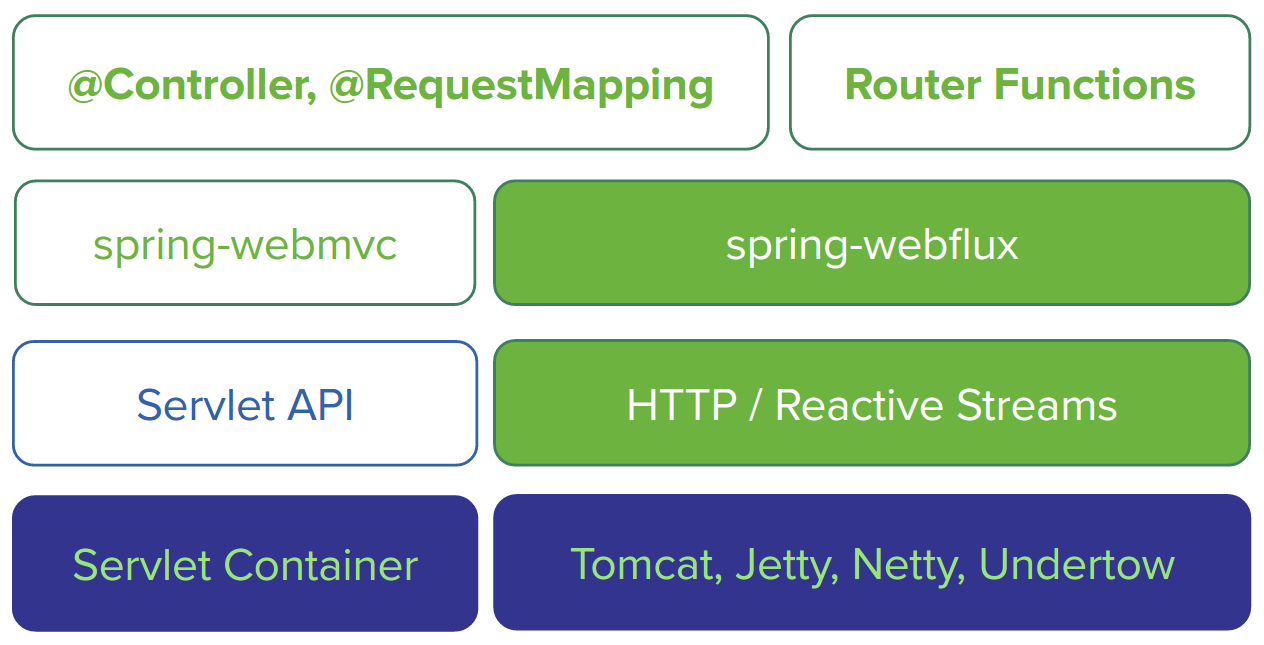
# SpringCloud Gateway

SpringCloud Gateway是建立在Spring生态(Spring 5、Spring Boot 2、Project Reactor)之上的Api网关。旨在提供一种简单有效的方式来对Api进行路由，并提供横切关注点的功能，如：安全校验、监控、统计指标和弹性扩展。

SpringCloud Gateway是基于Netty和WebFlux构建的。

## 0.WebFlux

Spring WebFlux是Spring5推出的响应式Web框架。



1）服务端技术栈

Spring提供了完整的支持响应式的服务端技术栈

如上图所示，左侧为基于spring-webmvc的技术栈，右侧为基于spring-webflux的技术栈，

Spring WebFlux是基于响应式流的，因此可以用来建立异步的、非阻塞的、事件驱动的服务。它采用Reactor作为首选的响应式流的实现库，不过也提供了对RxJava的支持。

由于响应式编程的特性，Spring WebFlux和Reactor底层需要支持异步的运行环境，比如Netty和Undertow；也可以运行在支持异步I/O的Servlet 3.1的容器之上，比如Tomcat（8.0.23及以上）和Jetty（9.0.4及以上）。

从图的纵向上看，spring-webflux上层支持两种开发模式：

类似于Spring WebMVC的基于注解（@Controller、@RequestMapping）的开发模式；

Java 8 lambda 风格的函数式开发模式。

Spring WebFlux也支持响应式的Websocket服务端开发。

2）响应式Http客户端

此外，Spring WebFlux也提供了一个响应式的Http客户端API WebClient。它可以用函数式的方式异步非阻塞地发起Http请求并处理响应。其底层也是由Netty提供的异步支持。

