Spring Cloud - *Gateway*

Spring Cloud - Gateway

- Spring Cloud Gateway, Spring WebFlux'un üzerine bir **API Ağ Geçidi(API Gateway)** oluşturmak için bir kitaplık sağlar.
- Spring Cloud Gateway, API'lere yönlendirmek için basit ama etkili bir yol sağlamayı ve onlara güvenlik, izleme/ölçümler ve esneklik gibi kesişen endişeler sağlamayı amaçlar.

Spring Cloud – Gateway Özellikleri

- Spring Framework 5, Project Reactor ve Spring Boot 2.0 üzerine inşa edilmiştir
- Herhangi bir istek özniteliğindeki rotaları eşleştirebilir
- Yüklemler ve filtreler rotalara özeldir
- Devre Kesici entegrasyonu
- Spring Cloud DiscoveryClient entegrasyonu
- Yüklemler ve Filtreler yazmak kolay
- İstek-Talep Oranı Sınırlaması
- Yol Yeniden Yazma

```java
@SpringBootApplication
public class DemogatewayApplication {
	@Bean
	public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
		return builder.routes()
			.route("path_route", r -> r.path("/get")
				.uri("http://httpbin.org"))
			.route("host_route", r -> r.host("*.myhost.org")
				.uri("http://httpbin.org"))
			.route("rewrite_route", r -> r.host("*.rewrite.org")
				.filters(f -> f.rewritePath("/foo/(?<segment>.*)", "/${segment}"))
				.uri("http://httpbin.org"))
			.route("hystrix_route", r -> r.host("*.hystrix.org")
				.filters(f -> f.hystrix(c -> c.setName("slowcmd")))
				.uri("http://httpbin.org"))
			.route("hystrix_fallback_route", r -> r.host("*.hystrixfallback.org")
				.filters(f -> f.hystrix(c -> c.setName("slowcmd").setFallbackUri("forward:/hystrixfallback")))
				.uri("http://httpbin.org"))
			.route("limit_route", r -> r
				.host("*.limited.org").and().path("/anything/**")
				.filters(f -> f.requestRateLimiter(c -> c.setRateLimiter(redisRateLimiter())))
				.uri("http://httpbin.org"))
			.build();
	}
}
```

COPY

Kendi ağ geçidinizi(gateway) çalıştırmak için **spring-cloud-starter-gateway** dependency kullanın.

Kod:


```java
@SpringBootApplication
public class DemogatewayApplication {
	@Bean
	public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
		return builder.routes()
			.route("path_route", r -> r.path("/get")
				.uri("http://httpbin.org"))
			.route("host_route", r -> r.host("*.myhost.org")
				.uri("http://httpbin.org"))
			.route("rewrite_route", r -> r.host("*.rewrite.org")
				.filters(f -> f.rewritePath("/foo/(?<segment>.*)", "/${segment}"))
				.uri("http://httpbin.org"))
			.route("hystrix_route", r -> r.host("*.hystrix.org")
				.filters(f -> f.hystrix(c -> c.setName("slowcmd")))
				.uri("http://httpbin.org"))
			.route("hystrix_fallback_route", r -> r.host("*.hystrixfallback.org")
				.filters(f -> f.hystrix(c -> c.setName("slowcmd").setFallbackUri("forward:/hystrixfallback")))
				.uri("http://httpbin.org"))
			.route("limit_route", r -> r
				.host("*.limited.org").and().path("/anything/**")
				.filters(f -> f.requestRateLimiter(c -> c.setRateLimiter(redisRateLimiter())))
				.uri("http://httpbin.org"))
			.build();
	}
}
```

LEARN-ÖĞREN

1. How to Include Spring Cloud Gateway (Spring Cloud Gateway Nasıl Dahil Edilir)

Spring Cloud Gateway'i projenize dahil etmek için **org.springframework.cloud** group ID ve **spring-cloud-starter-gateway** artifact ID başlatıcıyı kullanın. Mevcut Spring Cloud Release Train ile yapı sisteminizi kurma hakkında ayrıntılar için Spring Cloud Project sayfasına bakın.

Eğer başlatıcıyı eklediyseniz ancak gateway'ın etkinleştirilmesini istemiyorsanız, **spring.cloud.gateway.enabled=false** olarak ayarlayın.

Spring Cloud Gateway, Spring Boot 2.x, Spring WebFlux ve Project Reactor üzerine kurulmuştur. Sonuç olarak, bilinen birçok eşzamanlı kitaplık (örneğin, Spring Data ve Spring Security) ve bildiğiniz kalıplar, Spring Cloud Gateway kullandığınızda geçerli olmayabilir. Bu projelere aşina değilseniz, Spring Cloud Gateway ile çalışmadan önce bazı yeni kavramlara aşina olmak için belgelerini okuyarak başlamanızı öneririz.

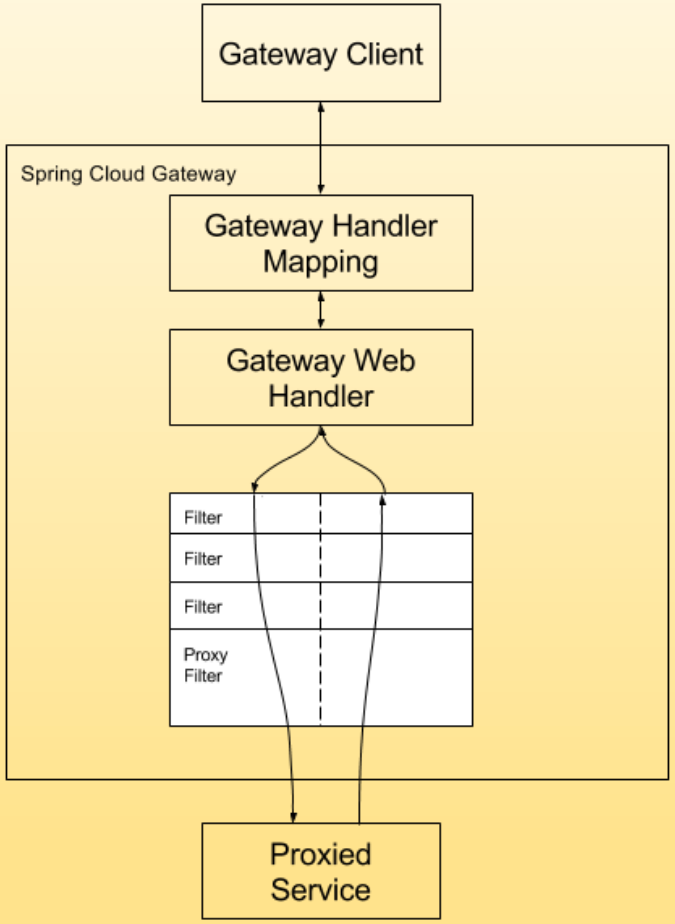
Spring Cloud Gateway, Spring Boot ve Spring Webflux tarafından sağlanan Netty çalışma zamanını gerektirir. Geleneksel bir Servlet Konteynerinde veya bir WAR olarak inşa edildiğinde çalışmaz.

2. Glossary (Sözlük)

- **Route:** Ağ geçidinin temel yapı taşı. Bir ID, bir hedef URI, bir tahminler koleksiyonu ve bir filtre koleksiyonu ile tanımlanır. Toplama yüklemi doğruysa bir rota eşleştirilir.
- **Predicate:** Bu bir Java 8 İşlev Yüklemidir. Giriş türü, Spring Framework **ServerWebExchange**'dir. Bu, üstbilgiler veya parametreler gibi HTTP isteğindeki herhangi bir şeyi eşleştirmenize olanak tanır.
- **Filter:** Bunlar, belirli bir fabrika ile oluşturulmuş **GatewayFilter** örnekleridir. Burada, aşağı akış isteğini göndermeden önce veya sonra istekleri ve yanıtları değiştirebilirsiniz.

3. How It Works (Nasıl Çalışır)

Aşağıdaki şema, Spring Cloud Gateway'in nasıl çalıştığına dair üst düzey bir genel bakış sağlar:



Client'ler Spring Cloud Gateway'e istekte bulunur. Gateway Handler Mapping, bir isteğin bir yolla eşleştiğini belirlerse, Gateway Web Handler'e gönderilir. Bu handler(işleyici), isteği, isteğe özel bir filtre zinciri aracılığıyla çalıştırır. Filtrelerin noktalı çizgiyle bölünmesinin nedeni, filtrelerin proxy isteği gönderilmeden önce ve gönderildikten sonra mantığı çalıştırabilmesidir. Tüm " pre« (ön) filtre mantığı yürütülür. Ardından proxy isteği yapılır. Proxy isteği yapıldıktan sonra "post" filtre mantığı çalıştırılır.

Bağlantı noktası olmayan rotalarda tanımlanan URI'ler, HTTP ve HTTPS URI'leri için sırasıyla 80 ve 443 varsayılan bağlantı noktası değerlerini alır.

4. Configuring Route Predicate Factories and Gateway Filter Factories

Tahminleri ve filtreleri yapılandırmanın iki yolu vardır: kısayollar ve tamamen genişletilmiş argümanlar. Aşağıdaki örneklerin çoğu kısayol yolunu kullanır.

Ad ve argüman adları, her bölümün ilk cümlesinde veya ikisinde **code** olarak listelenecektir. Bağımsız değişkenler, genellikle kısayol yapılandırması için gerekli olan sırayla listelenir.

4.1. Shortcut Configuration

Kısayol yapılandırması, filtre adıyla tanınır, ardından eşittir işareti (**=**), ardından virgülle (**,**) ayrılmış bağımsız değişken değerleri gelir.

application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: after_route
        uri: https://example.org
        predicates:
        - Cookie=mycookie,mycookievalue
```

Önceki örnek, iki bağımsız değişkenle Cookie Route Predicate Factory'yi tanımlar. **Cookie** ismi, **mycookie** ve **mycookievalue** ile eşleşecek değer.

4.2. Fully Expanded Arguments

Tamamen genişletilmiş bağımsız değişkenler, name/value çiftleriyle daha çok standart yaml yapılandırması gibi görünür. Tipik olarak, bir **name** anahtarı ve bir **args** anahtarı olacaktır. **args** anahtarı, yüklemi veya filtreyi yapılandırmak için anahtar değer çiftlerinin bir haritasıdır.

application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: after_route
        uri: https://example.org
        predicates:
        - name: Cookie
          args:
            name: mycookie
            regexp: mycookievalue
```

Bu, yukarıda gösterilen **Cookie** yükleminin kısayol yapılandırmasının tam yapılandırmasıdır.

5. Route Predicate Factories

Spring Cloud Gateway, Spring WebFlux **HandlerMapping** altyapısının bir parçası olarak rotaları eşleştirir. Spring Cloud Gateway, birçok yerleşik rota yüklemi fabrikasını içerir. Bu tahminlerin tümü, HTTP isteğinin farklı nitelikleriyle eşleşir. Birden çok rota yüklem fabrikasını mantıksal **and** deyimlerle birleştirebilirsiniz.

