微服务项目-京锋购 02 - Spring Cloud Gateway

京锋购

Spring Cloud Gateway

工作原理图

网关概述

入门

- 创建项目
- 编写规则
- 测试

网关路由规则详解

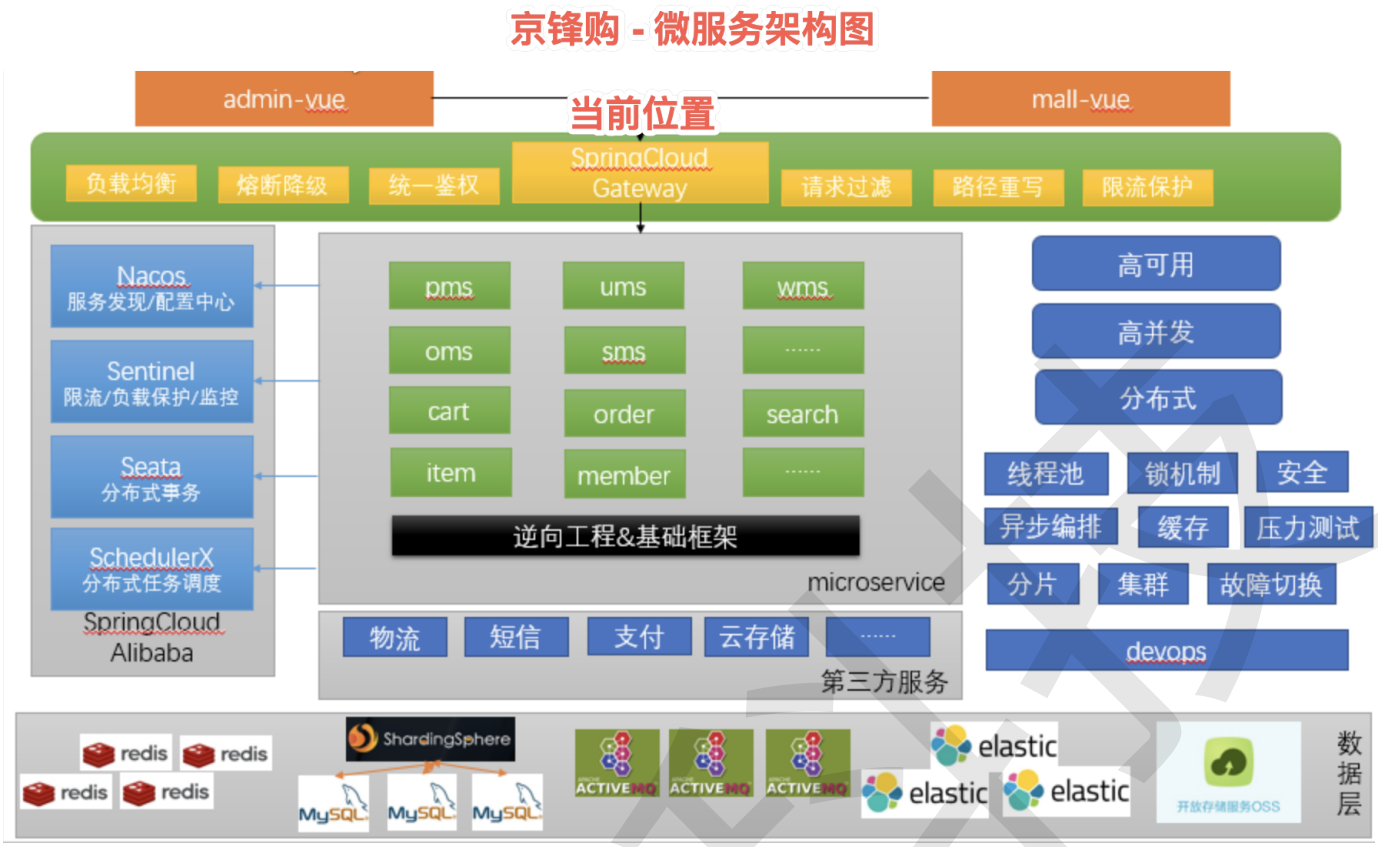
- 基本概念
- 路由断言工厂
- 过滤器工厂

面向服务的路由

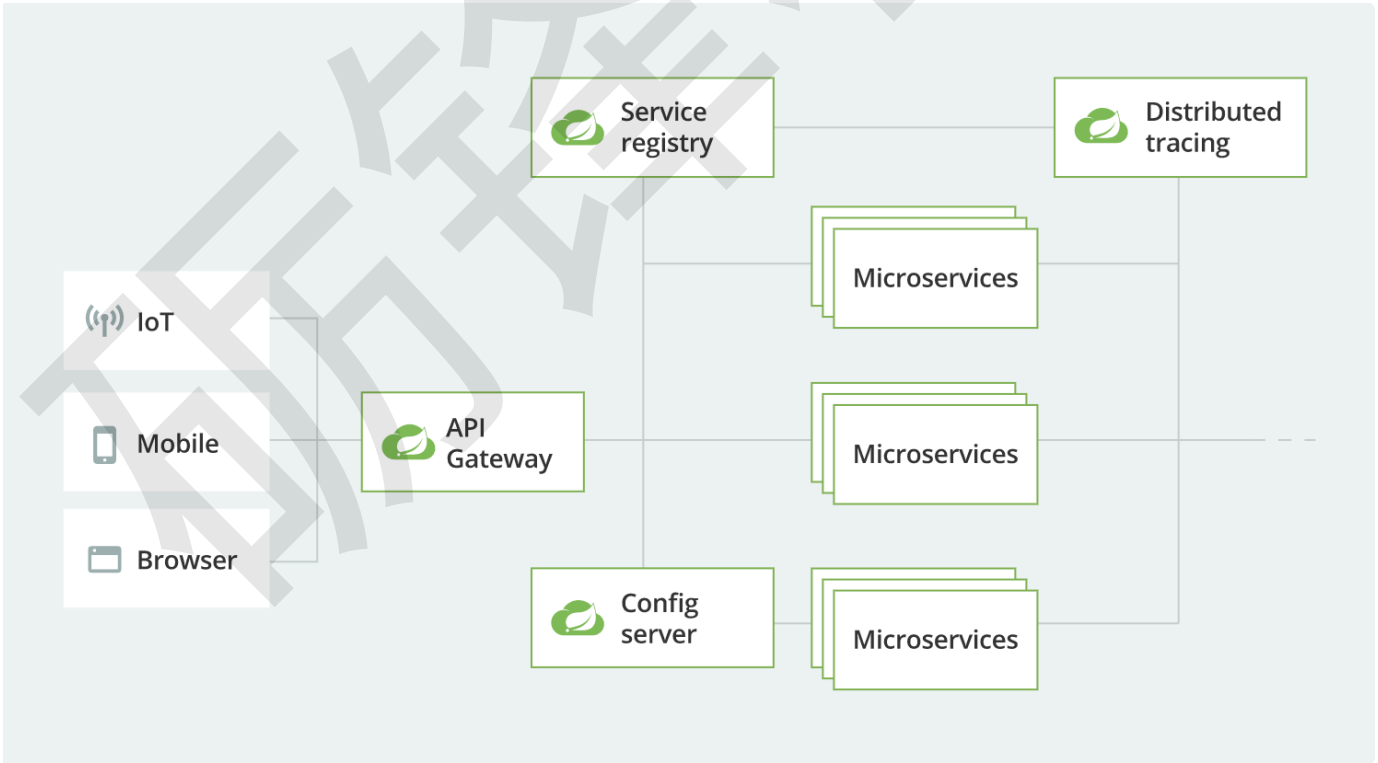
- 把网关服务注册到 Nacos
