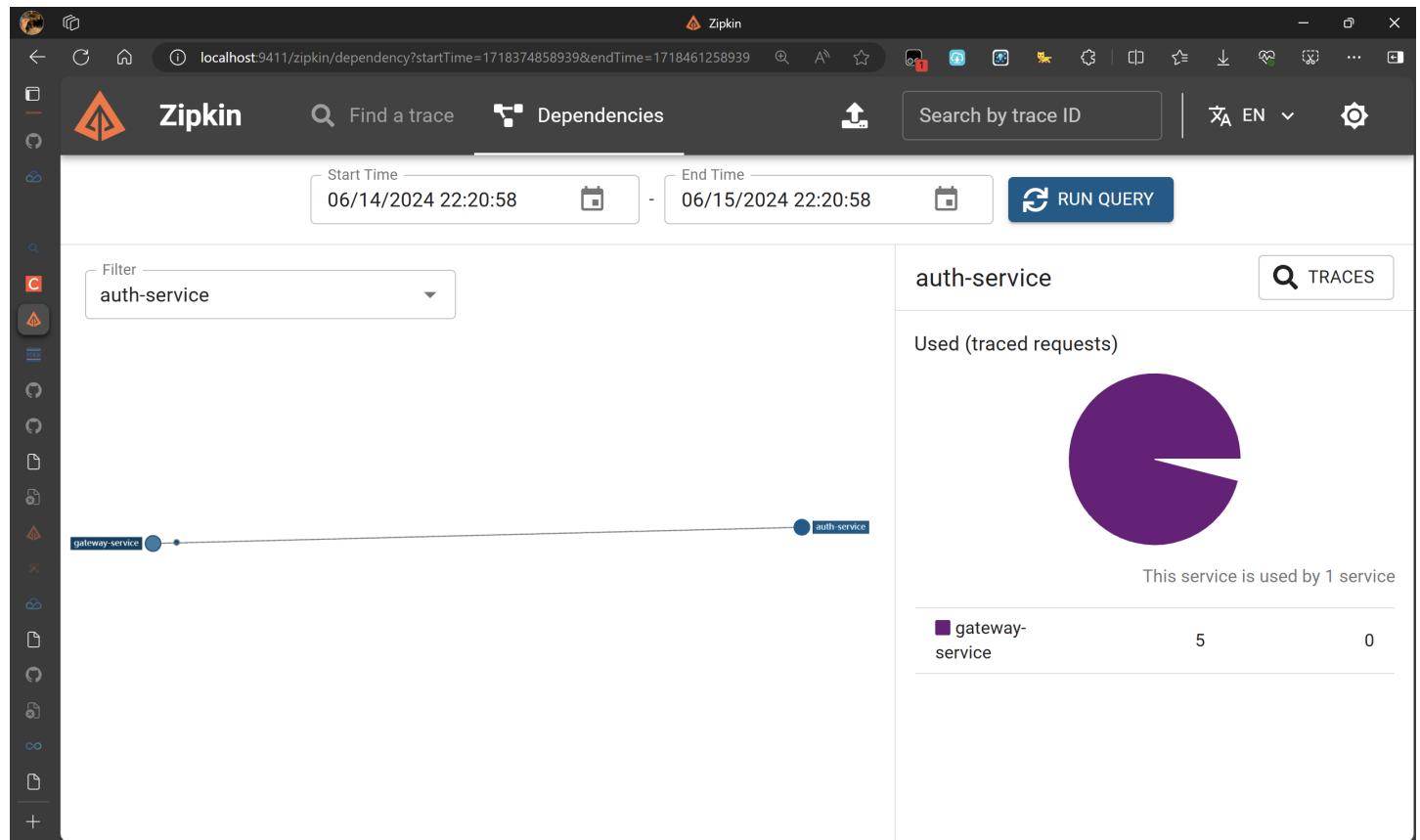
payment-service (2) warehouse-service (1)

Service filters payment-service Service filt...

