Spring Cloud Gateway实战

1 Spring Cloud Gateway的选择

1.1 什么是Spring Cloud Gateway

Spring Cloud Gateway(以下简称Gateway)是Spring生态系统中的一个API网关，它可以**处理HTTP请求和响应**，并**充当微服务架构中的入口点**。它是一个基于Spring Framework 5和Spring Boot 2.x的开源项目，采用响应式编程模型，旨在提供**高性能、高可靠性和易于使用的API网关**解决方案。

1.2 为什么选择Spring Cloud Gateway

Spring Cloud Gateway有以下几个主要的功能，笔者当时采用该组件的原因就是其灵活的路由以及过滤功能，还有就是它和springboot项目良好的契合性，可以算是简单易上手、无侵入性了。

* **请求路由**：当一个客户端发送请求到Spring Cloud Gateway时，Gateway会将请求路由到适当的微服务实例。Gateway会根据请求的URI和HTTP方法来匹配预定义的路由规则，并将请求转发给目标微服务。
* **过滤器**：Gateway还支持过滤器，开发人员可以编写过滤器来实现请求和响应的定制逻辑，例如鉴权、限流、日志记录等。
* **负载均衡**：Gateway可以通过Spring Cloud的负载均衡器来负载均衡请求，这可以确保请求被均匀分配到各个微服务实例中，从而提高系统的可用性和性能。
* **响应转换**：Gateway还支持响应转换，可以将微服务返回的响应转换成另一种格式，例如JSON或XML。
* **故障转移**：Gateway还支持故障转移，当微服务实例出现故障时，Gateway会自动将请求转发到备用实例，以确保系统的可用性。

2 Spring Cloud Gateway的使用

2.1 Spring Cloud Gateway在项目中的应用

> spring cloud gateway至少要求JDK8（函数式编程）；

新建一个maven项目，引入springboot的一些基础依赖，再引入关键依赖：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

> 笔者在使用maven打包时，最终打包出来的jar包不可用，在项目下加入以下编译配置就解决了

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <includeSystemScope>true</includeSystemScope>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

在使用Gateway时，有两种配置方式：配置文件配置和Java代码配置，在日常

的使用时，都是以配置文件配置为主（建议使用yaml文件）。

* 配置文件

```
server：
  port: 8000

spring:
  application:
    name: xxx-gateway
  cloud:
    gateway:
      routes: # 路由数组[路由 就是指定当请求满足什么条件的时候转到哪个微服务]
        - id: service-a # 当前路由的标识, 要求唯一
          uri: http://127.0.0.1:8080 # 请求要转发到的地址
          order: 1 # 路由的优先级,数字越小级别越高
          predicates: # 断言(就是路由转发要满足的条件)
            - Path=/a/**
            # 当请求路径满足Path指定的规则时,才进行路由转发
          filters:
             - StripPrefix=1
             #去掉一级前缀
        - id: service-bcd
          uri: http://127.0.0.1:8081
          predicates:
          - Path=/b/**,/c/**,/d/**
          # 多路径匹配时用英文逗号分隔
```

* Java Bean配置

```
import org.springframework.cloud.gateway.route.RouteLocator;
import
org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class GatewayConfig {
```

```
    @Bean
    public RouteLocator customRouteLocator(RouteLocatorBuilder
builder) {
        return builder.routes()
                .route("service-a", r -> r.order(1)
                        .path("/a/**")
                        .filters(f -> f.stripPrefix(1))
                        .uri("http://127.0.0.1:8080"))
                .route("service-bcd", r -> r.path("/b/**")
                        .or().path("/c/**")
                        .or().path("/d/**")
                        .uri("http://127.0.0.1:8081"))
                .build();
    }
}
```