5.1. The After Route Predicate Factory (After Route Yüklem Fabrikası)

After route yüklemi fabrikası, bir **datetime** (Java **ZonedDateTime** olan) bir parametre alır. Bu yüklem, belirtilen datatime'den sonra gerçekleşen isteklerle eşleşir. Aşağıdaki örnek, bir rota sonrası yüklemini yapılandırır:

Örnek 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]
```

Bu rota, 20 Ocak 2017 17:42 Dağ Saati'nden (Denver) sonra yapılan tüm isteklerle eşleşir.

5.2. The Before Route Predicate Factory

Önce rota yüklemi fabrikası, bir **datetime** (Java **ZonedDateTime** olan) bir parametre alır. Bu yüklem, belirtilen **datetime**'den önce gerçekleşen isteklerle eşleşir. Aşağıdaki örnek, bir önceki rota yüklemini yapılandırır:

Örnek 2. application.yml

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

Bu rota, 20 Ocak 2017 17:42 Mountain Time (Denver) tarihinden önce yapılan tüm isteklerle eşleşir.

5.3. The Between Route Predicate Factory

Between rota yüklemi fabrikası, java **ZonedDateTime** nesneleri olan **datetime1** ve **datetime2** olmak üzere iki parametre alır. Bu yüklem, **datetime1**'den sonra ve **datetime2**'den önce gerçekleşen isteklerle eşleşir. **datetime2** parametresi, **datetime1**'den sonra olmalıdır. Aşağıdaki örnek, bir rota yüklemi arasında yapılandırır:

Örnek 3. application.yml

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

Bu rota, 20 Ocak 2017 17:42 Dağ Saati (Denver) ve 21 Ocak 2017 17:42 Dağ Saati (Denver) öncesinde yapılan tüm isteklerle eşleşir. Bu, bakım pencereleri için yararlı olabilir.

5.4. The Cookie Route Predicate Factory

Cookie rota yüklem fabrikası, cookie **name** ve bir **regexp** (bir Java normal ifadesidir) olmak üzere iki parametre alır. Bu yüklem, verilen name'e sahip ve değerleri regular expression(normal ifade) ile eşleşen tanımlama bilgileriyle eşleşir. Aşağıdaki örnek, bir cookie rota yüklem fabrikasını yapılandırır:

Örnek 4. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: cookie_route
        uri: https://example.org
        predicates:
        - Cookie=chocolate, ch.p
```

Bu rota, değeri **ch.p** regular expression(normal ifade) ile eşleşen **chocolate** adlı bir cookie'lere sahip isteklerle eşleşir.

5.5. The Header Route Predicate Factory

Header rota yüklem fabrikası, **header** ve bir **regexp**(normal ifade) (bir Java regular expression-normal ifade- olan) olmak üzere iki parametre alır. Bu yüklem, değeri regular expression(normal ifade) ile eşleşen verilen name'e sahip bir header ile eşleşir. Aşağıdaki örnek, bir header rota yüklemini yapılandırır:

Örnek 5. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: header_route
        uri: https://example.org
        predicates:
        - Header=X-Request-Id, \d+
```

Bu yol, isteğin değeri **\d+** regular expression(normal ifade) ile eşleşen **X-Request-Id** adlı bir header varsa (yani, bir veya daha fazla basamak değerine sahipse) eşleşir.

5.6. The Host Route Predicate Factory

Host rota yüklem fabrikası bir parametre alır: Host adı **patterns**'in bir listesi. Desen, ayırıcı olarak **.** olan ant-style bir desendir. Bu, desenle eşleşen **host** header ile eşleşir. Aşağıdaki örnek, bir host yolu yüklemini yapılandırır:

Örnek 6. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: host_route
        uri: https://example.org
        predicates:
        - Host=**.somehost.org,**.anotherhost.org
```

URI şablon değişkenleri de (**{sub}.myhost.org** gibi) desteklenir.

Bu rota, isteğin **www.somehost.org** veya **beta.somehost.org** veya **www.anotherhost.org** değerine sahip bir Host header'ına sahip olması durumunda eşleşir.

Bu yüklem(predicate), URI şablon değişkenlerini (önceki örnekte tanımlanan **sub** gibi) bir ad ve değer haritası olarak çıkarır ve bunu **ServerWebExchangeUtils.URI_TEMPLATE_VARIABLES_ATTRIBUTE** içinde tanımlanan bir key ile **ServerWebExchange.getAttributes()** içine yerleştirir. Bu değerler daha sonra **GatewayFilter** fabrikaları tarafından kullanılabilir.

5.7. The Method Route Predicate Factory

Method Rota yüklem Fabrikası, bir veya daha fazla parametre olan bir **methods** bağımsız değişkeni alır: eşleştirilecek HTTP yöntemleri. Aşağıdaki örnek, bir method rota yüklemi yapılandırır:

Örnek 7. application.yml

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

Bu rota, istek yöntemi bir **GET** veya bir **POST** ise eşleşir.

5.8. The Path Route Predicate Factory

Path Route Predicate Factory iki parametre alır: Spring **PathMatcher** desenlerinin(**patterns**) bir listesi ve **matchTrailingSlash** adlı isteğe bağlı bir bayrak (varsayılanı **true**'dur). Aşağıdaki örnek, bir yol rotası yüklemini yapılandırır:

Örnek 8. application.yml

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

Bu rota, örneğin istek yolu şuysa eşleşir: **/red/1** veya **/red/1 /** veya **/red/blue** veya **/blue/green**.

Eğer **matchTrailingSlash false** olarak ayarlanırsa, istek yolu(request path) **/red/1/** eşleşmeyecektir.

Bu yüklem, URI şablon değişkenlerini (önceki örnekte tanımlanan **segment** gibi) bir ad ve değer haritası olarak çıkarır ve bunu **ServerWebExchangeUtils.URI_TEMPLATE_VARIABLES_ATTRIBUTE** içinde tanımlanan bir anahtarla **ServerWebExchange.getAttributes()** içine yerleştirir. Bu değerler daha sonra **GatewayFilter** fabrikaları tarafından kullanılabilir.
Bu değişkenlere erişimi kolaylaştırmak için bir yardımcı program yöntemi (**get** olarak adlandırılır) mevcuttur.
Aşağıdaki örnek, **get** metodunun nasıl kullanılacağını gösterir:

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

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

5.9. The Query Route Predicate Factory

Query yolu yüklem fabrikası iki parametre alır: gerekli bir **param** ve isteğe bağlı bir **regexp** (bir Java -regular expression- normal ifadesidir). Aşağıdaki örnek, bir query yolu yüklemi yapılandırır:

Örnek 9. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: https://example.org
        predicates:
        - Query=green
```

Önceki yol, istek **green** bir sorgu parametresi içeriyorsa eşleşir.

application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: https://example.org
        predicates:
        - Query=red, gree.
```

İstek, değeri **gree.** regexp ile eşleşen **red** bir sorgu parametresi içeriyorsa, **green** ve **greet** eşleşir.

5.10. The RemoteAddr Route Predicate Factory

RemoteAddr yol yüklem fabrikası, **192.168.0.1/16** gibi CIDR gösterimi (IPv4 veya IPv6) dizeleri olan kaynakların(**sources**) bir listesini (en az boyut 1) (Burada **192.168.0.1** bir IP adresidir ve **16** bir alt ağ maskesidir) alır.
Aşağıdaki örnek, RemoteAddr yol yüklemini yapılandırır:

Örnek 10. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: remoteaddr_route
        uri: https://example.org
        predicates:
        - RemoteAddr=192.168.1.1/24
```

Bu rota, isteğin remote(uzak) adresi ise eşleşir, örneğin **192.168.1.10**.

5.10.1. Modifying the Way Remote Addresses Are Resolved

Varsayılan olarak, RemoteAddr yol yüklemi fabrikası, gelen istekteki uzak adresi kullanır. Spring Cloud Gateway bir proxy katmanının arkasındaysa, bu gerçek client(alıcı) IP adresiyle eşleşmeyebilir.

Özel bir **RemoteAddressResolver** ayarlayarak uzak adresin çözümlenme şeklini özelleştirebilirsiniz. Spring Cloud Gateway, X-Forwarded-For Header'ına, **XForwardedRemoteAddressResolver**'a dayanan varsayılan olmayan bir uzak adres çözümleyici ile birlikte gelir.

XForwardedRemoteAddressResolver, güvenliğe farklı yaklaşımlar getiren iki statik oluşturucu yöntemine sahiptir:

- **XForwardedRemoteAddressResolver::trustAll**, her zaman **X-Forwarded-For** başlığında bulunan ilk IP adresini alan bir **RemoteAddressResolver** döndürür. Kötü niyetli bir client(alıcı), **X-Forwarded-For** için çözümleyici tarafından kabul edilecek bir başlangıç değeri ayarlayabildiğinden, bu yaklaşım kimlik sahtekarlığına karşı savunmasızdır.
- **XForwardedRemoteAddressResolver::maxTrustedIndex**, Spring Cloud Gateway'in önünde çalışan güvenilir altyapı sayısıyla ilişkili bir dizin alır. Spring Cloud Gateway, örneğin yalnızca HAProxy aracılığıyla erişilebilirse, 1 değeri kullanılmalıdır. Spring Cloud Gateway'e erişilmeden önce iki atlama güvenilir altyapısı gerekiyorsa, 2 değeri kullanılmalıdır.

Aşağıdaki başlık değerini göz önünde bulundurun:

```
X-Forwarded-For: 0.0.0.1, 0.0.0.2, 0.0.0.3
```

Aşağıdaki **maxTrustedIndex** değerleri aşağıdaki uzak adresleri verir:

maxTrustedIndex	result
[Integer.MIN_VALUE,0]	(invalid, IllegalArgumentException during initialization)
1	0.0.0.3
2	0.0.0.2
3	0.0.0.1
[4, Integer.MAX_VALUE]	0.0.0.1

Aşağıdaki örnek, aynı konfigürasyonun Java ile nasıl elde edileceğini gösterir:

Örnek 11. GatewayConfig.java

```
RemoteAddressResolver resolver = XForwardedRemoteAddressResolver
    .maxTrustedIndex(1);

...

.route("direct-route",
    r -> r.remoteAddr("10.1.1.1", "10.10.1.1/24")
        .uri("https://downstream1")
.route("proxied-route",
    r -> r.remoteAddr(resolver, "10.10.1.1", "10.10.1.1/24")
        .uri("https://downstream2")
)
```

5.11. The Weight Route Predicate Factory

Weight yolu yüklem fabrikası iki bağımsız değişken alır: **group** ve **weight** (bir int). Ağırlıklar grup başına hesaplanır. Aşağıdaki örnek, bir ağırlık yolu yüklemini yapılandırır:

Örnek 12. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: weight_high
        uri: https://weighthigh.org
        predicates:
        - Weight=group1, 8
      - id: weight_low
        uri: https://weightlow.org
        predicates:
        - Weight=group1, 2
```

Bu rota, trafiğin ~%80'ini Weighthigh.org'a ve ~%20'sini Weightlow.org'a yönlendirir.

5.12. The XForwarded Remote Addr Route Predicate Factory

XForwarded Remote Addr rota yüklemi fabrikası, **192.168.0.1/16** gibi CIDR gösterimi (IPv4 veya IPv6) dizeleri olan kaynakların(**sources**) bir listesini (en az boyut 1) (Burada **192.168.0.1** bir IP adresi ve **16** bir alt maskesidir) alır.
Bu rota yüklemi, isteklerin **X-Forwarded-For** HTTP başlığına göre filtrelenmesine izin verir.