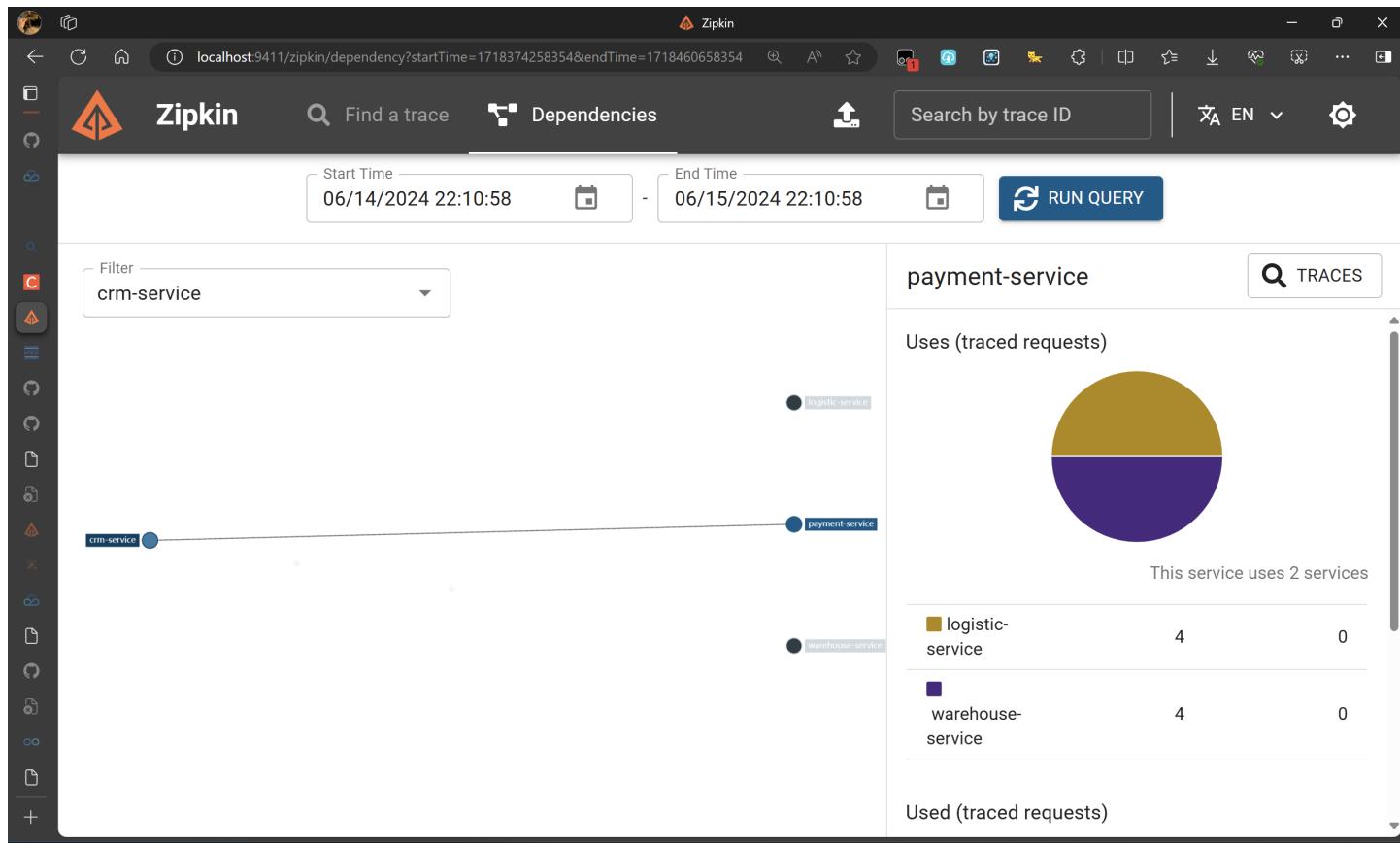




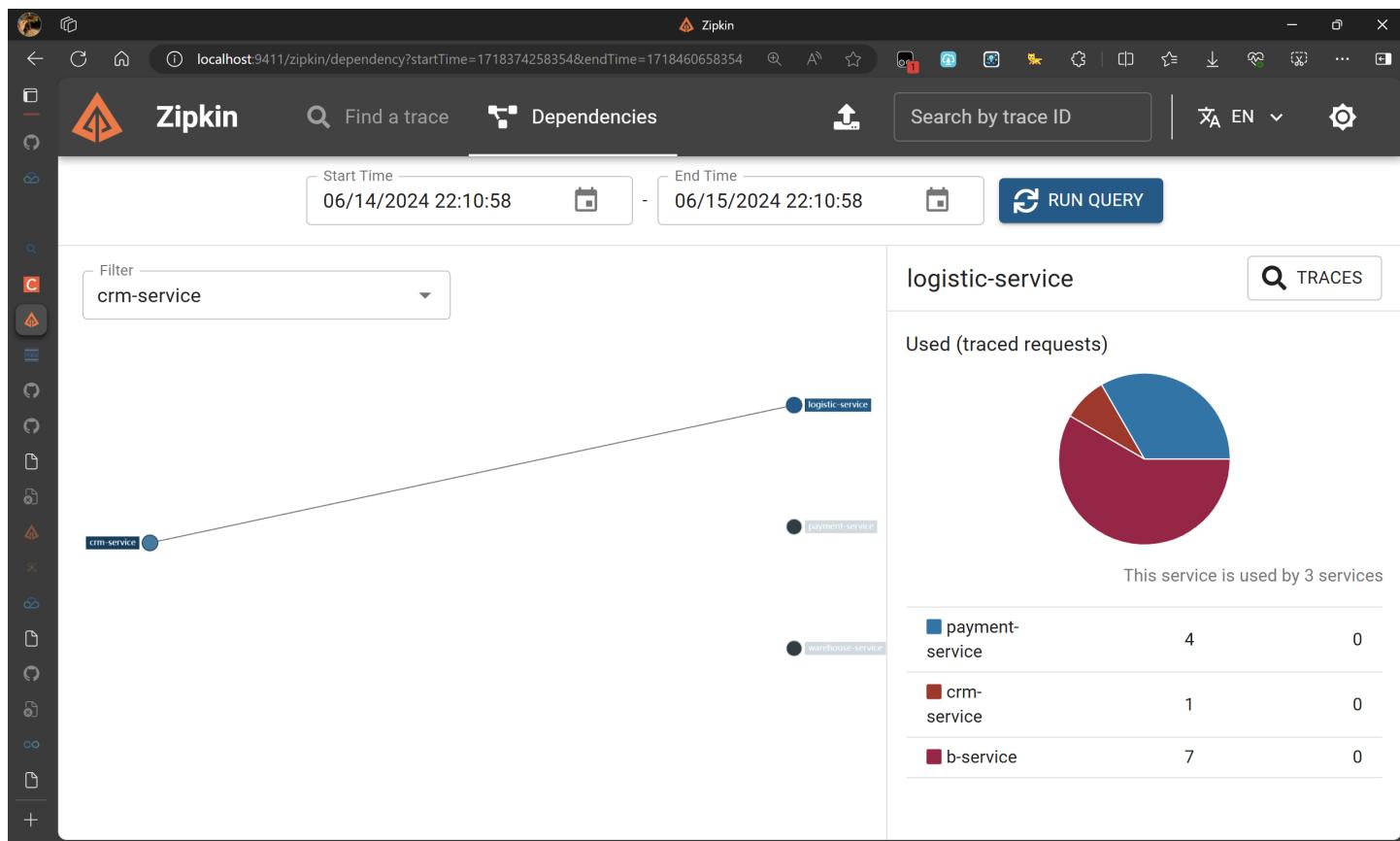
1. gateway-service 调用 auth-service



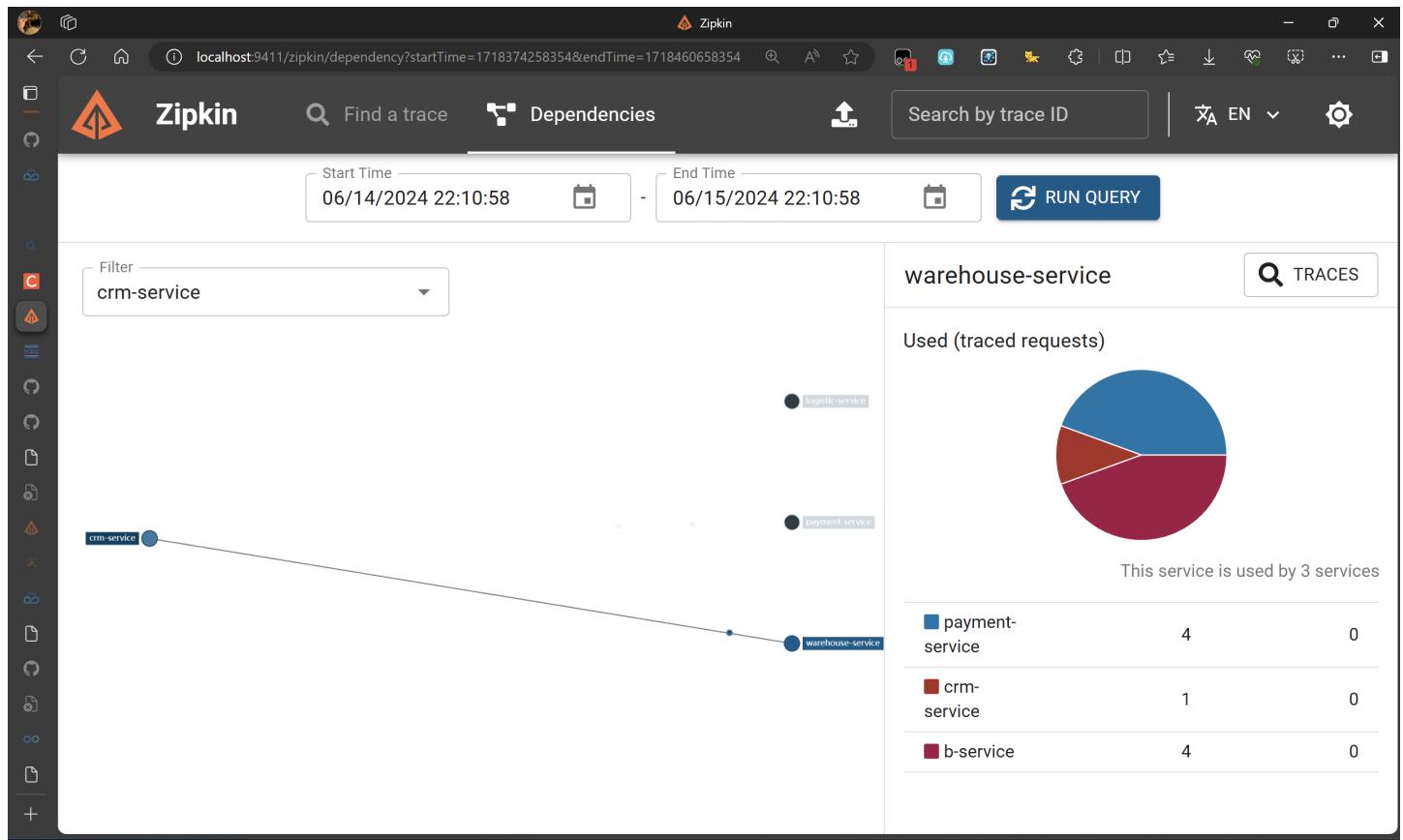
2. crm-service 调用 payment-service



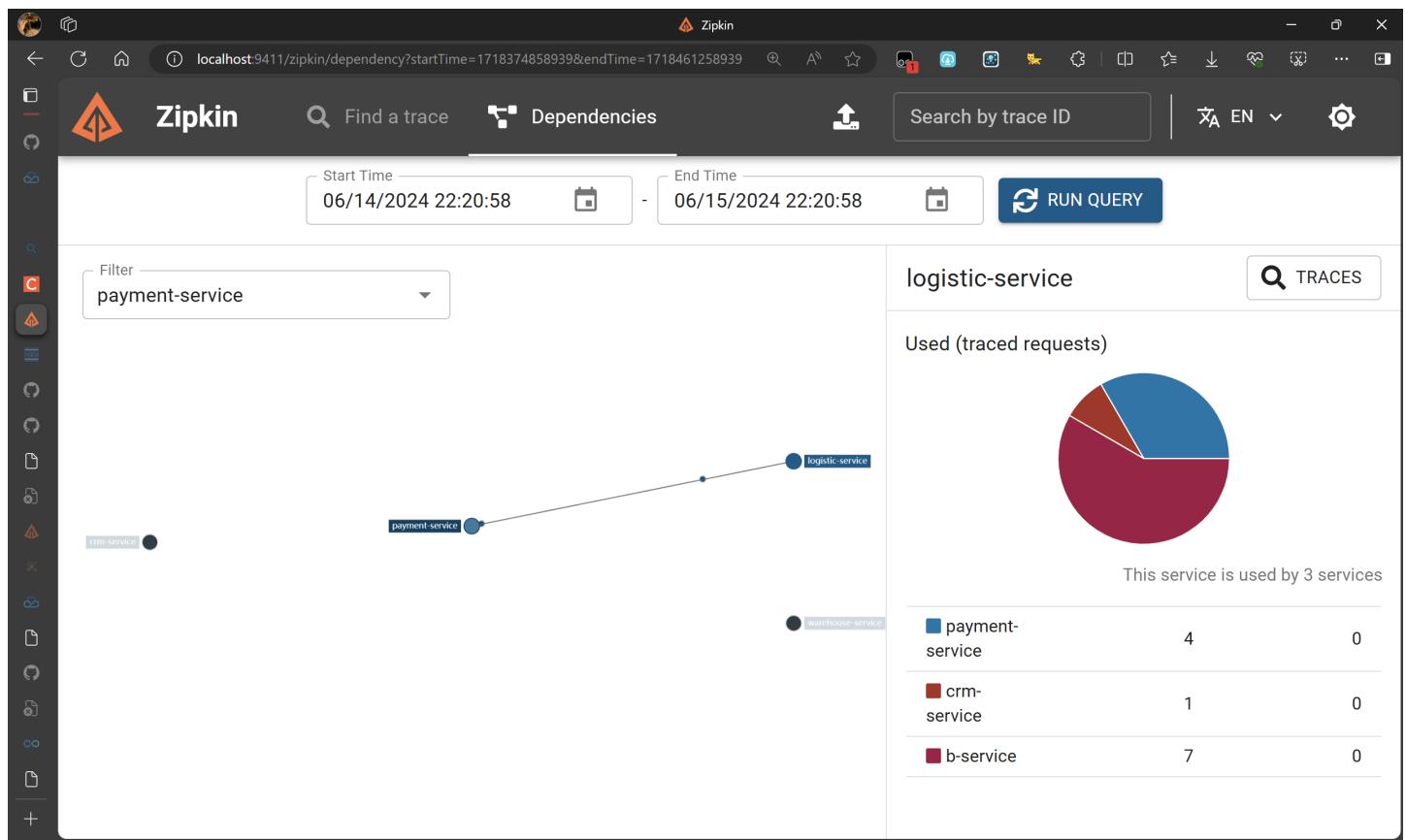
3. crm-service 调用 logistic-service



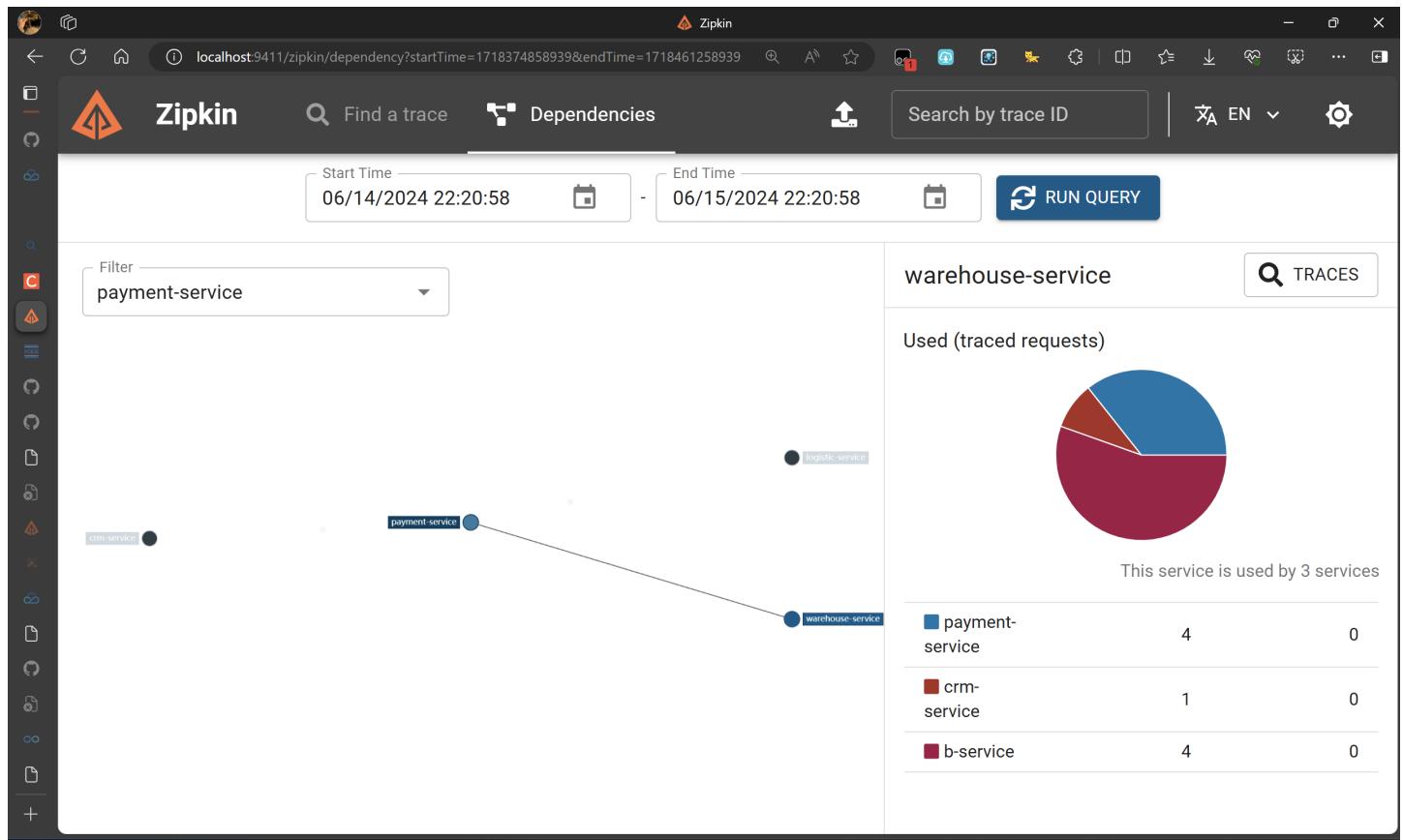
4. crm-service 调用 warehouse-service



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`

```

1 <dependencies>
2   <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-web</artifactId>
5   </dependency>
6   <dependency>
7     <groupId>org.springframework.cloud</groupId>
8     <artifactId>spring-cloud-config-server</artifactId>
9   </dependency>
10  <dependency>
11    <groupId>com.alibaba.cloud</groupId>
12    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
13  </dependency>
14 </dependencies>

```

`application.properties`

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

ConfigServerApplication.java

```
1 package com.example.configserver;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.config.server.EnableConfigServer;
6
7 @SpringBootApplication@EnableConfigServerpublic class ConfigServerApplication
8     {public static void main(String[] args) {
9         SpringApplication.run(ConfigServerApplication.class, args);
10    }
11 }
```

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

```
1 spring.cloud.config.uri=http://localhost:8888
2 spring.zipkin.base-url=http://localhost:9411
3 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：

```
1 <dependency>
2     <groupId>io.micrometer</groupId>
