8 - Spring Cloud Gateway

Spring Cloud Gateway implements the matching rules of Route routes through Predicate assertions. To put it simply, Predicate is the judgment condition of route forwarding. The request will be forwarded to the specified service for processing only if it meets the condition of Predicate.

Let's use an example to demonstrate how Predicate is used.

(1) Create a Spring Boot module named micro-service-cloud-gateway-9527 under the parent project DataEngineSwarm, and introduce relevant dependencies in its pom.xml. The configuration is as follows.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.luxbp</groupId>
        <artifactId>DataEngineSwarm</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <groupId>com.luxbp</groupId>
    <artifactId>micro-service-cloud-gateway-9527</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>micro-service-cloud-gateway-9527</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--Note: don't introduce spring-boot-starter-web into Gateway, otherwise it will bring up a error-->
        <!-- Spring cloud gateway dependency-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>
        <!--Eureka -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

(2) In the classpath (/resources directory) of micro-service-cloud-gateway-9527, create a new configuration file application.yml, with the configuration content as follows.

```yaml
server:
  port: 9527
spring:
  application:
    name: microServiceCloudGateway
  cloud:
    gateway:
      routes:
        #hide micro-service-cloud-provider-dept-8001
        - id: provider_dept_list_routh
          uri: http://localhost:8001
          predicates:
            # Predicate
            - Path=/dept/list/**
            - Method=GET
eureka:
  instance:
    instance-id: micro-service-cloud-gateway-9527
    hostname: micro-service-cloud-gateway
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

(3) On the main startup class of micro-service-cloud-gateway-9527, use the @ EnableEurekaClient annotation to enable the Eureka client function

By default, Spring Cloud Gateway will create a dynamic route for forwarding based on the service list maintained in the service registry (such as Eureka Server) and the service name (spring. application. name) as the path, so as to realize the dynamic routing function.

We can modify the uri address of Route to the following form in the configuration file.

```
lb://service-name
```

Modify the configuration of application.yml in micro-service-cloud-gateway-9527 and use the microservice name in the registry to create a dynamic route for forwarding. The configuration is as follows.

```
server:
  port: 9527
spring:
  application:
    name: microServiceCloudGateway

  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
      routes:
        - id: provider_dept_list_routh
          uri: lb://MICROSERVICECLOUDPROVIDERDEPT # dynamic routing
          predicates:
            - Path=/dept/list/**
            - Method=GET
eureka:
  instance:
    instance-id: micro-service-cloud-gateway-9527
    hostname: micro-service-cloud-gateway
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

Filter

Usually, for security reasons, the services provided by the server often have certain verification logic, such as user login status verification, signature verification, etc.

In the microservice architecture, the system consists of multiple microservices, all of which require these verification logics. At this point, we can write these verification logics into the Filter filter of Spring Cloud Gateway.

Spring Cloud Gateway provides the following two types of filters for fine-grained control over requests and responses.

Pre:
  This filter can intercept and modify the request before it is forwarded to the microservice, such as parameter verification, permission verification, traffic monitoring, log output, and
  protocol conversion.
Post:
  This filter can intercept and reprocess the response after the microservice responds to the request, such as modifying the response content or response header, log output, traffic
  monitoring, etc.

Set up a GatewayFilter in the configuration file (such as application.yml) is similar to that of Predicate, and the format is as follows.

```
spring:
  cloud:
    gateway:
      routes:
        - id: xxxx
          uri: xxxx
          predicates:
            - Path=xxxx
          filters:
            - AddRequestParameter=X-Request-Id,1024 #The filter factory will add a pair of request headers to the matching request headers, the name is X-Request-Id and the value is 1024
            - PrefixPath=/dept #add /dept to the front of request path
            ……
```

Spring Cloud Gateway has built-in as many as 31 GatewayFilter. See more details at their document pages.

atfer modified filtyer, a re-start is needed.

Spring cloud also support developer to create a globalFilter. GlobalFilter is a global filter that acts on all routes, through which we can implement some unified business functions, such as authority authentication, IP access restrictions, etc. When a request is matched by a route, all GlobalFilters will be combined with the GatewayFilter configured by the route itself to form a filter chain.

Spring Cloud Gateway provides us with a variety of default GlobalFilter, such as global filters related to forwarding, routing, load balancing, etc. But in actual project development, we usually customize some of our own GlobalFilter global filters to meet our own business needs, and rarely use Spring Cloud Config to provide these default GlobalFilter directly.
For details about the default global filter, please refer to the Spring Cloud official website.
Under 9527 com.luxbp.filter package, there is MyGlobalFilter.java, it gives a show case of how to create a globalFilter. The code in it has been commented. Once un-comment the code, it will take effect.
The filter filts all the request urls, to examine if the request carry a param “uname”, intercept all the request without param, and will let pass all the request that has the “uname” param.