我们可以把WebClient看做是响应式的RestTemplate，与后者相比，前者：

是非阻塞的，可以基于少量的线程处理更高的并发；

可以使用Java 8 lambda表达式；

支持异步的同时也可以支持同步的使用方式；

可以通过数据流的方式与服务端进行双向通信。

Mono和Flux是Reactor提供的响应式模型，Mono 是一个用来发送 0 或者单值数据的发布器，Flux 可以用来发送 0 到 N 个值。

## 1.引入SpringCloud Gateway

在pom文件添加下面启动器：

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

SpringBoot版本：2.0.6.RELEASE

SpringCloud版本：Finchley.SR2

当添加了starter，由于某些原因又不想启动网关，则可以设置：

spring.cloud.gateway.enabled=false

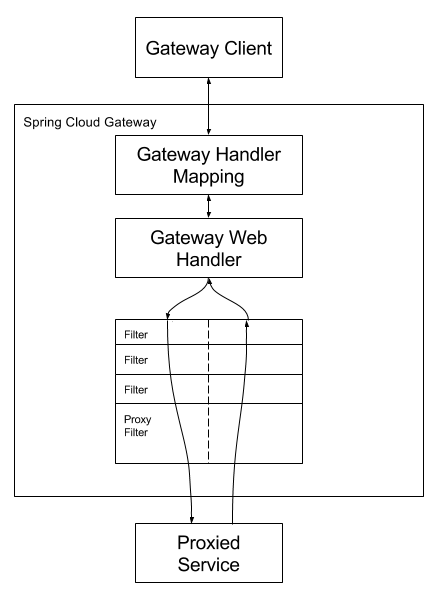
## 2.相关词汇

Route:路由是网关的基本组成，包含id、目标Url、谓词集合和过滤器集合，当聚合谓词返回true时，则匹配到对应路由。

Predicate:是java8的函数谓词,输入参数类型为: org.springframework.web.server. ServerWebExchange,允许开发人员匹配Http请求中的任何内容,如:请求头和参数。

Filter:使用特定工厂类创建的org.springframework.cloud.gateway.filter. GatewayFilter实例,在过滤器中,可以在请求发往下游时，修改请求和响应。

## 3.SpringCloud Gateway是如何运作的



客户端向SpringCloud Gateway发生请求,如果网关的Handler Mapping能够确定请求匹配到了路由,则会将请求发生给网关Web Handler。该处理程序会将请求在过滤器链上执行，过滤器有请求和响应两个阶段。执行完所有的pre类型的过滤器逻辑之后，会进行代理请求，在发出代理请求后，执行post类型的过滤器逻辑。

## 4.路由谓词工厂

SpringCloud Gateway将路由作为Spring WebFlux HandlerMapping的基础部分。Gateway包含了很多内置的Route Predicate Factory。所有的这些Predicate都会匹配Http请求的不同属性。可以组合多个Predicate，最后的结果是逻辑And。

### 4.1 After 路由谓词工厂

After路由谓词工厂需要一个datetime类型的参数,当请求时间发生在之后都会返回true。

spring:

cloud:

gateway:

routes:

- id: after\_route

uri: http://example.org

predicates:

- After=2017-01-20T17:42:47.789-07:00[America/Denver**]**

内置的路由谓词在包：

org.springframework.cloud.gateway.handler.predicate

类似的谓词工厂有：Before、Between

总结：匹配请求时间Before、After、Between

### 4.2 匹配Cookie

Cookie路由谓词工厂需要两个参数：cookie name和正则表达式。当请求含有指定名字的cookie并且值匹配了正则表达式，则返回true：

spring:

cloud:

gateway:

routes:

- id: cookie\_route

uri: http://example.org

predicates:

- Cookie=chocolate**,** ch.p

### 4.3 匹配请求头Header

Header路由谓词工厂需要两个参数：header name和正则表达式。当请求含有指定名字的header并且值匹配了正则表达式，则返回true：

spring:

cloud:

gateway:

routes:

- id: header\_route

uri: http://example.org

predicates:

- Header=X-Request-Id**,** \d+

### 4.4 匹配host

Host路由谓词工厂需要一个参数：host名称的匹配模式，该模式是Ant风格的，. 表示分隔。当请求含有Host的header并且值匹配了指定，则返回true：

spring:

cloud:

gateway:

routes:

- id: host\_route

uri: http://example.org

predicates:

- Host=\*\*.somehost.org

### 4.5 匹配请求方法

Method路由谓词工厂需要一个参数：Http请求方法,当请求方法与配置匹配了，则返回true。

spring:

cloud:

gateway:

routes:

- id: method\_route

uri: http://example.org

predicates:

- Method=GET

### 4.6 匹配路径

Path路由谓词工厂需要两个参数：一个PathMatcher列表和一个可选的matchOptionalTrailingSeparator标志。

spring:

cloud:

gateway:

routes:

- id: host\_route

uri: http://example.org

predicates:

- Path=/foo/{segment},/bar/{segment**}**

上述的定义会将Url路径变量segment，映射到ServerWebExchange的Attributes中，以map的形式存在，这些变量在GatewayFilter Factory中可以使用到：

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

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

### 4.7 查询参数匹配

Query路由谓词工厂需要两个参数：查询参数名称和值的正则模式(可选)

spring:

cloud:

gateway:

routes:

- id: query\_route

uri: http://example.org

predicates:

- Query=foo**,** ba.

