

Spring Boot Feign微服务技术栈

微服务介绍

技术介绍

- Spring Boot 2.7.9
- Spring Cloud 2021.0.6
- Eureka
- Feign
 - 入门案例
 - 负载均衡
 - 全局策略
 - 局部策略
 - GET/POST方法
 - 性能调优
 - HTTP协议
 - GZIP压缩传输
 - HTTP连接池
 - 日志配置
 - 请求超时
- mysql
- docker

Feign声明式服务调用

1. 学习目标



2. 什么是Feign

Feign 是 Spring Cloud Netflix 组件中的一个轻量级 RESTful 的 HTTP 服务客户端，实现了负载均衡和 Rest 调用的开源框架，封装了 Ribbon 和 RestTemplate，实现了 WebService 的面向接口编程，进一步降低了项目的耦合度。

Feign 内置了 Ribbon，用来做客户端负载均衡调用服务注册中心的服务。

Feign 本身并不支持 Spring MVC 的注解，它有一套自己的注解，为了更方便的使用，Spring Cloud 孵化了 OpenFeign。

Feign 是一种声明式、模板化的 HTTP 客户端（仅在 Consumer 中使用）。

Feign 支持的注解和用法请参考官方文档：<https://github.com/OpenFeign/feign> 或 spring.io 官网文档

Feign 的使用方式是：使用 Feign 的注解定义接口，调用这个接口，就可以调用服务注册中心的服务。

3. Feign解决什么问题

Feign 旨在使编写 JAVA HTTP 客户端变得更加容易，Feign 简化了 RestTemplate 代码，实现了 Ribbon 负载均衡，使代码变得更加简洁，也少了客户端调用的代码，使用 Feign 实现负载均衡是首选方案。只需要你创建一个接口，然后上面添加注解即可。

Feign 是声明式服务调用组件，其核心就是：像调用本地方法一样调用远程方法，无感知远程 HTTP 请求。

- 它解决了让开发者调用远程接口就跟调用本地方法一样的体验，开发者完全感知不到这是远程方法，更感知不到这是个 HTTP 请求。无需关注与远程的交互细节，更无需关注分布式环境开发。
- 它像 Dubbo 一样，Consumer 直接调用 Provider 接口方法，而不需要通过常规的 Http Client 构造请求再解析返回数据。

4. Feign vs OpenFeign

OpenFeign 是 Spring Cloud 在 Feign 的基础上支持了 Spring MVC 的注解，如 `@RequestMapping`、`@PathVariable` 等等。

OpenFeign 的 `@FeignClient` 可以解析 SpringMVC 的 `@RequestMapping` 注解下的接口，并通过动态代理的方式产生实现类，实现类中做负载均衡并调用服务。

5. Feign 入门案例

Feign 的使用主要分为以下几个步骤：

- 服务消费者添加 Feign 依赖；
- 创建业务层接口，添加 `@FeignClient` 注解声明需要调用的服务；
- 业务层抽象方法使用 SpringMVC 注解配置服务地址及参数；
- 启动类添加 `@EnableFeignClients` 注解激活 Feign 组件。

5.1 创建项目

PS：服务消费者通过 Eureka 注册中心获取服务，或者 Ribbon 点对点直连模式都可以使用 Feign 来实现。

我们创建聚合项目并使用 Eureka 注册中心来讲解 Feign，首先创建一个 pom 父工程。

创建过程的步骤忽略

5.2 添加依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>feign</artifactId>
  <version>1.0-SNAPSHOT</version>
  <modules>
    <module>eureka-server01</module>
    <module>eureka-server02</module>
    <module>service-provider</module>
    <module>service-consumer</module>
  </modules>
  <packaging>pom</packaging>

  <name>feign</name>
  <url>http://maven.apache.org</url>

  <!--继承spring boot parent 依赖-->
  <parent>
    <artifactId>spring-boot-starter-parent</artifactId>
    <groupId>org.springframework.boot</groupId>
    <version>2.7.9</version>
  </parent>

  <properties>
    <!--字符集-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--jdk 版本号-->
    <java.version>11</java.version>
    <!--spring cloud 版本号-->
    <spring-cloud.version>2021.0.6</spring-cloud.version>
    <!--spring boot 版本号-->
    <spring-boot.version>2.7.9</spring-boot.version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>${spring-boot.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-dependencies</artifactId>
        <version>${spring-cloud.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
</project>
```

```
</dependencyManagement>

</project>
```

5.3 注册中心eureka-server

创建eureka集群服务注册中心，分别为eureka-server01和eureka-server02

5.3.1 添加依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>feign</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>eureka-server01</artifactId>
  <packaging>jar</packaging>

  <name>eureka-server01</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <!--spring boot web 依赖包-->
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--netflix eureka server 依赖包-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
      <!--          <version>3.1.1</version>-->
      <version>3.0.2</version>
    </dependency>

    <!--spring boot test 依赖包-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
```

```

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
</dependency>
</dependencies>
</project>

```

5.3.2 配置文件

```

server:
  port: 8761

spring:
  application:
    name: eureka-server
  security:
    user:
      name: root
      password: 123456

eureka:
  server:
    enable-self-preservation: true
    eviction-interval-timer-in-ms: 6000

  instance:
    hostname: eureka01
    prefer-ip-address: true
    instance-id: ${spring.cloud.client.ip-address}:${server.port}

  client:
    service-url:
      defaultZone: http://root:123456@localhost:8762/eureka/
    register-with-eureka: true
    fetch-registry: true

```

5.3.3 安全认证

```

package com.example.config;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurer
Adapter;

```

```

/**
 * 功能描述：安全认证配置类
 *
 * @author lizongzai
 * @date 2023/02/24 16:31
 * @since 1.0.0
 */
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.csrf().ignoringAntMatchers("/eureka/**"); //忽略 eureka/**的所有请求安全认证
    }
}

```

5.3.4 项目启动类

```

package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer
@SpringBootApplication
public class EurekaServer01Application {

    public static void main(String[] args) {
        SpringApplication.run(EurekaServer01Application.class, args);
    }

}


```

以相同方式创建eureka-server02,并将eureka-server01/02集群服务注册中心启动

5.3.5 访问服务注册中心

<http://localhost:8761>

<http://localhost:8762>

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2023-02-27T11:05:23 +0800
Data center	default	Uptime	00:13
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	8

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - 192.168.126.1:8762, 192.168.126.1:8761

General Info

Name	Value
------	-------

5.4 服务提供者service-provider

5.4.1 添加依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>feign</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>service-provider</artifactId>
  <packaging>jar</packaging>

  <name>service-provider</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!--spring boot web 依赖-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--netflix eureka client 依赖-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
      <version>3.0.2</version>
    </dependency>
  </dependencies>
</project>
```

```

<!--lombok 依赖包-->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
</dependency>

<!--mysql 依赖-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.18</version>
</dependency>
<!--mybatis-plus 依赖-->
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>3.4.0</version>
</dependency>