> 上述配置文件和Java Bean的效果相同

Gateway提供了多种配置组合来完成对应的路由转发规则：

* uri：目标服务地址，可配置uri和lb://应用服务名称
* order：路由优先级，值越小优先级越高
* predicates：匹配条件，根据规则匹配是否请求该路由
* filters: 过滤规则，这个过滤包含多种过滤器
* StripPrefix=1，表示去掉几级前缀，例如上述配置中，我们访问 `http://127.0.0.1:8000/a/test` 将会转发到 `http://127.0.0.1:8080/test` 中。

其中predicates和filters是实现各种自定义转发的关键。

2.1.1 Route Predicate（路由谓词）

predicates拥有多种匹配条件，以下只列举常用的几种，详细的谓词介绍见[Route Predicate（路由谓词）工厂](http://cxyroad.com/"https://springdoc.cn/spring-cloud-gateway/#gateway-request-predicates-factories")。

时间谓词

* **After**，这个谓词匹配发生在指定日期时间之后的请求。
* **Before**，这个谓词匹配发生在指定日期时间之前的请求。
* **Between**，这个谓词匹配发生在两个时间之间的请求。

```
spring:
  cloud:
    gateway:
      routes:
      - id: after_route
        uri: https://example.org
        predicates:
        - After=2017-01-20
        # 匹配2017-01-20之后的请求
      - id: before_route
        uri: https://example.org
        predicates:
        - Before=2017-01-20
        # 匹配2017-01-20之前的请求
      - id: between_route
        uri: https://example.org
        predicates:
        - Between=2017-01-20, 2017-01-21
        # 匹配2017-01-20到2017-01-21之间的请求
```

Cookie

这个谓词匹配具有给定名称且其值符合正则表达式的cookie。

```
spring:
  cloud:
    gateway:
      routes:
        - id: cookie_route
          uri: https://example.org
          predicates:
            - Cookie=username,^[A-Za-z_]
```

```
            # 匹配cookie中要包含名为username且值必须以字母开头的参数
```

Header

这个谓词匹配具有给定名称且其值与正则表达式相匹配的请求头。

```
spring:
  cloud:
    gateway:
      routes:
      - id: header_route
        uri: https://example.org
        predicates:
        - Header=X-Request-Id, \d+
        # 匹配请求头中X-Request-Id的值为数字的请求
```

Host

这个谓词匹配符合该正则的主机名称的请求。

```
spring:
  cloud:
    gateway:
      routes:
      - id: host_route
        uri: https://example.org
        predicates:
        - Host=**.somehost.org
        # 匹配来自somehost.org子域名的请求

```

Method

这个谓词匹配指定请求方式的请求。

```
spring:
  cloud:
    gateway:
      routes:
      - id: method_route
        uri: https://example.org
        predicates:
        - Method=GET,POST
        # 匹配GET、POST请求
```

Path

这个谓词匹配指定请求路径的请求。

```
spring:
  cloud:
    gateway:
      routes:
      - id: path_route
        uri: https://example.org
        predicates:
        - Path=/a/*
        # 匹配所有请求路径/a/*的请求，例如/a/test
```

2.1.2 Route filters（路由过滤器）

Gateway 支持多种内置的过滤器，可以对请求进行多种处理。以下也只列举几种常用的filter，详细的filter见[GatewayFilter 工厂](http://cxyroad.com/"https://springdoc.cn/spring-cloud-gateway/#gatewayfilter-%E5%B7%A5%E5%8E%82")。

AddRequestHeader

该过滤器可以在转发请求时新增或替换指定请求头信息。

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_header_route
        uri: https://example.org
        filters:
        - AddRequestHeader=X-Request-color, blue
        # 该过滤器将 X-Request-color:blue 请求头附加在请求中
```

AddRequestParameter

该过滤器可以给转发的请求添加指定的查询参数。

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_parameter_route
        uri: https://example.org
        filters:
        - AddRequestParameter=color, blue
        # 该过滤器添加了 color=blue 的查询参数
```

