

1. 概述简介

1.1. 官网

1.2. 是什么

1.2.1. 概述

1.2.2. 一句话

1.3. 能干啥

1.4. 微服务架构中网关在哪

1.5. 有Zuul了怎么又出来Gateway

1.5.1. 为什么选择Gateway

1.5.2. Zuul 1.x模型

1.5.3. Gateway 模型

1.5.3.1. WebFlux 是什么

1.5.3.2. 说明

2. 三大核心概念

2.1. Route(路由)

2.2. Predicate(断言)

2.3. Filter(过滤)

2.4. 总结

3. Gateway工作流程

3.1. 官网总结

3.2. 核心逻辑

4. 入门配置

4.1. 新建Module

4.2. POM

4.3. YML

4.4. 主启动类

4.5. 9527网关如何做路由映射呢

4.6. YML新增网关配置

4.7. 测试

4.8. YML配置说明

4.8.1. 在YML配置文件中配置

4.8.2. 代码中注入RouteLocator的Bean

4.8.2.1. 官网案例

4.8.2.2. 实例演示

4.8.2.2.1. 业务需求

4.8.2.2.2. 编码实现

4.8.2.2.3. 测试

4.9. 注意事项

5. 通过服务名实现动态路由

5.1. 说明

5.2. POM

5.3. YML

5.4. 测试

6. Predict

6.1. 是什么

6.2. Route Predicate Factories

6.3. 常用的Route Predicate

6.3.1. After Route Predicate

6.3.1.1. 官方使用说明

6.3.1.2. 实例演示

6.3.1.2.1. 获取时间字符串

6.3.1.2.2. 修改YML

6.3.1.2.3. 测试

6.3.2. Before Route Predicate

6.3.2.1. 官方使用说明

6.3.2.2. 实例演示

6.3.2.2.1. 修改YML

6.3.2.2.2. 测试

6.3.3. Between Route Predicate

6.3.3.1. 官方使用说明

6.3.3.2. 实例演示

6.3.3.2.1. 修改YML

6.3.3.2.2. 测试

6.3.4. Cookie Route Predicate

6.3.4.1. 官方使用说明

6.3.4.2. 实例演示

6.3.4.2.1. 修改YML

6.3.4.2.2. 测试

6.3.5. Header Route Predicate

6.3.5.1. 官方使用说明

6.3.5.2. 实例演示

6.3.5.2.1. 修改YML

6.3.5.2.2. 测试

6.3.6. Host Route Predicate

6.3.6.1. 官方使用说明

6.3.6.2. 实例演示

6.3.6.2.1. 修改YML

6.3.6.2.2. 测试

6.3.7. Method Route Predicate

6.3.7.1. 官方使用说明

6.3.7.2. 实例演示

6.3.7.2.1. 修改YML

6.3.7.2.2. 测试

6.3.8. Path Route Predicate

6.3.8.1. 官方使用说明

6.3.8.2. 实例演示

6.3.8.2.1. 修改YML

6.3.8.2.2. 测试

6.3.9. Query Route Predicate

6.3.9.1. 官方使用说明

6.3.9.2. 实例演示

6.3.9.2.1. 修改YML

6.3.9.2.2. 测试

6.3.10. RemoteAddr Route Predicate

6.3.10.1. 官方使用说明

6.3.11. Weight Route Predicate

6.3.11.1. 官方使用说明

6.3.12. 小结

7. Filter的使用

7.1. 是什么

7.2. SpringCloud Gateway的filter

7.2.1. 生命周期(只有2个)

7.2.2. 种类(只有2类)

7.2.2.1. GatewayFilter

7.2.2.2. GlobalFilter

7.3. 常用的GatewayFilter

7.3.1. AddRequestParameter, AddRequestHeader

7.3.1.1. 修改cloud-gateway-gateway9527模块的yml配置文件

7.3.1.2. 修改cloud-provider-payment8001模块的PaymentController

7.3.1.3. 测试

7.4. 自定义过滤器

7.4.1. 两个主要接口介绍

7.4.2. 能干嘛

7.4.3. 案例代码

7.4.4. 测试

1. 概述简介

1.1. 官网

上一代zuul 1.x: <https://github.com/Netflix/zuul/wiki>

当前gateway: <https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.1.RELEASE/reference/html/>

1.2. 是什么

Reactive Stack
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.

Servlet Stack
Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.

Reactive Streams Adapters
Netty, Servlet 3.1+ Containers

Servlet API
Servlet Containers

Spring Security Reactive
Spring WebFlux

Spring Security
Spring MVC

Spring Data Reactive Repositories
Mongo, Cassandra, Redis, Couchbase

Spring Data Repositories
JDBC, JPA, NoSQL

Developers are constantly challenged with choosing the most effective runtime, programming model, and architecture for their application's requirements and team's skill set. For example, some use cases are best handled by a technology stack based on synchronous blocking I/O architecture, whereas others would be better served by an asynchronous, nonblocking stack built on the reactive design principles described in the [Reactive Streams Specification](#).

Reactive Spring represents a platform-wide initiative to deliver reactive support at every level of the development stack: web, security, data, messaging, etc. Spring Framework 5 delivers on this vision by providing a new reactive web stack called Spring WebFlux, which is offered side by side with the traditional Spring MVC web stack. The choice is yours!

[WebFlux Reference Documentation](#)

Spring Cloud Gateway

2.2.1.RELEASE

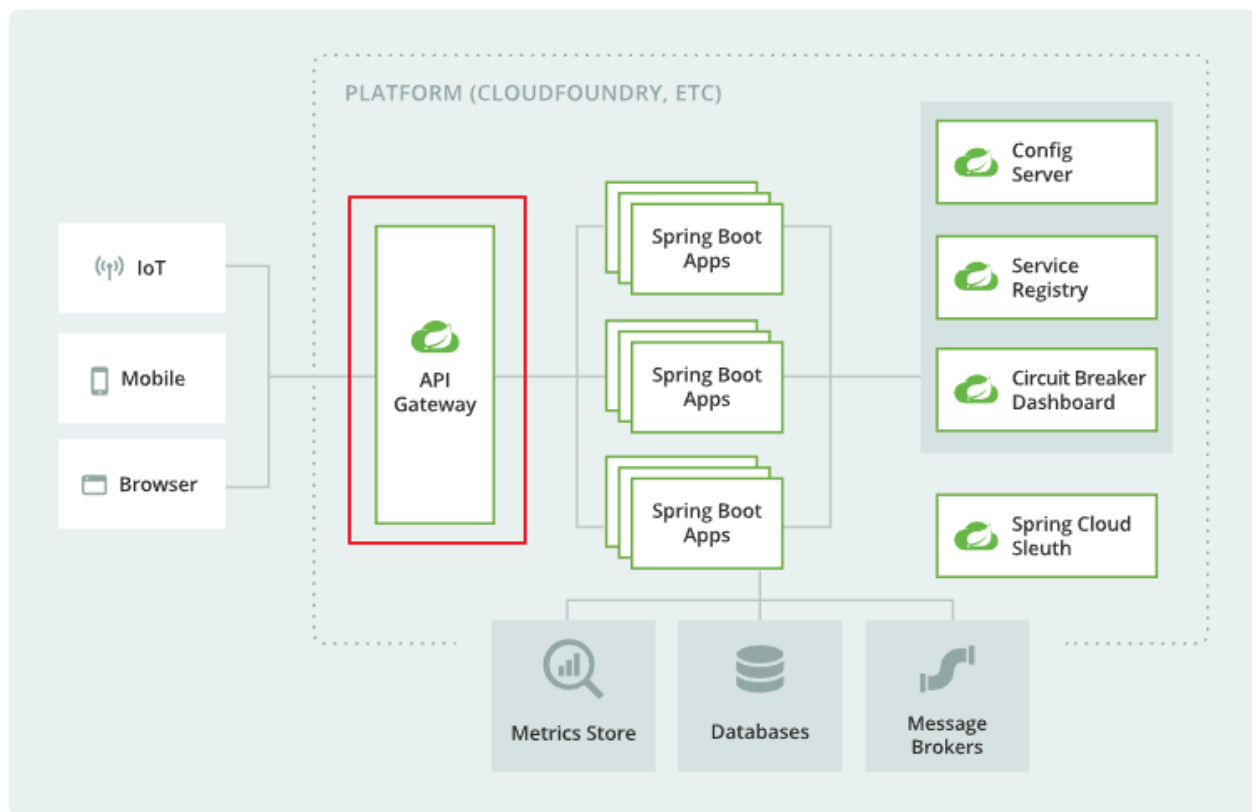
This project provides an API Gateway built on top of the Spring Ecosystem, including: Spring 5, Spring Boot 2 and Project Reactor. Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.

1.2.1. 概述

Cloud全家桶中有个很重要的组件就是网关，在1x版本中都是采用的zuul网关

但在2x版本中，zuul的升级一直跳票，Spring Cloud最后自己研发了一个网关替代zuul那就是Spring Cloud Gateway.

一句话：gateway是原zuul 1.x版的替代



Gateway是在 Spring生态系统之上构建的API网关服务，基于 Spring5，Spring Boot2和 Project Reactor等技术。

Gateway旨在提供一种简单而有效的方式来对API进行路由，以及提供一些强大的过滤器功能，例如：熔断、限流、重试等

Spring Cloud Gateway是 Spring Cloud的一个全新项目，基于 Spring5.0 + Spring Boot2.0 和 Project Reactor等技术开发的网关，它旨在为微服务架构提供一种简单有效的统一的API路由管理方式。

Spring Cloud Gateway作为 Spring Cloud生态系统中的网关，目标是替代zuul，在 Spring Cloud 2.0以上版本中，没有对新版本的Zuul 2.0以上最新高性能版本进行集成，仍然还是使用的 Zuul 1.x 非 Reactor模式的老版本。而为了提升网关的性能，Spring Cloud Gateway是基于 WebFlux框架实现的，而 WebFlux框架底层则使用了高性能的 Reactor模式通信框架Netty