上述的配置会匹配到查询参数中含有foo参数,并且参数值为baz，bas的请求。

### 4.8 请求地址匹配

RemoteAddr用于匹配请求的来源地址

spring:

cloud:

gateway:

routes:

- id: remoteaddr\_route

uri: http://example.org

predicates:

- RemoteAddr=192.168.1.1/24

192.168.1.1是ip，24是子网掩码。

上述的匹配会匹配到来之192.168.1.10的请求。

默认情况下，RemoteAddr会使用请求的remote address作为参数进行判断，如果SpringCloud Gateway是在代理层之后，则可能匹配不到真正的客户端ip地址。

可以通过自定义RemoteAddressResolver来解析remote address。SpringCloud Gateway附带了一个非默认的远程ip解析器：XForwardedRemoteAddressResolver基于X-Forwarded-For请求头进行解析。

XForwardedRemoteAddressResolver有两个静态构造函数：trustAll和maxTrustedIndex。trustAll会获取X-Forwarded-For的第一个ip地址作为客户端ip地址，这种方式容易被欺骗，因为客户端可以为该请求头设置一个初始值；maxTrustedIndex可以获取X-Forwarded-For指定下标的值，当请求经过多次代理的时候，X-Forwarded-For会有多个值，如：X-Forwarded-For: 0.0.0.1, 0.0.0.2, 0.0.0.3。

maxTrustedIndex为1时取0.0.0.3,2时取0.0.0.2，大于等于3时取0.0.01.下标是从1开始的。

## 5.GatewayFilter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。 路由过滤器的范围限定为特定路由。SpringCloud Gateway包含许多内置的GatewayFilter工厂。

### 5.1 AddRequestHeader:添加请求头

AddRequestHeader需要一个name和value参数：

spring:

cloud:

gateway:

routes:

- id: add\_request\_header\_route

uri: http://example.org

filters:

- AddRequestHeader=X-Request-Foo**,** Bar

这将为所有匹配请求的下游请求添加X-Request-Foo：Bar请求头。

### 5.2 AddRequestParameter:添加请求参数

AddRequestParameter需要一个name和value参数：

spring:

cloud:

gateway:

routes:

- id: add\_request\_parameter\_route

uri: http://example.org

filters:

- AddRequestParameter=foo**,** bar

这将为所有匹配请求的下游请求添加查询参数:foo=bar。

### 5.3 AddResponseHeader

spring:

cloud:

gateway:

routes:

- id: add\_request\_header\_route

uri: http://example.org

filters:

- AddResponseHeader=X-Response-Foo**,** Bar

### 5.4 Hystrix:熔断过滤器

Hystrix是Netflix的一个库，它实现了断路器模式。 Hystrix GatewayFilter允许您将断路器引入您的网关路由，保护您的服务免受级联故障的影响，并允许您在下游故障时提供回退响应。

要在项目中启用Hystrix GatewayFilters，需要添加对spring-cloud-starter-netflix-hystrix的依赖关系。

Hystrix GatewayFilter Factory需要一个名称参数，该参数是HystrixCommand的名称。

spring:

cloud:

gateway:

routes:

- id: hystrix\_route

uri: http://example.org

filters:

- Hystrix=myCommandName

这将使用命令名myCommandName将剩余的过滤器包装在HystrixCommand中。

Hystrix过滤器还可以接受可选的fallbackUri参数。 目前，仅支持forward：schemed URIs。 如果调用了回退，请求将被转发到与URI匹配的控制器。

spring:

cloud:

gateway:

routes:

- id: hystrix\_route

uri: lb://backing-service:8088

predicates:

- Path=/consumingserviceendpoint

filters:

- name: Hystrix

args:

name: fallbackcmd

fallbackUri: forward:/incaseoffailureusethis

- RewritePath=/consumingserviceendpoint**,** /backingserviceendpoint

当调用Hystrix fallback时，这将转发到/incaseoffailureusethis这个URI。 请注意，此示例还通过目标URI上的lb前缀演示（可选）Spring Cloud Netflix Ribbon负载平衡。

主要方案是将fallbackUri用于网关应用程序内的内部控制器或处理程序。 但是，也可以将请求重新路由到外部应用程序中的控制器或处理程序，如下所示：

spring:

cloud:

gateway:

routes:

- id: ingredients

uri: lb://ingredients

predicates:

- Path=//ingredients/\*\*

filters:

- name: Hystrix

args:

name: fetchIngredients

fallbackUri: forward:/fallback

- id: ingredients-fallback

uri: http://localhost:9994

predicates:

- Path=/fallback

在此示例中，网关应用程序中没有回退端点或处理程序，但是，在http:// localhost:9994下注册的另一个应用程序中有一个。

如果请求被转发到回退，Hystrix网关过滤器还会提供导致它的Throwable。 它作为ServerWebExchangeUtils.HYSTRIX\_EXECUTION\_EXCEPTION\_ATTR属性添加到ServerWebExchange中，可以在处理网关应用程序中的回退时使用该属性。