Bu, isteğe yalnızca bu ters proxy'ler tarafından kullanılan güvenilir bir IP adresleri listesinden geliyorsa izin verilmesi gereken yük dengeleyiciler veya web uygulaması güvenlik duvarları gibi ters proxy'lerle kullanılabilir.

Aşağıdaki örnek, bir XForwardedRemoteAddr yol yüklemini yapılandırır:

Örnek 13. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: xforwarded_remoteaddr_route
        uri: https://example.org
        predicates:
        - XForwardedRemoteAddr=192.168.1.1/24
```

Bu rota, **X-Forwarded-For** header içeriyorsa eşleşir, örneğin **192.168.1.10**.

6. GatewayFilter Factories

Rota filtreleri, bir şekilde gelen HTTP isteğinin veya giden HTTP yanıtının değiştirilmesine izin verir.
Rota filtreleri, belirli bir rotaya göre kapsamlandırılır. Spring Cloud Gateway, birçok yerleşik GatewayFilter Fabrikası içerir.

Aşağıdaki filtrelerden herhangi birinin nasıl kullanılacağına ilişkin daha ayrıntılı örnekler için unit test'lerine bakın.

6.1. The AddRequestHeader GatewayFilter Factory

AddRequestHeader GatewayFilter fabrikası bir **name** ve **value** parametresi alır.

Aşağıdaki örnek, bir **AddRequestHeader GatewayFilter**'ı yapılandırır:

Örnek 14. application.yml

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

Bu liste, tüm eşleşen istekler için downstream(aşağı akış) isteğinin başlıklarına X-Request-red:blue başlığı ekler.

AddRequestHeader, bir yolu veya host'u eşleştirmek için kullanılan URI değişkenlerinin farkındadır.

URI değişkenleri değerde kullanılabilir ve çalışma zamanında genişletilir.

Aşağıdaki örnek, bir değişken kullanan bir **AddRequestHeader GatewayFilter** yapılandırır:

Örnek 15. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_header_route
        uri: https://example.org
        predicates:
        - Path=/red/{segment}
        filters:
        - AddRequestHeader=X-Request-Red, Blue-{segment}
```

6.2. The AddRequestParameter GatewayFilter Factory

AddRequestParameter GatewayFilter Factory, bir **name** ve **value** parametresi alır.

Aşağıdaki örnek, bir **AddRequestParameter GatewayFilter**'ı yapılandırır:

Örnek 16. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_parameter_route
        uri: https://example.org
        filters:
        - AddRequestParameter=red, blue
```

Bu, tüm eşleşen istekler için downstream(aşağı akış) isteğinin sorgu dizesine **red=blue** ekleyecektir.

AddRequestParameter, bir yolu veya host'u eşleştirmek için kullanılan URI değişkenlerinin farkındadır.

URI değişkenleri değerde kullanılabilir ve çalışma zamanında genişletilir.

Aşağıdaki örnek, bir değişken kullanan bir **AddRequestParameter GatewayFilter** yapılandırır:

Örnek 17. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_parameter_route
        uri: https://example.org
        predicates:
        - Host: {segment}.myhost.org
        filters:
        - AddRequestParameter=foo, bar-{segment}
```

6.3. The AddResponseHeader GatewayFilter Factory

AddResponseHeader GatewayFilter Factory, bir **name** ve **value** parametresi alır.
Aşağıdaki örnek, bir **AddResponseHeader GatewayFilter**'ı yapılandırır:

Örnek 18. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_response_header_route
        uri: https://example.org
        filters:
        - AddResponseHeader=X-Response-Red, Blue
```

Bu, tüm eşleşen istekler için downstream(aşağı akış) yanıtının başlıklarına **X-Response-Red:Blue** başlığı ekler.

AddResponseHeader, bir yolu veya host'u eşleştirmek için kullanılan URI değişkenlerinin farkındadır.
URI değişkenleri değerde kullanılabilir ve çalışma zamanında genişletilir.
Aşağıdaki örnek, bir değişken kullanan bir **AddResponseHeader GatewayFilter** yapılandırır:

Örnek 19. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_response_header_route
        uri: https://example.org
        predicates:
        - Host: {segment}.myhost.org
        filters:
        - AddResponseHeader=foo, bar-{segment}
```

6.4. The DedupeResponseHeader GatewayFilter Factory

DedupeResponseHeader GatewayFilter fabrikası, bir **name** parametresi ve isteğe bağlı bir **strategy** parametresi alır.
name, başlık adlarının boşlukla ayrılmış bir listesini içerebilir.
Aşağıdaki örnek, bir **DedupeResponseHeader GatewayFilter**'ı yapılandırır:

Örnek 20. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: dedupe_response_header_route
        uri: https://example.org
        filters:
        - DedupeResponseHeader=Access-Control-Allow-Credentials Access-Control-Allow-Origin
```

Bu, **Access-Control-Allow-Credentials** ve **Access-Control-Allow-Origin** yanıt başlıklarının yinelenen değerlerini, hem ağ geçidi CORS mantığının hem de aşağı akış mantığının bunları eklediği durumlarda kaldırır.

DedupeResponseHeader filtresi ayrıca isteğe bağlı bir **strategy** parametresini de kabul eder.
Kabul edilen değerler **RETAIN_FIRST** (varsayılan), **RETAIN_LAST** ve **RETAIN_UNIQUE**'dir.

6.5. Spring Cloud CircuitBreaker GatewayFilter Factory

Spring Cloud CircuitBreaker GatewayFilter fabrikası, Gateway rotalarını bir circuit breaker'a(devre kesiciye) sarmak için Spring Cloud CircuitBreaker API'lerini kullanır. Spring Cloud CircuitBreaker, Spring Cloud Gateway ile kullanılabilen birden çok kitaplığı destekler. Spring Cloud, Resilience4J'yi kutudan çıktığı gibi destekler.

Spring Cloud CircuitBreaker filtresini etkinleştirmek için, **spring-cloud-starter-circuitbreaker-reactor-resilience4j'yi** sınıf yoluna yerleştirmeniz gerekir. Aşağıdaki örnek, bir Spring Cloud CircuitBreaker **GatewayFilter**'ı yapılandırır:

Örnek 21. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: circuitbreaker_route
        uri: https://example.org
        filters:
        - CircuitBreaker=myCircuitBreaker
```

Devre kesiciyi yapılandırmak için, kullandığınız temel devre kesici uygulamasının yapılandırmasına bakın. [Resilience4J Belgeleri]

Spring Cloud CircuitBreaker filtresi, isteğe bağlı bir **fallbackUri** parametresini de kabul edebilir. Şu anda yalnızca **forward:** planlı URI'ler desteklenmektedir. Geri dönüş çağrılırsa, istek URI tarafından eşleşen denetleyiciye iletilir. Aşağıdaki örnek, böyle bir yedeği yapılandırır:

Örnek 22. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: circuitbreaker_route
        uri: lb://backing-service:8088
        predicates:
        - Path=/consumingServiceEndpoint
        filters:
        - name: CircuitBreaker
          args:
            name: myCircuitBreaker
            fallbackUri: forward:/inCaseOfFailureUseThis
        - RewritePath=/consumingServiceEndpoint, /backingServiceEndpoint
```

Aşağıdaki liste Java'da aynı şeyi yapar:

Örnek 23. Application.java

```
@Bean
public RouteLocator routes(RouteLocatorBuilder builder) {
   return builder.routes()
      .route("circuitbreaker_route", r -> r.path("/consumingServiceEndpoint")
         .filters(f -> f.circuitBreaker(c -> c.name("myCircuitBreaker").fallbackUri("forward:/inCaseOfFailureUseThis"))
            .rewritePath("/consumingServiceEndpoint", "/backingServiceEndpoint")).uri("lb://backing-service:8088")
      .build();
}
```

Bu örnek, circuit breaker fallback çağrıldığında **/inCaseofFailureUseThis** URI'sine iletir. Bu örneğin (isteğe bağlı) Spring Cloud LoadBalancer yük dengelemesini de gösterdiğine dikkat edin (hedef URI'deki **lb** öneki ile tanımlanır).

Birincil senaryo, gateway uygulamasında dahili bir denetleyici veya işleyici tanımlamak için **fallbackUri**'yi kullanmaktır. Ancak, isteği harici bir uygulamadaki bir controller'e veya handler'e aşağıdaki gibi yeniden yönlendirebilirsiniz:

Örnek 24. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: ingredients
        uri: lb://ingredients
        predicates:
        - Path=//ingredients/**
        filters:
        - name: CircuitBreaker
          args:
            name: fetchIngredients
            fallbackUri: forward:/fallback
      - id: ingredients-fallback
        uri: http://localhost:9994
        predicates:
        - Path=/fallback
```

Bu örnekte, ağ geçidi (gateway) uygulamasında **fallback** bitiş noktası veya işleyici yoktur.
Ancak, başka bir uygulamada **localhost:9994** altında kayıtlı bir tane var.

İsteğin geri dönüşe iletilmesi durumunda, Spring Cloud CircuitBreaker Gateway filtresi, buna neden olan **Throwable**'ı da sağlar.
Ağ geçidi uygulamasında yedeği işlerken kullanılabilecek
ServerWebExchangeUtils.CIRCUITBREAKER_EXECUTION_EXCEPTION_ATTR özniteliği olarak **ServerWebExchange**'e eklenir.

Harici controller/handler(denetleyici/işleyici) senaryosu için, istisna ayrıntılarıyla başlıklar eklenebilir.
FallbackHeaders GatewayFilter Factory bölümünde bunu yapmakla ilgili daha fazla bilgi bulabilirsiniz.

6.5.1. Tripping The Circuit Breaker On Status Codes

Bazı durumlarda, tamamladığı rotadan döndürülen durum koduna bağlı olarak bir devre kesiciyi(circuit breaker'i) tetiklemek isteyebilirsiniz. Devre kesici yapılandırma nesnesi, döndürülürse devre kesicinin açılmasına neden olacak durum kodlarının bir listesini alır. Devre kesiciyi açmak istediğiniz durum kodlarını ayarlarken, durum kodu değerine sahip bir integer veya **HttpStatus** numaralandırmasının String gösterimini kullanabilirsiniz.