Spring Cloud Gateway的目标提供统一的路由方式且基于 Filter链的方式提供了网关基本的功能，例如：安全，监控/指标，和限流

1.2.2. 一句话

SpringCloud Gateway使用的是Webflux中的reactor-netty响应式编程组件,底层使用了Netty 通讯框架

源码架构

- cloud-gateway-gateway9527
 - Lifecycle
 - Plugins
 - Dependencies
 - org.springframework.cloud:spring-cloud-starter-gateway:2.2.1.RELEASE
 - org.springframework.cloud:spring-cloud-starter:2.2.1.RELEASE
 - org.springframework.cloud:spring-cloud-gateway-core:2.2.1.RELEASE
 - org.springframework.boot:spring-boot-starter-webflux:2.2.2.RELEASE
 - org.springframework.boot:spring-boot-starter:2.2.2.RELEASE (omitted for duplicate)
 - org.springframework.boot:spring-boot-starter-json:2.2.2.RELEASE
 - org.springframework.boot:spring-boot-starter-reactor-netty:2.2.2.RELEASE
 - org.springframework.boot:spring-boot-starter-validation:2.2.2.RELEASE
 - org.springframework:spring-web:5.2.2.RELEASE
 - org.springframework:spring-webflux:5.2.2.RELEASE
 - org.synchronoss.cloud:nio-multipart-parser:1.1.0
 - org.springframework.boot:spring-boot-starter-web:2.2.2.RELEASE
 - org.springframework.boot:spring-boot-starter-actuator:2.2.2.RELEASE
 - org.springframework.cloud:spring-cloud-starter-netflix-eureka-client:2.2.1.RELEASE

1.3. 能干啥

反向代理

鉴权

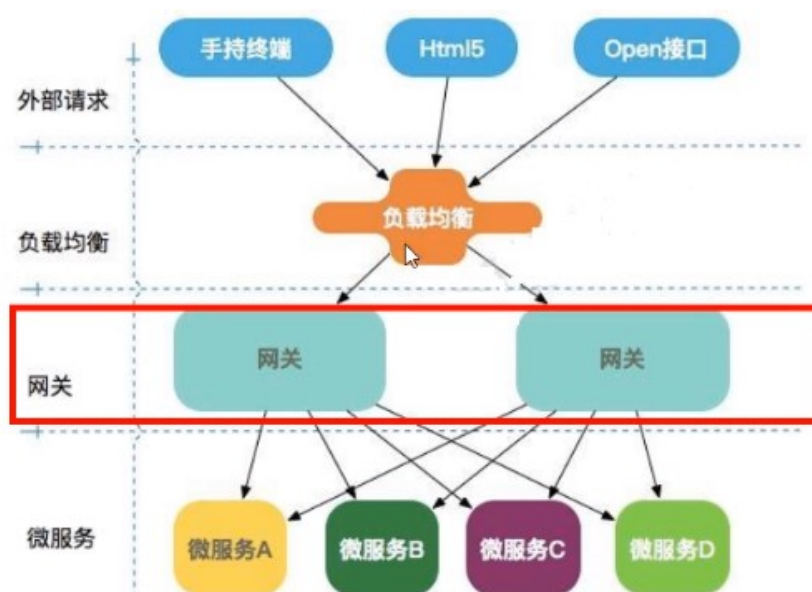
流量控制

熔断

日志监控

...

1.4. 微服务架构中网关在哪



1.5. 有Zuul了怎么又出来Gateway

1.5.1. 为什么选择Gateway

1.netflix不太靠谱,zuul2.0一直跳票,迟迟不发布

- 1) 一方面因为Zuul 1.0 已经进入了维护阶段, 而且 Gateway是 SpringCloud团队研发的, 是亲儿子产品, 值得信赖。
- 2) 而且很多功能zuul都没有,用起来也非常的简单便捷。
- 3) Gateway是基于异步非阻塞模型上进行开发的, 性能方面不需要担心。虽然Netflix早就发布了最新的Zuul 2.x, 但 Spring Cloud貌似没有整合计划。而且 Netflix相关组件都宣布进入维护期; 不知前景如何?
- 4)多方面综合考虑 Gateway是很理想的网关选择。

2.SpringCloud Gateway具有如下特性

- 1) 基于 Spring Framework5, Project Reactor和 Spring Boot2.0进行构建;
- 2) 动态路由: 能够匹配任何请求属性; 可以对路由指定 Predicate (断言) 和 Filter (过滤器)
- 3) 集成 Hystrix的断路器功能
- 4) 集成 Spring Cloud服务发现功能;
- 5) 易于编写的 Predicate (断言) 和 Filter (过滤器) ;
- 6) 请求限流功能;
- 7) 支持路径重写。

3.SpringCloud Gateway与Zuul的区别

- 1) Zuul 1.x, 是一个基于阻塞IO的 API Gateway
- 2) Zuul 1.x 基于 Servlet2.5 使用阻塞架构. 它不支持任何长连接 (如 WebSocket) .Zuul的设计模式和 Nginx较像, 每次IO操作都是从工作线程中选择一个执行, 请求线程被阻塞到工作线程完成, 但是差别是 Nginx用C++实现, Zuul用Java实现, 而JVM本身会有第一次加载较慢的情况, 使得Zuul的性能相对较差。
- 3) Zuul 2.x 理念更先进, 想基于Netty和支持长连接, 但 Spring Cloud目前还没有整合。Zuul 2.x 的性能较Zuul 1.x 有较大提升。在性能方面, 根据官方提供的基准测试, Spring Cloud Gateway的RPS (每秒请求数) 是Zuul的1.6倍。
- 4) Spring Cloud Gateway建立在 Spring Framework5、Project Reactor和 Spring Boot 2 之上, 使用非阻塞API
- 5) Spring Cloud Gateway还支持 WebSocket, 并且与 Spring紧密集成拥有更好的开发体验

1.5.2. Zuul 1.x模型

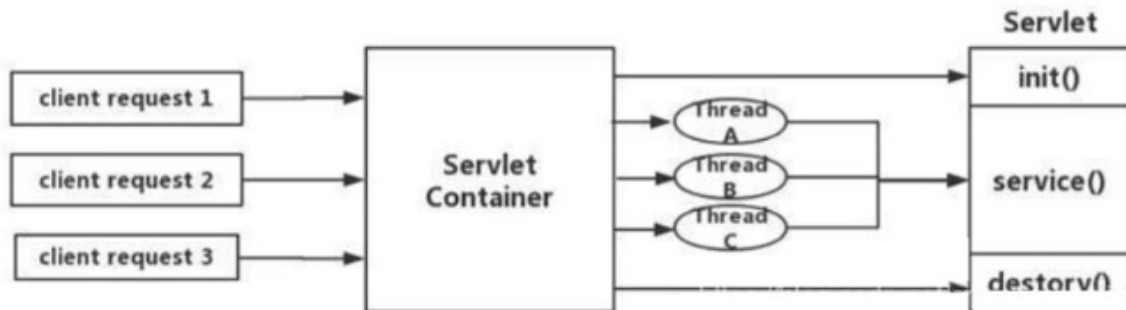
SpringCloud中所集成的Zuul版本, 采用的是 Tomcat容器, 使用的是传统的 Servlet IO处理模型。

servlet由 servlet container进行生命周期管理。

container启动时构造 servlet对象并调用 servlet init()进行初始化

container运行时接受请求，并为每个请求分配个线程（一般从线程池中获取空闲线程）然后调用 service()

container关闭时调用 servlet destroy()销毁 servlet



上述模式的缺点：

- servlet是一个简单的网络IO模型，当请求进入 servlet container时，servlet container就会为其绑定一个线程，在并发不高的场景下这种模型是适用的。但是一旦高并发(比如抽风用jmeter压)，线程数量就会上涨，而线程资源代价是昂贵的（上下文切换，内存消耗大）严重影响请求的处理时间.在一些简单业务场景下，不希望为每个 request分配一个线程，只需要1个或几个线程就能应对极大并发的请求，这种业务场景下Servlet模型没有优势
- 所以Zuul 1.x 是基于 servlet 之上的一个阻塞式处理模型，即 spring实现了处理所有 request请求的一个 servlet（DispatcherServlet）并由该 servlet阻塞式处理处理。所以SpringCloud Zuul无法摆脱 servlet模型的弊端

1.5.3. Gateway 模型

1.5.3.1. WebFlux 是什么

<https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html#spring-webflux>

[Back to index](#)
1. Spring WebFlux
1.1. Overview
1.2. Reactive Core
1.3. DispatcherHandler
1.4. Annotated Controllers
1.5. Functional Endpoints
1.6. URI Links
1.7. CORS
1.8. Web Security
1.9. View Technologies

This part of the documentation covers support for reactive-stack web applications built on a [Reactive Streams](#) API to run on non-blocking servers, such as Netty, Undertow, and Servlet 3.1+ containers. Individual chapters cover the [Spring WebFlux](#) framework, the reactive [WebClient](#), support for [testing](#), and [reactive libraries](#). For Servlet-stack web applications, see [Web on Servlet Stack](#).

1. Spring WebFlux

The original web framework included in the Spring Framework, Spring Web MVC, was purpose-built for the Servlet API and Servlet containers. The reactive-stack web framework, Spring WebFlux, was added later in version 5.0. It is fully non-blocking, supports [Reactive Streams](#) back pressure, and runs on such servers as Netty, Undertow, and Servlet 3.1+ containers.

Both web frameworks mirror the names of their source modules ([spring-webmvc](#) and [spring-webflux](#)) and co-exist side by side in the Spring Framework. Each module is optional. Applications can use one or the other module or, in some cases, both — for example, Spring MVC controllers with the reactive [WebClient](#).

1.5.3.2. 说明

传统的Web框架，比如说：struts2，springmvc等都是基于 Servlet API与 Servlet容器基础上运行的。

但是在 Servlet 3.1之后有了异步非阻塞的支持。而 WebFlux是个典型非阻塞异步的框架，它的核心是基于 Reactor的相关API 实现的。相对于传统的web框架来说，它可以运行在诸如 Netty，Undertow及支持 Servlet3.1的容器上。非阻塞式+函数式编程（Spring5必须让你使用 java8）

Spring WebFlux是 Spring5.0引入的新的响应式框架，区别于 Spring MVC，它不需要依赖 Servlet API，它是完全异步非阻塞的，并且基于 Reactor来实现响应式流规范。

2. 三大核心概念

2.1. Route(路由)

路由是构建网关的基本模块,它由ID,目标URI,一系列的断言和过滤器组成,如断言为true则匹配该路由

2.2. Predicate(断言)

参考的是Java8的java.util.function.Predicate

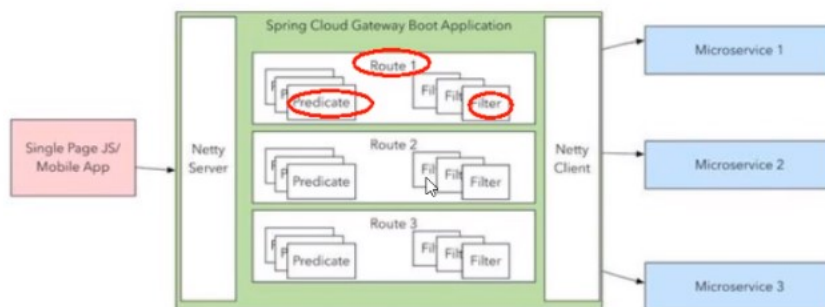
开发人员可以匹配HTTP请求中的所有内容(例如请求头或请求参数),如果请求与断言相匹配则进行路由

2.3. Filter(过滤)

指的是Spring框架中GatewayFilter的实例,使用过滤器,可以在请求被路由前或者之后对请求进行修改.