对于外部控制器/处理程序方案，可以添加包含异常详细信息的请求头。 您可以在FallbackHeaders GatewayFilter Factory部分中找到有关它的更多信息。

Hystrix设置（例如超时）可以使用全局默认值配置，也可以使用应用程序属性逐个路径配置，如Hystrix wiki中所述。

要为上面的示例路由设置5秒超时，将使用以下配置：

hystrix.command.fallbackcmd.execution.isolation.thread.timeoutInMilliseconds: 5000

### 5.5 FallbackHeaders

FallbackHeaders工厂允许您在转发到外部应用程序中的fallbackUri的请求的请求头中添加Hystrix执行异常详细信息，如下所示：

spring:

cloud:

gateway:

routes:

- id: ingredients

uri: lb://ingredients

predicates:

- Path=//ingredients/\*\*

filters:

- name: Hystrix

args:

name: fetchIngredients

fallbackUri: forward:/fallback

- id: ingredients-fallback

uri: http://localhost:9994

predicates:

- Path=/fallback

filters:

- name: FallbackHeaders

args:

executionExceptionTypeHeaderName: Test-Header

在此示例中，在运行HystrixCommand时发生执行异常后，请求将转发到在localhost：9994上运行的应用程序中的回退端点或处理程序。 带有异常类型，消息和rootcause的请求头会被FallbackHeaders过滤器添加到该请求中。

通过设置下面列出的参数的值及其默认值，可以在配置中覆盖请求头的名称：

executionExceptionTypeHeaderName ("Execution-Exception-Type")

executionExceptionMessageHeaderName ("Execution-Exception-Message")

rootCauseExceptionTypeHeaderName ("Root-Cause-Exception-Type")

rootCauseExceptionMessageHeaderName ("Root-Cause-Exception-Message")

您可以在Hystrix GatewayFilter Factory部分中找到有关Hystrix如何与Gateway配合使用的更多信息。

### 5.6 PrefixPath过滤器

PrefixPath GatewayFilter Factory需要prefix前缀参数:

spring:

cloud:

gateway:

routes:

- id: prefixpath\_route

uri: http://example.org

filters:

- PrefixPath=/mypath

这将添加/mypath前缀到所有匹配的的请求路径。所以对/hello的请求会被发送到/mypath/hello。

### 5.7 PreserveHostHeader

PreserveHostHeader GatewayFilter Factory没有参数。 此过滤器设置路由过滤器将检查的请求属性，以确定是否应发送原始主机头，而不是http客户端确定的主机头。

spring:

cloud:

gateway:

routes:

- id: preserve\_host\_route

uri: http://example.org

filters:

- PreserveHostHeader

### 5.8 RequestRateLimiter 请求限制过滤

RequestRateLimiter GatewayFilter Factory使用RateLimiter实现来确定是否允许当前请求继续。 如果不是，则返回HTTP 429 - Too Many Requests（默认情况下）的状态。

此过滤器采用可选的keyResolver参数和特定于速率限制器的参数（参见下文）。

keyResolver是一个实现KeyResolver接口的bean。 在配置中，使用SpEL按名称引用bean。 ＃{@ myKeyResolver}是一个引用名为myKeyResolver的bean的SpEL表达式。

**public** **interface** KeyResolver {

Mono<String> resolve(ServerWebExchange exchange);

}

KeyResolver接口允许可插拔策略派生用于限制请求的key。 在未来的里程碑中，将会有一些KeyResolver实现。

KeyResolver的默认实现是PrincipalNameKeyResolver，它从ServerWebExchange检索Principal并调用Principal.getName（）。

默认情况下，如果KeyResolver未找到key，则会拒绝请求。 可以使用spring.cloud.gateway.filter.request-rate-limiter.deny-empty-key（true或false）和spring.cloud.gateway.filter.request-rate-limiter.empty-key-来调整此行为。

RequestRateLimiter不能通过“快捷方式”表示法进行配置。 以下示例无效

spring.cloud.gateway.routes[0].filters[0]=RequestRateLimiter=2, 2, #{@userkeyresolver}

#### Redis RateLimiter

redis实现基于Stripe完成的工作。 它需要使用spring-boot-starter-data-redis-active Spring Boot启动器。

使用的算法是令牌桶算法。

redis-rate-limiter.replenishRate是您希望允许用户每秒执行多少请求，而不会丢弃任何请求。 这是令牌桶填充的速率。

redis-rate-limiter.burstCapacity是用户在一秒钟内允许执行的最大请求数。 这是令牌桶可以容纳的令牌数。 将此值设置为零将阻止所有请求。