AddResponseHeader

与AddRequestHeader相对应，该过滤器可以在接收请求的返回信息时新增或替换指定请求头信息。

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_response_header_route
        uri: https://example.org
        filters:
        - AddResponseHeader=X-Response-color, Blue
        # 该过滤器将 X-Request-color:blue 请求头附加在请求中


```

PrefixPath

该过滤器可以将所有转发的请求加上指定的前缀。

```
spring:
  cloud:
    gateway:
      routes:
      - id: prefixpath_route
        uri: https://example.org
        filters:
        - PrefixPath=/prefix
        # 经过该路由的请求都会加上 /prefix 前缀，例如原来为 /hello 的请求会转发到 /prefix/hello


```

RedirectTo

该过滤器可以将请求重定向至指定URL并附带300系列的重定向HTTP状态码。

```
spring:
  cloud:
```

```
    gateway:
      routes:
      - id: prefixpath_route
        uri: https://example.org
        filters:
        - RedirectTo=302, https://redirect.org
        # 该过滤器将发送一个带有 Location:https://redirect.org header的
302 状态码的响应，以执行重定向。
```

RewritePath

该过滤器可以将请求路径重写。

> 注意，由于YAML的规范，`$` 应该写成 `$\`。

```
spring:
  cloud:
    gateway:
      routes:
      - id: rewritepath_route
        uri: https://example.org
        predicates:
        - Path=/red/**
        filters:
        - RewritePath=/red/?(?<segment>.*), /$\{segment}
        # 请求为 /red/blue 的情况下会变成 /blue
```

StripPrefix

该过滤器可以去除请求路径中指定层级的前缀。

```
spring:
```

```
  cloud:
    gateway:
      routes:
      - id: stripprefix_route
        uri: https://example.org
        predicates:
        - Path=/name/**
        filters:
        - StripPrefix=1
        # 经过网关的请求 /name/aa/bb 会变成 /aa/bb，如果
StripPrefix=2，请求会变成 /bb，依此类推


```

Retry

该过滤器提供重试机制。该过滤器拥有以下参数：

* `retries`: 重试次数。默认为3。
* `statuses`: 需要重试的HTTP状态代码，用
 `org.springframework.http.HttpStatus` 表示。
* `methods`: 重试的HTTP方法，用
 `org.springframework.http.HttpMethod` 来表示。默认是GET请求。
* `series`: 要重试的状态代码系列，用
 `org.springframework.http.HttpStatus.Series` 表示。默认是5XX系列状态码
。
* `exceptions`: 要重试的异常类型。默认是`IOException` 和
 `TimeoutException`。
* `backoff`: 为重试配置的指数式backoff。重试是在 `backoff` 间隔
 `firstBackoff * (factor ^ n)` 之后进行的，其中 `n` 是迭代次数。如果配置了
 `maxBackoff`，应用的最大 backoff 时间被限制在 `maxBackoff`。如果
 `basedOnPreviousValue` 为 `true`，则通过使用 `prevBackoff * factor` 来
计算backoff。默认未启用。

```
spring:
  cloud:
    gateway:
      routes:
      - id: retry_test
        uri: http://localhost:8080/flakey
        predicates:
        - Host=*.retry.com
```

```
        filters:
        - name: Retry
          args:
            retries: 3
            statuses: 500
            methods: GET,POST
            backoff:
              firstBackoff: 10ms
              maxBackoff: 50ms
              factor: 2
              basedOnPreviousValue: false
              # 请求发生500错误时，在10ms、40ms后重试两次


```

全局过滤器 default-filters

如果要添加指定的filter并将其应用于所有的路由，可以使用 `spring.cloud.gateway.default-filters`。

```
spring:
  cloud:
    gateway:
      default-filters:
      - AddResponseHeader=X-Response-Default-Color, Blue
      - PrefixPath=/aa
      # 所有请求会添加 /aa 的请求前缀，还会在返回报文的请求头处加入 X-Response-Default-Color:Blue 的请求头参数


```

2.1.3 SSL证书配置

在使用Gateway进行路由转发时，不可避免的会涉及到需要SSL证书的情况，Gateway可以通过遵循通常的 Spring server configuration 来监听 HTTPS 请求。