- 修改配置，通过服务名路由
- 测试

京锋购

京东 X 砺锋 = 京锋购商城

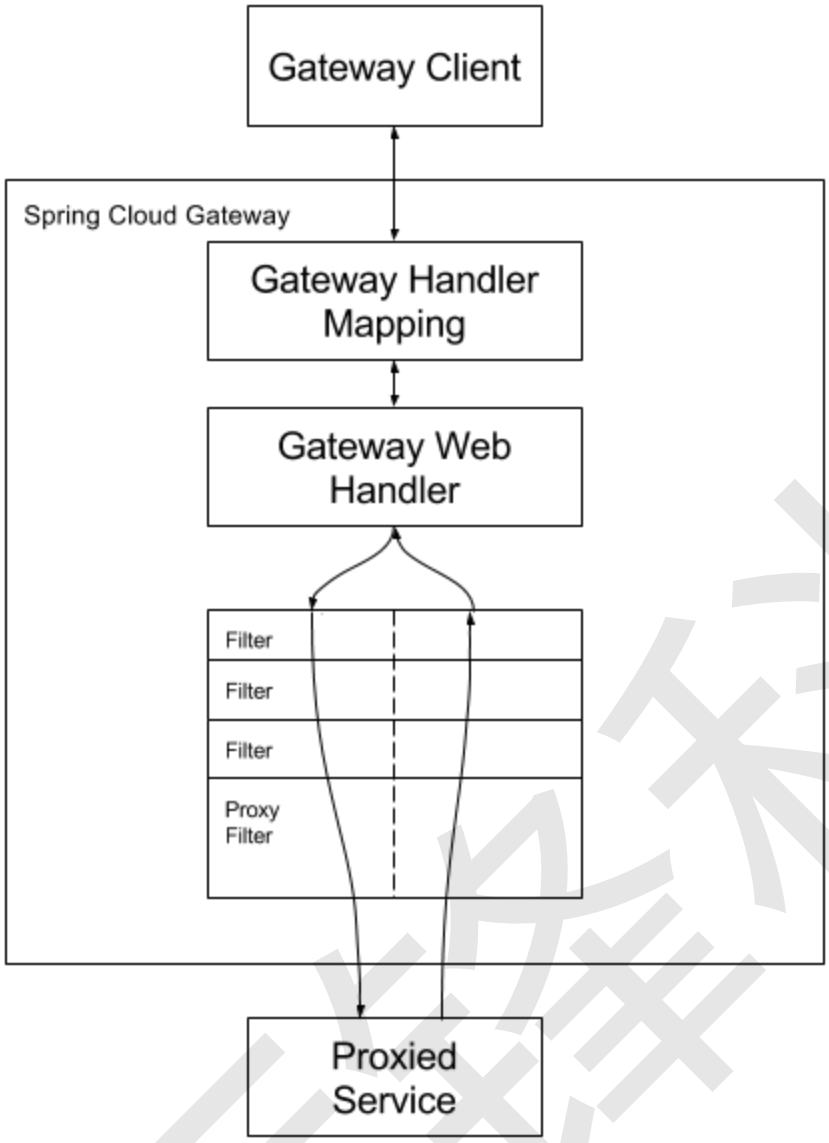


Spring Cloud Gateway



服务网关，官方文档：https://spring.io/projects/spring-cloud-gateway

工作原理图



网关概述

API 网关出现的原因是微服务架构的出现，不同的微服务一般会有不同的网络地址，而外部客户端可能需要调用多个服务的接口才能完成一个业务需求。