通过在replenishRate和burstCapacity中设置相同的值来实现稳定的速率。 通过将burstCapacity设置为高于replenishRate，可以允许临时突发。 在这种情况下，需要在突发之间允许速率限制器一段时间（根据replenishRate），因为2个连续的突发将导致请求被丢弃（HTTP 429 - Too Many Requests）。

spring:

cloud:

gateway:

routes:

- id: requestratelimiter\_route

uri: http://example.org

filters:

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 10

redis-rate-limiter.burstCapacity: 20

*@Bean*

KeyResolver userKeyResolver() {

**return** exchange -> Mono.just(exchange.getRequest().getQueryParams().getFirst("user"));

}

这定义了每个用户10的请求率限制。 允许突发20，但下一秒只有10个请求可用。 KeyResolver是一个简单的获取用户请求参数（注意：这不建议用于生产）。

速率限制器也可以定义为实现RateLimiter接口的bean。 在配置中，使用SpEL按名称引用bean。 ＃{@ myRateLimiter}是一个引用名为myRateLimiter的bean的SpEL表达式。

spring:

cloud:

gateway:

routes:

- id: requestratelimiter\_route

uri: http://example.org

filters:

- name: RequestRateLimiter

args:

rate-limiter: "#{@myRateLimiter}"

key-resolver: "#{@userKeyResolver}"

### 5.9 RedirectTo

RedirectTo GatewayFilter Factory采用status和url参数。 状态应该是300系列重定向http代码，例如301. url应该是有效的URL。 这将是Location请求头的值。

spring:

cloud:

gateway:

routes:

- id: prefixpath\_route

uri: http://example.org

filters:

- RedirectTo=302**,** http://acme.org

这将发送带有Location：http：//acme.org header,响应码为30的重定向。

### 5.10 RemoveNonProxyHeaders

RemoveNonProxyHeaders GatewayFilter Factory从转发的请求中删除header。 删除的默认标头列表来自IETF。

默认删除的header有：

* Connection
* Keep-Alive
* Proxy-Authenticate
* Proxy-Authorization
* TE
* Trailer
* Transfer-Encoding
* Upgrade

要更改此设置，请将spring.cloud.gateway.filter.remove-non-proxy-headers.headers属性设置为要删除的标头名称列表。

### 5.11 RemoveRequestHeader

RemoveRequestHeader GatewayFilter Factory需要一个name参数。 它是要删除的request header的名称。

spring:

cloud:

gateway:

routes:

- id: removerequestheader\_route

uri: http://example.org

filters:

- RemoveRequestHeader=X-Request-Foo

这将在向下游发送之前删除X-Request-Foo 请求头。

### 5.12 RemoveResponseHeader

spring:

cloud:

gateway:

routes:

- id: removeresponseheader\_route

uri: http://example.org

filters:

- RemoveResponseHeader=X-Response-Foo

### 5.13 RewritePath

RewritePath GatewayFilter Factory采用路径regexp参数和替换参数。 这使用Java正则表达式来灵活地重写请求路径。

spring:

cloud:

gateway:

routes:

- id: rewritepath\_route

uri: http://example.org

predicates:

- Path=/foo/\*\*

filters:

- RewritePath=/foo/(?<segment>.\*)**,** /$\{segment**}**

对于/foo/bar的请求路径，这将在发出下游请求之前将路径设置为/bar。 注意由于YAML规范，$\替换为$。

### 5.14 RewriteResponseHeader

RewriteResponseHeader GatewayFilter Factory采用名称，正则表达式和替换参数。 它使用Java正则表达式以灵活的方式重写响应头值。

spring:

cloud:

gateway:

routes:

- id: rewriteresponseheader\_route

uri: http://example.org

filters:

- RewriteResponseHeader=X-Response-Foo**,** **,** password=[^&]+**,** password=\*\*\*

对于header值/42user=ford＆password=omg!what＆flag=true，在发出下游请求后，它将被设置为/42?user=ford＆password=\*\*\*＆flag=true。 由于YAML规范，请使用$\表示$。

### 5.15 SaveSession

SaveSession GatewayFilter Factory在转发下游调用之前强制执行WebSession::save操作。 当使用类似Spring Session和惰性数据存储的东西时，这是特别有用的，并且需要确保在转发调用之前已保存会话状态。

spring:

cloud:

gateway:

routes:

- id: save\_session

uri: http://example.org

predicates:

- Path=/foo/\*\*

filters:

- SaveSession

如果要将Spring Security与Spring Session集成，并且希望确保将安全性详细信息转发到远程进程，则这很关键。

### 5.16 SecureHeaders

SecureHeaders GatewayFilter Factory为响应添加了许多header。

这些header包括:

* X-Xss-Protection:1; mode=block
* Strict-Transport-Security:max-age=631138519
* X-Frame-Options:DENY
* X-Content-Type-Options:nosniff
* Referrer-Policy:no-referrer
* Content-Security-Policy:default-src 'self' https:; font-src 'self' https: data:; img-src 'self' https: data:; object-src 'none'; script-src https:; style-src 'self' https: 'unsafe-inline'
* X-Download-Options:noopen
* X-Permitted-Cross-Domain-Policies:none