<!--swagger2 依赖-->
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.7.0</version>
</dependency>
<!--swagger 第三方ui依赖-->
<dependency>
  <groupId>com.github.xiaoymin</groupId>
  <artifactId>swagger-bootstrap-ui</artifactId>
  <version>1.9.6</version>
</dependency>

<!--spring boot actuator 依赖-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<!--JUnit 测试单元依赖-->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
</dependencies>
</project>

```

5.4.2 配置文件

```

server:
  port: 8080

spring:
  application:

```



```

# 应用名称(集群内相同)
name: service-provider
main:
  allow-circular-references: true
mvc:
  pathmatch:
    matching-strategy: ANT_PATH_MATCHER
datasource:
  driver-class-name: com.mysql.cj.jdbc.Driver
  url: "jdbc:mysql://localhost:3306/micro_service?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai"
  username: root
  password: password

# 配置Eureka Server注册中心
eureka:
  instance:
    # 主机名, 不配置的时候将根据操作系统的主机名称来获取
    hostname: localhost
    # 是否开启IP地址注册
    prefer-ip-address: true
    # 主机地址+端口号
    #instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
    instance-id: ${spring.cloud.client.ip-address}:${server.port}
  client:
    serviceUrl:
      # 注册中心对外暴露的注册地址
      defaultZone:
http://root:123456@localhost:8761/eureka/,http://root:123456@localhost:8762/eureka/
    register-with-eureka: true
    fetch-registry: true

# 度量指标监控与健康检测
management:
  endpoints:
    web:
      exposure:
        # 开启shutdown端点访问
        include: shutdown
  endpoint:
    shutdown:
      # 开启shutdown实现优雅停止服务
      enabled: true

```

5.4.3 业务代码

业务代码 **忽略** , 具体请看github: <https://github.com/lizongzai/feign>

5.4.4 启动项目

忽略

5.4.5 访问服务注册中心

http://localhost:8761

http://localhost:8762

Data centerdefault

Uptime00:16

Lease expiration enabledfalse

Renews threshold5

Renews (last min)4

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - 192.168.126.1:8762 , 192.168.126.1:8761
SERVICE-PROVIDER	n/a (1)	(1)	UP (1) - 192.168.126.1:8080

General Info

Name	Value
total-avail-memory	1280mb
num-of-cpus	12
current-memory-usage	315mb (24%)

访问业务代码接口： http://localhost:8080/product/list

[{"id":149,"productName":"Apple iPhone 14pro","productNum":1,"productPrice":8699,0}]

5.5 服务消费者service-consumer

5.5.1 添加依赖

☐ 添加spring-cloud-starter-openfeign依赖

```
<!--spring cloud openfeign 依赖-->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>feign</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>service-consumer</artifactId>
  <packaging>jar</packaging>

  <name>service-consumer</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!--spring boot web 依赖-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--netflix eureka client 依赖-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
      <version>3.0.2</version>
    </dependency>

    <!--lombok 依赖包-->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
    </dependency>

    <!--mysql 依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.18</version>
    </dependency>

    <!--mybatis-plus 依赖-->
    <dependency>
```

```

        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus-boot-starter</artifactId>
        <version>3.4.0</version>
    </dependency>

    <!--swagger2 依赖-->
    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>2.7.0</version>
    </dependency>
    <!--swagger 第三方ui依赖-->
    <dependency>
        <groupId>com.github.xiaoymin</groupId>
        <artifactId>swagger-bootstrap-ui</artifactId>
        <version>1.9.6</version>
    </dependency>

    <!--spring boot actuator 依赖-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <!--JUnit 测试单元依赖-->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>

    <!--Ribbon 依赖包-->
    <dependency>
        <groupId>com.netflix.ribbon</groupId>
        <artifactId>ribbon-loadbalancer</artifactId>
        <version>2.3.0</version>
    </dependency>

</dependencies>
</project>

```

5.5.2 配置文件

```

server:
  port: 8081

spring:
  application:
    # 应用名称
    name: service-consumer
  main:
    allow-circular-references: true
  mvc:
    pathmatch:
      matching-strategy: ANT_PATH_MATCHER

```

```

datasource:
  driver-class-name: com.mysql.cj.jdbc.Driver
  url: "jdbc:mysql://localhost:3306/micro_service?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai"
  username: root
  password: password

# 配置Eureka Server注册中心
eureka:
  instance:
    # 主机名, 不配置的时候将根据操作系统的主机名称来获取
    hostname: localhost
    # 是否开启IP地址注册
    prefer-ip-address: true
    # 主机地址+端口号
    #instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
    instance-id: ${spring.cloud.client.ip-address}:${server.port}
  client:
    serviceUrl:
      # 注册中心对外暴露的注册地址
      defaultZone:
http://root:123456@localhost:8761/eureka/,http://root:123456@localhost:8762/eureka/
    # 是否将自己注册到注册中心, 默认为true
    register-with-eureka: true
    # 表示Eureka Client 间隔多长时间服务器来取注册信息, 默认为30秒
    registry-fetch-interval-seconds: 10

# 负载均衡策略
# service-provider 是指被调用者的微服务名称
service-provider:
  ribbon:
    NFLoadBalancerRuleClassName: com.netflix.loadBalancer.RandomRule

```

5.5.3 业务代码

业务代码 **忽略** , 具体请看github: <https://github.com/lizongzai/feign>

5.5.4 启动项目

忽略

5.5.5 访问服务注册中心

<http://localhost:8761>

<http://localhost:8762>

Lease expiration enabled	true
Renews threshold	6
Renews (last min)	8

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - 192.168.126.1:8762, 192.168.126.1:8761
SERVICE-CONSUMER	n/a (1)	(1)	UP (1) - 192.168.126.1:8081
SERVICE-PROVIDER	n/a (1)	(1)	UP (1) - 192.168.126.1:8080

General Info

Name	Value
total-avail-memory	1280mb
num-of-cpus	12
current-memory-usage	339mb (26%)
192.168.126.1:8081/actuator/info	00:27

```
@FeignClient(name="sss",url="localhost:8761",configuration=FeignConfiguration.class)
public interface IfeignInter {

    //1.@GetMapping("/eureka/apps/{serviceName}")@PathVariable("serviceName")

    //2.@RequestMapping(value="/eureka/apps/{serviceName}",
method=RequestMethod.GET)@PathVariable("必须有值")

    //3.@RequestMapping("GET /eureka/apps/{serviceName}")@Param("serviceName")

    @RequestMapping("GET /eureka/apps/{serviceName}")
    public String getServiceInfoByserviceName(@Param("serviceName") String
serviceName) ;
}
```

5.6 feign核心代码

5.6.1 服务消费者启动类

添加@EnableFeignClients注解功能

```
package com.example;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@EnableFeignClients
@SpringBootApplication
@MapperScan("com.example.mapper")
public class ServiceConsumerApplication {
```

```

    public static void main(String[] args) {
        SpringApplication.run(ServiceConsumerApplication.class, args);
    }
}

```

5.6.2 创建被调用服务者接口

在consumer消费者项目，添加被调用者接口