```
```

```
server:
  ssl:
    enabled: true
    key-alias: scg
    key-store-password: scg1234
    key-store: classpath:scg-keystore.p12
    key-store-type: PKCS12
```

如果需要转发到使用HTTPS的下游，你可以将Gateway配置为信任所有下游的证书。

> 只建议在内网环境且下游是后端服务时使用。

```
spring:
  cloud:
    gateway:
      httpclient:
        ssl:
          useInsecureTrustManager: true
```

在生产等对安全要求较高的环境中，可以配置指定网关所信任的证书。

```
spring:
  cloud:
    gateway:
      httpclient:
        ssl:
          trustedX509Certificates:
          - cert1.pem
          - cert2.pem
```

如果 Spring Cloud Gateway 没有配置受信任的证书，就会使用默认的 trust store（即启动项目的JDK中配置的证书，可以通过设置 javax.net.ssl.trustStore 系统属性来配置它）。

2.1.4 CORS跨域配置

Global CORS 配置，也就是对所有向Gateway发起的请求的跨域配置

```
spring:
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowedOrigins: "https://example.org"
            allowedMethods:
            - GET
            # 允许来自 https://example.org 的GET请求
```

2.1.5 网关日志记录

在本文中，笔者采用重写 GlobalFilter 的方式来实现日志记录，这也是官方提供的一种自定义 Filter 的方式。

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.http.server.reactive.ServerHttpRequest;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;

@Component
@Slf4j
```

```
public class LoggingFilter implements GlobalFilter {

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
GatewayFilterChain chain) {
        ServerHttpRequest request = exchange.getRequest();

        // 获取客户端IP地址
        String ip = request.getRemoteAddress() != null ?
request.getRemoteAddress().getAddress().getHostAddress() :
"Unknown";

        // 记录IP和请求行
        log.info("Request from IP: {}, Method: {}, Path: {}", ip,
request.getMethodValue(), request.getPath());

        String realIp = request.getHeaders().getFirst("X-Forwarded-For");
        log.info("X-Forwarded-For: {}", realIp);

        // 继续 filter 链的处理，如果没有这步操作会导致 Gateway 转发失效
        return chain.filter(exchange);
    }
}

```

> 上述代码只是简单的记录了一下请求的来源、请求类型、请求路径以及可能经过nginx转发的真实IP。需要注意的是如果想要记录POST、PUT等请求body里面的内容时，需要额外实现数据流重复读取的机制，否则会导致下游服务无法读取到body信息。

3 性能影响

在实际项目中，大多会有一层Nginx进行的路由转发，如果再加上Gateway的转发消耗的话，可能会对原有的接口性能造成影响。

参考以下大佬做的测试，对实际的请求还是有一定影响的，但是相比于Netflix的zuul，已经好很多了。读者可以参考自己的业务需求再决定是否使用Spring Cloud Gateway。

服务	平均时延(ms)	RPS(每秒请求数)	性能损失
Origin	2.94	33014.70	0
Netflix Zuul	12.39	13175.21	60.09%
Spring Cloud Gateway	4.95	21685.14	34.32%

本文参考
====

* [手把手快速入门springcloud之Gateway网关](http://cxyroad.com/”https://zhuanlan.zhihu.com/p/613486761”)
* [spring gateway性能损耗 -- 记一次性能测试](http://cxyroad.com/”https://blog.csdn.net/ilyz5609/article/details/109514894”)
* [Spring Cloud Gateway（一）为什么用网关、能做什么、为什么选择Gateway、谓词工厂、过滤器配置](http://cxyroad.com/”https://blog.csdn.net/Extraordinarylife/article/details/115168526”)
* [Spring Cloud Gateway 中文文档](http://cxyroad.com/”https://springdoc.cn/spring-cloud-gateway/”)