2.4. 总结

|



web请求，通过一些匹配条件，定位到真正的服务节点。并在这个转发过程的前后，进行一些精细化控制。

predicate就是我们的匹配条件；而filter，就可以理解为一个无所不能的拦截器。有了这两个元素，再加上目标uri，就可以实现一个具体的路由了

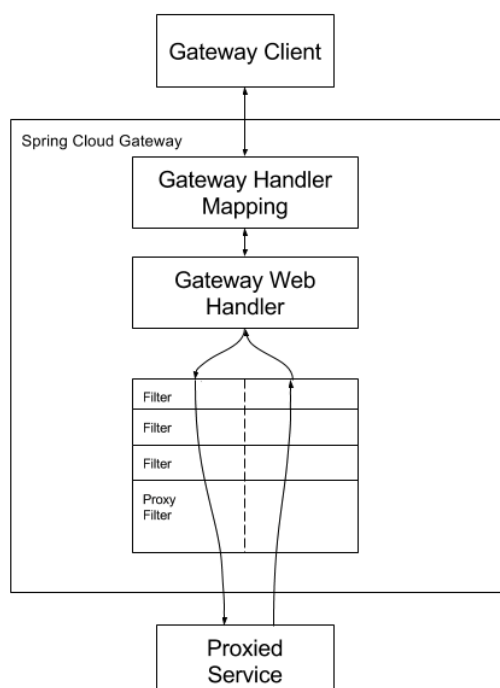
web请求，通过一些匹配条件，定位到真正的服务节点，并在这个转发过程的前后，进行一些精细化控制，predicate就是我们的匹配条件：而filter，就可以理解为一个无所不能的拦截器。有了这两个元素，再加上目标uri，就可以实现一个具体的路由了

3. Gateway工作流程

3.1. 官网总结

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.3.RELEASE/reference/html/#gateway-how-it-works>

The following diagram provides a high-level overview of how Spring Cloud Gateway works:



Clients make requests to Spring Cloud Gateway. If the Gateway Handler Mapping determines that a request matches a route, it is sent to the Gateway Web Handler. This handler runs the request through a filter chain that is specific to the request. The reason the filters are divided by the dotted line is that filters can run logic both before and after the proxy request is sent. All "pre" filter logic is executed. Then the proxy request is made. After the proxy request is made, the "post" filter logic is run.



URIs defined in routes without a port get default port values of 80 and 443 for the HTTP and HTTPS URIs, respectively.

客户端向 Spring Cloud Gateway发出请求。然后在 Gateway Handler Mapping中找到与请求相匹配的路由，将其发送到 Gateway Web handler。

Handler再通过指定的过滤器链来将请求发送到我们实际的服务执行业务逻辑，然后返回

过滤器之间用虚线分开是因为过滤器可能会在发送代理请求之前（"pre"）或之后（"post"）执行业务逻辑

Filter在"pre"类型的过滤器可以做参数校验、权限校验、流量监控、日志输出、协议转换等在"post"类型的过滤器中可以做响应内容、响应头的修改，日志的输出，流量监控等有着非常重要的作用。

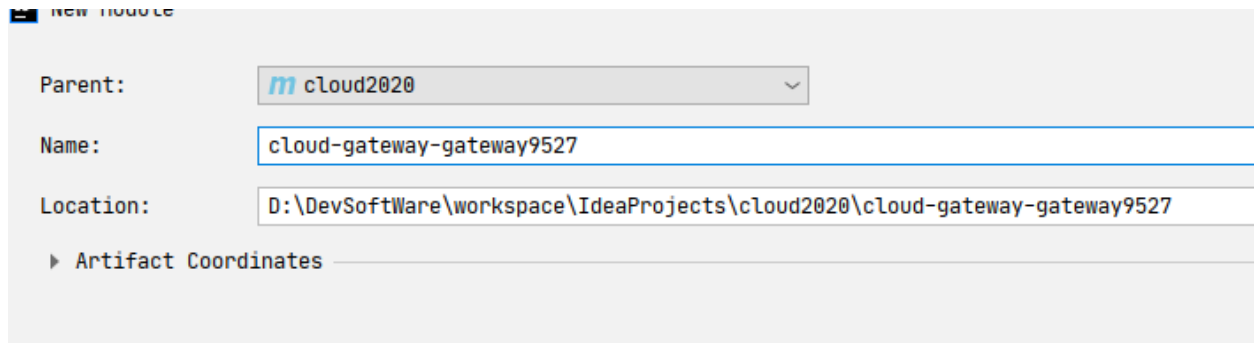
3.2. 核心逻辑

路由转发+执行过滤器链

4. 入门配置

4.1. 新建Module

新建cloud-gateway-gateway9527



Parent: m cloud2020

Name: cloud-gateway-gateway9527

Location: D:\DevSoftWare\workspace\IdeaProjects\cloud2020\cloud-gateway-gateway9527

► Artifact Coordinates

4.2. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0.xsd">
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>cloud-gateway-gateway9527</artifactId>
13
14    <dependencies>
15        <!--新增gateway-->
16        <dependency>
17            <groupId>org.springframework.cloud</groupId>
18            <artifactId>spring-cloud-starter-gateway</artifactId>
19        </dependency>
20        <!--eureka-client-->
21        <dependency>
22            <groupId>org.springframework.cloud</groupId>
```

```

23         <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
24     </dependency>
25     <!--API通用包-->
26     <dependency>
27         <groupId>cn.sitedev.springcloud</groupId>
28         <artifactId>cloud-api-commons</artifactId>
29         <version>1.0-SNAPSHOT</version>
30     </dependency>
31     <!--基础配置类-->
32     <dependency>
33         <groupId>org.springframework.boot</groupId>
34         <artifactId>spring-boot-devtools</artifactId>
35         <scope>runtime</scope>
36         <optional>true</optional>
37     </dependency>
38
39     <dependency>
40         <groupId>org.projectlombok</groupId>
41         <artifactId>lombok</artifactId>
42         <optional>true</optional>
43     </dependency>
44     <dependency>
45         <groupId>org.springframework.boot</groupId>
46         <artifactId>spring-boot-starter-test</artifactId>
47         <scope>test</scope>
48     </dependency>
49
50 </dependencies>
51
52 </project>

```

4.3. YML

```

1 server:
2   port: 9527
3 spring:
4   application:
5     name: cloud-gateway
6 eureka:
7   instance:
8     hostname: cloud-gateway-service

```

```

9   client: # 服务提供者provider注册进eureka服务列表内
10  service-url:
11      register-with-eureka: true
12      fetch-registry: true
13      defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka

```

4.4. 主启动类

```

1  package cn.sitedev.springcloud;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6
7  @SpringBootApplication
8  @EnableEurekaClient
9  public class GatewayMain9527 {
10     public static void main(String[] args) {
11         SpringApplication.run(GatewayMain9527.class, args);
12     }
13 }

```

4.5. 9527网关如何做路由映射呢

cloud-provider-payment8001看看controller的访问地址

<http://localhost:8001/payment/lb>

<http://localhost:8001/get/{id}>

我们目前不想暴露8001端口,希望在8001外面套一层9527

4.6. YML新增网关配置

修改内容:

```

1  spring:
2      cloud:
3          gateway:
4              discovery:
5                  locator:

```

```

6         enabled: true # 开启从注册中心动态创建路由的功能，利用微服务名称进行路由
7     routes:
8         - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
9           #匹配后提供服务的路由地址
10          uri: http://localhost:8001
11          predicates:
12              - Path=/payment/get/** # 断言，路径相匹配的进行路由
13         - id: payment_route2
14          uri: http://localhost:8001
15          predicates:
16              Path=/payment/lb/** #断言,路径相匹配的进行路由

```

完整内容:

```

1 server:
2     port: 9527
3 spring:
4     application:
5         name: cloud-gateway
6     cloud:
7         gateway:
8             discovery:
9                 locator:
10                    enabled: true # 开启从注册中心动态创建路由的功能，利用微服务名称进行路由
11            routes:
12                - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
13                  #匹配后提供服务的路由地址
14                  uri: http://localhost:8001
15                  predicates:
16                      - Path=/payment/get/** # 断言，路径相匹配的进行路由
17                - id: payment_route2
18                  uri: http://localhost:8001
19                  predicates:
20                      Path=/payment/lb/** #断言,路径相匹配的进行路由
21 eureka:
22     instance:
23         hostname: cloud-gateway-service
24     client: # 服务提供者provider注册进eureka服务列表内
25         service-url:
26             register-with-eureka: true
27             fetch-registry: true
28             defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka

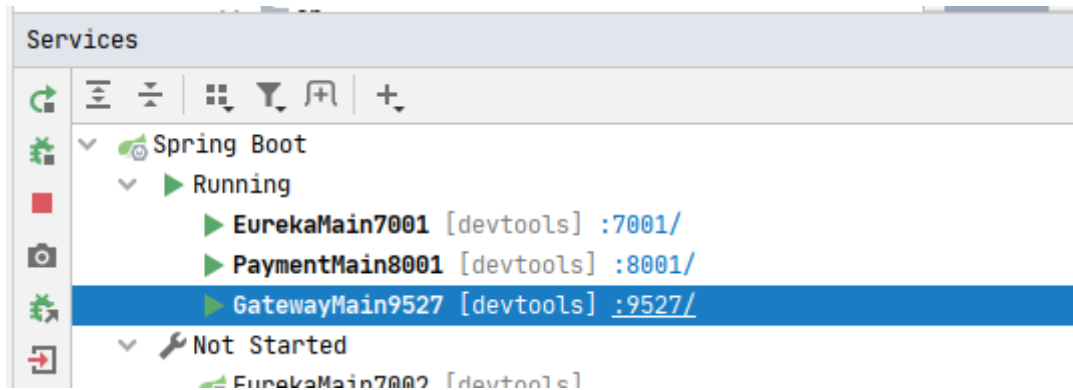
```

4.7. 测试

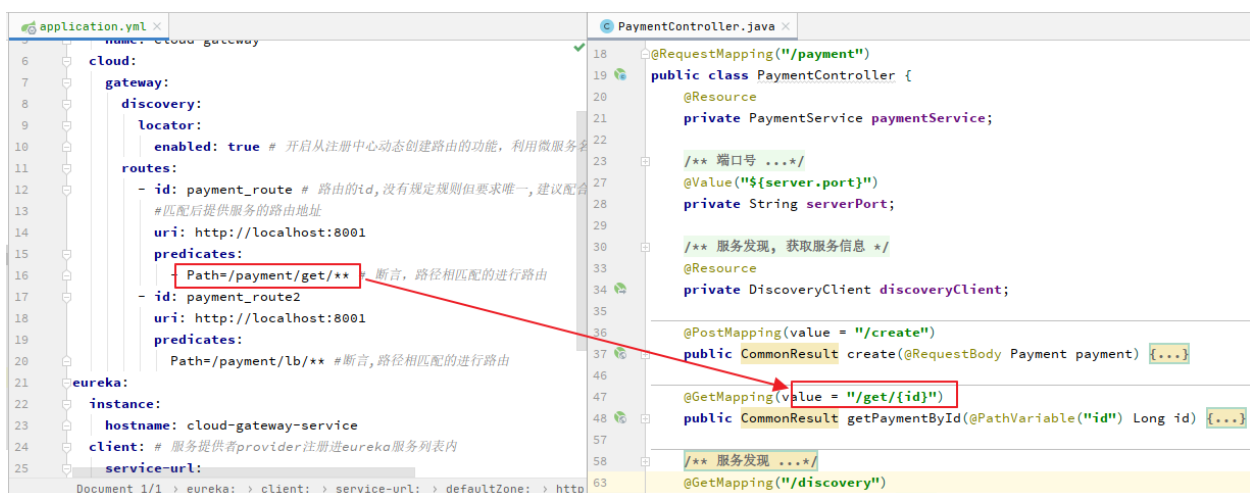
启动EurekaMain7001

启动PaymentMain8001

启动GatewayMain9527



访问说明:



添加网关前, 访问地址: <http://localhost:8001/payment/get/1>



添加网关后, 访问地址: <http://localhost:9527/payment/get/1>



4.8. YML配置说明

网关路由有两种配置方式

4.8.1. 在YML配置文件中配置