```

package com.example.service;

import com.example.pojo.Product;
import java.util.List;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * 功能描述：调用服务生产者的微服务名称,例如：service-provider
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
@FeignClient("service-provider")
public interface IProductService {

    /**
     * 功能描述：获取商品列表
     *
     * @return
     */
    //配置需要调用的微服务地址和参数
    @GetMapping("/product/list")
    List<Product> selectProductList();

}

```

5.6.3 服务消费者接口实现

```

package com.example.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.mapper.OrderMapper;
import com.example.pojo.Order;
import com.example.pojo.Product;
import com.example.service.IOrderService;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * <p>

```

```

* 服务实现类
* </p>
*
* @author lizongzai
* @date 2023/02/27 11:50
* @since 1.0.0
*/
@Service
public class OrderServiceImpl extends ServiceImpl<OrderMapper, Order> implements
IOrderService {

    @Autowired
    private OrderMapper orderMapper;
    @Autowired
    private IProductService productService;

    /**
     * 获取订单
     *
     * @param id
     * @return
     */
    @Override
    public Order selectOrderById(Integer id) {

        //LoadBalancerClient负载均衡调用微服务
        List<Product> productList = productService.selectProductList();

        //获取订单信息
        Order mapperOrderById = orderMapper.getOrderById(id);
        Order order = new Order();
        order.setId(mapperOrderById.getId());
        order.setOrderNo(mapperOrderById.getOrderNo());
        order.setOrderAddress(mapperOrderById.getOrderAddress());
        order.setTotalPrice(mapperOrderById.getTotalPrice());
        order.setProductList(productList);
        return order;
    }
}

```

5.6.4 访问订单接口

<http://localhost:8081/order/1>


```
{
  "id": 1,
  "orderNo": "1001",
  "orderAddress": "上海市浦东新区陆家嘴金茂大厦",
  "totalPrice": 8699.0,
  "productList": [{
    "id": 149,
    "productName": "Apple iPhone 14pro",
    "productNum": 1,
    "productPrice": 8699.0
  }]
}
```

6. Feign负载均衡

Feign 封装了 Ribbon 自然也就集成了负载均衡的功能，默认采用轮询策略。如何修改负载均衡策略呢？与之前学习 Ribbon 时讲解的配置是一致的。

6.1 全局策略

在启动类或配置类中注入负载均衡策略对象, 所有服务请求均使用该随机策略(), 默认为轮询策略。

服务消费者 `service-consumer` 中，添加代码如下：

```
@Bean
public RandomRule randomRule() {
    return new RandomRule();
}
```

6.2 局部策略

在服务消费者-->修改配置文件指定服务的负载均衡策略, 格式：`微服务应用名.ribbon.NFLoadBalancerRuleClassName`

```
# 负载均衡策略
# service-provider 是指被调用者的微服务名称
service-provider:
  ribbon:
    NFLoadBalancerRuleClassName: com.netflix.loadBalancer.RandomRule
```

7. Feign请求参数

7.1 GET方法

使用 `@PathVariable` 注解和 `@RequestParam` 注解接收请求参数

7.1.1 服务提供者

7.1.1.1 ProductController

```
package com.example.controller;

import com.example.pojo.Product;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * <p>
 * 前端控制器
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    /**
     * 功能描述：根据主键查询商品
     *
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Product selectProductById(@PathVariable("id") Integer id) {
        return productService.selectProductById(id);
    }

}
```

7.1.1.2 IProductService

```
package com.example.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.example.pojo.Product;
import java.util.List;
```

```

/**
 * <p>
 * 服务类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
public interface IProductService extends IService<Product> {

    /**
     * 功能描述：根据主键查询商品
     *
     * @return
     */
    Product selectProductById(Integer id);

}

```

7.1.1.3 ProductServiceImpl

```

package com.example.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.mapper.ProductMapper;
import com.example.pojo.Product;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@Service
public class ProductServiceImpl extends ServiceImpl<ProductMapper, Product> implements
    IProductService {

    @Autowired
    private ProductMapper productMapper;

    /**
     * 功能描述：根据主键查询商品
     *
     * @param id
     * @return
     */
    @Override
    public Product selectProductById(Integer id) {
        return productMapper.selectProductById(id);
    }
}

```

```
}  
}
```

7.1.1.4 ProductMapper

```
package com.example.mapper;  
  
import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
import com.example.pojo.Product;  
import java.util.List;  
import org.apache.ibatis.annotations.Param;  
  
/**  
 * <p>  
 * Mapper 接口  
 * </p>  
 *  
 * @author lizongzai  
 * @since 2023-02-23  
 */  
public interface ProductMapper extends BaseMapper<Product> {  
  
    /**  
     * 功能描述：根据主键查询商品  
     *  
     * @param id  
     * @return  
     */  
    Product selectProductById(@Param("id") Integer id);  
}
```

7.1.1.5 Mybatis

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="com.example.mapper.ProductMapper">  
  
    <!-- 通用查询映射结果 -->  
    <resultMap id="BaseResultMap" type="com.example.pojo.Product">  
        <id column="id" property="id"/>  
        <result column="productName" property="productName"/>  
        <result column="productNum" property="productNum"/>  
        <result column="productPrice" property="productPrice"/>  
    </resultMap>  
  
    <!-- 通用查询结果列 -->  
    <sql id="Base_Column_List">  
        id  
        , productName, productNum, productPrice  
    </sql>  
  
    <!-- 根据主键查询商品-->  
    <select id="selectProductById" resultType="com.example.pojo.Product">
```

```

        select *
        from t_product
        where id = #{id};
    </select>

</mapper>

```

7.1.2 服务消费者

7.1.2.1 IProductService(★)

```

package com.example.service;

import com.example.pojo.Product;
import java.util.List;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

/**
 * 功能描述：调用服务生产者的微服务名称,例如：service-provider
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
@FeignClient("service-provider")
public interface IProductService {

    /**
     * 功能描述：根据主键查询商品
     *
     * @param id
     * @return
     */
    @GetMapping("/product/{id}")
    Product selectProductById(@PathVariable("id") Integer id);

}

```

7.1.2.2 OrderServiceImpl(★)

```

package com.example.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.mapper.OrderMapper;
import com.example.pojo.Order;
import com.example.pojo.Product;
import com.example.service.IOrderService;
import com.example.service.IProductService;
import java.util.Collections;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

```

```

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
@Service
public class OrderServiceImpl extends ServiceImpl<OrderMapper, Order> implements
IOrderService {

    @Autowired
    private OrderMapper orderMapper;
    @Autowired
    private IProductService productService;

    /**
     * 功能描述：获取订单
     *
     * @param id
     * @return
     */
    @Override
    public Order selectOrderById(Integer id) {

        //查询商品列表
        Product productById = productService.selectProductById(id);

        //获取订单信息
        Order mapperOrderById = orderMapper.getOrderById(id);
        Order order = new Order();
        order.setId(mapperOrderById.getId());
        order.setOrderNo(mapperOrderById.getOrderNo());
        order.setOrderAddress(mapperOrderById.getOrderAddress());
        order.setTotalPrice(mapperOrderById.getTotalPrice());
        order.setProductList(Collections.singletonList(productById));
        return order;
    }
}