要更改默认值，请在spring.cloud.gateway.filter.secure-headers命名空间中设置相应的属性：

可以更改的属性：

* xss-protection-header
* strict-transport-security
* frame-options
* content-type-options
* referrer-policy
* content-security-policy
* download-options
* permitted-cross-domain-policies

### 5.17 SetPath

SetPath GatewayFilter Factory采用路径模板参数。 它提供了一种通过允许模板化路径段来操作请求路径的简单方法。 这使用了Spring Framework中的uri模板。 允许多个匹配的段

spring:

cloud:

gateway:

routes:

- id: setpath\_route

uri: http://example.org

predicates:

- Path=/foo/{segment**}**

filters:

- SetPath=/{segment**}**

对于/foo/bar的请求路径，这将在发出下游请求之前将路径设置为/bar。

### 5.18 SetResponseHeader

SetResponseHeader GatewayFilter Factory需要name和value参数。

spring:

cloud:

gateway:

routes:

- id: setresponseheader\_route

uri: http://example.org

filters:

- SetResponseHeader=X-Response-Foo**,** Bar

此GatewayFilter将替换具有给定名称的所有header，而不是添加。 因此，如果下游服务器以X-Response-Foo：1234响应，则将替换为X-Response-Foo：Bar，这是网关客户端将接收的内容。

### 5.19 SetStatus

SetStatus GatewayFilter Factory采用单个状态参数。 它必须是有效的Spring HttpStatus。 它可以是整数值404或枚举NOT\_FOUND的字符串表示。

spring:

cloud:

gateway:

routes:

- id: setstatusstring\_route

uri: http://example.org

filters:

- SetStatus=BAD\_REQUEST

- id: setstatusint\_route

uri: http://example.org

filters:

- SetStatus=401

在任何一种情况下，响应的HTTP状态都将设置为401。

### 5.20 StripPrefix

StripPrefix GatewayFilter Factory采用一个参数，即parks。 parts参数指示在向下游发送之前从请求中剥离的路径中的部分数。

spring:

cloud:

gateway:

routes:

- id: nameRoot

uri: http://nameservice

predicates:

- Path=/name/\*\*

filters:

- StripPrefix=2

当通过网关向/name/bar/foo发出请求时，对nameservice的请求将类似于http://nameservice/foo。

### 5.21 Retry

Retry GatewayFilter Factory将retries，statuses，methods和series作为参数。

retries: 应尝试的重试次数

statuses: 应该重试的HTTP状态代码，使用

org.springframework.http.HttpStatus表示

statuses: 应该重试的HTTP方法，使用org.springframework.http.HttpMethod表示

series: 要重试的一系列状态代码，使用

org.springframework.http.HttpStatus.Series表示

spring:

cloud:

gateway:

routes:

- id: retry\_test

uri: http://localhost:8080/flakey

predicates:

- Host=\*.retry.com

filters:

- name: Retry

args:

retries: 3

statuses: BAD\_GATEWAY

重试过滤器当前不支持使用正文重试（例如，对于具有正文的POST或PUT请求）。

当使用带有转发前缀URL的重试过滤器时，应仔细编写目标端点，以便在出现错误时不会执行任何可能导致响应发送到客户端并提交的操作。 例如，如果目标端点是带注释的控制器，则目标控制器方法不应返回带有错误状态代码的ResponseEntity。 相反，它应抛出异常，或发出错误信号，例如 通过Mono.error（ex）返回值，可以将重试过滤器配置为通过重试来处理。

### 5.22 RequestSize

当请求大小大于允许的限制时，RequestSize GatewayFilter Factory可以限制请求到达下游服务。 过滤器将RequestSize作为参数，该参数是以字节为单位定义的请求的允许大小限制。

spring:

cloud:

gateway:

routes:

- id: request\_size\_route

uri: http://localhost:8080/upload

predicates:

- Path=/upload

filters:

- name: RequestSize

args:

maxSize: 5000000

当请求因大小而被拒绝时，RequestSize GatewayFilter Factory将响应状态设置为413 Payload Too Large，并附加标头errorMessage。 以下是此类errorMessage的示例。

errorMessage : Request size is larger than permissible limit. Request size is 6.0 MB where permissible limit is 5.0 MB

如果未在路由定义中提供过滤器参数，则默认请求大小将设置为5 MB

### 5.23 Modify Request Body

此过滤器被视为BETA，API可能在将来发生变化.