Örnek 25. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: circuitbreaker_route
        uri: lb://backing-service:8088
        predicates:
        - Path=/consumingServiceEndpoint
        filters:
        - name: CircuitBreaker
          args:
            name: myCircuitBreaker
            fallbackUri: forward:/inCaseOfFailureUseThis
            statusCodes:
              - 500
              - "NOT_FOUND"
```

Örnek 26. Application.java

```
@Bean
public RouteLocator routes(RouteLocatorBuilder builder) {
   return builder.routes()
      .route("circuitbreaker_route", r -> r.path("/consumingServiceEndpoint")
         .filters(f -> f.circuitBreaker(c -> c.name("myCircuitBreaker").fallbackUri("forward:/inCaseOfFailureUseThis").addStatusCode("INTERNAL_SERVER_ERROR"))
            .rewritePath("/consumingServiceEndpoint", "/backingServiceEndpoint")).uri("lb://backing-service:8088")
      .build();
}
```

6.6. The FallbackHeaders GatewayFilter Factory

FallbackHeaders fabrikası, aşağıdaki senaryoda olduğu gibi, harici bir uygulamada bir **fallbackUri**'ye iletilen bir isteğin üstbilgilerine Spring Cloud CircuitBreaker yürütme istisnası ayrıntılarını eklemenize olanak tanır:

Örnek 27. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: ingredients
        uri: lb://ingredients
        predicates:
        - Path=//ingredients/**
        filters:
        - name: CircuitBreaker
          args:
            name: fetchIngredients
            fallbackUri: forward:/fallback
      - id: ingredients-fallback
        uri: http://localhost:9994
        predicates:
        - Path=/fallback
        filters:
        - name: FallbackHeaders
          args:
            executionExceptionTypeHeaderName: Test-Header
```

Bu örnekte, devre kesici çalıştırılırken bir yürütme istisnası meydana geldikten sonra, istek, **localhost:9994** üzerinde çalışan bir uygulamada **fallback** uç noktasına veya header'e iletilir. İstisna tipi, mesaj ve (varsa) kök neden istisna tipi ve mesajına sahip başlıklar, **FallbackHeaders** filtresi tarafından bu isteğe eklenir.

Aşağıdaki bağımsız değişkenlerin (varsayılan değerleriyle gösterilir) değerlerini ayarlayarak yapılandırmadaki başlıkların adlarının üzerine yazabilirsiniz:

- ❑ executionExceptionTypeHeaderName ("Execution-Exception-Type")
- ❑ executionExceptionMessageHeaderName ("Execution-Exception-Message")
- ❑ rootCauseExceptionTypeHeaderName ("Root-Cause-Exception-Type")
- ❑ rootCauseExceptionMessageHeaderName ("Root-Cause-Exception-Message")

Devre kesiciler(circuit breaker) ve ağ geçidi(gateway) hakkında daha fazla bilgi için Spring Cloud CircuitBreaker Factory bölümüne bakın.

6.7. The MapRequestHeader GatewayFilter Factory

MapRequestHeader GatewayFilter fabrikası, **fromHeader** ve **toHeader** parametrelerini alır. Yeni bir adlandırılmış üstbilgi (**toHeader**) oluşturur ve değer, gelen http isteğinden mevcut bir adlandırılmış üstbilgiden (**fromHeader**) çıkarılır. Giriş başlığı yoksa, filtrenin etkisi yoktur. Yeni adlandırılmış başlık zaten mevcutsa, değerleri yeni değerlerle artırılır. Aşağıdaki örnek, bir **MapRequestHeader**'ı yapılandırır:

Örnek 28. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: map_request_header_route
        uri: https://example.org
        filters:
        - MapRequestHeader=Blue, X-Request-Red
```

Bu, gelen HTTP isteğinin **Blue** header'inden güncellenmiş değerlerle aşağı akış isteğine **X-Request-Red:<values>** header'ini ekler.

6.8. The PrefixPath GatewayFilter Factory

PrefixPath GatewayFilter fabrikası, tek bir **prefix** parametresi alır.

Aşağıdaki örnek, bir **PrefixPath GatewayFilter**'ı yapılandırır:

Örnek 29. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: prefixpath_route
        uri: https://example.org
        filters:
        - PrefixPath=/mypath
```

Bu, eşleşen tüm isteklerin yolunun önüne **/mypath**'i ekleyecektir. Böylece **/hello**'ya bir istek **/mypath/hello**'ya gönderilir.

6.9. The PreserveHostHeader GatewayFilter Factory

PreserveHostHeader GatewayFilter fabrikasında parametre yoktur. Bu filtre, HTTP client'i(alıcısı) tarafından belirlenen host header yerine orijinal host header'in gönderilip gönderilmeyeceğini belirlemek için yönlendirme filtresinin denetlediği bir istek özniteliği ayarlar. Aşağıdaki örnek, bir **PreserveHostHeader GatewayFilter**'ı yapılandırır:

Örnek 30. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: preserve_host_route
        uri: https://example.org
        filters:
        - PreserveHostHeader
```

6.10. The RequestRateLimiter GatewayFilter Factory

RequestRateLimiter GatewayFilter fabrikası, geçerli isteğin devam etmesine izin verilip verilmediğini belirlemek için bir **RateLimiter** uygulaması kullanır. Değilse, **HTTP 429 – Too Many Requests**(varsayılan olarak) durumu döndürülür.
Bu filtre, isteğe bağlı bir **keyResolver** parametresini ve hız sınırlayıcıya özgü parametreleri alır (bu bölümün ilerleyen kısımlarında anlatılacaktır).
keyResolver, **KeyResolver** arabirimini uygulayan bir çekirdektir. Konfigürasyonda, SpEL kullanarak fasulyeye adıyla başvurun. **#{@myKeyResolver}**, **myKeyResolver** adlı bir bean'e başvuran bir SpEL ifadesidir. Aşağıdaki liste, **KeyResolver** arabirimini gösterir:

Örnek 31. KeyResolver.java

```
public interface KeyResolver {
   Mono<String> resolve(ServerWebExchange exchange);
}
```

KeyResolver arabirimi, takılabilir stratejilerin, istekleri sınırlamak için anahtarı türetmesine izin verir. Gelecekteki dönüm noktası sürümlerinde, bazı **KeyResolver** uygulamaları olacaktır.
KeyResolver'ın varsayılan uygulaması, **Principal**'ı **ServerWebExchange**'den alan ve **Principal.getName()**'i çağıran **PrincipalNameKeyResolver**'dır.
Varsayılan olarak, **KeyResolver** bir anahtar bulamazsa, istekler reddedilir. **spring.cloud.gateway.filter.request-rate-limiter.deny-empty-key** (**true** veya **false**) ve **spring.cloud.gateway.filter.request-rate-limiter.empty-key-status-code** özelliklerini ayarlayarak bu davranışı ayarlayabilirsiniz.
RequestRateLimiter, " shortcut" gösterimi ile yapılandırılamaz. Aşağıdaki örnek geçersiz:

Örnek 32. application.properties

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

6.10.1. The Redis RateLimiter

Redis uygulaması, Stripe'ta yapılan çalışmaları temel alır. **spring-boot-starter-data-redis-reactive** Spring Boot starter'ın kullanılmasını gerektirir.
Kullanılan algoritma Token Bucket Algoritmasıdır.
redis-rate-limiter.replenishRate özelliği, bir kullanıcının bırakılan istekler olmadan saniyede kaç istek yapmasına izin verilmesini istediğinizdir. Bu, jeton kovasının doldurulma hızıdır.
redis-rate-limiter.burstCapacity özelliği, bir kullanıcının bir saniyede yapmasına izin verilen maksimum istek sayısıdır. Bu, jeton kovasının tutabileceği jeton sayısıdır. Bu değeri sıfıra ayarlamak tüm istekleri engeller.
redis-rate-limiter.requestedTokens özelliği, bir isteğin kaç jeton maliyeti olduğunu gösterir. Bu, her istek için paketten alınan jeton sayısıdır ve varsayılan olarak **1**'dir.
Sabit bir oran, **replenishRate** ve **burstCapacity**'de aynı değeri ayarlayarak elde edilir. **burstCapacity**, **replenishRate**'den daha yüksek bir değere ayarlanarak geçici patlamalara izin verilebilir. Bu durumda, art arda iki çoğuşma isteklerin düşmesine neden olacağından (**HTTP 429 – Too Many Request**) hız sınırlayıcının patlamalar arasında bir süre geçmesine izin verilmesi gerekir (**repleishRate**'e göre). Aşağıdaki liste bir **redis-rate-limiter** yapılandırır:
1 request/s'nin altındaki oran sınırları, **replenishRate**'i istenen istek sayısına, **requestTokens**'ı saniye cinsinden zaman aralığına ve **burstCapacity**'yi **replenishRate** ve **requestTokens** ürününe ayarlayarak gerçekleştirilir, örn. **replenishRate=1**, **requestTokens=60** ve **burstCapacity=60** ayarı, **1 request/min** sınırına neden olur.

Örnek 33. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: requestratelimiter_route
        uri: https://example.org
        filters:
        - name: RequestRateLimiter
          args:
            redis-rate-limiter.replenishRate: 10
            redis-rate-limiter.burstCapacity: 20
            redis-rate-limiter.requestedTokens: 1
```

Aşağıdaki örnek, Java'da bir KeyResolver'ı yapılandırır:

Örnek 34. Config.java

```
@Bean
KeyResolver userKeyResolver() {
    return exchange -> Mono.just(exchange.getRequest().getQueryParams().getFirst("user"));
}
```

Bu, kullanıcı başına 10'luk bir istek oranı limiti tanımlar. 20'lik bir patlamaya izin verilir, ancak sonraki saniyede yalnızca 10 istek kullanılabilir. **KeyResolver**, **user** request parametresini alan basit bir parametredir (bunun üretim için önerilmediğini unutmayın).

Ayrıca **RateLimiter** arabirimini uygulayan bir çekirdek olarak bir hız sınırlayıcı tanımlayabilirsiniz. Konfigürasyonda, SpEL'i kullanarak bean'a adıyla başvurabilirsiniz. **#{@myRateLimiter}**, **myRateLimiter** adlı bir bean'e başvuran bir SpEL ifadesidir.
Aşağıdaki liste, önceki listede tanımlanan **KeyResolver**'ı kullanan bir hız sınırlayıcıyı tanımlar:

Örnek 35. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: requestratelimiter_route
        uri: https://example.org
        filters:
        - name: RequestRateLimiter
          args:
            rate-limiter: "#{@myRateLimiter}"
            key-resolver: "#{@userKeyResolver}"
```

6.11. The RedirectTo GatewayFilter Factory

RedirectTo GatewayFilter fabrikası, **status** ve **url** olmak üzere iki parametre alır. **Status** parametresi, 301 gibi bir 300 serisi yönlendirme HTTP kodu olmalıdır. **Url** parametresi geçerli bir URL olmalıdır. Bu, **Location** header değeridir. Göreli yönlendirmeler için, rota tanımınızın uri'si olarak **uri: no://op** kullanmalısınız. Aşağıdaki liste, bir **RedirectTo GatewayFilter**'ı yapılandırır:

Örnek 36. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: prefixpath_route
        uri: https://example.org
        filters:
        - RedirectTo=302, https://acme.org
```

Bu, bir yönlendirme gerçekleştirmek için **Location:https://acme.org** başlığına sahip bir 302 durumu gönderir.

6.12. The RemoveRequestHeader GatewayFilter Factory

RemoveRequestHeader GatewayFilter fabrikası bir **name** parametresi alır. Kaldırılacak başlığın adıdır.