```

7.1.2.3 访问订单接口

<http://localhost:8081/order/1>

```
{
  "id": 1,
  "orderNo": "1001",
  "orderAddress": "上海市浦东新区陆家嘴金茂大厦",
  "totalPrice": 8699.0,
  "productList": [{
    "id": 1,
    "productName": "Apple iPhone 14pro",
    "productNum": 1,
    "productPrice": 8699.0
  }]
}
```

7.2 POST方法

使用 `@RequestBody` 注解接收请求参数

7.2.1 服务提供者

7.2.1.1 ProductController

```
package com.example.controller;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * <p>
 * 前端控制器
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    /**
     * 功能描述：使用POST方法,根据主键查询商品
     *

```

```

    * @param id
    * @return
    */
@PostMapping("/single")
public Product queryProductById(@RequestBody Integer id) {
    return productService.queryProductById(id);
}

/**
 * 功能描述：使用POST方法,添加商品
 *
 * @param product
 * @return
 */
@PostMapping("/save")
public RespBean addProduct(@RequestBody Product product) {

    if (product.getProductName() == null || product.getProductNum() == null ||
product.getProductPrice() == null) {
        return RespBean.success("添加失败");
    }
    return productService.addProduct(product);
}
}

```

7.2.1.2 IProductService

```

package com.example.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;

/**
 * <p>
 * 服务类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
public interface IProductService extends IService<Product> {

    /**
     * 功能描述：使用POST方法,根据主键查询商品
     *
     * @param id
     * @return
     */
    Product queryProductById(Integer id);

    /**

```



```

    * 功能描述：使用POST方法,添加商品
    *
    * @param product
    * @return
    */
    RespBean addProduct(Product product);
}

```

7.2.1.3 ProductServiceImpl

```

package com.example.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.mapper.ProductMapper;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@Service
public class ProductServiceImpl extends ServiceImpl<ProductMapper, Product> implements
    IProductService {

    @Autowired
    private ProductMapper productMapper;

    /**
     * 功能描述：使用POST方法,根据主键查询商品
     *
     * @param id
     * @return
     */
    @Override
    public Product queryProductById(Integer id) {
        return productMapper.queryProductById(id);
    }

    /**
     * 功能描述：使用POST方法,添加商品
     *
     * @param product
     * @return
     */
    @Override
    public RespBean addProduct(Product product) {

```

```

        int result = productMapper.addProduct(product);
        System.out.println("添加商品 = " + product);
        if (result > 0 ) {
            return RespBean.success("添加成功!");
        } else {
            return RespBean.success("添加失败!");
        }
    }
}

```

7.2.1.4 ProductMapper

```

package com.example.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;
import java.util.Map;
import org.apache.ibatis.annotations.Param;

/**
 * <p>
 * Mapper 接口
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
public interface ProductMapper extends BaseMapper<Product> {

    /**
     * 功能描述：使用POST方法, 根据主键查询商品
     *
     * @param id
     * @return
     */
    Product queryProductById(@Param("id") Integer id);

    /**
     * 功能描述：使用POST方法, 添加商品
     *
     * @param product
     * @return
     */
    int addProduct(Product product);
}

```

7.2.1.5 Mybatis

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.mapper.ProductMapper">

    <!-- 通用查询映射结果 -->
    <resultMap id="BaseResultMap" type="com.example.pojo.Product">
        <id column="id" property="id"/>
        <result column="productName" property="productName"/>
        <result column="productNum" property="productNum"/>
        <result column="productPrice" property="productPrice"/>
    </resultMap>

    <!-- 通用查询结果列 -->
    <sql id="Base_Column_List">
        id
        , productName, productNum, productPrice
    </sql>

    <!-- 功能描述：使用POST方法,根据主键查询商品-->
    <select id="queryProductById" resultType="com.example.pojo.Product">
        select *
        from t_product
        where id = #{id};
    </select>

    <!-- 使用POST方法,添加商品-->
    <insert id="addProduct">
        insert into t_product(id, productName, productNum, productPrice)
        values (#{id}, #{productName}, #{productNum}, #{productPrice});
    </insert>

</mapper>
```

7.2.2 服务消费者

7.2.2.1 ProductController

```
package com.example.controller;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
```

```

private IProductService productService;

/**
 * 功能描述：使用POST方法,根据主键查询商品
 *
 * @param id
 * @return
 */
@PostMapping("/info")
public Product queryProductById(Integer id) {
    return productService.queryProductById(id);
}

/**
 * 功能描述：使用POST方法,添加商品
 *
 * @param product
 * @return
 */
@PostMapping("/save")
public RespBean addProduct(@RequestBody Product product) {

    if (product.getProductName() == null || product.getProductNum() == null ||
product.getProductPrice() == null) {
        return RespBean.success("添加失败");
    }
    return productService.addProduct(product);
}
}

```

7.2.2.2 IProductService(★)

```

package com.example.service;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * 功能描述：调用服务生产者的微服务名称,例如：service-provider
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
@FeignClient("service-provider")
public interface IProductService {

    /**
     * 功能描述：使用POST方法,根据主键查询商品
     *
     * @param id
     */
}

```

```

* @return
*/
@PostMapping("/product/single")
Product queryProductById(Integer id);

/**
 * 功能描述：使用POST方法,添加商品
 *
 * @param product
 * @return
 */
@PostMapping("/product/save")
RespBean addProduct(Product product);
}

```

7.2.2.3 访问订单接口

The image shows two screenshots. The top screenshot is from Postman, showing a POST request to `http://localhost:8081/product/save` with a JSON body containing product details. The response is a JSON object with a success message. The bottom screenshot is from IntelliJ IDEA, showing the project structure and a database query result in the `t_product` table.

Postman Request Details:

- Method: POST
- URL: `http://localhost:8081/product/save`
- Body (JSON):


```

{
  "productName": "海尔冰箱",
  "productNum": 1,
  "productPrice": 15999
}

```
- Status: 200 OK
- Response (JSON):