如果让客户端直接与各个微服务通信，会有以下的问题：

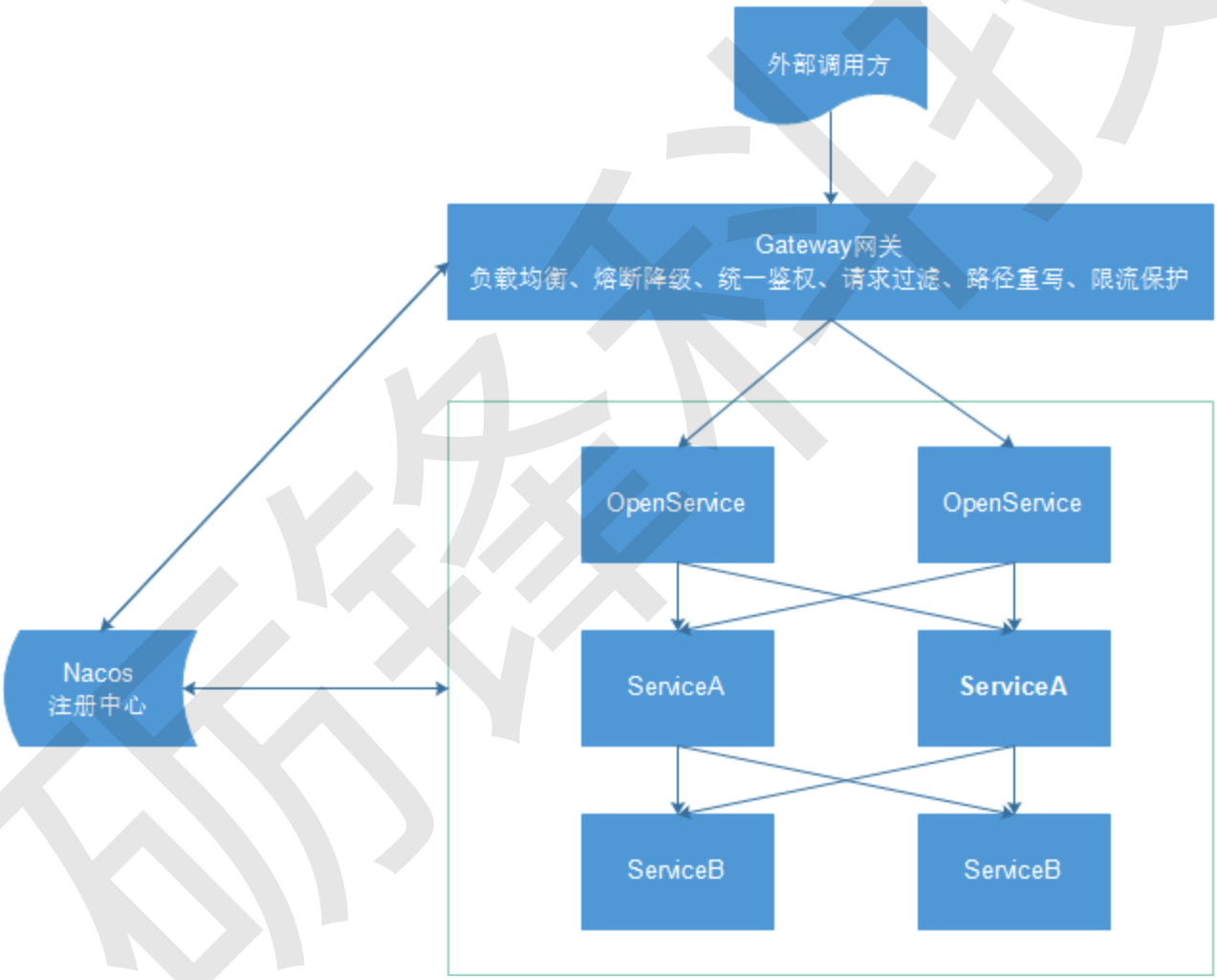
1. 破坏了服务无状态特点。
    a. 为了保证对外服务的安全性，我们需要实现对服务访问的权限控制，而开放服务的权限控制机制将会贯穿并污染整个开放服务的业务逻辑，这会带来的最直接问题是，破坏了服务集群中 RESTful API 无状态的特点。

b. 从具体开发和测试的角度来说，在工作中除了要考虑实际的业务逻辑之外，还需要额外考虑对接口访问的控制处理。

2. 无法直接复用既有接口。
   a. 当我们需要对一个即有的集群内访问接口，实现外部服务访问时，我们不得不通过在原有接口上增加校验逻辑，或增加一个代理调用来实现权限控制，无法直接复用原有的接口。

以上的问题可以借助 API 网关解决。API 网关是介于客户端和服务器端之间的中间层，所有的外部请求都会先经过 API 网关这一层。也就是说，API 的实现方面更多的考虑业务逻辑，而安全、性能、监控可以交由 API 网关来做，这样既提高业务灵活性又不缺安全性。