Aşağıdaki liste, **RemoveRequestHeader GatewayFilter**'ı yapılandırır:

Örnek 37. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: removerequestheader_route
        uri: https://example.org
        filters:
        - RemoveRequestHeader=X-Request-Foo
```

Bu, aşağı akışa gönderilmeden önce **X-Request-Foo** header'i kaldırır.

6.13. RemoveResponseHeader GatewayFilter Factory

RemoveResponseHeader GatewayFilter fabrikası bir **name** parametresi alır. Kaldırılacak başlığın adıdır.
Aşağıdaki liste, **RemoveResponseHeader GatewayFilter**'ı yapılandırır:

Örnek 38. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: removeresponseheader_route
        uri: https://example.org
        filters:
        - RemoveResponseHeader=X-Response-Foo
```

Bu, ağ geçidi istemcisine döndürülmeden önce **X-Response-Foo** üstbilgisini yanıttan kaldırır.

Her türlü hassas başlığı kaldırmak için, bu filtreyi, bunu yapmak isteyebileceğiniz tüm rotalar için yapılandırmanız gerekir. Ayrıca, bu filtreyi **spring.cloud.gateway.default** filtrelerini kullanarak bir kez yapılandırabilir ve tüm tesisatlara uygulanmasını sağlayabilirsiniz.

6.14. The RemoveRequestParameter GatewayFilter Factory

RemoveRequestParameter GatewayFilter fabrikası bir **name** parametresi alır. Kaldırılacak sorgu parametresinin adıdır.

Aşağıdaki örnek, bir **RemoveRequestParameter GatewayFilter** yapılandırır:

Örnek 39. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: removerequestparameter_route
        uri: https://example.org
        filters:
        - RemoveRequestParameter=red
```

Bu, aşağı yönde gönderilmeden önce **red** parametreyi kaldıracaktır.

6.15. RequestHeaderSize GatewayFilter Factory

RequestHeaderSize GatewayFilter fabrikası, **maxSize** ve **errorHeaderName** parametrelerini alır. **maxSize** parametresi, istek başlığının izin verilen maksimum veri boyutudur (anahtar ve değer dahil). **errorHeaderName** parametresi, bir hata mesajı içeren yanıt başlığının adını ayarlar, varsayılan olarak "errorMessage"dır. Aşağıdaki liste, bir **RequestHeaderSize GatewayFilter** yapılandırır:

Örnek 40. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: requestheadersize_route
        uri: https://example.org
        filters:
        - RequestHeaderSize=1000B
```

Herhangi bir istek başlığının boyutu 1000 Bayttan büyükse, bu durum 431 gönderir.

6.16. The RewritePath GatewayFilter Factory

RewritePath GatewayFilter fabrikası, bir yol **regexp** parametresi ve bir **replacement** parametresi alır. Bu, istek yolunu yeniden yazmanın esnek bir yolu için Java düzenli ifadelerini kullanır.
Aşağıdaki liste, bir **RewritePath GatewayFilter**'ı yapılandırır:

Örnek 41. application.yml

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

/red/blue istek yolu için, bu, aşağı akış(downstream) isteğini yapmadan önce **/blue** yolunu ayarlar. YAML belirtimi nedeniyle **$** öğesinin **$** ile değiştirilmesi gerektiğini unutmayın.

6.17. RewriteLocationResponseHeader GatewayFilter Factory

RewriteLocationResponseHeader GatewayFilter fabrikası, genellikle arka uca özgü ayrıntılardan kurtulmak için **Location** yanıtı üstbilgisinin değerini değiştirir. **stripVersionMode**, **locationHeaderName**, **hostValue** ve **protocolsRegex** parametrelerini alır. Aşağıdaki liste, bir **RewriteLocationResponseHeader GatewayFilter**'ı yapılandırır:

Örnek 42. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: rewritelocationresponseheader_route
        uri: http://example.org
        filters:
        - RewriteLocationResponseHeader=AS_IN_REQUEST, Location, ,
```

Örneğin, bir **POST api.example.com/some/object/name** isteği için, **object-service.prod.example.net/v2/some/object/id** öğesinin **Location** yanıtı üstbilgi değeri **api.example .com/some/object/id** olarak yeniden yazılır.

stripVersionMode parametresi aşağıdaki olası değerlere sahiptir: **NEVER_STRIP**, **AS_IN_REQUEST** (varsayılan) ve **ALWAYS_STRIP**.

- **NEVER_STRIP:** Orijinal istek yolu sürüm içermese bile sürüm çıkarılmaz.
- **AS_IN_REQUEST** Sürüm, yalnızca orijinal istek yolu sürüm içermiyorsa çıkarılır.
- **ALWAYS_STRIP** Orijinal istek yolu sürüm içerse bile sürüm her zaman çıkarılır.

Sağlanmışsa, **hostValue** parametresi, yanıt **Location** başlığının **host:port** kısmını değiştirmek için kullanılır. Sağlanmazsa, **Host** request başlığının değeri kullanılır.

ProtocolsRegex parametresi, protokol adının eşleştirildiği, geçerli bir normal ifade **String**'i olmalıdır. Eşleşmezse, filtre hiçbir şey yapmaz. Varsayılan, **http|https|ftp|ftps**'dir.

6.18. The RewriteResponseHeader GatewayFilter Factory

RewriteResponseHeader GatewayFilter fabrikası **name**, **regexp** ve **replacement** parametrelerini alır. Yanıt başlığı değerini yeniden yazmanın esnek bir yolu için Java reguler expression'larını(düzenli ifadelerini) kullanır.
Aşağıdaki örnek, bir **RewriteResponseHeader GatewayFilter**'ı yapılandırır:

Örnek 42. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: rewriteresponseheader_route
        uri: https://example.org
        filters:
        - RewriteResponseHeader=X-Response-Red, , password=[^&]+, password=***
```

/42?user=ford&password=omg!what&flag=true başlık değeri için, aşağı akış(downstream) isteği yapıldıktan sonra **/42?user=ford&password=***&flag=true** olarak ayarlanır. YAML özelliğinden dolayı **$** demek için **$** kullanmalısınız.

6.19. The SaveSession GatewayFilter Factory

SaveSession GatewayFilter fabrikası, çağrıyı aşağı akışa(downstream'a) iletmeden önce bir **WebSession::save** işlemini zorlar. Bu, özellikle Spring Session gibi bir şeyi tembel bir veri deposuyla kullanırken kullanışlıdır ve yönlendirilen aramayı yapmadan önce session durumunun kaydedildiğinden emin olmanız gerekir.
Aşağıdaki örnek, bir **SaveSession GatewayFilter**'ı yapılandırır:

Örnek 44. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: save_session
        uri: https://example.org
        predicates:
        - Path=/foo/**
        filters:
        - SaveSession
```

Spring Security'yi Spring Session ile entegre ederseniz ve güvenlik ayrıntılarının uzak sürece iletildiğinden emin olmak istiyorsanız, bu çok önemlidir.

6.20. The SecureHeaders GatewayFilter Factory

SecureHeaders GatewayFilter fabrikası, bu blog gönderisinde yapılan öneriye göre yanıta bir dizi başlık ekler.

Aşağıdaki başlıklar (varsayılan değerleriyle gösterilir) eklenir:

- **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)**

Varsayılan değerleri değiştirmek için, **spring.cloud.gateway.filter.secure-headers** ad alanında uygun özelliği ayarlayın. Aşağıdaki özellikler mevcuttur:

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

Varsayılan değerleri devre dışı bırakmak için, **spring.cloud.gateway.filter.secure-headers.disable** özelliğini virgülle ayrılmış değerlerle ayarlayın. Aşağıdaki örnek bunun nasıl yapılacağını gösterir:

```
spring.cloud.gateway.filter.secure-headers.disable=x-frame-options,strict-transport-security
```

Devre dışı bırakmak için güvenli başlığın küçük tam adının kullanılması gerekir..

6.21. The SetPath GatewayFilter Factory

SetPath GatewayFilter fabrikası bir yol **template** parametresi alır. Yolun template'li(şablonlu) bölümlerine izin vererek istek yolunu değiştirmek için basit bir yol sunar. Bu, Spring Framework'teki URI şablonlarını kullanır.
Birden çok eşleşen segmente izin verilir.
Aşağıdaki örnek, bir **SetPath GatewayFilter**'ı yapılandırır:

Örnek 45. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setpath_route
        uri: https://example.org
        predicates:
        - Path=/red/{segment}
        filters:
        - SetPath=/{segment}
```

/red/blue istek yolu için, bu, aşağı akış isteğini yapmadan önce **/blue** yolunu ayarlar.

6.22. The SetRequestHeader GatewayFilter Factory

SetRequestHeader GatewayFilter fabrikası, **name** ve **value** parametrelerini alır.

Aşağıdaki liste, bir **SetRequestHeader GatewayFilter**'ı yapılandırır:

Örnek 46. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setrequestheader_route
        uri: https://example.org
        filters:
        - SetRequestHeader=X-Request-Red, Blue
```

Bu **GatewayFilter**, tüm başlıkları verilen adla değiştirir (eklemek yerine). Dolayısıyla, aşağı akış sunucusu bir **X-Request-Red:1234** ile yanıt verirse, bu, aşağı akış hizmetinin alacağı **X-Request-Red:Blue** ile değiştirilir.

SetRequestHeader, bir yolu veya ana bilgisayarı eşleştirmek için kullanılan URI değişkenlerinin farkındadır. URI değişkenleri değerde kullanılabilir ve çalışma zamanında genişletilir.

Aşağıdaki örnek, bir değişken kullanan bir **SetRequestHeader GatewayFilter** yapılandırır:

Örnek 47. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setrequestheader_route
        uri: https://example.org
        predicates:
        - Host: {segment}.myhost.org
        filters:
        - SetRequestHeader=foo, bar-{segment}
```

6.23. The SetResponseHeader GatewayFilter Factory

SetResponseHeader GatewayFilter fabrikası, **name** ve **value** parametrelerini alır. Aşağıdaki liste, bir **SetResponseHeader GatewayFilter**'ı yapılandırır:

Örnek 48. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setresponseheader_route
        uri: https://example.org
        filters:
        - SetResponseHeader=X-Response-Red, Blue
```

Bu GatewayFilter, tüm başlıkları verilen adla değiştirir (eklemek yerine). Bu nedenle, aşağı akış sunucusu bir **X-Response-Red:1234** ile yanıt verdiyse, bu, ağ geçidi istemcisinin alacağı **X-Response-Red:Blue** ile değiştirilir.
SetResponseHeader, bir yolu veya ana bilgisayarı eşleştirmek için kullanılan URI değişkenlerinin farkındadır.
URI değişkenleri değerde kullanılabilir ve çalışma zamanında genişletilecektir.
Aşağıdaki örnek, bir değişken kullanan bir **SetResponseHeader GatewayFilter** yapılandırır:

Örnek 49. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setresponseheader_route
        uri: https://example.org
        predicates:
        - Host: {segment}.myhost.org
        filters:
        - SetResponseHeader=foo, bar-{segment}
```

6.24. The SetStatus GatewayFilter Factory

SetStatus GatewayFilter fabrikası tek bir parametre, **status** alır. Geçerli bir Spring **HttpStatus** olmalıdır. **404** tamsayı değeri veya numaralandırmanın string temsili olabilir: **NOT_FOUND**.
Aşağıdaki liste, bir **SetStatus GatewayFilter**'ı yapılandırır:

Örnek 50. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setstatusstring_route
        uri: https://example.org
        filters:
        - SetStatus=UNAUTHORIZED
      - id: setstatusint_route
        uri: https://example.org
        filters:
        - SetStatus=401
```

Her iki durumda da yanıtın HTTP durumu 401 olarak ayarlanır.
SetStatus GatewayFilter'ı, yanıttaki bir üstbilgideki proxy istekten orijinal HTTP durum kodunu döndürecek şekilde yapılandırabilirsiniz. Başlık, aşağıdaki özellik ile yapılandırılırsa yanıta eklenir:

Örnek 51. application.yml

```
spring:
  cloud:
    gateway:
      set-status:
        original-status-header-name: original-http-status
```

6.25. The StripPrefix GatewayFilter Factory

StripPrefix GatewayFilter fabrikası bir parametre alır, **parts**. **parts** parametresi, aşağı akışa göndermeden önce istekten çıkarılacak yoldaki part(parça) sayısını gösterir. Aşağıdaki liste, bir **StripPrefix GatewayFilter**'ı yapılandırır:

Örnek 52. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: nameRoot
        uri: https://nameservice
        predicates:
        - Path=/name/**
        filters:
        - StripPrefix=2
```

/name/blue/red'e ağ geçidi üzerinden bir istek yapıldığında, **nameservice** yapılan istek ad **nameservice/red** gibi görünür.