```
1 spring:
2   cloud:
3     gateway:
4       discovery:
5         locator:
6           enabled: true # 开启从注册中心动态创建路由的功能，利用微服务名称进行路由
7       routes:
8         - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
9           #匹配后提供服务的路由地址
10          uri: http://localhost:8001
11          predicates:
12            - Path=/payment/get/** # 断言，路径相匹配的进行路由
13        - id: payment_route2
14          uri: http://localhost:8001
15          predicates:
16            Path=/payment/lb/** #断言,路径相匹配的进行路由
```

4.8.2. 代码中注入RouteLocator的Bean

4.8.2.1. 官网案例

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.3.RELEASE/reference/html/#modifying-the-way-remote-addresses-are-resolved>

参见 Example 12. GatewayConfig.java

```

1 RemoteAddressResolver resolver = XForwardedRemoteAddressResolver
2   .maxTrustedIndex(1);
3
4 ...
5
6 .route("direct-route",
7   r -> r.remoteAddr("10.1.1.1", "10.10.1.1/24")
8     .uri("https://downstream1")
9 .route("proxied-route",
10   r -> r.remoteAddr(resolver, "10.10.1.1", "10.10.1.1/24")
11     .uri("https://downstream2")
12 )

```

4.8.2.2. 实例演示

4.8.2.2.1. 业务需求

通过9527网关访问到外网的百度新闻网址<https://news.baidu.com/guonei>

4.8.2.2.2. 编码实现

在cloud-gateway-gateway9527模块内添加一个配置bean

```

1 package cn.sitedev.springcloud.config;
2
3 import org.springframework.cloud.gateway.route.RouteLocator;
4 import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 @Configuration
9 public class GatewayConfig {
10     /**
11      * 配置了一个id为path_route_sitedev的路由规则，
12      * <p>
13      * 当访问http://localhost:9527/guonei时，会自动转发到地址https://news.baidu.com/
14      *
15      * @param routeLocatorBuilder
16      * @return
17      */
18     @Bean

```

```

19     public RouteLocator customRouteLocator(RouteLocatorBuilder routeLocatorBuilder)
20         RouteLocatorBuilder.Builder routes = routeLocatorBuilder.routes();
21         routes.route("path_route_sitedev", r -> r.path("/guonei").uri("https://news.
22         return routes.build();
23     }
24
25     @Bean
26     public RouteLocator customRouteLocator2(RouteLocatorBuilder routeLocatorBuilder)
27         RouteLocatorBuilder.Builder routes = routeLocatorBuilder.routes();
28         routes.route("path_route_sitedev2", r -> r.path("/guoji").uri("https://news.
29         return routes.build();
30     }
31 }

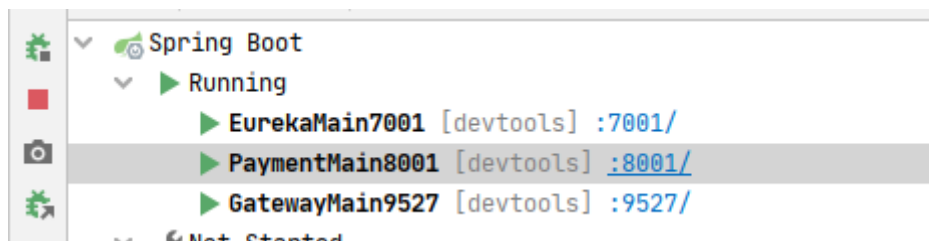
```

4.8.2.2.3. 测试

启动EurekaMain7001

启动PaymentMain8001

启动GatewayMain9527



浏览器访问: <http://localhost:9527/guonei>



浏览器访问<http://localhost:9527/guojia>



4.9. 注意事项

如果在启动GatewayMain9527时, 报如下错误:

```
1 *****
2 APPLICATION FAILED TO START
3 *****
4
5 Description:
6
7 Parameter 0 of method modifyResponseBodyGatewayFilterFactory in org.springframework.
8
9
10 Action:
11
12 Consider defining a bean of type 'org.springframework.http.codec.ServerCodecConfigur
13
14
15 Process finished with exit code 0
```

是因为依赖中引入了spring-boot-starter-web, 因为Spring Cloud Gateway 是使用 netty+webflux实现, webflux与web是冲突的。因此我们需要移除spring-boot-starter-web的依赖

5. 通过服务名实现动态路由

5.1. 说明

默认情况下Gateway会根据注册中心注册的服务列表, 以注册中心上微服务名为路径创建动态路由进行转发,从而实现动态路由的功能

5.2. POM

修改内容:

```
1      <!--eureka-client-->
2      <dependency>
3          <groupId>org.springframework.cloud</groupId>
4          <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
5      </dependency>
```

完整内容:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0"
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>cloud-gateway-gateway9527</artifactId>
13
14    <dependencies>
15        <!--新增gateway-->
16        <dependency>
17            <groupId>org.springframework.cloud</groupId>
18            <artifactId>spring-cloud-starter-gateway</artifactId>
19        </dependency>
20        <!--eureka-client-->
21        <dependency>
22            <groupId>org.springframework.cloud</groupId>
23            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
24        </dependency>
```

```

25     <!--API通用包-->
26     <dependency>
27         <groupId>cn.sitedev.springcloud</groupId>
28         <artifactId>cloud-api-commons</artifactId>
29         <version>1.0-SNAPSHOT</version>
30     </dependency>
31     <!--基础配置类-->
32     <dependency>
33         <groupId>org.springframework.boot</groupId>
34         <artifactId>spring-boot-devtools</artifactId>
35         <scope>runtime</scope>
36         <optional>true</optional>
37     </dependency>
38
39     <dependency>
40         <groupId>org.projectlombok</groupId>
41         <artifactId>lombok</artifactId>
42         <optional>true</optional>
43     </dependency>
44     <dependency>
45         <groupId>org.springframework.boot</groupId>
46         <artifactId>spring-boot-starter-test</artifactId>
47         <scope>test</scope>
48     </dependency>
49
50 </dependencies>
51
52 </project>

```

5.3. YML

说明:

需要注意的是uri的协议lb,表示启用Gateway的负载均衡功能.

lb://serverName是spring cloud gateway在微服务中自动为我们创建的负载均衡uri

修改内容:

```

1 spring:
2   cloud:
3     gateway:
4     discovery:

```

```

5     locator:
6         enabled: true # 开启从注册中心动态创建路由的功能，利用微服务名称进行路由
7     routes:
8         - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
9           #匹配后提供服务的路由地址
10    #       uri: http://localhost:8001
11           uri: lb://cloud-payment-service
12           predicates:
13             - Path=/payment/get/** # 断言，路径相匹配的进行路由
14         - id: payment_route2
15    #       uri: http://localhost:8001
16           uri: lb://cloud-payment-service
17           predicates:
18             Path=/payment/lb/** #断言,路径相匹配的进行路由

```

完整内容:

```

1 server:
2     port: 9527
3 spring:
4     application:
5         name: cloud-gateway
6     cloud:
7         gateway:
8             discovery:
9                 locator:
10                    enabled: true # 开启从注册中心动态创建路由的功能，利用微服务名称进行路由
11            routes:
12                - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
13                  #匹配后提供服务的路由地址
14    #       uri: http://localhost:8001
15                  uri: lb://cloud-payment-service
16                  predicates:
17                    - Path=/payment/get/** # 断言，路径相匹配的进行路由
18                - id: payment_route2
19    #       uri: http://localhost:8001
20                  uri: lb://cloud-payment-service
21                  predicates:
22                    Path=/payment/lb/** #断言,路径相匹配的进行路由
23 eureka:
24     instance:
25         hostname: cloud-gateway-service

```



```
26 client: # 服务提供者provider注册进eureka服务列表内
27 service-url:
28     register-with-eureka: true
29     fetch-registry: true
30     defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
```