此过滤器可用于在网关向下游发送请求主体之前对其进行修改。

*@Bean*

**public** RouteLocator routes(RouteLocatorBuilder builder) {

**return** builder.routes()

.route("rewrite\_request\_obj", r -> r.host("\*.rewriterequestobj.org")

.filters(f -> f.prefixPath("/httpbin")

.modifyRequestBody(String.**class**, Hello.**class**, MediaType.APPLICATION\_JSON\_VALUE,

(exchange, s) -> **return** Mono.just(**new** Hello(s.toUpperCase())))).uri(uri))

.build();

}

**static** **class** Hello {

String message;

**public** Hello() { }

**public** Hello(String message) {

**this**.message = message;

}

**public** String getMessage() {

**return** message;

}

**public** **void** setMessage(String message) {

**this**.message = message;

}

}

### 5.24 Modify Response Body

此过滤器被视为BETA，API可能在将来发生变化

此过滤器可用于在将响应主体发送回客户端之前对其进行修改。

此过滤器只能使用Java DSL进行配置

*@Bean*

**public** RouteLocator routes(RouteLocatorBuilder builder) {

**return** builder.routes()

.route("rewrite\_response\_upper", r -> r.host("\*.rewriteresponseupper.org")

.filters(f -> f.prefixPath("/httpbin")

.modifyResponseBody(String.**class**, String.**class**,

(exchange, s) -> Mono.just(s.toUpperCase()))).uri(uri)

.build();

}

## 6 全局过滤器

GlobalFilter接口与GatewayFilter具有相同的签名。 这些是有条件地应用于所有路由的特殊过滤器。（此接口和用法可能会在未来的里程碑中发生变化）。

### 6.1 组合全局过滤器和GatewayFilter

当请求进入（并匹配路由）时，Filtering Web Handler会将GlobalFilter的所有实例和GatewayFilter的所有实例添加到过滤器链中。 这个组合的过滤器链由org.springframework.core.Ordered接口排序，可以通过实现getOrder（）方法或使用@Order注释来设置。

由于Spring Cloud Gateway区分了过滤器逻辑执行的“前”和“后”阶段（请参阅：工作原理），具有最高优先级的过滤器将是“前”阶段中的第一个和“后”阶段中的最后一个。

*@Bean*

*@Order(-1)*

**public** GlobalFilter a() {

**return** (exchange, chain) -> {

log.info("first pre filter");

**return** chain.filter(exchange).then(Mono.fromRunnable(() -> {

log.info("third post filter");

}));

};

}

*@Bean*

*@Order(0)*

**public** GlobalFilter b() {

**return** (exchange, chain) -> {

log.info("second pre filter");

**return** chain.filter(exchange).then(Mono.fromRunnable(() -> {

log.info("second post filter");

}));

};

}

*@Bean*

*@Order(1)*

**public** GlobalFilter c() {

**return** (exchange, chain) -> {

log.info("third pre filter");

**return** chain.filter(exchange).then(Mono.fromRunnable(() -> {

log.info("first post filter");

}));

};

}

### 6.2 Forward Routing Filter

ForwardRoutingFilter在exchange attribute ServerWebExchangeUtils.GATEWAY\_REQUEST\_URL\_ATTR中查找URI。 如果url具有forward方案（即forward:///localendpoint），它将使用Spring DispatcherHandler来处理请求。

请求URL的路径部分将被转发URL中的路径覆盖。 未修改的原始URL将附加到ServerWebExchangeUtils.GATEWAY\_ORIGINAL\_REQUEST\_URL\_ATTR属性中的列表中。

### 6.3 LoadBalancerClient Filter

LoadBalancerClientFilter在exchange attribute ServerWebExchangeUtils.GATEWAY\_REQUEST\_URL\_ATTR中查找URI。 如果url具有lb方案（即lb：// myservice），它将使用Spring Cloud LoadBalancerClient将名称（前一示例中的myservice）解析为实际主机和端口，并替换相同属性中的URI。

未修改的原始URL将附加到ServerWebExchangeUtils.GATEWAY\_ORIGINAL\_REQUEST\_URL\_ATTR属性中的列表中。 过滤器还将查看ServerWebExchangeUtils.GATEWAY\_SCHEME\_PREFIX\_ATTR属性以查看它是否等于lb，然后应用相同的规则。

spring:

cloud:

gateway:

routes:

- id: myRoute

uri: lb://service

predicates:

- Path=/service/\*\*

默认情况下，当在LoadBalancer中找不到服务实例时，将返回503。 您可以通过设置spring.cloud.gateway.loadbalancer.use404 = true来配置网关以返回404。

从LoadBalancer返回的ServiceInstance的isSecure值将覆盖在对网关发出的请求中指定的方案。 例如，如果请求通过HTTPS进入网关但ServiceInstance指示它不安全，则下游请求将通过HTTP进行。 相反的情况也适用。 但是，如果为网关配置中的路由指定了GATEWAY\_SCHEME\_PREFIX\_ATTR，则将剥离前缀，并且路由URL中的结果方案将覆盖ServiceInstance配置。