典型的架构图如图所示：



入门

创建项目

通过 idea 创建 maven 项目，需要引入依赖：

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

编写规则

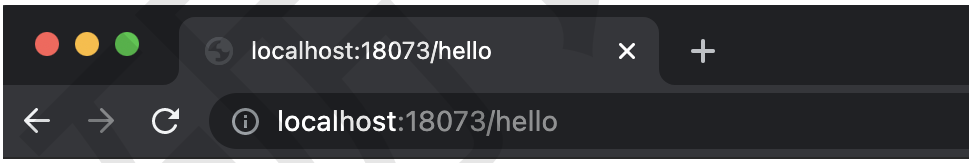
为了演示路由到不同服务，这里把消费者和生产者都配置在网关中

```yaml
server:
  port: 18073

spring:
  cloud:
    gateway:
      routes:
        - id: nacos-consumer
          uri: http://127.0.0.1:18072
          predicates:
            - Path=/hi
        - id: nacos-provider
          uri: http://127.0.0.1:18071
          predicates:
            - Path=/hello
```

测试

通过网关路径访问消费者或者生产者。



hello nacos-dev, redis-url=redis//xxxx, jdbc-url=jdbc//xxx

网关路由规则详解

基本概念

- Route
  - 路由网关的基本构建块。

    - 它由ID，目的URI，断言（Predicate）集合和过滤器（filter）集合组成。如果断言聚合为真，则匹配该路由。
- Predicate
    - 这是一个 Java 8 函数式断言。
    - 允许开发人员匹配来自HTTP请求的任何内容，例如请求头或参数。
- 过滤器
    - 过滤器可以在发送下游请求之前或之后修改请求和响应。

总的来说，路由根据断言进行匹配，匹配成功就会转发请求给URI，在转发请求之前或者之后可以添加过滤器。

路由断言工厂

Spring Cloud Gateway包含许多内置的 Route Predicate 工厂。所有这些断言都匹配 HTTP 请求的不同属性。多路由断言工厂通过 and 组合。

官方提供的路由工厂：

5. Route Predicate Factories

5.1. The After Route Predicate Factory

5.2. The Before Route Predicate Factory

5.3. The Between Route Predicate Factory

5.4. The Cookie Route Predicate Factory

5.5. The Header Route Predicate Factory

5.6. The Host Route Predicate Factory

5.7. The Method Route Predicate Factory

5.8. The Path Route Predicate Factory

5.9. The Query Route Predicate Factory

5.10. The RemoteAddr Route Predicate Factory

5.11. The Weight Route Predicate Factory

有人整理了如下谓语和工厂对应表：

重点掌握的时候请求路径路由断言的配置方式：

5.8. The Path Route Predicate Factory

The `Path` Route Predicate Factory takes two parameters: a list of Spring `PathMatcher` `patterns` and an optional flag called `matchTrailingSlash` (defaults to `true`). The following example configures a path route predicate:

Example 8. application.yml

```yaml
spring:
  cloud:
    gateway:
      routes:
      - id: path_route
        uri: https://example.org
        predicates:
        - Path=/red/{segment},/blue/{segment}
```

This route matches if the request path was, for example: `/red/1` or `/red/1/` or `/red/blue` or `/blue/green`.

If `matchTrailingSlash` is set to `false`, then request path `/red/1/` will not be matched.

This predicate extracts the URI template variables (such as `segment`, defined in the preceding example) as a map of names and values and places it in the `ServerWebExchange.getAttributes()` with a key defined in `ServerWebExchangeUtils.URI_TEMPLATE_VARIABLES_ATTRIBUTE`. Those values are then available for use by `GatewayFilter` factories

A utility method (called `get`) is available to make access to these variables easier. The following example shows how to use the `get` method:

```java
Map<String, String> uriVariables = ServerWebExchangeUtils.getPathPredicateVariables(exchange);

String segment = uriVariables.get("segment");
```

这个路由匹配以 /red 或者 /blue 开头的路径，转发到 https:example.org。例如 `/red/1` or `/red/1/` or `/red/blue` or /blue/green.

