服务网关

在一个微服务架构中，每个服务都会有不同的网络地址，外部客户端在完成某个业务需求时可能需要同时调用多个服务，一个订餐系统，需要同时调用多个微服务才能完成一次业务。

这种方式会带来以下问题：

- 客户端多次请求不同的微服务，会增加客户端的复杂性。
- 认证复杂，每个微服务都需要进行认证。
- HTTP 请求增加，效率不高。
- 存在跨域请求，处理起来会比较复杂。

如何解决？在客户端和服务端之间添加一个 API 网关，所有的外部请求只需要跟网关进行交互，同时由网关完成各个微服务的调用，这就是网关原理，微服务网关是介于客户端和服务端之间的，外部请求会先经过微服务网关，再由网关映射到各个微服务。

网关的优点：

- 减少客户端和微服务之间的调用，提高效率。
- 便于监控，可在网关中监控数据，完成统一的切面任务处理。
- 便于认证，只需要在网关中进行认证，无需每个微服务都进行认证。
- 降低了客户端与服务端的耦合度。

Spring Cloud Zuul 来实现微服务网关的落地。

Zuul

Zuul 是 Netflix 开源的一个 API GateWay 服务器，本质上是一个 Web Servlet 应用，Zuul 在云平台上提供动态路由，监控，安全等服务，Zuul 相当于客户端和 Netflix 网站后端所有请求的中间层，它可以和 Eureka、Ribbon、Hystrix 等组件配合使用，Zuul 可以通过加载动态过滤机制实现以下功能：

- 验证与安全保障。
- 审查和监控。
- 动态路由。
- 压力测试。
- 负载分配。
- 静态响应处理。
- 多区域弹性。

代码实现

- pom.xml

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

- application.yml

```
server:
  port: 8030
spring:
  application:
    name: gateway
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
zuul:
  routes:
    provider: /p/**
```

`zuul.routes.provider`：自定义微服务的访问路径，`zuul.routes.provider: /p/**` 表示 provider 微服务就会被映射到 gateway 的 /p/** 路径。

- 启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@EnableZuulProxy
@EnableAutoConfiguration
public class GateWayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GateWayApplication.class,args);
    }
}
```

> 注解说明

`@EnableZuulProxy` 包含 `@EnableZuulServer` 的功能，而且还加入了 `@EnableCiruitBreaker` 和 `@EnableDiscoveryClient`。

`@EnableAutoConfiguration` 可以帮助 Spring Boot 应用将所有符合条件的 `@Configuration` 配置都加载到当前 Spring Boot 创建并使用的 IoC 容器。

同时 Zuul 自带了负载均衡的功能。

Ribbon 负载均衡

什么是 Ribbon

Ribbon 是 Spring Cloud 的负载均衡组件，Ribbon 是 Netflix 发布的负载均衡器，Spring Cloud Ribbon 是基于 Netflix Ribbon 实现的，是一个用于对 HTTP 和 TCP 请求进行控制的负载均衡客户端。

Ribbon 的使用同样需要结合 Eureka Server，即需要将 Ribbon 在注册中心进行注册，为 Ribbon 注册完成网路信息之后，Ribbon 就可以基于某种负载均衡算法，如轮询，随机等自动帮助服务消费者去调用接口，除了 Ribbon 默认提供的这些负载均衡算法外，我们也可以为 Ribbon 实现自定义的负载均衡算法。在 Spring Cloud 中，当 Ribbon 和 Eureka 配合使用时，Ribbon 可以自动从 Eureka Server 获取服务提供者的地址列表，并基于负载均衡算法进行挑选（负载均衡）。

如何使用?

- pom.xml

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

- application.yml

```
server:
  port: 8040
spring:
  application:
    name: ribbon
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
```

- 创建启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

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

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```

`@LoadBalanced`：声明一个基于 Ribbon 的负载均衡。

- RibbonHandler

```
package com.southwind.controller;

import com.southwind.entity.Student;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

import java.util.Collection;

@RestController
@RequestMapping("/ribbon")
public class RibbonHandler {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/findAll")
    public Collection<Student> findAll(){
        return
restTemplate.getForObject("http://provider/student/findAll",Collection.class
);
    }