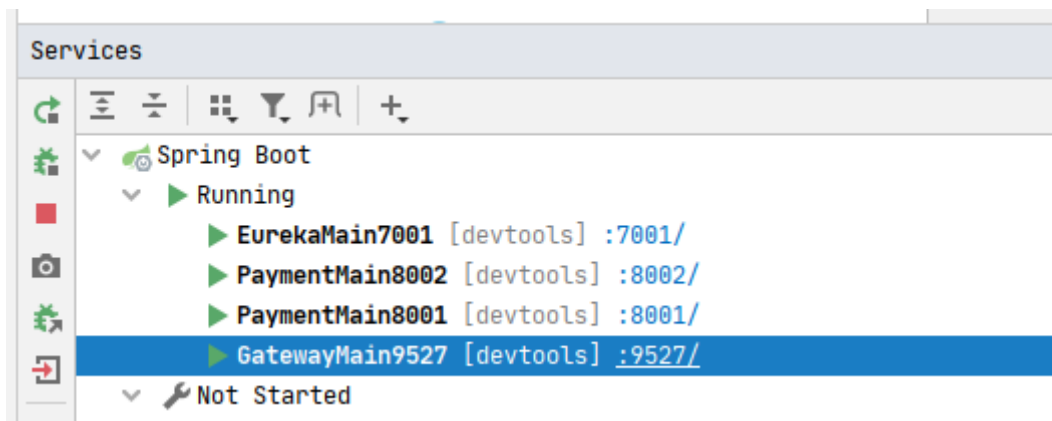
5.4. 测试

启动EurekaMain7001

启动PaymentMain8002

启动PaymentMain8001

启动GatewayMain9527



浏览器访问: <http://localhost:9527/payment/lb>



1 | 8001



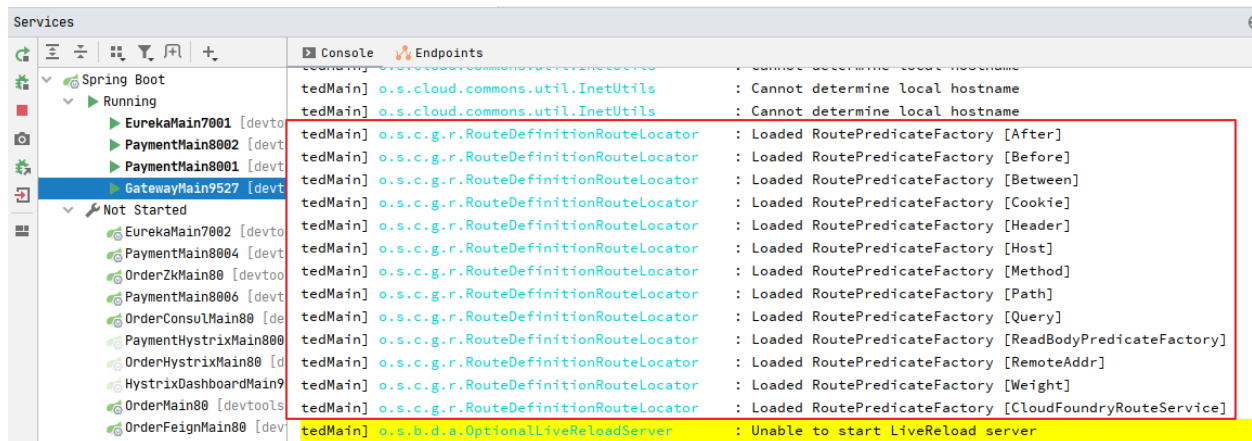
1 | 8002

可以看到: 8001/8002两个端口交替切换

6. Predict

6.1. 是什么

启动我们的gateway9527, 查看控制台输出:



6.2. Route Predicate Factories

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.1.RELEASE/reference/html/#gateway-request-predicates-factories>

4. Route Predicate Factories

4.1. The After Route Predicate Factory

Spring Cloud Gateway matches routes as part of the Spring WebFlux `HandlerMapping` infrastructure. Spring Cloud Gateway includes many built-in route predicate factories. All of these predicates match on different attributes of the HTTP request. You can combine multiple route predicate factories with logical `and` statements.

The after route predicate factory takes one parameter, a datetime. This predicate matches requests that happen after the specified datetime. The following example configures an after route predicate:

Example 1. application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id: after_route
          uri: https://example.org
          predicates:
            - After=2017-01-20T17:42:47.789-07:00[America/Denver]
```

This route matches any request made after Jan 20, 2017 17:42 Mountain Time (Denver).

4.2. The Before Route Predicate Factory

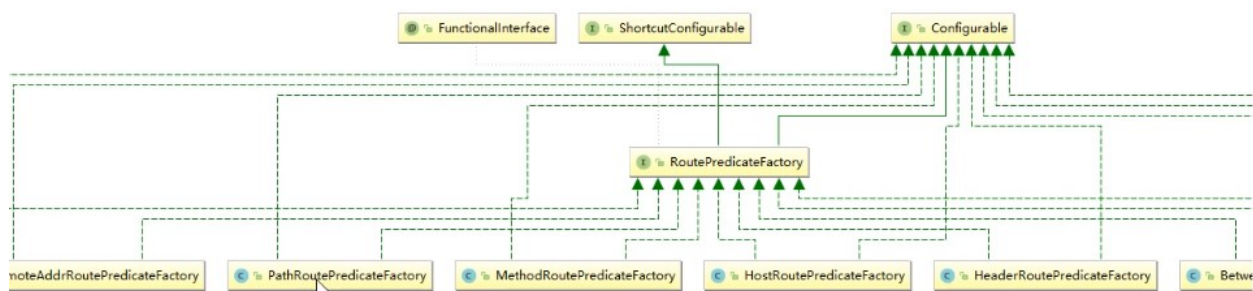
Spring Cloud Gateway将路由匹配作为 Spring WebFlux HandlerMapping基础架构的部分。

Spring Cloud Gateway包括许多内置的 Route Predicate工厂。所有这些 Predicate都与HTTP请求的不同属性匹配。多个 Route Predicate工厂可以进行组合

Spring Cloud Gateway创建 Route对象时, 使用 RoutePredicateFactory创建 Predicate对象, Predicate对象可以赋值给Route。Spring Cloud Gateway包含许多内置的 Route Predicate Factories.

所有这些谓词都匹配HTTP请求的不同属性。多种谓词工厂可以组合, 并通过逻辑and

6.3. 常用的Route Predicate



6.3.1. After Route Predicate

6.3.1.1. 官方使用说明

The after route predicate factory takes one parameter, a datetime. This predicate matches requests that happen after the specified datetime. The following example configures an after route predicate:

Example 1. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: after_route
6           uri: https://example.org
7           predicates:
8             - After=2017-01-20T17:42:47.789-07:00[America/Denver]
```

This route matches any request made after Jan 20, 2017 17:42 Mountain Time (Denver).

6.3.1.2. 实例演示

6.3.1.2.1. 获取时间字符串

```
1 package cn.sitedev.springcloud;
2
3 import java.time.ZoneId;
4 import java.time.ZonedDateTime;
5
6 public class ZonedDateTimeDemo {
```

```

7   public static void main(String[] args) {
8       // 获取当前时间(默认时区)
9       ZonedDateTime zonedDateTime = ZonedDateTime.now();
10      System.out.println(zonedDateTime);
11
12      // 获取当前时间(指定时区)
13      zonedDateTime = ZonedDateTime.now(ZoneId.of("America/New_York"));
14      System.out.println(zonedDateTime);
15  }
16 }

```

```

Run: ZonedDateTimeDemo x
D:\DevSoftWare\Java\jdk8\bin\java.exe ...
2020-06-28T14:44:45.583+08:00[Asia/Shanghai]
2020-06-28T02:44:45.591-04:00[America/New_York]
Process finished with exit code 0

```

6.3.1.2.2. 修改YML

修改内容:

```

1  spring:
2      cloud:
3          gateway:
4              routes:
5                  - id: payment_route2
6                    # uri: http://localhost:8001
6                    uri: lb://cloud-payment-service
7                  predicates:
8                      - Path=/payment/lb/** #断言,路径相匹配的进行路由
9                      - After=2020-06-28T15:00:00.000+08:00[Asia/Shanghai]
10

```

6.3.1.2.3. 测试

启动 EurekaMain7001

启动 PaymentMain8002

启动 PaymentMain8001

启动 GatewayMain9527

This route matches any request made before Jan 20, 2017 17:42 Mountain Time (Denver).

6.3.2.2. 实例演示

6.3.2.2.1. 修改YML

修改内容:

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8           predicates:
9             - Path=/payment/lb/** #断言,路径相匹配的进行路由
10            - Before=2020-06-28T15:05:00.000+08:00[Asia/Shanghai]
```

6.3.2.2.2. 测试

在Before断言的时间之前, 命令行执行 `curl http://localhost:9527/payment/lb`

```
C:\Users\qchen>curl http://localhost:9527/payment/1b8001
C:\Users\qchen>
```

在Before断言的时间之后, 命令行执行 `curl http://localhost:9527/payment/lb`

[illegible]

6.3.3. Between Route Predicate

6.3.3.1. 官方使用说明

The `between` route predicate factory takes two parameters, `datetime1` and `datetime2`. This predicate matches requests that happen after `datetime1` and before

`datetime2` . The `datetime2` parameter must be after `datetime1` . The following example configures a between route predicate:

Example 3. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: between_route
6           uri: https://example.org
7           predicates:
8             - Between=2017-01-20T17:42:47.789-07:00[America/Denver], 2017-01-21T17:42:47.789-07:00[America/Denver]
```

This route matches any request made after Jan 20, 2017 17:42 Mountain Time (Denver) and before Jan 21, 2017 17:42 Mountain Time (Denver). This could be useful for maintenance windows.

6.3.3.2. 实例演示

6.3.3.2.1. 修改YML

修改内容:

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8           predicates:
9             - Path=/payment/lb/** #断言,路径相匹配的进行路由
10            - Between=2020-06-28T15:05:00.000+08:00[Asia/Shanghai], 2020-06-28T15:10:00.000+08:00[Asia/Shanghai]
```

6.3.3.2.2. 测试

在Between断言的时间之内, 命令行执行 `curl http://localhost:9527/payment/lb`

```
C:\Users\qchen>curl http://localhost:9527/payment/1b8001
C:\Users\qchen>
```

在Between断言的时间之外, 命令行执行 `curl http://localhost:9527/payment/lb`

```
C:\Users\qchen>curl http://localhost:9527/payment/1b
{"timestamp": "2020-06-28T07:12:00.187+0000", "path": "/payment/1b", "status": 404, "error": "Not Found", "message": null, "requestId": "49b8bdaf5", "trace": "org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND\r\n\tat org.springframework.rk.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\tSuppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: \r\nError has been observed at the following site(s):\r\n\t|_ checkpoint $?org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\r\n\t|_ checkpoint $?HTTP GET /\r\npayment/1b/\r\n[ExceptionHandlerWebHandler]\r\nStack trace:\r\n\r\n\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\r\n\tat reactor.core.publisher.MonoDefer.subscribe(MonoDefer.java:44)\r\n\r\n\tat reactor.core.publisher.Mono.subscribe(Mono.java:4105)\r\n\r\n\tat reactor.core.publisher.FluxSwitchIfEmpty$SwitchIfEmptySubscriber.onComplete(FluxSwitchIfEmpty.java:75)\r\n\r\n\tat reactor.core.publisher.MonoFlatMap$FlatMapMain.oNComplete(MonoFlatMap.java:174)\r\n\r\n\tat reactor.core.publisher.MonoNext$NextSubscriber.onComplete(MonoNext.java:96)\r\n\r\n"}

```

6.3.4. Cookie Route Predicate

6.3.4.1. 官方使用说明

The cookie route predicate factory takes two parameters, the cookie name and a regular expression. This predicate matches cookies that have the given name and whose values match the regular expression. The following example configures a cookie route predicate factory:

Example 4. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: cookie_route
6           uri: https://example.org
7           predicates:
8             - Cookie=chocolate, ch.p
```

This route matches requests that have a cookie named `chocolate` whose value matches the `ch.p` regular expression.