3     <artifactId>micrometer-tracing-brave</artifactId>
4 </dependency>
5 <dependency>
6     <groupId>io.zipkin.brave</groupId>
7     <artifactId>brave-instrumentation-http</artifactId>
```

```
8 </dependency>
9 <dependency>
10    <groupId>org.springframework.boot</groupId>
11    <artifactId>spring-boot-starter-actuator</artifactId>
12 </dependency>
13 <dependency>
14    <groupId>org.springframework.cloud</groupId>
15    <artifactId>spring-cloud-starter-zipkin</artifactId>
16 </dependency>
```

2. 配置 application.yml

确保在 application.yml 中正确配置了 Zipkin：

```
1 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：

```
1 package com.example.config;
2
3 import io.micrometer.tracing.Tracer;
4 import io.micrometer.tracing.brave.bridge.BraveTracer;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7 import zipkin2.reporter.AsyncReporter;
8 import zipkin2.reporter.okhttp3.OkHttpSender;
9
10 @Configurationpublic class TracingConfig {
11     @Beanpublic Tracer tracer() {OkHttpSender sender =
12         OkHttpSender.create("http://localhost:9411/api/v2/spans");
13         AsyncReporter<zipkin2.Span> spanReporter =
14         AsyncReporter.create(sender);
15         brave.Tracing braveTracing = brave.Tracing.newBuilder()
16             .localServiceName("logistic-service")
17             .spanReporter(spanReporter)
18             .build();return new BraveTracer(braveTracing);
19     }
20 }
```

4. 配置 RestTemplate

确保 `RestTemplate` 配置正确：

```
1 package com.example.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.client.RestTemplate;