    @GetMapping("/findById/{id}")
```

```
    public Student findById(@PathVariable("id") long id){
        return
restTemplate.getForObject("http://provider/student/findById/{id}",Student.cl
ass,id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student){

 restTemplate.postForObject("http://provider/student/save",student,Student.c
lass);
    }

    @PutMapping("/update")
    public void update(@RequestBody Student student){
        restTemplate.put("http://provider/student/update",student);
    }

    @DeleteMapping("/deleteById/{id}")
    public void deleteById(@PathVariable("id") long id){
        restTemplate.delete("http://provider/student/deleteById/{id}",id);
    }
}
```

Feign

什么是 Feign?

与 Ribbon 一样，Feign 也是由 Netflix 提供的，Feign 是一个声明式，模版化的 HTTP 客户端，它简化了我们编写 Web 服务客户端的操作，可以帮助开发者更加快捷、方便地调用 HTTP API。Spring Cloud Feign 是基于 Netflix Feign 实现的，整合了 Spring Cloud Ribbon 和 Sprint Cloud Hystrix，使它具有可插拔、负载均衡、服务熔断等一系列便捷的功能，在 Spring Cloud 中使用 Feign 非常简单，只需要创建一个接口，同时在接口上添加相关注解即可完成服务提供方的接口绑定，简化了在使用 Spring Cloud Ribbon 时自行封装服务调用客户端的开发量，Feign 支持多种注解，包括 Feign 注解、JAX-RS 注解、Spring MVC 注解等，Spring Cloud 对 Feign 进行了优化，整合了 Ribbon 和 Eureka ，从而让 Feign 的使用更加方便。

Feign 是一种比 Ribbon 更加方便好用的 Web 服务客户端，二者的区别。

简单理解 Ribbon 是一个通用的 HTTP 客户端工具，Feign 是基于 Ribbon 实现的，同时它更加灵活，使用起来更加方便，Ribbon + RestTemplate 实现了服务调用的负载均衡，相比这种方式，使用 Feign 可以直接通过声明式接口的形式来调用服务。

Feign 的特点

- Feign 是一个声明式、模版化的 HTTP 客户端。
- 支持 Feign 注解、JAX-RS 注解、Spring MVC 注解。
- Feign 基于 Ribbon 实现，使用起来更加简便。

- Feign 集成了 Hystrix，具备服务熔断功能。

如何使用?

- pom.xml

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

- application.yml

```
server:
  port: 8050
spring:
  application:
    name: feign
eureka:
  client:
    service-url:
      defaulZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
```

- 启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.cloud.openfeign.FeignClient;

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

`@EnableFeignClients`：声明启用 Feign。

- 声明式接口

```
package com.southwind.feign;

import com.southwind.entity.Student;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@FeignClient(value = "provider")
public interface FeignProviderClient {

    @GetMapping("/student/index")
    public String index();

    @GetMapping("/student/findAll")
    public Collection<Student> findAll();

    @GetMapping("/student/findById/{id}")
    public Student findById(@PathVariable("id") long id);

    @PostMapping("/student/save")
    public void save(@RequestBody Student student);

    @PutMapping("/student/update")
    public void update(@RequestBody Student student);

    @DeleteMapping("/student/deleteById/{id}")
    public void deleteById(@PathVariable("id") long id);
}
```

- FeignHandler

```
package com.southwind.controller;

import com.southwind.entity.Student;
import com.southwind.feign.FeignProviderClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@RestController
@RequestMapping("/feign")
public class FeignHandler {
```

```
    @Autowired
    private FeignProviderClient feignProviderClient;

    @GetMapping("/findAll")
    public Collection<Student> findAll(){
        return feignProviderClient.findAll();
    }

    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id") long id){
        return feignProviderClient.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student){
        feignProviderClient.save(student);
    }

    @PutMapping("/update")
    public void update(@RequestBody Student student){
        feignProviderClient.update(student);
    }

    @DeleteMapping("/deleteById/{id}")
    public void deleteById(@PathVariable("id") long id){
        feignProviderClient.deleteById(id);
    }

    @GetMapping("/index")
    public String index(){
        return feignProviderClient.index();
    }
}
```