Cookie Route Predicate需要两个参数，一个是 Cookie name，一个是正则表达式

路由规则会通过获取对应的 Cookie name值和正则表达式去匹配，如果匹配上就会执行路由，如果没有匹配上则不执行

6.3.4.2. 实例演示

6.3.4.2.1. 修改YML

修改内容:

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8           predicates:
9             - Path=/payment/lb/** #断言,路径相匹配的进行路由
10            - Cookie=username, sitedev

```

6.3.4.2.2. 测试

不添加cookie, 命令行执行 `curl http://localhost:9527/payment/lb`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb
{"timestamp":"2020-06-28T07:17:40.459+0000","path":"/payment/lb","status":404,"error":"Not Found","message":null,"requestId":"52b2b23f","trace":"org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND\r\n\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\tSuppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: \nError has been observed at the following site(s):\n\t|_ checkpoint 鈹?org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\n\t|_ checkpoint 鈹?HTTP GET \"/payment/lb/" [ExceptionHandlerWebHandler]\nStack trace:\r\n\t\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\t\t\tat reactor.core.publisher.MonoDefer.subscribe(MonoDefer.java:405)\r\n\t\t\t\tat reactor.core.publisher.FluxSwitchIfEmpty$

```

添加cookie, 命令行执行 `curl http://localhost:9527/payment/lb --cookie "username=sitedev"`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb --cookie "username=sitedev"
8001
C:\Users\qchen>

```

6.3.5. Header Route Predicate

6.3.5.1. 官方使用说明

The header route predicate factory takes two parameters, the header name and a regular expression. This predicate matches with a header that has the given name whose value matches the regular expression. The following example configures a header route predicate:

Example 5. application.yml

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: header_route
6           uri: https://example.org

```



```

7     predicates:
8     - Header=X-Request-Id, \d+

```

This route matches if the request has a header named `X-Request-Id` whose value matches the `\d+` regular expression (that is, it has a value of one or more digits).

两个参数：一个是属性名称和一个正则表达式，这个属性值和正则表达式匹配则执行。

6.3.5.2. 实例演示

6.3.5.2.1. 修改YML

修改内容:

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8           predicates:
9             - Path=/payment/lb/** #断言,路径相匹配的进行路由
10            - Header=X-Request-Id, \d+

```

6.3.5.2.2. 测试

不添加指定请求头, 命令行执行 `curl http://localhost:9527/payment/lb -H "X-Request-Id:sitedev"`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb -H "X-Request-Id:sitedev"
{"timestamp":"2020-06-28T07:23:58.683+0000","path":"/payment/lb","status":404,"error":"Not Found","message":null,"requestId":"d484400a","trace":"org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND\r\n\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\tSuppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: \nError has been observed at the following site(s):\n\t|_ checkpoint 鈹?org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\n\t|_ checkpoint 鈹?HTTP GET \"/payment/lb/" [ExceptionHandlerWebHandler]\nStack trace:\r\n\t\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\t\t\tat reactor.core.publisher.MonoDefer.subscribe(MonoDefer.java:

```

添加指定请求头, 命令行执行 `curl http://localhost:9527/payment/lb -H "X-Request-Id:123"`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb -H "X-Request-Id:123"
8001
C:\Users\qchen>

```

6.3.6. Host Route Predicate

6.3.6.1. 官方使用说明

The host route predicate factory takes one parameter: a list of host name patterns. The pattern is an Ant-style pattern with `.` as the separator. This predicate matches the `Host` header that matches the pattern. The following example configures a host route predicate:

Example 6. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: host_route
6           uri: https://example.org
7           predicates:
8             - Host=**.somehost.org,**.anotherhost.org
```

URI template variables (such as `{sub}.myhost.org`) are supported as well.

This route matches if the request has a `Host` header with a value of `www.somehost.org` or `beta.somehost.org` or `www.anotherhost.org`.

This predicate extracts the URI template variables (such as `sub`, 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`.

Host Route Predicate接收一组参数，一组匹配的域名列表，这个模板是一个ant分隔的模板，用号作为分隔符。它通过参数中的主机地址作为匹配规则

6.3.6.2. 实例演示

6.3.6.2.1. 修改YML

修改内容:

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8           predicates:
```

```
9 - Path=/payment/lb/** #断言,路径相匹配的进行路由
10 - Host=**.sitedev.cn
```

6.3.6.2.2. 测试

不添加指定HOST, 命令行执行 `curl http://localhost:9527/payment/lb -H "Host:www.baidu.com"`

```
C:\Users\qchen>curl http://localhost:9527/payment/lb -H "Host:www.baidu.com"
{"timestamp":"2020-06-28T07:29:31.698+0000","path":"/payment/lb","status":404,"error":"Not Found","message":null,"requestId":"dab33c3f","trace":"org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND\r\n\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\tSuppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: \nError has been observed at the following site(s):\n\t|_ checkpoint #R?org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\n\t|_ checkpoint #R?HTTP GET \"/
```

添加指定HOST, 命令行执行 `curl http://localhost:9527/payment/lb -H "Host:www.sitedev.cn"`

```
C:\Users\qchen>curl http://localhost:9527/payment/lb -H "Host:www.sitedev.cn"
8001
C:\Users\qchen>
```

6.3.7. Method Route Predicate

6.3.7.1. 官方使用说明

The Method Route Predicate Factory takes one or more parameters: the HTTP methods to match. The following example configures a method route predicate:

Example 7. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: method_route
6           uri: https://example.org
7           predicates:
8             - Method=GET,POST
```

This route matches if the request method was a `GET` or a `POST`.

6.3.7.2. 实例演示

6.3.7.2.1. 修改YML

修改内容:

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8         predicates:
9           - Path=/payment/lb/** #断言,路径相匹配的进行路由
10          - Method=GET

```

6.3.7.2.2. 测试

不使用指定的Method, 命令行执行 `curl http://localhost:9527/payment/lb -X POST`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb -X POST
{"timestamp":"2020-06-28T07:34:46.623+0000","path":"/payment/lb","status":404,"error":"Not Found","message":null,"requestId":"b49fafab","trace":"org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND\r\n\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\t\tSuppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: \nError has been observed at the following site(s):\n\t|_ checkpoint 鈔?org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\n\t|_ checkpoint 鈔?HTTP POST \"/payment/lb/" [ExceptionHandlerWebHandler]\nStack trace:\r\n\t\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\t\t\tat reactor.core.publisher.MonoDefer.subscribe(MonoDefer.java:44)\r\n\t\t\t\tat reactor.core.publisher.Mono.subscribe(Mono.java:4105)\r\n\t\t\t\t\tat reactor.core.publisher.FluxSwitchIfEmpty

```

使用指定的Method, 命令行执行 `curl http://localhost:9527/payment/lb -X GET`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb -X GET
8001
C:\Users\qchen>

```

6.3.8. Path Route Predicate

6.3.8.1. 官方使用说明

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

Example 8. application.yml

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: host_route
6           uri: https://example.org
7         predicates:
8           - Path=/red/{segment},/blue/{segment}

```

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

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:

```
1 Map<String, String> uriVariables = ServerWebExchangeUtils.getPathPredicateVariables(
2
3 String segment = uriVariables.get("segment");
```

6.3.8.2. 实例演示

6.3.8.2.1. 修改YML

修改内容:

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8           predicates:
9             - Path=/payment/lb/** #断言,路径相匹配的进行路由
```

6.3.8.2.2. 测试

不使用指定的Path, 命令行执行 `curl http://localhost:9527/payment/lbx`

```
C:\Users\qchen>curl http://localhost:9527/payment/lbx  
{"timestamp": "2020-06-28T07:36:32.569+0000", "path": "/payment/lbx", "status": 404, "error": "Not Found", "message": null, "requestId": "4a85e996a", "trace": "org.springframework.web.server.ResponseStatusException: 404 NOT FOUND\\r\\n\\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\\r\\n\\tat reactor.core.publisher.FluxOnAssembly$OnAssemblyException:\\nError has been observed at the following site(s):\\n|_ checkpoint $R$or g.org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\\n|_ checkpoint $R$HTTP GET \\ /payment/lbx\" [ExceptionHandlerWebHandler]\\nStack trace: \\r\\n\\tat org.springframework.web.reactive.resource.Resource
```

使用指定的Path, 命令行执行 `curl http://localhost:9527/payment/lb`

```
C:\Users\qchen>curl http://localhost:9527/payment/1b8002
C:\Users\qchen>
```

6.3.9. Query Route Predicate

6.3.9.1. 官方使用说明

The query route predicate factory takes two parameters: a required `param` and an optional `regex`. The following example configures a query route predicate:

Example 9. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: query_route
6           uri: https://example.org
7           predicates:
8             - Query=green
```

The preceding route matches if the request contained a `green` query parameter.

application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: query_route
6           uri: https://example.org
7           predicates:
8             - Query=red, gree.
```

The preceding route matches if the request contained a `red` query parameter whose value matched the `gree.` regexp, so `green` and `greet` would match.

6.3.9.2. 实例演示

6.3.9.2.1. 修改YML

修改内容:

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           # uri: http://localhost:8001
7           uri: lb://cloud-payment-service
8         predicates:
9           - Path=/payment/lb/** #断言,路径相匹配的进行路由
10          - Query=username, \d+

```

6.3.9.2.2. 测试

不使用指定的Query, 命令行执行 `curl http://localhost:9527/payment/lb?`
`username=sitedev`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb?username=sitedev
{"timestamp":"2020-06-28T07:42:35.585+0000","path":"/payment/lb","status":404,"error":"Not Found","message":null,"requestId":"d7968dd5","trace":"org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND\r\n\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\tSuppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException: \nError has been observed at the following site(s):\n\t|_ checkpoint 鈹?org.springframework.cloud.gateway.filter.WeightCalculatorWebFilter [DefaultWebFilterChain]\n\t|_ checkpoint 鈹?HTTP GET \"/payment/lb?username=sitedev\" [ExceptionHandlerWebHandler]\nStack trace:\r\n\t\tat org.springframework.web.reactive.resource.ResourceWebHandler.lambda$handle$0(ResourceWebHandler.java:325)\r\n\t\tat reactor.core.publisher.MonoDefer.subscribe(MonoDefer.java:44)\r\n\t\tat reactor.core.publisher.Mono.subscribe(Mono.java:4105)\r\n\t\tat reactor.core.publisher.F

```

使用指定的Query, 命令行执行 `curl http://localhost:9527/payment/lb?username=123`