6
7 @Configurationpublic class AppConfig {
8     @Beanpublic RestTemplate restTemplate() {return new RestTemplate();}
9 }
10 }
```

5. 控制器类

确保你的控制器类正确注入并使用 `RestTemplate`：

```
1 package com.example.springboot;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6 import org.springframework.web.client.RestTemplate;
7
8 @RestControllerpublic class AController {
9     @Autowiredprivate RestTemplate restTemplate;
10    private static final String SERVICE_PROVIDER_ADDRESS = "http://logistic-
11    service";
12    @GetMapping("/test")public String callServiceB() {return
13        restTemplate.getForObject(SERVICE_PROVIDER_ADDRESS + "/test-service-b",
14            String.class);
15    }
16 }
```

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:

```
1 [  
2     {"id": 1,"name": "Alice","email": "alice@example.com"},  
3     {"id": 2,"name": "Bob","email": "bob@example.com"}  
4 ]
```

- 测试:

The screenshot shows the Postman interface with a GET request to `http://localhost:8081/customers`. The response body is displayed in a JSONpretty-printed format, showing a list of two customer objects: Alice and Bob.

```
1 [  
2     {  
3         "name": "Alice",  
4         "id": 1,  
5         "email": "alice@example.com"  
6     },  
7     {  
8         "name": "Bob",  
9         "id": 2,  
10        "email": "bob@example.com"  
11    }  
12 ]
```

- Swagger:

GET /customers 获取所有客户

Parameters

No parameters

Responses

Code Description Links

200 OK No links

Media type

/* Controls Accept header.