6.26. The Retry GatewayFilter Factory

Retry GatewayFilter fabrikası aşağıdaki parametreleri destekler:

- **retries**: Denenmesi gereken yeniden deneme sayısı.
- **statuses**: **org.springframework.http.HttpStatus** kullanılarak temsil edilen, yeniden denenmesi gereken HTTP durum kodları.
- **methods**: **org.springframework.http.HttpMethod** kullanılarak temsil edilen, yeniden denenmesi gereken HTTP yöntemleri.
- **series**: **org.springframework.http.HttpStatus.Series** kullanılarak temsil edilen, yeniden denenecek durum kodu dizisi.
- **exceptions**: Retries gereken atılan istisnaların listesi.
- **backoff**: Retries için yapılandırılmış üstel geri çekilme. Retries **firstBackoff * (factor ^ n)** değerinde bir geri çekilme aralığından sonra gerçekleştirilir; burada **n** yinelemedir. Eğer **maxBackoff** yapılandırılırsa, uygulanan maksimum geri çekilme **maxBackoff** ile sınırlıdır. Eğer **baseOnPreviousValue** true ise, backoff(geri çekilme) **prevBackoff * factor** kullanılarak hesaplanır.

Etkinleştirilmişse, **Retry** filtresi için aşağıdaki varsayılanlar yapılandırılır:

- **retries**: Üç kez
- **series**: 5XX serisi
- **methods**: GET yöntemi
- **exceptions**: **IOException** ve **TimeoutException**
- **backoff**: devre dışı (disabled)

Aşağıdaki liste, bir **Retry GatewayFilter**'ı yapılandırır:

Örnek 53. application.yml

```
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
            methods: GET,POST
            backoff:
              firstBackoff: 10ms
              maxBackoff: 50ms
              factor: 2
              basedOnPreviousValue: false
```

Bir **forward:** prefixed URL ile retries filtresini kullanırken, hedef bitiş noktası dikkatli bir şekilde yazılmalıdır, böylece bir hata durumunda, istemciye bir yanıtın gönderilmesine ve taahhüt edilmesine neden olabilecek hiçbir şey yapmaz. Örneğin, hedef uç nokta açıklamalı bir denetleyiciyse, hedef denetleyici yöntemi bir hata durum koduyla **ResponseEntity** döndürmemelidir. Bunun yerine, retries filtresinin yeniden denenerek işlemek üzere yapılandırılabileceği bir **Exception** oluşturmalı veya bir hata sinyali vermelidir (örneğin, bir **Mono.error(ex)** dönüş değeri aracılığıyla).

Bir gövdeye sahip herhangi bir HTTP yöntemiyle retries filtresini kullanırken, gövde önbelleğe alınır ve ağ geçidi bellek kısıtlı hale gelir. Gövde, **ServerWebExchangeUtils.CACHED_REQUEST_BODY_ATTR** tarafından tanımlanan bir istek özniteliğinde önbelleğe alınır. Nesnenin türü bir **org.springframework.core.io.buffer.DataBuffer**'dır.

Tek bir **status** ve **method** ile basitleştirilmiş bir " shortcut" gösterimi eklenebilir. Aşağıdaki iki örnek eşdeğerdir:

Örnek 54. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: retry_route
        uri: https://example.org
        filters:
        - name: Retry
          args:
            retries: 3
            statuses: INTERNAL_SERVER_ERROR
            methods: GET
            backoff:
              firstBackoff: 10ms
              maxBackoff: 50ms
              factor: 2
              basedOnPreviousValue: false

      - id: retryshortcut_route
        uri: https://example.org
        filters:
        - Retry=3,INTERNAL_SERVER_ERROR,GET,10ms,50ms,2,false
```

6.27. The RequestSize GatewayFilter Factory

İstek boyutu izin verilen sınırdan büyük olduğunda, **RequestSize GatewayFilter** fabrikası, bir isteğin aşağı akış hizmetine ulaşmasını kısıtlayabilir. Filtre bir **maxSize** parametresi alır. **maxSize** bir **DataSize** türüdür, bu nedenle değerler bir sayı ve ardından 'KB' veya 'MB' gibi isteğe bağlı bir DataUnit son eki olarak tanımlanabilir. Baytlar için varsayılan 'B'dir. Bayt cinsinden tanımlanan isteğin izin verilen boyut sınırıdır.
Aşağıdaki liste, bir **RequestSize GatewayFilter**'ı yapılandırır:

Örnek 55. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: request_size_route
        uri: http://localhost:8080/upload
        predicates:
        - Path=/upload
        filters:
        - name: RequestSize
          args:
            maxSize: 5000000
```

RequestSize GatewayFilter fabrikası, istek boyut nedeniyle reddedildiğinde, yanıt durumunu ek bir **errorMessage** başlığıyla **413 Payload Too Large** olarak ayarlar. Aşağıdaki örnekte böyle bir **errorMessage** gösterilmektedir:

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

Yol tanımında bir filtre bağımsız değişkeni olarak sağlanmadıysa, varsayılan istek boyutu 5 MB olarak ayarlanır.

6.28. The SetRequestHostHeader GatewayFilter Factory

Ana bilgisayar başlığının geçersiz kılınması gerekebileceği belirli durumlar vardır. Bu durumda, **SetRequestHostHeader GatewayFilter** fabrikası, mevcut host header belirtilen bir değerle değiştirebilir. Filtre bir **host** parametresi alır. Aşağıdaki liste, bir **SetRequestHostHeader GatewayFilter**'ı yapılandırır:

Örnek 56. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: set_request_host_header_route
        uri: http://localhost:8080/headers
        predicates:
        - Path=/headers
        filters:
        - name: SetRequestHostHeader
          args:
            host: example.org
```

SetRequestHostHeader GatewayFilter fabrikası, ana bilgisayar başlığının değerini **example.org** ile değiştirir.

6.29. Modify a Request Body GatewayFilter Factory

Ağ geçidi tarafından aşağı akışa gönderilmeden önce istek gövdesini değiştirmek için **ModifyRequestBody** filtre filtresini kullanabilirsiniz.

Bu filtre yalnızca Java DSL kullanılarak yapılandırılabilir.

Aşağıdaki liste, bir istek gövdesinin nasıl değiştirileceğini gösterir **GatewayFilter**:

```
@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;
   }
}
```

Eğer isteğin gövdesi yoksa, **RewriteFilter null** olarak geçirilir. İstekte eksik bir gövde atamak için **Mono.empty()** döndürülmelidir.

6.30. Modify a Response Body GatewayFilter Factory

İstemciye geri gönderilmeden önce yanıt gövdesini değiştirmek için **ModifyResponseBody** filtresini kullanabilirsiniz.
Bu filtre yalnızca Java DSL kullanılarak yapılandırılabilir.
Aşağıdaki liste, bir yanıt gövdesinin nasıl değiştirileceğini gösterir **GatewayFilter**:

```
@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();
}
```

Eğer yanıtın gövdesi yoksa, **RewriteFilter null** olarak geçirilir. Yanıtta eksik bir gövde atamak için **Mono.empty()** döndürülmelidir.

6.31. Token Relay GatewayFilter Factory

Belirteç Aktarımı, bir OAuth2 tüketicisinin Client olarak hareket ettiği ve gelen belirteci giden resource isteklerine ilettiği yerdir. Tüketici, saf bir Serveri (bir SSO uygulaması gibi) veya bir Resource Serveri olabilir.
Spring Cloud Gateway, OAuth2 access tokens, proxy yaptığı hizmetlere akış aşağı iletebilir.
Bu işlevi ağ geçidine eklemek için **TokenRelayGatewayFilterFactory**'yi şu şekilde eklemeniz gerekir:

App.java

```
@Bean
public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
   return builder.routes()
         .route("resource", r -> r.path("/resource")
               .filters(f -> f.tokenRelay())
               .uri("http://localhost:9000"))
         .build();
}
```

veya bu

application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: resource
        uri: http://localhost:9000
        predicates:
        - Path=/resource
        filters:
        - TokenRelay=
```

ve (kullanıcının oturum açmasına ve bir belirteç almasına ek olarak) kimlik doğrulama belirtecini hizmetlere (bu durumda **/resource**) iletir.
Bunu Spring Cloud Gateway için etkinleştirmek için aşağıdaki bağımlılıkları ekleyin

- **org.springframework.boot:spring-boot-starter-oauth2-client**

Peki bu nasıl çalışır?
{githubmaster}/src/main/java/org/springframework/cloud/gateway/security/TokenRelayGatewayFilterFactory.java[filter], o anda kimliği doğrulanmış kullanıcıdan bir erişim belirteci çıkarır ve onu aşağı akış istekleri için bir istek başlığına koyar.
Tam bir çalışma örneği için bu projeye bakın.

Bir **TokenRelayGatewayFilterFactory** çekirdeği yalnızca, bir **ReactiveClientRegistrationRepository** çekirdeğinin oluşturulmasını tetikleyecek uygun **spring.security.oauth2.client.*** özellikleri ayarlanmışsa oluşturulur.
TokenRelayGatewayFilterFactory tarafından kullanılan **ReactiveOAuth2AuthorizedClientService**'in varsayılan uygulaması, bir bellek içi veri deposu kullanır. Daha sağlam bir çözüme ihtiyacınız varsa, kendi uygulamanızı **ReactiveOAuth2AuthorizedClientService** sağlamanız gerekecektir.

6.32. The CacheRequestBody GatewayFilter Factory

Gövde okuması gereken bazı durumlar vardır. İstek gövdesi akışı yalnızca bir kez okunabildiğinden, istek gövdesini önbelleğe almamız gerekir. İstek gövdesini aşağı akışa göndermeden önce önbelleğe almak ve gövdeyi exchagne özniteliğinden almak için **CacheRequestBody** filtresini kullanabilirsiniz.