过滤器工厂

路由过滤器允许以某种方式修改传入的 HTTP 请求或传出的 HTTP 响应。

路径过滤器的范围限定为特定路由。Spring Cloud Gateway 包含许多内置的 GatewayFilter 工厂，而且不断新增，目前有 31 个。

6. GatewayFilter Factories

6.1. The AddRequestHeader GatewayFilter Factory

6.2. The AddRequestParameter GatewayFilter Factory

6.3. The AddResponseHeader GatewayFilter Factory

6.4. The DedupeResponseHeader GatewayFilter Factory

6.5. Spring Cloud CircuitBreaker GatewayFilter Factory

6.6. The FallbackHeaders GatewayFilter Factory

6.7. The MapRequestHeader GatewayFilter Factory

6.8. The PrefixPath GatewayFilter Factory

6.9. The PreserveHostHeader GatewayFilter Factory

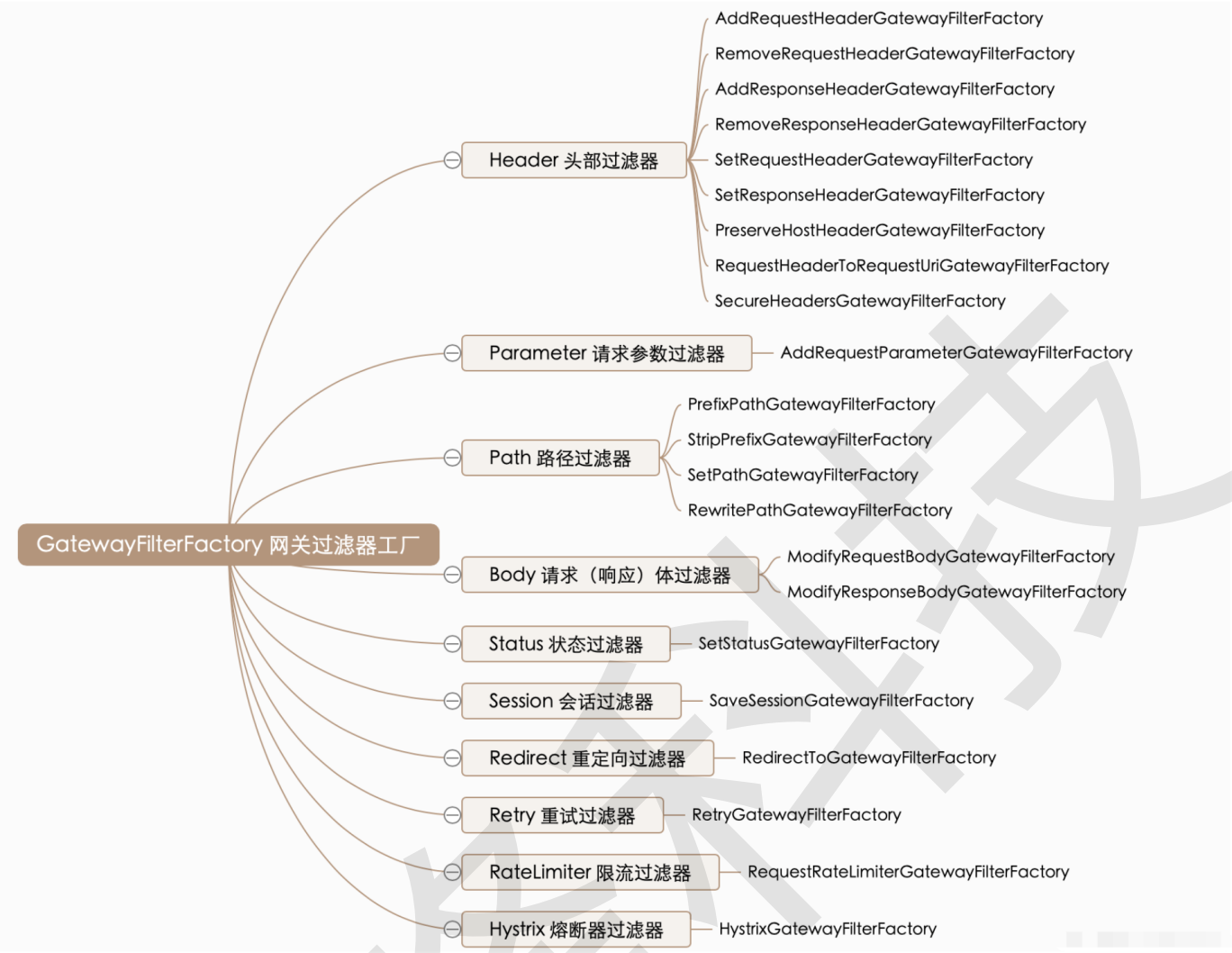
6.10. The RequestRateLimiter GatewayFilter Factory