Example Value | Schema

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

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:

POST | http://localhost:8081/customers

Params | Authorization | Headers (9) | **Body** | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw (selected) | binary | GraphQL | **JSON** | Beautify

```

1  {
2    "name": "Alice",
3    "email": "alice@example.com"
4  }

```

Body | Cookies | Headers (5) | Test Results | 200 OK | 24 ms | 217 B | Save as example | ⋮

Pretty | Raw | Preview | Visualize | **JSON** | `copy` | `copy` | `copy`

```

1  {
2    "name": "Alice",
3    "id": "1",
4    "email": "alice@example.com"
5  }

```

- **swagger:**

POST /customers 创建客户

Parameters

No parameters

Request body **required**

application/json

Example Value | Schema

```
{
  "additionalProp1": "string",
  "additionalProp2": "string",
  "additionalProp3": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header.

Example Value | Schema

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:

The screenshot shows the Postman interface. The method is set to **PUT**, the URL is **http://localhost:8081/customers/1**, and the **Body** tab is selected. The body contains the following JSON:

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

The response status is **200 OK** with a time of **12 ms** and a size of **231 B**. There are 5 headers listed under **Headers (5)**.

- swagger:

The screenshot shows the Swagger UI for the **/customers/{id}** endpoint. The method is **PUT**. The **Parameters** section shows a required parameter **id** of type **integer(int32)**. The **Responses** section shows a **200 OK** response with a media type of ***/***. The example response body is:

```
{ "additionalProp1": 0, "additionalProp2": 0, "additionalProp3": 0 }
```

2.1.4 DELETE /customers/{id}

- 描述: 删除一个客户
- 参数: 无

- 返回值:

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

- 测试:

The screenshot shows the Postman interface with a successful DELETE request to `http://localhost:8081/customers/1`. The response body is a JSON object with the message "Customer deleted successfully".

- Swagger:

The screenshot shows the Swagger UI for the `/customers/{id}` endpoint. It includes sections for Parameters (with a required `id` parameter), Responses (with a 200 OK response example), and Example Value (showing a JSON object with three string properties: `additionalProp1`, `additionalProp2`, and `additionalProp3`).

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:

```
1 [  
2     {"id": 1,"description": "Shipment 123","status": "Delivered"},  
3     {"id": 2,"description": "Shipment 456","status": "In Transit"}  
4 ]
```

- 测试:

The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET' selected, the URL 'http://localhost:8081/logistics', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Cookies' tab is also visible. Under the 'Params' tab, there is a table with columns 'Key', 'Value', 'Description', and 'Bulk Edit'. There are two rows in the table, both labeled 'Key' and 'Value' with 'Description'. In the main area, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Body' tab is selected and shows a JSON response. The response is a JSON array with two elements. Each element is an object with properties: 'status' (values 'Delivered' and 'In Transit'), 'id' (values 1 and 2), and 'description' (values 'Shipment 123' and 'Shipment 124'). The 'Test Results' tab shows a successful response with status code 200 OK, 9 ms duration, and 284 B size. There is also a 'Save as example' button.

```
1 [  
2     {  
3         "status": "Delivered",  
4         "id": 1,  
5         "description": "Shipment 123"  
6     },  
7     {  
8         "status": "In Transit",  
9         "id": 2,  
10        "description": "Shipment 124"  
11    }  
12]
```

- Swagger:

The screenshot shows the Swagger UI for the '/logistics' endpoint. At the top, it says 'GET /logistics 获取所有物流信息'. Below that is a 'Parameters' section with a note 'No parameters'. To the right is a 'Try it out' button. The next section is 'Responses'. It lists a '200 OK' response with a 'Media type' dropdown set to '*/*'. A note below says 'Controls Accept header.' There are buttons for 'Example Value' and 'Schema'. The 'Example Value' section shows a JSON array with two objects, each having an 'id' (1 or 2), 'description' ('Shipment 123' or 'Shipment 124'), and an 'additionalProp' object with three properties: 'additionalProp1', 'additionalProp2', and 'additionalProp3'. The 'Links' column for the 200 response says 'No links'.

Code	Description	Links
200	OK	No links

```
[  
  {  
    "id": 1,  
    "description": "Shipment 123",  
    "additionalProp": {}  
  },  
  {  
    "id": 2,  
    "description": "Shipment 124",  
    "additionalProp": {}  
  }]
```

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:

The screenshot shows the Postman interface for a POST request to `http://localhost:8081/logistics`. The request body is set to raw JSON:

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

The response details show a 200 OK status with the following data:

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

- Swagger:

POST /logistics 创建物流信息

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{  
    "additionalProp1": "string",  
    "additionalProp2": "string",  
    "additionalProp3": "string"  
}
```

Responses

Code	Description	Links
200	OK	No links

Media type

/*

Controls Accept header.

Example Value | Schema

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:

PUT | http://localhost:8081/logistics/1

Params Authorization Headers (9) Body • Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

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

Body Cookies Headers (5) Test Results 200 OK 12 ms 230 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "Delivered",
3   "id": 1,
4   "description": "Shipment 123 Updated"
5 }
```

- Swagger:

PUT /logistics/{id} 更新物流信息

Parameters

Name	Description
id <small>required</small>	物流ID (path)
id	

Request body required

application/json

Example Value | Schema

```
{
  "additionalProp1": "string",
  "additionalProp2": "string",
  "additionalProp3": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

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:

DELETE http://localhost:8081/logistics/1

Params Authorization Headers (9) Body **raw** JSON Cookies

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON `{ "message": "Logistic deleted successfully" }`

200 OK 7 ms 207 B Save as example

- Swagger:

DELETE /logistics/{id} 删除物流信息

Parameters

Name	Description
id * required	integer(\$int32) 物流ID (path)
id	

Responses

Code	Description	Links
200	OK	No links

Media type `*/*`
Controls Accept header.

Example Value Schema

```
{
  "additionalProp1": "string",
  "additionalProp2": "string",
  "additionalProp3": "string"
}
```

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:

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

3   {"id": 2,"name": "Warehouse B","location": "Location B"}  

4 ]
```

- 测试:

GET http://localhost:8081/warehouses

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Beautify

Body

1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 [
2 {
3 "name": "Warehouse A",
4 "location": "Location A",
5 "id": 1
6 },
7 {
8 "name": "Warehouse B",
9 "location": "Location B",
10 "id": 2
11 }
12]

Status: 200 OK Time: 93 ms Size: 273 B Save as example

- Swagger:

GET /warehouses 获取所有仓库信息

Parameters

No parameters

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header.

Example Value | Schema

```
[  
  {  
    "additionalProp1": {},  
    "additionalProp2": {},  
    "additionalProp3": {}  
  }  
]
```

2.3.2 POST /warehouses

- 描述: 创建新的仓库信息
- 参数:

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

- 返回值:

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

- 测试:

The screenshot shows the Postman interface with a POST request to `http://localhost:8081/warehouses`. The request body is set to raw JSON with the value `{"name": "Warehouse A", "location": "Location A"}`. The response status is 200 OK with a time of 19 ms and a size of 217 B. The response body is identical to the request body.

- Swagger:

The screenshot shows the Swagger UI for the `/warehouses` endpoint. It includes sections for Parameters (none), Request body (example value: `{"additionalProp1": "string", "additionalProp2": "string", "additionalProp3": "string"}`), and Responses (200 OK with media type `*/*`). The `application/json` dropdown is selected.

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:

The screenshot shows a Postman interface with the following details:

- Method: PUT
- URL: <http://localhost:8081/warehouses/1>
- Body tab selected
- Content type: raw
- JSON content:

```
1 {"name": "Warehouse A Updated","location": "Location A Updated"}
```
- Response status: 200 OK
- Response time: 12 ms
- Response size: 233 B

- Swagger:

The screenshot shows the Swagger UI for the `PUT /warehouses/{id}` endpoint. The interface includes:

- Method: PUT
- Path: `/warehouses/{id}` (labeled as "更新仓库信息")
- Parameters:

 - Name: id (required, integer, path)

- Request body (required): application/json

 - Example Value:

```
{ "additionalProp1": "string", "additionalProp2": "string", "additionalProp3": "string" }
```


- Responses:

 - Code: 200 Description: OK Links: No links

2.3.4 DELETE /warehouses/{id}

- 描述: 删除仓库信息
- 参数: 无
- 返回值:

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

- 测试:

The screenshot shows the Postman interface with a successful DELETE request to `http://localhost:8081/warehouses/1`. The response status is `200 OK` with a size of `208 B`. The response body is displayed in JSON format:

```
1 {
2     "message": "Warehouse deleted successfully"
3 }
```

- Swagger:

The screenshot shows the Swagger UI for the `/warehouses/{id}` endpoint. It includes a parameter table and a response table.

Parameters

Name	Description
<code>id</code> <small>* required</small>	仓库ID (path)

Responses

Code	Description	Links
200	OK	No links

For the 200 OK response, the media type is set to `*/*`. The example value is a JSON object:

```
{  
    "additionalProp1": "string",  
    "additionalProp2": "string",  
    "additionalProp3": "string"  
}
```

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:

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

4]

- 测试:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8081/payments`. The response body is a JSON array containing two payment objects:

```
1 [  
2   {  
3     "id": 1,  
4     "status": "Completed",  
5     "orderId": "123",  
6     "amount": 100.0  
7   },  
8   {  
9     "id": 2,  
10    "status": "Pending",  
11    "orderId": "124",  
12    "amount": 200.0  
13  }  
14 ]
```

- Swagger:

The screenshot shows the Swagger UI for the `/payments` endpoint. It includes sections for Parameters, Responses, and Example Value.

Parameters: No parameters.

Responses:

Code	Description	Links
200	OK	No links

Example Value (Media type: `*/*`):

```
[  
  {  
    "additionalProp1": {},  
    "additionalProp2": {},  
    "additionalProp3": {}  
  }  
]
```

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:

The screenshot shows the Postman application interface. A POST request is being made to the URL `http://localhost:8081/payments`. The request body is set to raw JSON, containing the payload `{\"orderId\": \"123\", \"amount\": 100.0, \"status\": \"Pending\"}`. The response received is a 200 OK status with a response time of 14 ms and a size of 222 B. The response body is displayed in pretty JSON format, showing the returned object with fields `id`, `status`, `orderId`, and `amount`.

- Swagger:

The screenshot shows the Swagger UI for the `/payments` endpoint. The endpoint is described as "创建支付信息".
Parameters: No parameters.
Request body (required): application/json
Example Value | Schema:

```
{  "additionalProp1": {},  "additionalProp2": {},  "additionalProp3": {}}
```

Responses:

Code	Description	Links
200	OK	No links

Media type: */*
Controls Accept header.
Example Value | Schema:

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

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:

The screenshot shows the Postman interface for a PUT request to `http://localhost:8081/payments/1`. The request method is set to `PUT`, and the URL is `http://localhost:8081/payments/1`. The `Body` tab is selected, showing the JSON payload:

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

The response tab shows the following details:

- Status: 200 OK
- Time: 17 ms
- Size: 224 B
- Save as example

The response body is displayed in Pretty, Raw, Preview, and Visualize formats, and is also shown as JSON:

```
1 {
2   "id": 1,
3   "status": "Completed",
4   "orderId": "123",
5   "amount": 100.0
6 }
```

- Swagger:

PUT /payments/{id} 更新支付信息

Parameters

Name	Description
id <small>required</small>	integer(\$int32) 支付ID (path)

Request body required

application/json

Example Value | Schema

```
{
  "additionalProp1": {},
  "additionalProp2": {},
  "additionalProp3": {}
}
```

Responses

Code	Description	Links
200	OK	No links

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:

DELETE <http://localhost:8081/payments/1> Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 9 ms Size: 206 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Payment deleted successfully"
3 }
```

- Swagger:

DELETE /payments/{id} 删除支付信息

Parameters

Name Description

id required
integer(\$int32) 支付ID
(path)

Try it out

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header.

Example Value | Schema

```
{ "additionalProp1": "string", "additionalProp2": "string", "additionalProp3": "string" }
```

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:

```
1 {
2   "access_token": "access_token_value",
3   "token_type": "bearer",
4   "expires_in": 3600,
5   "refresh_token": "refresh_token_value",
6   "scope": "read write"
7 }
```

- 测试:

```

POST http://localhost:8081/oauth/token

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies
Body none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "grant_type": "password",
3   "username": "your_username",
4   "password": "your_password",
5   "client_id": "your_client_id",
6   "client_secret": "your_client_secret"
7 }

```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 27 ms Size: 300 B Save as example

```

Pretty Raw Preview Visualize JSON
1 {
2   "access_token": "access_token_value",
3   "scope": "read write",
4   "token_type": "bearer",
5   "expires_in": 3600,
6   "refresh_token": "refresh_token_value"
7 }

```

- Swagger:

POST /oauth/token 获取OAuth令牌

Parameters

No parameters

Request body required

application/json

```
{
  "additionalProp1": "string",
  "additionalProp2": "string",
  "additionalProp3": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header.

Example Value | Schema

```
{}
```

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:

```

1 {
2   "active": true,
3   "exp": 1609459200,
4   "user_name": "user",
5   "authorities": ["ROLE_USER"]

```

6 }

- 测试:

The screenshot shows a Postman interface with a POST request to `http://localhost:8081/oauth/check_token`. The request body is set to raw JSON, containing the following payload:

```
1 {  
2   "token": "your_token"  
3 }
```

The response status is 200 OK, with a response body showing a JSON object with fields `exp`, `active`, `user_name`, and `authorities`.

```
1 {  
2   "exp": 1609459200,  
3   "active": true,  
4   "user_name": "user",  
5   "authorities": [  
6     "ROLE_USER"  
7   ]  
8 }
```

- Swagger:

The screenshot shows the Swagger UI for the `/oauth/check_token` endpoint. The request body is required and has a schema of `application/json`. An example value is provided:

```
{  
  "additionalProp1": "string",  
  "additionalProp2": "string",  
  "additionalProp3": "string"  
}
```

The responses section shows a 200 OK response with a media type of `*/*`. An example value is shown as:

```
{  
  "additionalProp1": "string",  
  "additionalProp2": "string",  
  "additionalProp3": "string"  
}
```

三、关键代码

3.1 CRM Service

CrmServiceApplication.java

```
1 package com.example.crmService;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6 import org.springframework.cloud.openfeign.EnableFeignClients;
7
8 @SpringBootApplication@EnableDiscoveryClient@EnableFeignClientspublic class
9     CrmServiceApplication {public static void main(String[] args) {
10         SpringApplication.run(CrmServiceApplication.class, args);
11     }
12 }
```

Customer.java

```
1 package com.example.crmService.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entitypublic class Customer {@Id@GeneratedValue(strategy =
9 GenerationType.IDENTITY)private Long id;private String name;private String
email;
10 // getters and setters
11 }
```

CustomerRepository.java

```
1 package com.example.crmService.repository;
2
3 import com.example.crmService.model.Customer;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface CustomerRepository extends JpaRepository<Customer, Long> {
7 }
```

CustomerService.java