```

{
  "code": 200,
  "message": "添加成功!",
  "object": null
}

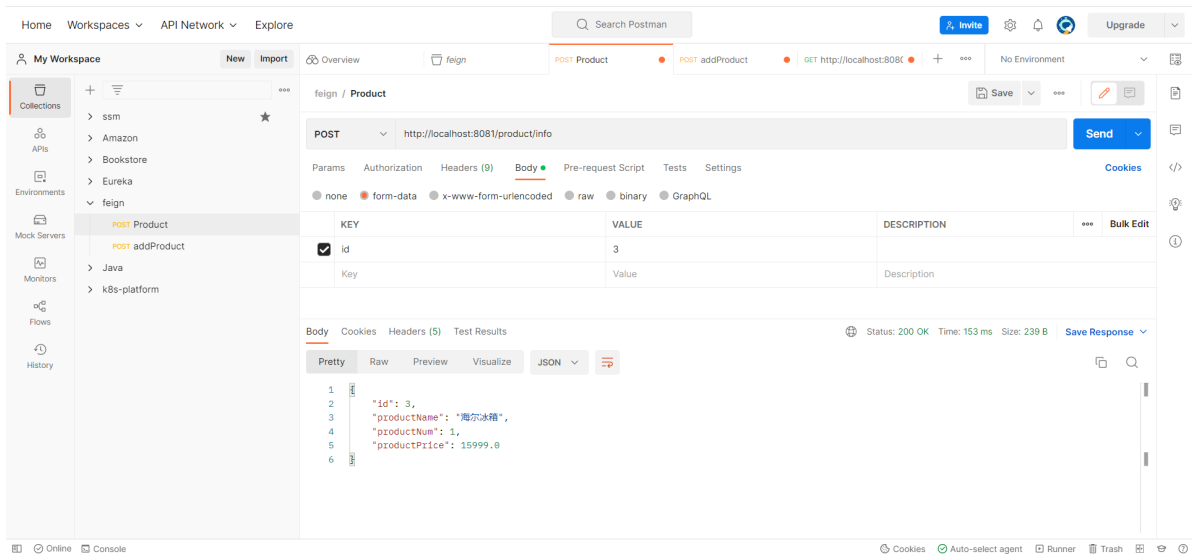
```

IntelliJ IDEA Database View:

Database: `micro_service@localhost` | Table: `t_product`

ID	productName	productNum	productPrice
1	Apple iPhone 14pro	1	8699
2	笔记本电脑	1	29999
3	海尔冰箱	1	15999

Services running: `micro_service@localhost` | `t_product` | `console`

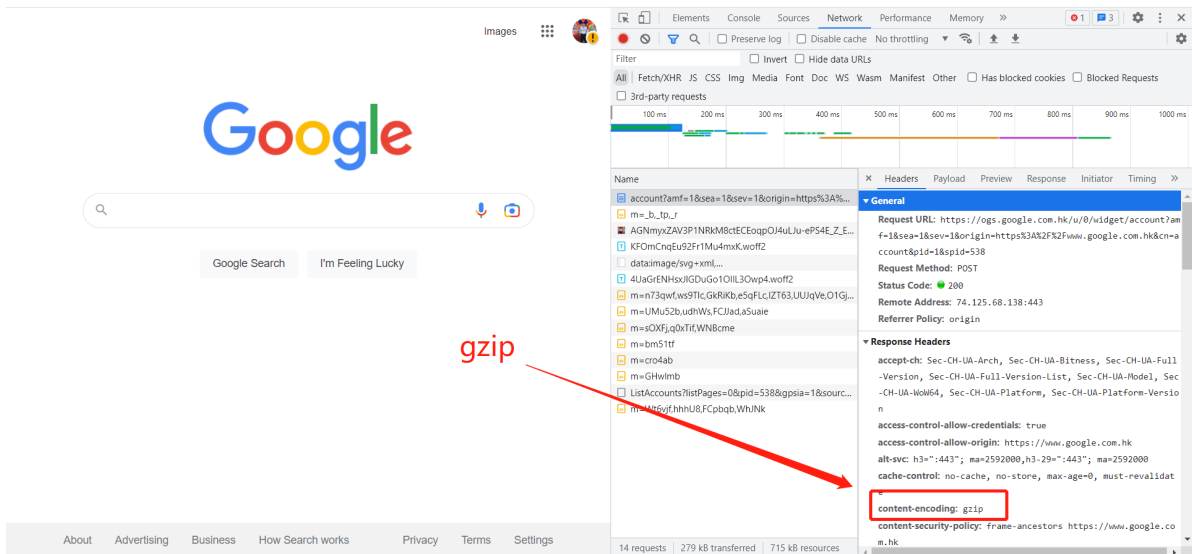


8. Feign性能调优

gzip 介绍：gzip 是一种数据格式，采用 deflate 算法压缩数据；gzip 是一种流行的文件压缩算法，应用十分广泛，尤其是在 Linux 平台。

gzip 能力：当 Gzip 压缩一个纯文本文件时，效果是非常明显的，大约可以减少 70% 以上的文件大小。

gzip 作用：网络数据经过压缩后实际上降低了网络传输的字节数，最明显的好处就是可以加快网页加载的速度。网页加载速度加快的好处不言而喻，除了节省流量，改善用户的浏览体验外，另一个潜在的好处是 Gzip 与搜索引擎的抓取工具有着更好的关系。例如 Google 就可以通过直接读取 gzip 文件来比普通手工抓取更快地检索网页。



8.1 HTTP协议关于压缩传输的规定

1. 客户端向服务器请求中带有：`Accept-Encoding:gzip, deflate` 字段，向服务器表示客户端支持的压缩格式（gzip 或者 deflate），如果不发送该消息头，服务端默认是不会压缩的。
2. 服务端在收到请求之后，如果发现请求头中含有 `Accept-Encoding` 字段，并且支持该类型压缩，就会对响应报文压缩之后返回给客户端，并且携带 `Content-Encoding:gzip` 消息头，表示响应报文是根据该格式进行压缩的。
3. 客户端接收到请求之后，先判断是否有 `Content-Encoding` 消息头，如果有，按该格式解压报文。否则按正常报文处理。

8.2 GZIP压缩案例

8.2.1 局部策略

只需要配置service-consumer通过Feign到provider的请求与响应的GZIP压缩。

服务消费者(application.yml)

```
# Feign gzip 压缩
feign:
  compression:
    request:
      mime-types: text/xml,application/xml,application/json # 配置压缩支持的 MIME TYPE
      min-request-size: 512 # 配置压缩数据大小的最小阈值，默认 2048
      enabled: true # 请求是否开启 gzip 压缩
    response:
      enabled: true # 响应是否开启 gzip 压缩
```

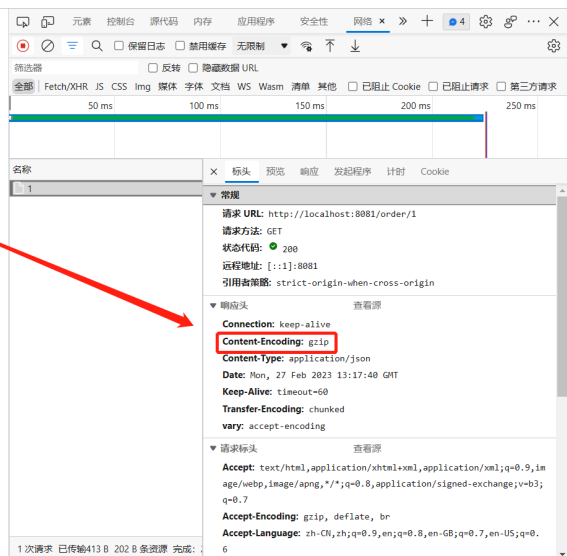
8.2.2 全局策略

对客户端浏览器的请求以及Consumer对Provider的请求与响应都实现GZIP压缩。

服务消费者(application.yml)