```

C:\Users\qchen>curl http://localhost:9527/payment/lb?username=123
8001
C:\Users\qchen>_

```

6.3.10. RemoteAddr Route Predicate

6.3.10.1. 官方使用说明

The RemoteAddr route predicate factory takes a list (min size 1) of CIDR-notation (IPv4 or IPv6) strings, such as `192.168.0.1/16` (where `192.168.0.1` is an IP address and `16` is a subnet mask). The following example configures a RemoteAddr route predicate:

Example 10. application.yml

```

1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: remoteaddr_route
6           uri: https://example.org
7         predicates:

```

This route matches if the remote address of the request was, for example, `192.168.1.10`.

6.3.11. Weight Route Predicate

6.3.11.1. 官方使用说明

The weight route predicate factory takes two arguments: group and weight. The weights are calculated per group. The following example configures a weight route predicate:

Example 11. application.yml

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: weight_high
6           uri: https://weighthigh.org
7           predicates:
8             - Weight=group1, 8
9         - id: weight_low
10          uri: https://weightlow.org
11          predicates:
12            - Weight=group1, 2
```

This route would forward ~80% of traffic to weighthigh.org and ~20% of traffic to weightlow.org

6.3.12. 小结

目前已测试过的配置

```
1 server:
2   port: 9527
3 spring:
4   application:
5     name: cloud-gateway
6   cloud:
7     gateway:
8       discovery:
```



```

9      locator:
10        enabled: true # 开启从注册中心动态创建路由的功能, 利用微服务名称进行路由
11      routes:
12        - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
13          #匹配后提供服务的路由地址
14          # uri: http://localhost:8001
15          uri: lb://cloud-payment-service
16          predicates:
17            - Path=/payment/get/** # 断言, 路径相匹配的进行路由
18        - id: payment_route2
19          # uri: http://localhost:8001
20          uri: lb://cloud-payment-service
21          predicates:
22            - Path=/payment/lb/** #断言,路径相匹配的进行路由
23            # - After=2020-06-28T15:00:00.000+08:00[Asia/Shanghai] # 指定时间之后才能
24            # - Before=2020-06-28T15:05:00.000+08:00[Asia/Shanghai] # 指定时间之前才能
25            # - Between=2020-06-28T15:05:00.000+08:00[Asia/Shanghai], 2020-06-28T15:0
26            # - Cookie=username, sitedev # cookie中包含指定参数/参数值
27            # - Header=X-Request-Id, \d+ # header中包含指定参数/参数值
28            # - Host=**.sitedev.cn # host为指定的host
29            # - Method=GET # 请求方式为指定方式
30            - Query=username, \d+ # 传参中包含指定参数/参数值
31      eureka:
32        instance:
33          hostname: cloud-gateway-service
34        client: # 服务提供者provider注册进eureka服务列表内
35          service-url:
36            register-with-eureka: true
37            fetch-registry: true
38            defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka

```

说白了,Predicate就是为了实现一组匹配规则, 让请求过来找到对应的Route进行处理

7. Filter的使用

7.1. 是什么

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.1.RELEASE/reference/html/#gatewayfilter-factories>

5. GatewayFilter Factories

Route filters allow the modification of the incoming HTTP request or outgoing HTTP response in some manner. Route filters are scoped to a particular route. Spring Cloud Gateway includes many built-in GatewayFilter Factories.



For more detailed examples of how to use any of the following filters, take a look at the [unit tests](#).

路由过滤器可用于修改进入的HTTP请求和返回的HTTP响应，路由过滤器只能指定路由进行使用。

Spring Cloud Gateway内置了多种路由过滤器，他们都由 GatewayFilter的工厂类来产生

7.2. SpringCloud Gateway的filter

7.2.1. 生命周期(只有2个)

pre

post

7.2.2. 种类(只有2类)

7.2.2.1. GatewayFilter

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.1.RELEASE/reference/html/#gatewayfilter-factories>

达31种之多:

5. GatewayFilter Factories

- 5.1. The AddRequestHeader GatewayFilter Factory
- 5.2. The AddRequestParamter GatewayFilter Factory
- 5.3. The AddResponseHeader GatewayFilter Factory
- 5.4. The DedupeResponseHead GatewayFilter Factory
- 5.5. The Hystrix GatewayFilter Factory
- 5.6. Spring Cloud CircuitBreaker GatewayFilter Factory
- 5.7. The FallbackHeaders GatewayFilter Factory
- 5.8. The MapRequestHeader GatewayFilter Factory
- 5.9. The PrefixPath GatewayFilter Factory
- 5.10. The PreserveHostHeader GatewayFilter Factory

5. GatewayFilter Factories

Route filters allow the modification of the incoming HTTP request or outgoing HTTP response. Filters are scoped to a particular route. Spring Cloud Gateway includes many built-in filters.



For more detailed examples of how to use any of the following filters, see the [Spring Cloud Gateway documentation](#).

5.1. The AddRequestHeader GatewayFilter Factory

The `AddRequestHeader GatewayFilter` factory takes a name and value parameter. The `AddRequestHeader GatewayFilter` is used to add headers to the request.

Example 13. application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id: add_request_header_route
          uri: https://example.org
          filters:
            - AddRequestHeader=X-Request-red, blue
```

This listing adds `X-Request-red:blue` header to the downstream request's headers.

7.2.2.2. GlobalFilter

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.2.1.RELEASE/reference/html/#global-filters>

9种:

Factories

6. Global Filters

6.1. Combined Global Filter and GatewayFilter Ordering

6.2. Forward Routing Filter

6.3. The LoadBalancerClient Filter

6.4. The ReactiveLoadBalancer

6.5. The Netty Routing Filter

6.6. The Netty Write Response Filter

6.7. The RouteToRequestUrl Filter


6.8. The Websocket Routing Filter

6.9. The Gateway Metrics Filter

6.10. Marking An Exchange As Routed

6. Global Filters

The `GlobalFilter` interface has the same signature as `GatewayFilter`, applied to all routes.

 This interface and its usage are subject to change in future

6.1. Combined Global Filter and `GatewayFilter`

When a request matches a route, the filtering web handler adds all instances of `GatewayFilter` to a filter chain. This combined filter chain interface, which you can set by implementing the `getOrder()` method

As Spring Cloud Gateway distinguishes between “pre” and “post” ([Works](#)), the filter with the highest precedence is the first in the “pre”

The following listing configures a filter chain:

Example 56. ExampleConfiguration.java

```
@Bean
```

7.3. 常用的GatewayFilter

7.3.1. AddRequestParameter, AddRequestHeader

AddRequestParameter: 添加请求参数

AddRequestHeader: 添加请求头

7.3.1.1. 修改cloud-gateway-gateway9527模块的yml配置文件

修改内容:

```
1 spring:
2   cloud:
3     gateway:
4       routes:
5         - id: payment_route2
6           uri: lb://cloud-payment-service
7           predicates:
8             - Path=/payment/lb/** #断言,路径相匹配的进行路由
```

```

9      filters:
10         # 添加请求参数
11         - AddRequestParameter=MyParameter, myParameter
12         # 添加请求头
13         - AddRequestHeader=MyHeader, myHeader

```

完整内容:

```

1  server:
2     port: 9527
3  spring:
4     application:
5         name: cloud-gateway
6     cloud:
7         gateway:
8             discovery:
9                 locator:
10                    enabled: true # 开启从注册中心动态创建路由的功能，利用微服务名称进行路由
11        routes:
12            - id: payment_route # 路由的id,没有规定规则但要求唯一,建议配合服务名
13              #匹配后提供服务的路由地址
14              #          uri: http://localhost:8001
15              uri: lb://cloud-payment-service
16              predicates:
17                  - Path=/payment/get/** # 断言，路径相匹配的进行路由
18            - id: payment_route2
19              #          uri: http://localhost:8001
20              uri: lb://cloud-payment-service
21              predicates:
22                  - Path=/payment/lb/** #断言,路径相匹配的进行路由
23                  #          - After=2020-06-28T15:00:00.000+08:00[Asia/Shanghai]
24                  #          - Before=2020-06-28T15:05:00.000+08:00[Asia/Shanghai]
25                  #          - Between=2020-06-28T15:05:00.000+08:00[Asia/Shanghai], 2020-06-28T15:05:00.000+08:00[Asia/Shanghai]
26                  #          - Cookie=username, sitedev
27                  #          - Header=X-Request-Id, \d+
28                  #          - Host=**.sitedev.cn
29                  #          - Method=GET
30                  #          - Query=username, \d+
31        filters:
32            # 添加请求参数
33            - AddRequestParameter=MyParameter, myParameter
34            # 添加请求头

```

```

35         - AddRequestHeader=MyHeader, myHeader
36 eureka:
37     instance:
38         hostname: cloud-gateway-service
39     client: # 服务提供者provider注册进eureka服务列表内
40     service-url:
41         register-with-eureka: true
42         fetch-registry: true
43         defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka
44

```

7.3.1.2. 修改cloud-provider-payment8001模块的PaymentController

```

1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.entities.CommonResult;
4 import cn.sitedev.springcloud.entities.Payment;
5 import cn.sitedev.springcloud.service.PaymentService;
6 import lombok.extern.slf4j.Slf4j;
7 import org.springframework.beans.factory.annotation.Value;
8 import org.springframework.cloud.client.ServiceInstance;
9 import org.springframework.cloud.client.discovery.DiscoveryClient;
10 import org.springframework.web.bind.annotation.*;
11
12 import javax.annotation.Resource;
13 import javax.servlet.http.HttpServletRequest;
14 import java.util.List;
15 import java.util.concurrent.TimeUnit;
16
17 @RestController
18 @Slf4j
19 @RequestMapping("/payment")
20 public class PaymentController {
21     @Resource
22     private PaymentService paymentService;
23
24     /**
25      * 端口号
26      * 查看负载均衡效果
27      */
28     @Value("${server.port}")

```

```

29     private String serverPort;
30
31     /**
32      * 服务发现，获取服务信息
33      */
34     @Resource
35     private DiscoveryClient discoveryClient;
36
37     @PostMapping(value = "/create")
38     public CommonResult create(@RequestBody Payment payment) {
39         int result = paymentService.create(payment);
40         log.info("*****插入结果: " + result);
41         if (result > 0) {
42             return new CommonResult(200, "插入数据库成功, serverPort: " + serverPort);
43         } else {
44             return new CommonResult(444, "插入数据库失败", null);
45         }
46     }
47
48     @GetMapping(value = "/get/{id}")
49     public CommonResult getPaymentById(@PathVariable("id") Long id) {
50         Payment payment = paymentService.getPaymentById(id);
51         log.info("*****查询结果: " + payment);
52         if (payment != null) {
53             return new CommonResult(200, "查询成功, serverPort: " + serverPort, payment);
54         } else {
55             return new CommonResult(444, "没有对应记录, 查询id: " + id, null);
56         }
57     }
58
59     /**
60      * 服务发现
61      *
62      * @return
63      */
64     @GetMapping("/discovery")
65     public Object discovery() {
66         List<String> services = discoveryClient.getServices();
67         for (String element : services) {
68             log.info("*****element: " + element);
69         }
70         // 一个微服务下的全部实例
71         List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-PAYMENT-SERVICE");

```