6.11. The RedirectTo GatewayFilter Factory

6.12. The RemoveRequestHeader GatewayFilter Factory

6.13. RemoveResponseHeader GatewayFilter Factory

根据过滤器工厂的用途来划分，可以分为以下几种：Header、Parameter、Path、Body、Status、Session、Redirect、Retry、RateLimiter和Hystrix



这里重点掌握 PrefixPath GatewayFilter Factory

6.15. The `RewritePath` `GatewayFilter` Factory

The `RewritePath` `GatewayFilter` factory takes a path `regexp` parameter and a `replacement` parameter. This uses Java regular expressions for a flexible way to rewrite the request path. The following listing configures a `RewritePath` `GatewayFilter`:

Example 39. application.yml

```yaml
spring:
  cloud:
    gateway:
      routes:
      - id: rewritepath_route
        uri: https://example.org
        predicates:
        - Path=/red/**
        filters:
        - RewritePath=/red/?(?<segment>.*), /$\{segment}
```

For a request path of `/red/blue`, this sets the path to `/blue` before making the downstream request. Note that the `$` should be replaced with `$\` because of the YAML specification.

上面的配置中，所有的 /red/** 开始的路径都会命中配置的 router，并执行过滤器的逻辑，在案例中配置了 RewritePath 过滤器工厂，此工厂将 /red/(?.*)重写为{segment}，然后转发到 https://example.org。

比如在网页上请求localhost:8090/red/forezp，此时会将请求转发到 http://example.org/forezp 的页面

在开发中由于所有微服务的访问都要经过网关，为了区分不同的微服务，通常会在路径前加上一个标识，例如：访问服务提供方：`http://localhost:18090/provider/hello` ；访问服务消费方：`http://localhost:18090/consumer/hi` 如果不重写地址，直接转发的话，转发后的路径为：`http://localhost:18070/provider/hello`和`http://localhost:18080/consumer/hi`明显多了一个provider或者consumer，导致转发失败。

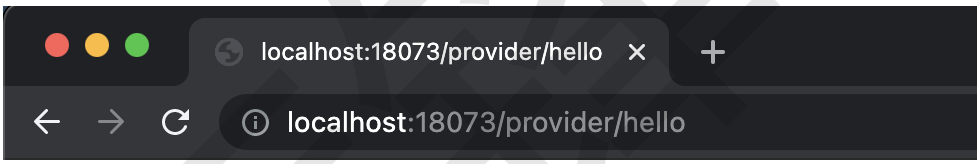
比如，我们用上了路径重写，配置如下：

```yaml
server:
  port: 18073
spring:
  cloud:
    gateway:
      routes:
        - id: nacos-consumer
          uri: http://127.0.0.1:18072
          predicates:
            - Path=/consumer/**
          filters:
            - RewritePath=/consumer/(?<segment>.*),/$\{segment}
        - id: nacos-provider
          uri: http://127.0.0.1:18071
          predicates:
            - Path=/provider/**
          filters:
            - RewritePath=/provider/(?<segment>.*),/$\{segment}
```