```
server:
  port: 8081
  compression:
    # 是否开启压缩请求
    enabled: true
    # 配置支持类型
    mime-types: application/json,application/xml,text/html,text/html,text/plain
```

```
[{"id":1,"orderNo":"1001","orderAddress":"上海市浦东新区陆家嘴金茂大厦","totalPrice":8699.0,"productList":
[{"id":1,"productName":"Apple iPhone 14pro","productNum":1,"productPrice":8699.0}]}
```



8.3 HTTP连接池

为什么HTTP连接池能提升性能?

8.3.1 HTTP的背景原理

- 两台服务器建立 HTTP 连接的过程是很复杂的一个过程，涉及到多个数据包的交换，很耗时间。
- HTTP 连接需要的 3 次握手 4 次挥手开销很大，这一开销对于大量的比较小的 HTTP 消息来说更大。

8.3.2 解决方案

采用 HTTP 连接池，可以节约大量的 3 次握手 4 次挥手，这样能大大提升吞吐量。

Feign 的 HTTP 客户端支持 3 种框架：HttpURLConnection、HttpClient、OkHttp；默认是 HttpURLConnection。可以通过查看源码 `org.springframework.cloud.openfeign.ribbon.FeignRibbonClientAutoConfiguration.java` 得知。

- 传统的 HttpURLConnection 是 JDK 自带的，并不支持连接池，如果要实现连接池的机制，还需要自己来管理连接对象。对于网络请求这种底层相对复杂的操作，如果有可用的其他方案，没有必要自己去管理连接对象。
- HttpClient 相比传统 JDK 自带的 HttpURLConnection，它封装了访问 HTTP 的请求头，参数，内容体，响应等等；它不仅使客户端发送 HTTP 请求变得容易，而且也方便了开发人员测试接口（基于 HTTP 协议的），既提高了开发的效率，又提高了代码的健壮性；另外高并发大量的请求网络的时候，也是用“连接池”提升吞吐量。

8.3.3 HttpClient

- 将Feign的HTTP客户端工具修改为HttpClient。
- 修改Consumer项目，添加两个依赖，因为本文使用的spring cloud版本已默认集成了apache http依赖，所以需要添加一个依赖即可。

```
<!-- 当前版本已经默认集成了 apache httpclient 依赖 -->
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.11</version>
</dependency>
<!-- feign apache httpclient 依赖 -->
<dependency>
  <groupId>io.github.openfeign</groupId>
  <artifactId>feign-httpclient</artifactId>
  <version>10.7.4</version>
</dependency>
```



```

<!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.14</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.github.openfeign/feign-httpclient -->
<dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-httpclient</artifactId>
    <version>11.10</version>
</dependency>

```

8.3.4 配置文件

```

feign:
  httpclient:
    enabled: true # 开启httpClient

```

PS: 如果使用 HttpClient 作为 Feign 的客户端工具。那么在定义接口上的注解是需要注意的，如果传递的参数是一个自定义的对象（对象会使用 JSON 格式来专递），需要配置参数类型，例如：`@GetMapping(value = "/single/pojo", consumes = MediaType.APPLICATION_JSON_VALUE)`。本文中使用的 Spring Cloud 版本，已无需手动配置。并且使用了 HttpClient 客户端以后，我们还可以通过 GET 请求传递对象参数。

8.3.5 服务提供者

8.3.5.1 ProductController

```

package com.example.controller;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * <p>
 * 前端控制器
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */

```

```

@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    /**
     * 功能描述：接收商品对象参数
     *
     * @param product
     * @return
     */
    @GetMapping("/pojo")
    public Product selectProductByPojo(@RequestBody Product product) {
        return productService.selectProductByPojo(product);
    }
}

```

8.3.5.2 IProductService

```

package com.example.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;

/**
 * <p>
 * 服务类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
public interface IProductService extends IService<Product> {

    /**
     * 功能描述：接收商品对象参数
     *
     * @param product
     * @return
     */
    Product selectProductByPojo(Product product);

}

```

8.3.5.3 ProductServiceImpl(★)

```

package com.example.service.impl;

import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;

```

```

import com.example.mapper.ProductMapper;
import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * <p>
 * 服务实现类
 * </p>
 *
 * @author lizongzai
 * @since 2023-02-23
 */
@Service
public class ProductServiceImpl extends ServiceImpl<ProductMapper, Product> implements
    IProductService {

    /**
     * 功能描述：接收商品对象参数
     *
     * @param product
     * @return
     */
    @Override
    public Product selectProductByPojo(Product product) {
        System.out.println("接收商品对象参数 = " + product);
        return product;
    }
}

```

8.3.6 服务消费者

8.3.6.1 ProductController

```

package com.example.controller;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import com.example.service.IProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;
}

```

```

/**
 * 功能描述：接收商品对象参数
 *
 * @param product
 * @return
 */
@GetMapping("/pojo")
public Product selectProductByPojo(@RequestBody Product product) {
    return productService.selectProductByPojo(product);
}
}

```

8.3.6.2 IProductService

```

package com.example.service;

import com.example.pojo.Product;
import com.example.pojo.RespBean;
import java.util.List;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

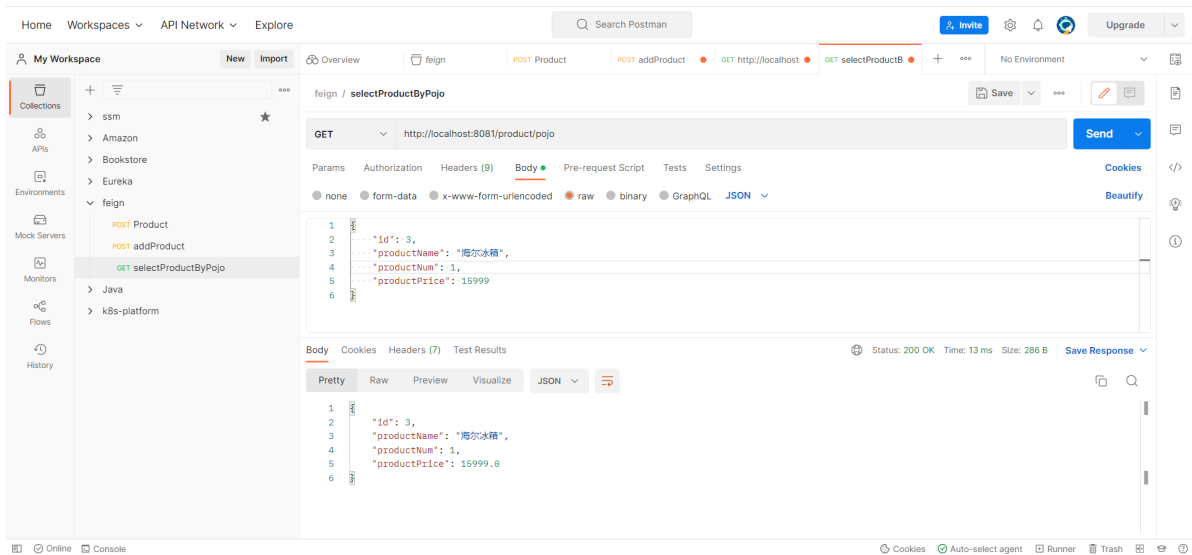
/**
 * 功能描述：调用服务生产者的微服务名称,例如：service-provider
 *
 * @author lizongzai
 * @date 2023/02/27 11:50
 * @since 1.0.0
 */
@FeignClient("service-provider")
public interface IProductService {