```
1 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 @Service public class CustomerService { @Autowired private 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
   { @Autowired private CustomerService service;
11 @GetMapping public List<Customer> getAll() { return service.findAll();
12 }
13 @PostMapping public Customer create(@RequestBody Customer customer) { return
   service.save(customer);
14 }
```

```
15 @PutMapping("/{id}")public Customer update(@PathVariable Long id, @RequestBody  
Customer customer) {return service.update(id, customer);  
16 }  
17 @DeleteMapping("/{id}")public String delete(@PathVariable Long id) {  
18     service.delete(id);return "Customer deleted successfully";  
19 }  
20 }
```

3.2 Logistic Service

LogisticServiceApplication.java

```
1 package com.example.logisticservice;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
6  
7 @SpringBootApplication@EnableDiscoveryClientpublic class  
LogisticServiceApplication {public static void main(String[] args) {  
8     SpringApplication.run(LogisticServiceApplication.class, args);  
9 }  
10 }
```

Logistic.java

```
1 package com.example.logisticservice.model;  
2  
3 import jakarta.persistence.Entity;  
4 import jakarta.persistence.GeneratedValue;  
5 import jakarta.persistence.GenerationType;  
6 import jakarta.persistence.Id;  
7  
8 @Entitypublic class Logistic {@Id@GeneratedValue(strategy =  
GenerationType.IDENTITY)private Long id;private String description;private  
String status;  
9 // getters and setters  
10 }
```

LogisticRepository.java

```
1 package com.example.logisticservice.repository;
2
3 import com.example.logisticservice.model.Logistic;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface LogisticRepository extends JpaRepository<Logistic, Long> {
7 }
```

LogisticService.java

```
1 package com.example.logisticservice.service;
2
3 import com.example.logisticservice.model.Logistic;
4 import com.example.logisticservice.repository.LogisticRepository;
5 import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import java.util.List;
10
11 @Servicepublic class LogisticService {@Autowiredprivate LogisticRepository
repository;
12 @CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public
List<Logistic> findAll() {return repository.findAll();
13 }
14 public Logistic save(Logistic logistic) {return repository.save(logistic);
15 }
16 public Logistic update(Long id, Logistic logistic) {Logistic existingLogistic =
repository.findById(id).orElseThrow(() -> new
ResourceNotFoundException("Logistic not found"));
17         existingLogistic.setDescription(logistic.getDescription());
18         existingLogistic.setStatus(logistic.getStatus());return
repository.save(existingLogistic);
19 }
20 public void delete(Long id) {
21         repository.deleteById(id);
22 }
23 public List<Logistic> fallbackMethod(Throwable throwable) {
24         System.out.println("Fallback method called due to: " +
throwable.getMessage());return List.of();
25 }
26 }
```

LogisticController.java

```
1 package com.example.logisticservice.controller;
2
3 import com.example.logisticservice.model.Logistic;
4 import com.example.logisticservice.service.LogisticService;
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("/logistics")public class LogisticController
11     {@Autowiredprivate LogisticService service;
12     @GetMappingpublic List<Logistic> getAll() {return service.findAll();}
13     @PostMappingpublic Logistic create(@RequestBody Logistic logistic) {return
14         service.save(logistic);}
15     @PutMapping("/{id}")public Logistic update(@PathVariable Long id, @RequestBody
16         Logistic logistic) {return service.update(id, logistic);}
17     @DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
18         service.delete(id);return "Logistic deleted successfully";
19     }
20 }
```

3.3 Warehouse Service

`WarehouseServiceApplication.java`

```
1 package com.example.warehouseservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication@EnableDiscoveryClientpublic class
8     WarehouseServiceApplication {public static void main(String[] args) {
9         SpringApplication.run(WarehouseServiceApplication.class, args);
10    }}
```

`Warehouse.java`

```
1 package com.example.warehouseservice.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entitypublic class Warehouse {@Id@GeneratedValue(strategy =
9 GenerationType.IDENTITY)private Long id;private String name;private String
10 location;
11 // getters and setters
12 }
```

WarehouseRepository.java

```
1 package com.example.warehouseservice.repository;
2
3 import com.example.warehouseservice.model.Warehouse;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface WarehouseRepository extends JpaRepository<Warehouse, Long> {
7 }
```

WarehouseService.java

```
1 package com.example.warehouseservice.service;
2
3 import com.example.warehouseservice.model.Warehouse;
4 import com.example.warehouseservice.repository.WarehouseRepository;
5 import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import java.util.List;
10
11 @Servicepublic class WarehouseService {@Autowiredprivate WarehouseRepository
12 repository;
13 @CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public
14 List<Warehouse> findAll() {return repository.findAll();
15 }
16 public Warehouse save(Warehouse warehouse) {return repository.save(warehouse);
17 }
```

```

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

```

WarehouseController.java

```

1 package com.example.warehousestest.controller;
2
3 import com.example.warehousestest.model.Warehouse;
4 import com.example.warehousestest.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
11 {@Autowiredprivate WarehouseService service;
12 @GetMappingpublic List<Warehouse> getAll() {return service.findAll();
13 }
14 @PostMappingpublic Warehouse create(@RequestBody Warehouse warehouse) {return
15     service.save(warehouse);
16 }
17 @PutMapping("/{id}")public Warehouse update(@PathVariable Long id, @RequestBody
18     Warehouse warehouse) {return service.update(id, warehouse);
19 }
20 @DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
21     service.delete(id);return "Warehouse deleted successfully";
22 }
23
24 }

```

3.4 Payment Service

PaymentServiceApplication.java

```
1 package com.example.paymentservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication@EnableDiscoveryClientpublic class
8 PaymentServiceApplication {public static void main(String[] args) {
9     SpringApplication.run(PaymentServiceApplication.class, args);
10    }
11 }
```

Payment.java

```
1 package com.example.paymentservice.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entitypublic class Payment {@Id@GeneratedValue(strategy =
9 GenerationType.IDENTITY)private Long id;private String orderId;private double
10 amount;private String status;
11 // getters and setters
12 }
```

PaymentRepository.java

```
1 package com.example.paymentservice.repository;
2
3 import com.example.paymentservice.model.Payment;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface PaymentRepository extends JpaRepository<Payment, Long> {
7 }
```

PaymentService.java

```

1 package com.example.paymentservice.service;
2
3 import com.example.paymentservice.model.Payment;
4 import com.example.paymentservice.repository.PaymentRepository;
5 import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import java.util.List;
10
11 @Servicepublic class PaymentService {@Autowiredprivate PaymentRepository
12   repository;
13   @CircuitBreaker(name = "default", fallbackMethod = "fallbackMethod")public
14   List<Payment> findAll() {return repository.findAll();}
15   }
16   public Payment save(Payment payment) {return repository.save(payment);}
17   }
18   public Payment update(Long id, Payment payment) {Payment existingPayment =
19   repository.findById(id).orElseThrow(() -> new
20   ResourceNotFoundException("Payment not found"));
21   existingPayment.setOrderId(payment.getOrderId());
22   existingPayment.setAmount(payment.getAmount());
23   existingPayment.setStatus(payment.getStatus());return
24   repository.save(existingPayment);
25   }
26   public void delete(Long id) {
27     repository.deleteById(id);
28   }
29   public List<Payment> fallbackMethod(Throwable throwable) {
30     System.out.println("Fallback method called due to: " +
31     throwable.getMessage());return List.of();
32   }
33 }
```

PaymentController.java