```

72     for (ServiceInstance instance : instances) {
73         log.info(instance.getServiceId() + "\t" + instance.getHost() + "\t" + instance.getPort());
74     }
75     return discoveryClient;
76 }
77
78 @GetMapping(value = "/lb")
79 public String getPaymentLB(HttpServletRequest request) {
80     String header = request.getHeader("MyHeader");
81     System.out.println("来自gateway9527的header: " + header);
82     String parameter = request.getParameter("MyParameter");
83     System.out.println("来自gateway9527parameter: " + parameter);
84     return serverPort;
85 }
86
87 @GetMapping(value = "/feign/timeout")
88 public String paymentFeignTimeout() {
89     try {
90         TimeUnit.SECONDS.sleep(3);
91     } catch (InterruptedException e) {
92         e.printStackTrace();
93     }
94     return serverPort;
95 }
96 }

```

```

@GetMapping(value = "/lb")
public String getPaymentLB(HttpServletRequest request) {
    String header = request.getHeader("MyHeader");
    System.out.println("来自gateway9527的header: " + header);
    String parameter = request.getParameter("MyParameter");
    System.out.println("来自gateway9527parameter: " + parameter);
    return serverPort;
}

```

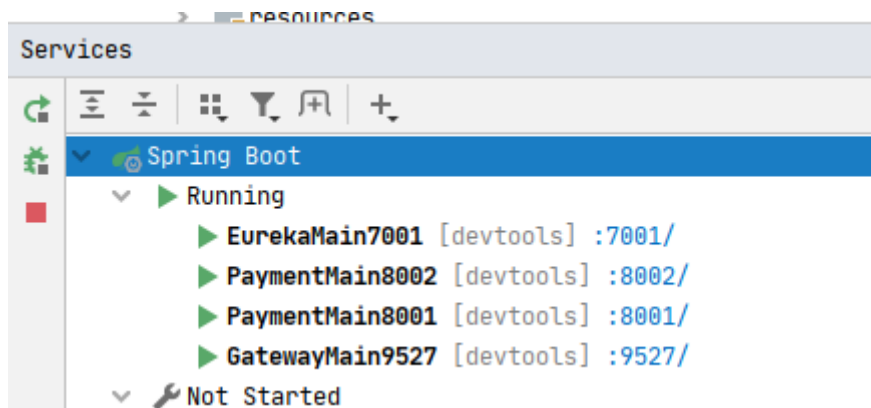
7.3.1.3. 测试

启动EurekaMain7001

启动PaymentMain8002

启动PaymentMain8001

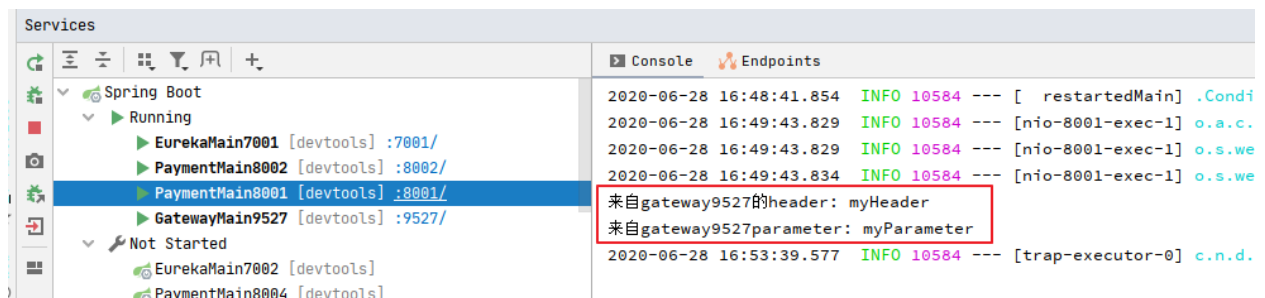
启动GatewayMain9527



浏览器访问: <http://localhost:9527/payment/lb>



当返回结果是8001时, 查看payment8001服务控制台输出:



7.4. 自定义过滤器

7.4.1. 两个主要接口介绍

```
1 implements GlobalFilter, Ordered {
```

GlobalFilter: 全局过滤器接口

Ordered: 排序接口

7.4.2. 能干嘛

全局日志记录

统一网关鉴权

...

7.4.3. 案例代码

在cloud-gateway-gateway9527模块中新增一个bean

...

```
1 package cn.sitedev.springcloud.filter;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.springframework.cloud.gateway.filter.GatewayFilterChain;
5 import org.springframework.cloud.gateway.filter.GlobalFilter;
6 import org.springframework.core.Ordered;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.stereotype.Component;
9 import org.springframework.web.server.ServerWebExchange;
10 import reactor.core.publisher.Mono;
11
12 import java.util.Date;
13
14 /**
15  * 自定义全局过滤器
16  */
17 @Component
18 @Slf4j
19 public class MyLogGatewayFilter implements GlobalFilter, Ordered {
20     @Override
21     public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
22         log.info("*****come in MyLogGatewayFilter: " + new Date());
23         String uname = exchange.getRequest().getQueryParams().getFirst("uname");
24         if (uname == null) {
25             log.info("*****用户名为null, 非法用户...");
26             exchange.getResponse().setStatusCode(HttpStatus.NOT_ACCEPTABLE);
27             return exchange.getResponse().setComplete();
28         }
29         return chain.filter(exchange);
30     }
31
32     @Override
33     public int getOrder() {
34         return 0;
35     }
36 }
```

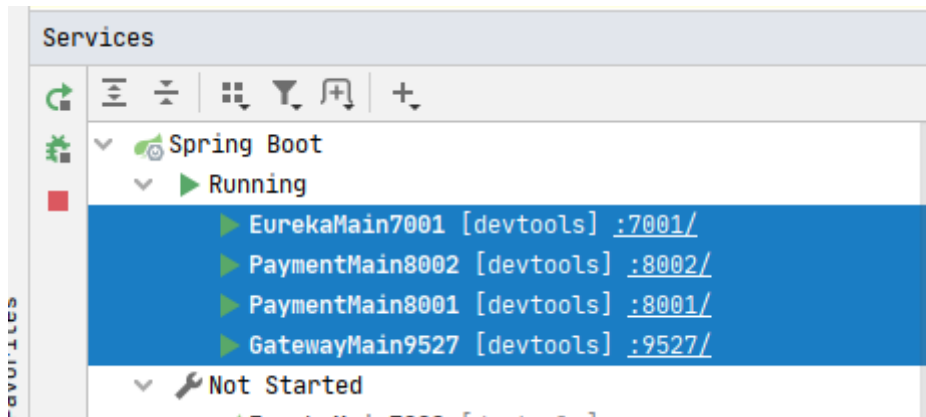
7.4.4. 测试

启动EurekaMain7001

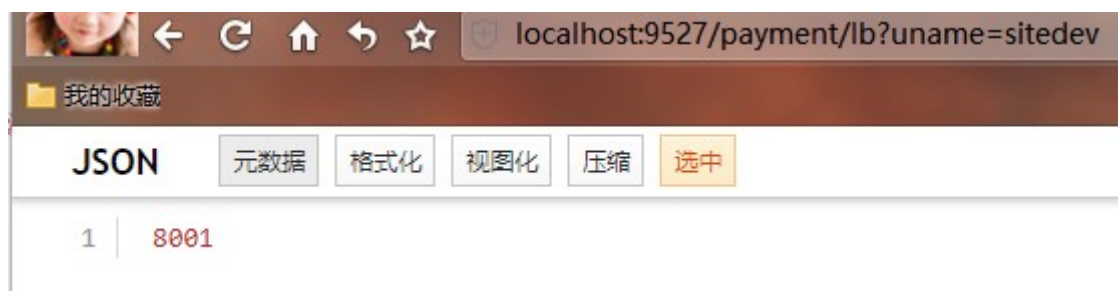
启动PaymentMain8002

启动PaymentMain8001

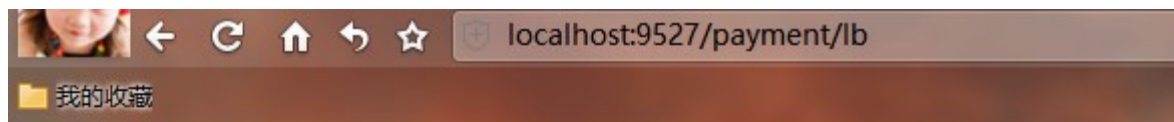
启动GatewayMain9527



测试正常场景: 浏览器访问 <http://localhost:9527/payment/lb?uname=sitedev>



测试异常场景: 浏览器访问 <http://localhost:9527/payment/lb>



该网页无法正常运作

如果问题仍然存在，请与网站所有者联系。

HTTP ERROR 406

重新加载

查看gateway9527服务控制台输出:

```
Spring Boot
├── Running
│   ├── EurekaMain7001 [devtools] :7001/
│   ├── PaymentMain8002 [devtools] :8002/
│   ├── PaymentMain8001 [devtools] :8001/
│   └── GatewayMain9527 [devtools] :9527/
└── Not Started
    ├── EurekaMain7002 [devtools]
    ├── PaymentMain8004 [devtools]
    ├── OrderZkMain80 [devtools]
    ├── PaymentMain8006 [devtools]
    └── OrderConsulMain80 [devtools]

EnabledFilter : Shutdown hook installed for: nrLoadBalancerFilterImpl-cloud-payment-s
useLoadBalancer : Client: cloud-payment-service instantiated a LoadBalancer: DynamicSer
adBalancer : Using serverListUpdater PollingServerListUpdater
ynamicProperty : Flipping property: cloud-payment-service.ribbon.ActiveConnectionsLimi
adBalancer : DynamicServerListLoadBalancer for client cloud-payment-service initia
Successful connection failure:0; Total blackout seconds:0; Last connection made:TI
ection failure:0; Total blackout seconds:0; Last connection made:Thu Jan 01 08:00:1
List@7662774d
ynamicProperty : Flipping property: cloud-payment-service.ribbon.ActiveConnectionsLimi
Filter : *****come in MyLogGatewayFilter: Sun Jun 28 16:11:30 CST 2020
Filter : *****用户名为null, 非法用户...
```