注意Path=/consumer/** 及 Path=/provider/**的变化。

此时，后台没有 /provider/hello 这个接口，只有 /hello 这个接口。

测试访问 /provider/hello 也可以成功，真正访问的是 /hello：



hello nacos-dev, redis-url=redis//xxxx, jdbc-url=jdbc//xxx

面向服务的路由

当前，我们使用 ip 地址访问服务，无法做到负载均衡，如果要做到负载均衡，则必须把网关工程注册到 nacos 注册中心，然后通过服务名访问。

把网关服务注册到 Nacos

1. 引入 Nacos 的相关依赖：

```xml

<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>

<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>

<!-- SpringCloud的依赖 -->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Hoxton.SR9</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>com.alibaba.cloud</groupId>
      <artifactId>spring-cloud-alibaba-dependencies</artifactId>
      <version>2.2.6.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

配置 nacos 服务地址及服务名

bootstrap.yml

```yaml
spring:
  application:
    name: gateway-demo
  cloud:
    nacos:
      config:
        server-addr: 127.0.0.1:8848
```

application.yml中的配置：

YAML 复制代码

```yaml
server:
  port: 18073
spring:
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
    gateway:
      routes:
        - id: nacos-consumer
          uri: http://127.0.0.1:18072
          predicates:
            - Path=/consumer/**
          filters:
            - RewritePath=/consumer/(?<segment>.*),/$\{segment}
        - id: nacos-provider
          uri: http://127.0.0.1:18071
          predicates:
            - Path=/provider/**
          filters:
            - RewritePath=/provider/(?<segment>.*),/$\{segment}
```

注入 nacos：

Java 复制代码

```java
package com.rushuni.gatewaydemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

/**
 * @author rushuni
 * @date 2021/8/20
 */
@SpringBootApplication
@EnableDiscoveryClient
public class GatewayDemoApplication {

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

}
```

修改配置，通过服务名路由

YAML 复制代码

```yaml
server:
  port: 18073
spring:
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
    gateway:
      routes:
        - id: nacos-consumer
          uri: lb://nacos-consumer
          predicates:
            - Path=/consumer/**
          filters:
            - RewritePath=/consumer/(?<segment>.*),/$\{segment}
        - id: nacos-provider
          uri: lb://nacos-provider
          predicates:
            - Path=/provider/**
          filters:
            - RewritePath=/provider/(?<segment>.*),/$\{segment}
```

语法：lb://服务名

lb：LoadBalance，代表访问方式为负载均衡。

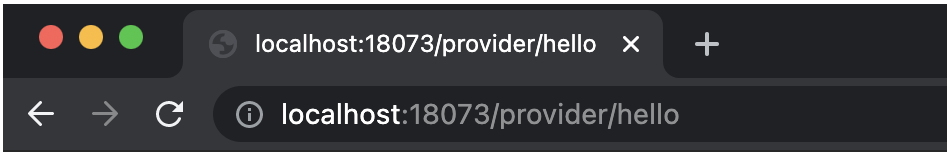
服务名取决于 Nacos 的服务列表中的服务名

服务名
nacos-consumer
nacos-provider

启动后：

```
2021-08-20 10:24:59.396  INFO 83999 --- [ctor-http-nio-2] c.netflix.loadbalancer.BaseLoadBalancer  : Client: nacos-provider instantiated a
↘LoadBalancer: DynamicServerListLoadBalancer:{NFLoadBalancer:name=nacos-provider,current list of Servers=[],Load balancer stats=Zone stats: {},Server
↘stats: []}ServerList:null
2021-08-20 10:24:59.401  INFO 83999 --- [ctor-http-nio-2] c.n.l.DynamicServerListLoadBalancer      : Using serverListUpdater PollingServerListUpdater
2021-08-20 10:24:59.414  INFO 83999 --- [ctor-http-nio-2] c.netflix.config.ChainedDynamicProperty  : Flipping property: nacos-provider.ribbon
  .ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647
2021-08-20 10:24:59.416  INFO 83999 --- [ctor-http-nio-2] c.n.l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client
  nacos-provider initialized: DynamicServerListLoadBalancer:{NFLoadBalancer:name=nacos-provider,current list of Servers=[192.168.1.37:18071],Load
  balancer stats=Zone stats: {unknown=[Zone:unknown;	Instance count:1;	Active connections count: 0;	Circuit breaker tripped count: 0;	Active
  connections per server: 0.0;]
```

测试



hello nacos-dev, redis-url=redis//xxxx, jdbc-url=jdbc//xxx