```

1 package com.example.paymentservice.controller;
2
3 import com.example.paymentservice.model.Payment;
4 import com.example.paymentservice.service.PaymentService;
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("/payments")public class PaymentController
11     {@Autowiredprivate PaymentService service;
12     }
13     @PostMappingpublic Payment create(@RequestBody Payment payment) {return
14         service.save(payment);
15     }
16     @PutMapping("/{id}")public Payment update(@PathVariable Long id, @RequestBody
17     Payment payment) {return service.update(id, payment);
18     }
19     @DeleteMapping("/{id}")public String delete(@PathVariable Long id) {
20         service.delete(id);return "Payment deleted successfully";
21     }
22 }
```

3.5 Auth Service

AuthServiceApplication.java

```
1 package com.example.authservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6
7 @SpringBootApplication@EnableDiscoveryClientpublic class AuthServiceApplication
8     {public static void main(String[] args) {
9         SpringApplication.run(AuthServiceApplication.class, args);
10    }
11 }
```

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
7     org.springframework.security.config.annotation.web.configuration.EnableAuthorizationServer;
8     import
9     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()
```

```
40     .username("user")
41     .password("password")
42     .roles("USER")
43     .build();return new InMemoryUserDetailsManager(user);
44   }
45 }
```

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

```
1 import io.github.resilience4j.circuitbreaker.CircuitBreakerConfig;
2 import io.github.resilience4j.circuitbreaker.CircuitBreakerRegistry;
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5
6 @Configuration
7 public class Resilience4jConfig {
8
9     @Bean
10    public CircuitBreakerRegistry circuitBreakerRegistry() {
11        CircuitBreakerConfig config = CircuitBreakerConfig.custom()
12            .failureRateThreshold(50)
13            .waitDurationInOpenState(java.time.Duration.ofMillis(1000))
14            .slidingWindowSize(2)
15            .build();
16        return CircuitBreakerRegistry.of(config);
17    }
18 }
19
```

2. 模拟断路控制器

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

```

2 import org.springframework.web.bind.annotation.GetMapping;
3 import org.springframework.web.bind.annotation.RestController;
4 import java.util.Map;
5
6 @RestController
7 public class SimpleController {
8
9     @CircuitBreaker(name = "simpleCircuitBreaker", fallbackMethod =
10      "fallbackMethod")
11     @GetMapping("/unstable")
12     public Map<String, String> unstableEndpoint() {
13         // 模拟一个不稳定的服务调用
14         if (Math.random() > 0.5) {
15             throw new RuntimeException("Service failure");
16         }
17         return Map.of("message", "Service call successful");
18     }
19
20     public Map<String, String> fallbackMethod(Throwable t) {
21         return Map.of("message", "Service is currently unavailable, please try
22 again later");
23     }

```

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 9 ms Size: 459 B Save as example ...

Pretty Raw Preview Visualize JSON  

```

1 {
2     "code": 502,
3     "msg": "Service failure"
4 }

```

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

```

1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 import
4     org.springframework.security.config.annotation.web.builders.HttpSecurity;
5 import org.springframework.security.core.userdetails.User;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.security.core.userdetails.UserDetailsService;
8 import org.springframework.security.provisioning.InMemoryUserDetailsManager;

```

```

8 import org.springframework.security.web.SecurityFilterChain;
9
10 @Configuration
11 public class SecurityConfig {
12
13     @Bean
14     public UserDetailsService userDetailsService() {
15         UserDetails user = User.withDefaultPasswordEncoder()
16             .username("user")
17             .password("password")
18             .roles("USER")
19             .build();
20
21         return new InMemoryUserDetailsManager(user);
22     }
23
24     @Bean
25     public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
26         throws Exception {
27         http
28             .authorizeRequests(authorizeRequests ->
29                 authorizeRequests.anyRequest().authenticated()
30             )
31             .formLogin().and()
32             .oauth2ResourceServer().jwt();
33
34     }

```

2. 配置授权服务器

```

1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 import
4     org.springframework.security.config.annotation.web.builders.HttpSecurity;
5 import org.springframework.security.core.userdetails.User;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.security.core.userdetails.UserDetailsService;
8 import org.springframework.security.provisioning.InMemoryUserDetailsManager;
9
10 @Configuration
11 public class SecurityConfig {
12
13     @Bean

```

```

14     public UserDetailsService userDetailsService() {
15         UserDetails user = User.withDefaultPasswordEncoder()
16             .username("user")
17             .password("password")
18             .roles("USER")
19             .build();
20         return new InMemoryUserDetailsManager(user);
21     }
22
23     @Bean
24     public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
25         throws Exception {
26         http
27             .authorizeRequests(authorizeRequests ->
28                 authorizeRequests.anyRequest().authenticated()
29             )
30             .formLogin().and()
31             .oauth2ResourceServer().jwt();
32         return http.build();
33     }
34 }
```

3. 配置OAuth2客户端

```

1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 import
4     org.springframework.security.config.annotation.web.builders.HttpSecurity;
5 import
6     org.springframework.security.config.annotation.web.configuration.OAuth2ClientCo
7 nfiguration;
8 import
9     org.springframework.security.oauth2.client.InMemoryOAuth2AuthorizedClientServic
10 e;
11 import
12     org.springframework.security.oauth2.client.OAuth2AuthorizedClientService;
13 import
14     org.springframework.security.oauth2.client.registration.ClientRegistration;
15 import
16     org.springframework.security.oauth2.client.registration.InMemoryClientRegistrat
17 ionRepository;
18
19
20 @Configuration
21 public class OAuth2ClientConfig {
```

```
12
13     @Bean
14     public InMemoryClientRegistrationRepository clientRegistrationRepository() {
15         return new
16             InMemoryClientRegistrationRepository(this.googleClientRegistration());
17
18     @Bean
19     public OAuth2AuthorizedClientService authorizedClientService() {
20         return new
21             InMemoryOAuth2AuthorizedClientService(clientRegistrationRepository());
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
```

```
11     public SecurityFilterChain
12         authorizationServerSecurityFilterChain(HttpSecurity http) throws Exception {
13             OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
14             return http.formLogin().and().build();
15         }
16 }
```

5. 验证信息

The screenshot shows a Postman request response. At the top, there are tabs for Body, Cookies, Headers (5), Test Results, and a status bar indicating Status: 200 OK, Time: 95 ms, Size: 204 B, Save as example, and more options. Below the tabs, there are buttons for Pretty, Raw, Preview, Visualize, and JSON (which is selected). The JSON response body is displayed in a code editor-like interface:

```
1 {  
2   "message": "Hello, authenticated user!"  
3 }
```

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

当前集群没有开启鉴权，请参考[文档](#)开启鉴权~

服务列表

public

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阀值	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 >



英 拼 2024/6/15