    /**
     * 功能描述：接收商品对象参数
     *
     * @param product
     * @return
     */
    @GetMapping("/product/pojo")
    Product selectProductByPojo(Product product);

}

```

8.3.6.3 访问订单接口



8.4 状态查看

浏览器发起的请求我们可以借助 F12 **Devtools** 中的 **Network** 来查看请求和响应信息。对于微服务中每个接口我们又该如何查看 URL，状态码和耗时信息？我们可以使用配置日志的方式进行查看。

8.4.1 logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="60 seconds" debug="false">

    <!-- 应用名称：和统一配置中的项目代码保持一致（小写） -->
    <property name="APP_NAME" value="app"/>
    <contextName>${APP_NAME}</contextName>

    <!-- 彩色日志依赖的渲染类 -->
    <conversionRule conversionWord="clr"
        converterClass="org.springframework.boot.logging.logback.ColorConverter"/>
    <conversionRule conversionWord="wex"
        converterClass="org.springframework.boot.logging.logback.WhitespaceThrowableProxyConverter"/>
    <conversionRule conversionWord="wEx"
        converterClass="org.springframework.boot.logging.logback.ExtendedWhitespaceThrowableProxyConverter"/>
    <property name="Console_Log_Pattern"
        value="${CONSOLE_LOG_PATTERN:-%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint}
%clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint}
%clr(%-40.40logger{50}){cyan} %clr(:){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:-
wEx}}"/>

    <!-- 日志文件保留天数 -->
    <property name="LOG_MAX_HISTORY" value="30"/>
    <!-- 定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径 -->

    <!-- 应用日志文件保存路径 -->
    <!-- 在没有定义 ${LOG_HOME} 系统变量的时候，可以设置此本地变量。 -->
```

```

<property name="LOG_HOME" value="{catalina.base}/service-consumer/logs" />
<property name="INFO_PATH" value="{LOG_HOME}/info" />
<property name="DEBUG_PATH" value="{LOG_HOME}/debug" />
<property name="ERROR_PATH" value="{LOG_HOME}/error" />
<!--<property name="LOG_HOME" msg="/home/logs/${APP_NAME}" />-->

<!--===== 按照每天生成日志文件：默认配置
===== -->

<!-- 控制台输出 -->
<appender name="console" class="ch.qos.logback.core.ConsoleAppender">
  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
    <level>DEBUG</level>
  </filter>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <!--格式化输出：%d表示日期，%c类名，%t表示线程名，%L行， %p日志级别 %msg：日志消息，%n是换行
符 -->
    <pattern>%black(%contextName - %d{yyyy-MM-dd HH:mm:ss}) %green([%c][%t][%L])
%highlight(%-5level) - %gray(%msg%n)
    <charset>UTF-8</charset>
  </pattern>
  </encoder>
</appender>

<!-- 按照每天生成日志文件：主项目日志 -->
<appender name="APP_DEBUG" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <!--日志文件输出的文件名 -->
    <FileNamePattern>${DEBUG_PATH}/debug-%d{yyyy-MM-dd}.log</FileNamePattern>
    <!--日志文件保留天数 -->
    <MaxHistory>${LOG_MAX_HISTORY}</MaxHistory>
  </rollingPolicy>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <!--格式化输出：%d表示日期，%c类名，%t表示线程名，%L行， %p日志级别 %msg：日志消息，%n是换行
符 -->
    <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%c][%t][%L][%p] - %msg%n</pattern>
    <charset>UTF-8</charset>
  </encoder>
  <!-- 此日志文件只记录debug级别的 -->
  <filter class="ch.qos.logback.classic.filter.LevelFilter">
    <level>debug</level>
    <onMatch>ACCEPT</onMatch>
    <onMismatch>DENY</onMismatch>
  </filter>
</appender>

<!-- 按照每天生成日志文件：主项目日志 -->
<appender name="APP_INFO" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <!--日志文件输出的文件名 -->
    <FileNamePattern>${INFO_PATH}/info-%d{yyyy-MM-dd}.log</FileNamePattern>
    <!--日志文件保留天数 -->
    <MaxHistory>${LOG_MAX_HISTORY}</MaxHistory>
  </rollingPolicy>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <!--格式化输出：%d表示日期，%c类名，%t表示线程名，%L行， %p日志级别 %msg：日志消息，%n是换行
符 -->
    <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%c][%t][%L][%p] - %msg%n</pattern>
    <charset>UTF-8</charset>

```

```

</encoder>

<!-- 此日志文件只记录info级别的 -->
<filter class="ch.qos.logback.classic.filter.LevelFilter">
    <level>info</level>
    <onMatch>ACCEPT</onMatch>
    <onMismatch>DENY</onMismatch>
</filter>
</appender>

<!-- 按照每天生成日志文件：主项目日志 -->
<appender name="APP_ERROR" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- 日志文件输出的文件名 -->
        <fileNamePattern>${ERROR_PATH}/error-%d{yyyy-MM-dd}.log</fileNamePattern>
        <!-- 日志文件保留天数 -->
        <maxHistory>${LOG_MAX_HISTORY}</maxHistory>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <!-- 格式化输出：%d表示日期，%c类名，%t表示线程名，%L行， %p日志级别 %msg：日志消息，%n是换行符 -->
        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%c][%t][%L][%p] - %msg%n</pattern>
        <charset>UTF-8</charset>
    </encoder>
    <!-- 此日志文件只记录error级别的 -->
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
        <level>error</level>
        <onMatch>ACCEPT</onMatch>
        <onMismatch>DENY</onMismatch>
    </filter>
</appender>

<!-- 日志输出到文件-->
<root level="info">
    <appender-ref ref="APP_DEBUG"/>
    <appender-ref ref="APP_INFO"/>
    <appender-ref ref="APP_ERROR"/>
    <appender-ref ref="console"/>
</root>

<!-- mybatis 日志级别 -->
<logger name="com.pm.health" level="debug"/>
</configuration>

```

8.4.2 全局策略

Consumer项目启动类中注入Feign的Logger对象

```

package com.example;

import com.netflix.loadbalancer.RandomRule;
import feign.Logger;
import feign.Logger.Level;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;