### 6.4 Netty Routing Filter

如果位于ServerWebExchangeUtils.GATEWAY\_REQUEST\_URL\_ATTR属性中的URL具有http或https方案，则运行Netty路由过滤器。 它使用Netty HttpClient发出下游代理请求。 响应放在ServerWebExchangeUtils.CLIENT\_RESPONSE\_ATTR属性中，以便在以后的过滤器中使用。 （有一个实验性的WebClientHttpRoutingFilter执行相同的功能，但不需要netty）

### 6.5 Netty Write Response Filter

如果ServerWebExchangeUtils.CLIENT\_RESPONSE\_ATTR属性中存在Netty HttpClientResponse，则运行NettyWriteResponseFilter。 它在所有其他过滤器完成后运行，并将代理响应写回网关客户端响应。 （有一个实验性的WebClientWriteResponseFilter执行相同的功能，但不需要netty）

### 6.6 RouteToRequestUrl Filter

如果ServerWebExchangeUtils.GATEWAY\_ROUTE\_ATTR属性中存在Route对象，则运行RouteToRequestUrlFilter。 它根据请求URI创建一个新URI，但使用Route对象的URI属性进行更新。 新URI放在ServerWebExchangeUtils.GATEWAY\_REQUEST\_URL\_ATTR属性中。

如果URI具有方案前缀，例如lb：ws：// serviceid，则lb方案将从URI中剥离并放置在ServerWebExchangeUtils.GATEWAY\_SCHEME\_PREFIX\_ATTR中，以便稍后在过滤器链中使用。

### 6.7 Websocket Routing Filter

如果位于ServerWebExchangeUtils.GATEWAY\_REQUEST\_URL\_ATTR交换属性中的URL具有ws或wss方案，则运行Websocket路由过滤器。 它使用Spring Web Socket基础结构将Websocket请求转发到下游。

可以通过在URI前面添加lb来对Websockets进行负载平衡，例如lb：ws：// serviceid。

如果您使用SockJS作为普通http的后备，则应配置正常的HTTP路由以及Websocket路由。

spring:

cloud:

gateway:

routes:

*# SockJS route*

- id: websocket\_sockjs\_route

uri: http://localhost:3001

predicates:

- Path=/websocket/info/\*\*

*# Normwal Websocket route*

- id: websocket\_route

uri: ws://localhost:3001

predicates:

- Path=/websocket/\*\*

### 6.8 Gateway Metrics Filter

要启用Gateway Metrics，请将spring-boot-starter-actuator添加为项目依赖项。 然后，默认情况下，只要属性spring.cloud.gateway.metrics.enabled未设置为false，网关度量标准筛选器就会运行。 此过滤器添加名为“gateway.requests”的计时器度量标准，其中包含以下标记：

* routeId: The route id
* routeUri: The URI that the API will be routed to
* outcome: Outcome as classified by HttpStatus.Series
* status: Http Status of the request returned to the client

然后可以从/actuator/metrics/gateway.requests中删除这些指标，并可以轻松地与Prometheus集成以创建Grafana仪表板。

要启用pometheus端点，请将micrometer-registry-prometheus添加为项目依赖项。

### 6.9 Making An Exchange As Routed

在Gateway路由了ServerWebExchange之后，它会通过将gatewayAlreadyRouted添加到exchange属性来将该交换标记为“路由”。 一旦请求被标记为路由，其他路由过滤器将不会再次路由请求，实质上是跳过过滤器。 您可以使用便捷方法将交换标记为路由，或检查交换是否已路由。

* ServerWebExchangeUtils.isAlreadyRouted 接受ServerWebExchange对象并检查它是否已“路由”
* ServerWebExchangeUtils.setAlreadyRouted获取ServerWebExchange对象并将其标记为“路由”

## 7. TLS / SSL

可以通过下面的配置使网关可以来监听https上的请求。 例：

server:

ssl:

enabled: **true**

key-alias: scg

key-store-password: scg1234

key-store: classpath:scg-keystore.p12

key-store-type: PKCS12

网关路由可以路由到http和https后端。 如果路由到https后端，则可以将网关配置为信任具有以下配置的所有下游证书：

spring:

cloud:

gateway:

httpclient:

ssl:

useInsecureTrustManager: **true**

使用不安全的信任管理器不适合生产。 对于生产部署，可以使用以下配置配置一组可信任的已知证书：

spring:

cloud:

gateway:

httpclient:

ssl:

trustedX509Certificates:

- cert1.pem

- cert2.pem

如果Spring Cloud Gateway未配置可信证书，则使用默认信任库（可以使用系统属性javax.net.ssl.trustStore覆盖）。

### 7.1 TLS Handshake TLS握手

网关维护一个客户端池，用于路由到后端。 通过https进行通信时，客户端会启动TLS握手。 这次握手会有很多超时。 可以配置这些超时（显示默认值）：

spring:

cloud:

gateway:

httpclient:

ssl:

handshake-timeout-millis: 10000

close-notify-flush-timeout-millis: 3000

close-notify-read-timeout-millis: 0

## 8 配置