```

```

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;

@EnableFeignClients
@SpringBootApplication
@MapperScan("com.example.mapper")
public class ServiceConsumerApplication {

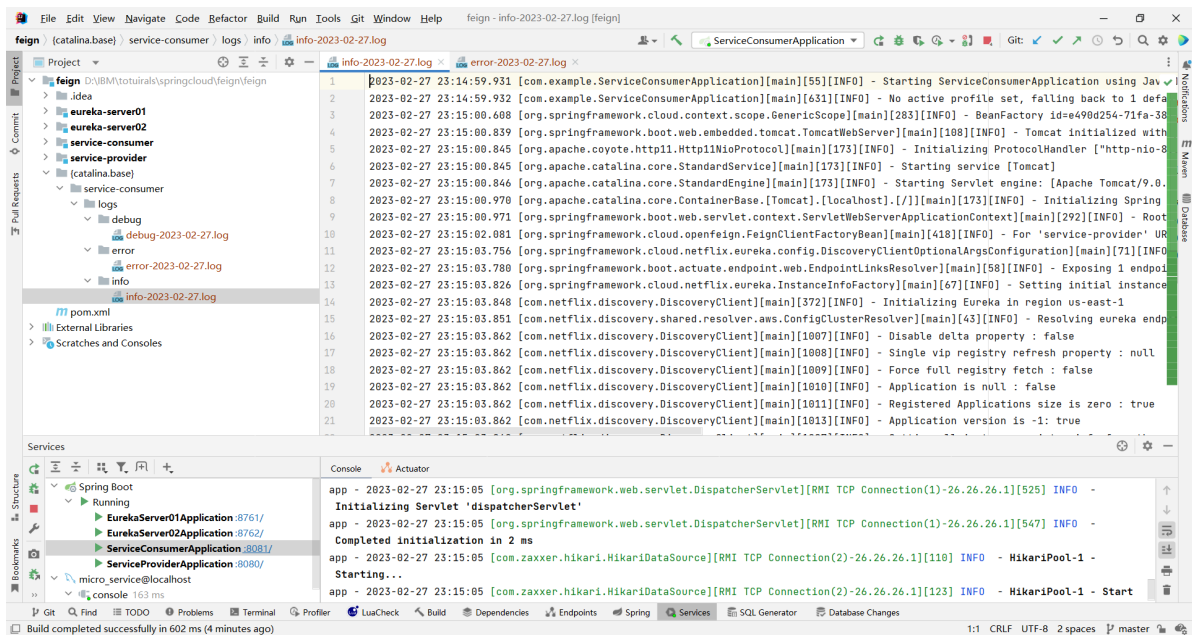
    @Bean
    public Logger.Level getLog() {
        return Level.FULL;
    }

    @Bean
    public RandomRule randomRule() {
        return new RandomRule();
    }

    public static void main(String[] args) {
        SpringApplication.run(ServiceConsumerApplication.class, args);
    }

}

```



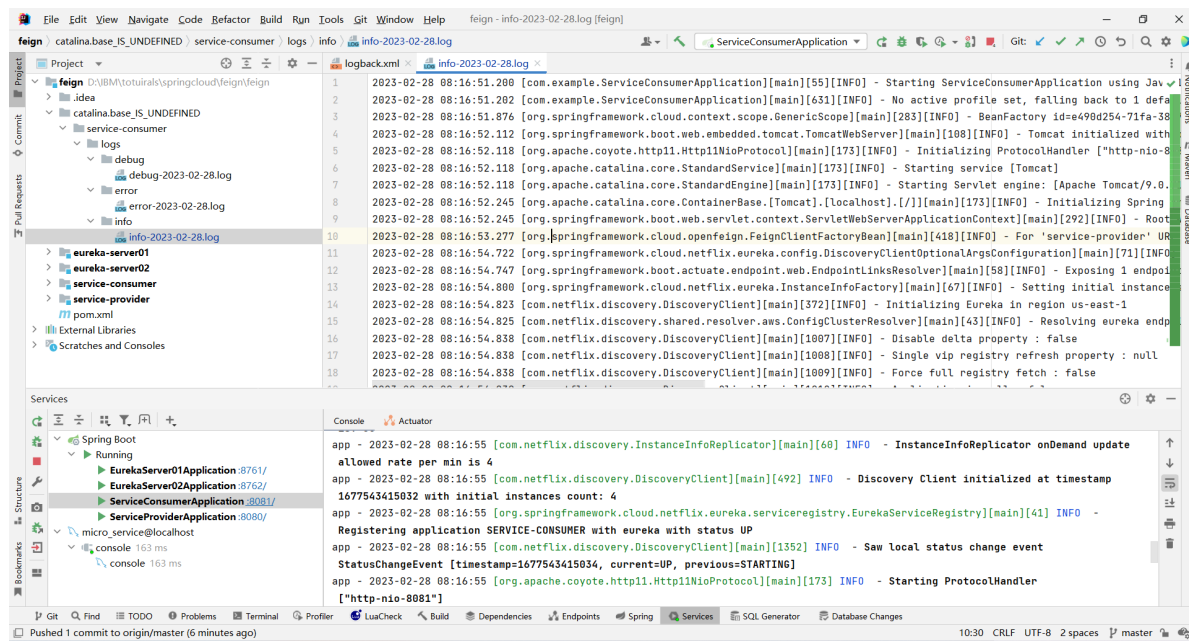
8.4.3 局部策略

```

feign:
  httpclient:
    enabled: true # 开启httpclient
  client:
    config:
      service-provider: # 需要调用的服务名称
      logger-level: Full

```


8.4.4 测试



8.5 请求超时

Feign 的负载均衡底层用的就是 Ribbon，所以这里的请求超时配置其实就是配置 Ribbon。

分布式项目中，服务压力比较大的情况下，可能处理服务的过程需要花费一定的时间，而默认情况下请求超时的配置是 1s 所以我们需要调整该配置延长请求超时时间。

service-provider项目中，添加测试代码

```
/**
 * 功能描述：使用GET方法，根据主键查询商品
 *
 * @param id
 * @return
 */
@GetMapping("/{id}")
public Product selectProductById(@PathVariable("id") Integer id) {
    try {
        // 设置超时验证
        Thread.sleep(2000L);
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
    return productService.selectProductById(id);
}
```

8.5.1 全局策略

(23条消息) 微服务化之各种超时时间配置效果 [JavaBoy_XJ的博客-CSDN博客](#) 微服务超时时间

<https://blog.csdn.net/xj80231314/article/details/88853369>

consumer项目中配置请求超时的处理。

```
ribbon:  
  ConnectionTimeout: 5000 # 请求连接超时时间,默认为1秒  
  ReadTimeout: 5000 # 请求处理的超时时间
```

8.5.2 局部策略

```
ribbon:  
  OkToRetryOnAllOperations: false #对所有操作请求都进行重试,默认false  
  ReadTimeout: 5000 #负载均衡超时时间,默认值5000  
  ConnectTimeout: 3000 #ribbon请求连接的超时时间,默认值2000  
  MaxAutoRetries: 3 #对当前实例的重试次数,默认0  
  MaxAutoRetriesNextServer: 1 #对切换实例的重试次数,默认1
```