Aşağıdaki liste, istek gövdesi **GatewayFilter**'ın nasıl önbelleğe alınacağını gösterir:

```
@Bean
public RouteLocator routes(RouteLocatorBuilder builder) {
   return builder.routes()
      .route("cache_request_body_route", r -> r.path("/downstream/**")
         .filters(f -> f.prefixPath("/httpbin")
            .cacheRequestBody(String.class).uri(uri))
      .build();
}
```

Örnek 57. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: cache_request_body_route
        uri: lb://downstream
        predicates:
        - Path=/downstream/**
        filters:
        - name: CacheRequestBody
          args:
            bodyClass: java.lang.String
```

CacheRequestBody, istek gövdesini çıkarır ve onu gövde sınıfına dönüştürür (önceki örnekte tanımlanan **Java.lang.String** gibi). sonra onu **ServerWebExchangeUtils.CACHED_REQUEST_BODY_ATTR** içinde tanımlanan bir anahtarla **ServerWebExchange.getAttributes()** içine yerleştirir.

Bu filtre yalnızca http isteğiyle (https dahil) çalışır.

6.33. Default Filters

Bir filtre eklemek ve bunu tüm rotalara uygulamak için **spring.cloud.gateway.default-filters**'ı kullanabilirsiniz. Bu özellik bir filtre listesi alır. Aşağıdaki liste, bir dizi varsayılan filtreyi tanımlar:

Örnek 58. application.yml

```
spring:
  cloud:
    gateway:
      default-filters:
      - AddResponseHeader=X-Response-Default-Red, Default-Blue
      - PrefixPath=/httpbin
```

7. Global Filters

GlobalFilter arabirimi, GatewayFilter ile aynı imzaya sahiptir. Bunlar, tüm rotalara koşullu olarak uygulanan özel filtrelerdir. Bu arayüz ve kullanımı, gelecekteki dönüm noktası sürümlerinde değişikliğe tabidir.

7.1. Combined Global Filter and GatewayFilter Ordering

Bir istek bir rotayla eşleştiğinde, filtreleme web işleyicisi tüm **GlobalFilter** örneklerini ve tüm rotaya özgü **GatewayFilter** örneklerini bir filtre zincirine ekler. Bu birleşik filtre zinciri, **getOrder()** yöntemini uygulayarak ayarlayabileceğiniz **org.springframework.core.Ordered** arabirimine göre sıralanır.

Spring Cloud Gateway, filtre mantığı yürütmesi için " pre" ve " post" aşamaları arasında ayrım yaptığından (bkz. Nasıl Çalışır), en yüksek önceliğe sahip filtre, " pre " –phase ‘te ilk ve " post " -phase ‘ta sonuncudur.

Aşağıdaki liste bir filtre zincirini yapılandırır:

Örnek 59. ExampleConfiguration.java

```
@Bean
public GlobalFilter customFilter() {
    return new CustomGlobalFilter();
}

public class CustomGlobalFilter implements GlobalFilter, Ordered {

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        log.info("custom global filter");
        return chain.filter(exchange);
    }

    @Override
    public int getOrder() {
        return -1;
    }
}
```

7.2. Forward Routing Filter

ForwardRoutingFilter, **ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR** değişim özniteliğinde bir URI arar. URL'nin bir **forward** şeması varsa (örneğin **forward:///localendpoint**), isteği işlemek için Spring **DispatcherHandler**'ı kullanır. İstek URL'sinin yol kısmı, yönlendirme URL'sindeki yolla geçersiz kılınır. Değiştirilmemiş orijinal URL, **ServerWebExchangeUtils.GATEWAY_ORIGINAL_REQUEST_URL_ATTR** özelliğindeki listeye eklenir.

7.3. The ReactiveLoadBalancerClientFilter

ReactiveLoadBalancerClientFilter, **ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR** adlı değişim özniteliğinde bir URI arar. URL'nin bir **lb** şeması varsa (**lb://myservice** gibi), adı (bu örnekte **myservice**) gerçek bir host'a ve bağlantı noktasına çözümlemek için Spring Cloud **ReactorLoadBalancer**'ı kullanır ve aynı öznitelikte URI'yi değiştirir. Değiştirilmemiş orijinal URL, **ServerWebExchangeUtils.GATEWAY_ORIGINAL_REQUEST_URL_ATTR** özelliğindeki listeye eklenir. Filtre ayrıca **lb**'ye eşit olup olmadığını görmek için **ServerWebExchangeUtils.GATEWAY_SCHEME_PREFIX_ATTR** özniteliğine bakar. Eğer öyleyse, aynı kurallar geçerlidir. Aşağıdaki liste, bir **ReactiveLoadBalancerClientFilter**'ı yapılandırır:

Örnek 60. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: myRoute
        uri: lb://service
        predicates:
        - Path=/service/**
```

Varsayılan olarak, **ReactorLoadBalancer** tarafından bir hizmet örneği bulunamadığında bir **503** döndürülür. **Spring.cloud.gateway.loadbalancer.use404=true** ayarını yaparak ağ geçidini bir **404** döndürecek şekilde yapılandırabilirsiniz.

ReactiveLoadBalancerClientFilter'dan döndürülen **ServiceInstance**'ın **isSecure** değeri, Ağ Geçidine(gateway) yapılan istekte belirtilen düzeni geçersiz kılar. Örneğin, istek **HTTPS** üzerinden Ağ Geçidine gelirse ancak **ServiceInstance** bunun güvenli olmadığını belirtirse, aşağı akış isteği **HTTP** üzerinden yapılır. Tersi durum da geçerli olabilir. Ancak, Ağ Geçidi(gateway) yapılandırmasında yol için **GATEWAY_SCHEME_PREFIX_ATTR** belirtilirse, önek çıkarılır ve yol URL'sinden elde edilen şema **ServiceInstance** yapılandırmasını geçersiz kılar.

Gateway, tüm LoadBalancer özelliklerini destekler. Bunlar hakkında daha fazla bilgiyi Spring Cloud Commons belgelerinde okuyabilirsiniz.

7.4. The Netty Routing Filter

Netty yönlendirme filtresi, **ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR** değişim özniteliğinde bulunan URL bir **http** veya **https** şemasına sahipse çalışır. Aşağı akış(downstream) proxy isteğini yapmak için Netty **HttpClient**'i kullanır. Yanıt, daha sonraki bir filtrede kullanılmak üzere **ServerWebExchangeUtils.CLIENT_RESPONSE_ATTR** değişim özniteliğine yerleştirilir. (Aynı işlevi yerine getiren ancak Netty gerektirmeyen deneysel bir **WebClientHttpRoutingFilter** da vardır.)

7.5. The Netty Write Response Filter

NettyWriteResponseFilter, **ServerWebExchangeUtils.CLIENT_RESPONSE_ATTR** değişim özniteliğinde bir Netty **HttpClientResponse** varsa çalışır. Diğer tüm filtreler tamamlandıktan sonra çalışır ve proxy yanıtını ağ geçidi istemci yanıtına geri yazar. (Aynı işlevi yerine getiren ancak Netty gerektirmeyen deneysel bir **WebClientWriteResponseFilter** da vardır.)

7.6. The RouteToRequestUrl Filter

ServerWebExchangeUtils.GATEWAY_ROUTE_ATTR değişim özniteliğinde bir **Route** nesnesi varsa, **RouteToRequestUrlFilter** çalışır. İstek URI'sini temel alan ancak **Route** nesnesinin URI özniteliği ile güncellenen yeni bir URI oluşturur. Yeni URI, **ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR** değişim özniteliğine yerleştirilir.

URI'nin **lb:ws://serviceid** gibi bir şema öneki varsa, **lb** şeması URI'den çıkarılır ve daha sonra filtre zincirinde kullanılmak üzere **ServerWebExchangeUtils.GATEWAY_SCHEME_PREFIX_ATTR**'ye yerleştirilir.

7.7. The Websocket Routing Filter

ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR değişim özniteliğinde bulunan URL bir **ws** veya **wss** şemasına sahipse, websocket yönlendirme filtresi çalışır. Websocket isteğini akış yönünde iletmek için Spring WebSocket altyapısını kullanır.

URI'nin önüne **lb:ws://serviceid** gibi **lb** ekleyerek web yuvalarının yükünü dengeleyebilirsiniz.

Normal HTTP üzerinden bir geri dönüş olarak SockJS kullanıyorsanız, websocket Rotasının yanı sıra normal bir HTTP rotasını da yapılandırmanız gerekir.

Aşağıdaki liste, bir websocket yönlendirme filtresini yapılandırır:

Örnek 61. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      # SockJS route
      - id: websocket_sockjs_route
        uri: http://localhost:3001
        predicates:
        - Path=/websocket/info/**
      # Normal Websocket route
      - id: websocket_route
        uri: ws://localhost:3001
        predicates:
        - Path=/websocket/**
```

7.8. The Gateway Metrics Filter

Ağ Geçiti(Gateway) metriklerini etkinleştirmek için, proje bağımlılığı olarak spring-boot-starter-actuator ekleyin. Ardından, varsayılan olarak, ağ geçidi metrikleri filtresi, **spring.cloud.gateway.metrics.enabled** özelliği **false** olarak ayarlanmadığı sürece çalışır. Bu filtre, aşağıdaki etiketlerle birlikte **spring.cloud.gateway.requests** adlı bir zamanlayıcı metriği ekler:

- **routeId**: Yol kimliği.
- **routeUri**: API'nin yönlendirildiği URI.
- **outcome**: HttpStatus.Series tarafından sınıflandırıldığı şekliyle sonuç.
- **status**: Client'a(alıcıya) döndürülen isteğin HTTP durumu.
- **httpStatusCode**: Client'a(alıcıya) döndürülen isteğin HTTP Durumu.
- **httpMethod**: İstek için kullanılan HTTP yöntemi.

Ayrıca, **spring.cloud.gateway.metrics.tags.path.enabled** (varsayılan olarak false olarak ayarlanmıştır) özelliği aracılığıyla, etiketle fazladan bir metriği etkinleştirebilirsiniz:

- **path**: İsteğin yolu.

Bu metrikler daha sonra **/actuator/metrics/spring.cloud.gateway.requests** adresinden alınabilir ve bir Grafana panosu oluşturmak için Prometheus ile kolayca entegre edilebilir.

Prometheus uç noktasını etkinleştirmek için micrometer-registry-Prometheus'u proje bağımlılığı olarak ekleyin.

7.9. Marking An Exchange As Routed

Ağ geçidi bir **ServerWebExchange**'i yönlendirdikten sonra, exchange özniteliklerine **gatewayAlreadyRouted** ekleyerek bu değişimi "yönlendirildi(routed)" olarak işaretler. Bir istek yönlendirildi(routed) olarak işaretlendikten sonra, diğer yönlendirme filtreleri isteği yeniden yönlendirmez ve esasen filtreyi atlar. Bir değişimi yönlendirildi(routed) olarak işaretlemek veya bir santralin zaten yönlendirilip yönlendirilmediğini kontrol etmek için kullanabileceğiniz kolaylık yöntemleri vardır.

- **ServerWebExchangeUtils.isAlreadyRouted**, bir **ServerWebExchange** nesnesi alır ve "yönlendirilip yönlendirilmediğini" kontrol eder.
- **ServerWebExchangeUtils.setAlreadyRouted**, bir **ServerWebExchange** nesnesi alır ve onu "yönlendirilmiş "(routed) olarak işaretler.

8. HttpHeadersFilters

HttpHeadersFilters, **NettyRoutingFilter**'da olduğu gibi, isteklere aşağı akışa(downstream) gönderilmeden önce uygulanır.

8.1. Forwarded Headers Filter

Forwarded Headers(İletilen Başlıklar) Filtresi, aşağı akış hizmetine göndermek için bir **Forwarded** Header oluşturur. Mevcut isteğin **Host** header'ini(başlığını), şemasını ve portunu mevcut herhangi bir **Forwarded** başlığına ekler.

8.2. RemoveHopByHop Headers Filter

RemoveHopByHop Başlık Filtresi, iletilen isteklerden başlıkları kaldırır. Kaldırılan başlıkların varsayılan listesi IETF'den gelir.

Kaldırılan varsayılan başlıklar şunlardır:

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

Bunu değiştirmek için, **spring.cloud.gateway.filter.remove-hop-by-hop.headers** özelliğini kaldırılacak başlık adları listesine ayarlayın.

8.3. XForwarded Headers Filter

XForwarded Headers Filter, aşağı akış hizmetine göndermek için çeşitli **X-Forwarded-*** başlıkları oluşturur. Çeşitli başlıklar oluşturmak için mevcut isteğin Host başlığını, şemasını, bağlantı noktasını ve yolunu kullanır.

Bireysel başlıkların oluşturulması, aşağıdaki boole özellikleri tarafından kontrol edilebilir (varsayılanı true'dur):

- **spring.cloud.gateway.x-forwarded.for-enabled**
- **spring.cloud.gateway.x-forwarded.host-enabled**
- **spring.cloud.gateway.x-forwarded.port-enabled**
- **spring.cloud.gateway.x-forwarded.proto-enabled**
- **spring.cloud.gateway.x-forwarded.prefix-enabled**

Birden çok üstbilgi eklemek, aşağıdaki boole özellikleriyle kontrol edilebilir (varsayılanı true'dur):

- **spring.cloud.gateway.x-forwarded.for-append**
- **spring.cloud.gateway.x-forwarded.host-append**
- **spring.cloud.gateway.x-forwarded.port-append**
- **spring.cloud.gateway.x-forwarded.proto-append**
- **spring.cloud.gateway.x-forwarded.prefix-append**

9. TLS and SSL

Ağ geçidi, olağan Spring server yapılandırmasını izleyerek HTTPS'deki istekleri dinleyebilir.
Aşağıdaki örnek bunun nasıl yapılacağını gösterir:

Örnek 62. application.yml

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

Ağ geçidi yollarını hem HTTP hem de HTTPS arka uçlarına yönlendirebilirsiniz. Bir HTTPS arka ucuna yönlendirme yapıyorsanız, ağ geçidini aşağıdaki yapılandırmayla tüm aşağı akış sertifikalarına güvenecek şekilde yapılandırabilirsiniz:

Örnek 63. application.yml

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

Güvensiz bir güven yöneticisi kullanmak üretim için uygun değildir. Bir üretim dağıtımı için ağ geçidini, aşağıdaki yapılandırmayla güvenebileceği bir dizi bilinen sertifikayla yapılandırabilirsiniz:

Örnek 64. application.yml

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

Spring Cloud Gateway'e güvenilir sertifikalar sağlanmazsa, varsayılan güven deposu kullanılır (**javax.net.ssl.trustStore** sistem özelliğini ayarlayarak geçersiz kılabilirsiniz).

9.1. TLS Handshake

Ağ geçidi, arka uçlara yönlendirmek için kullandığı bir client(alıcı) havuzunu korur. HTTPS üzerinden iletişim kurarken, client(alıcı) bir TLS anlaşması başlatır. Bu el sıkışma ile ilişkili bir dizi zaman aşımı vardır.

Bu zaman aşımlarını aşağıdaki gibi yapılandırabilirsiniz (varsayılanlar gösterilir):

Örnek 65. application.yml

```
spring:
  cloud:
    gateway:
      httpclient:
        ssl:
          handshake-timeout-millis: 10000
          close-notify-flush-timeout-millis: 3000
          close-notify-read-timeout-millis: 0
```

10. Configuration (Yapılandırma)

Spring Cloud Gateway yapılandırması, **RouteDefinitionLocator** örnekleri koleksiyonu tarafından yürütülür.
Aşağıdaki liste, **RouteDefinitionLocator** arabiriminin tanımını gösterir:

Örnek 66. RouteDefinitionLocator.java

```
public interface RouteDefinitionLocator {
   Flux<RouteDefinition>
getRouteDefinitions();
}
```

Varsayılan olarak, bir **PropertiesRouteDefinitionLocator**, Spring Boot'un **@ConfigurationProperties** mekanizmasını kullanarak özellikleri yükler.
Önceki yapılandırma örneklerinin tümü, adlandırılmış olanlar yerine konumsal bağımsız değişkenleri kullanan bir kısayol gösterimi kullanır. Aşağıdaki iki örnek eşdeğerdir:

Örnek 67. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: setstatus_route
        uri: https://example.org
        filters:
        - name: SetStatus
          args:
            status: 401
      - id: setstatusshortcut_route
        uri: https://example.org
        filters:
        - SetStatus=401
```

Ağ geçidinin bazı kullanımları için özellikler yeterlidir, ancak bazı üretim kullanım durumları, yapılandırmayı veritabanı gibi harici bir kaynaktan yüklemekten yararlanır. Gelecekteki kilometre taşı sürümlerinde Redis, MongoDB ve Cassandra gibi Spring Data Repositories'e dayalı **RouteDefinitionLocator** uygulamaları olacaktır.

10.1. RouteDefinition Metrics

RouteDefinition metriklerini etkinleştirmek için, proje bağımlılığı olarak spring-boot-starter-actuator ekleyin. Ardından, varsayılan olarak, **spring.cloud.gateway.metrics.enabled** özelliği **true** olarak ayarlandığı sürece metrikler kullanılabilir olacaktır. Değeri **RouteDefinitions** sayısı olan **spring.cloud.gateway.routes.count** adlı bir gösterge metriği eklenecektir. Bu metrik, **/actuator/metrics/spring.cloud.gateway.routes.count** adresinden edinilebilir.

11. Route Metadata Configuration

Meta verileri kullanarak her yol için ek parametreleri aşağıdaki gibi yapılandırabilirsiniz:

Örnek 68. application.yml

```
spring:
  cloud:
    gateway:
      routes:
      - id: route_with_metadata
        uri: https://example.org
        metadata:
          optionName: "OptionValue"
          compositeObject:
            name: "value"
          iAmNumber: 1
```

Bir Exchange'den (takastan) tüm meta veri özelliklerini aşağıdaki gibi alabilirsiniz:

```
Route route = exchange.getAttribute(GATEWAY_ROUTE_ATTR);
// get all metadata properties
route.getMetadata();
// get a single metadata property
route.getMetadata(someKey);
```

12. Http timeouts configuration (Http zaman aşımları yapılandırması)

Http timeouts(response-yanıt ve connect-bağlantı) tüm rotalar için yapılandırılabilir ve her belirli rota için geçersiz kılınabilir.

12.1. Global timeouts (Genel zaman aşımları)

Global(genel) http zaman aşımlarını yapılandırmak için:

connect-timeout milisaniye cinsinden belirtilmelidir.

response-timeout, bir java.time.Duration olarak belirtilmelidir

global http zaman timouts(genel http zaman aşımı) örneği

```
spring:
  cloud:
    gateway:
      httpclient:
        connect-timeout: 1000
        response-timeout: 5s
```

12.2. Per-route timeouts (Rota başına zaman aşımları)

Rota başına zaman aşımlarını yapılandırmak için:
connect-timeout milisaniye cinsinden belirtilmelidir.
response-timeout milisaniye cinsinden belirtilmelidir.

yapılandırma yoluyla per-route http timeouts(rota başına http zaman aşımları) yapılandırması

```
- id: per_route_timeouts
      uri: https://example.org
      predicates:
       - name: Path
         args:
           pattern: /delay/{timeout}
      metadata:
        response-timeout: 200
        connect-timeout: 200
```

Java DSL kullanarak rota başına zaman aşımı(per-route http timeouts) yapılandırması

```
import static org.springframework.cloud.gateway.support.RouteMetadataUtils.CONNECT_TIMEOUT_ATTR;
import static org.springframework.cloud.gateway.support.RouteMetadataUtils.RESPONSE_TIMEOUT_ATTR;

    @Bean
    public RouteLocator customRouteLocator(RouteLocatorBuilder routeBuilder){
      return routeBuilder.routes()
           .route("test1", r -> {
             return r.host("*.somehost.org").and().path("/somepath")
                  .filters(f -> f.addRequestHeader("header1", "header-value-1"))
                  .uri("http://someuri")
                  .metadata(RESPONSE_TIMEOUT_ATTR, 200)
                  .metadata(CONNECT_TIMEOUT_ATTR, 200);
           })
           .build();
    }
```

Negatif bir değere sahip rota başına **response-timeout**, genel **response-timeout** değerini devre dışı bırakır.

```
- id: per_route_timeouts
      uri: https://example.org
      predicates:
       - name: Path
         args:
           pattern: /delay/{timeout}
      metadata:
        response-timeout: -1
```

12.3. Fluent Java Routes API

Java'da basit yapılandırmaya izin vermek için **RouteLocatorBuilder** bean, akıcı bir API içerir. Aşağıdaki liste nasıl çalıştığını gösterir:

Example 69. GatewaySampleApplication.java

```
// static imports from GatewayFilters and RoutePredicates
@Bean
public RouteLocator customRouteLocator(RouteLocatorBuilder builder, ThrottleGatewayFilterFactory throttle) {
   return builder.routes()
         .route(r -> r.host("**.abc.org").and().path("/image/png")
            .filters(f ->
                  f.addResponseHeader("X-TestHeader", "foobar"))
            .uri("http://httpbin.org:80")
         )
         .route(r -> r.path("/image/webp")
            .filters(f ->
                  f.addResponseHeader("X-AnotherHeader", "baz"))
            .uri("http://httpbin.org:80")
            .metadata("key", "value")
         )
         .route(r -> r.order(-1)
            .host("**.throttle.org").and().path("/get")
            .filters(f -> f.filter(throttle.apply(1,
                  1,
                  10,
                  TimeUnit.SECONDS)))
            .uri("http://httpbin.org:80")
            .metadata("key", "value")
         )
         .build();
}
```

Bu stil ayrıca daha özel yüklem iddialarına izin verir. **RouteDefinitionLocator** çekirdekleri tarafından tanımlanan yüklemler, mantıksal **and** kullanılarak birleştirilir. Akıcı Java API'sini kullanarak, **Predicate** sınıfında **and()**, **or()** ve **negate()** operatörlerini kullanabilirsiniz.

12.4. The DiscoveryClient Route Definition Locator

Ağ geçidini, **DiscoveryClient** uyumlu hizmet kayıt defterine kayıtlı hizmetlere dayalı olarak rotalar oluşturacak şekilde yapılandırabilirsiniz.

Bunu etkinleştirmek için **spring.cloud.gateway.discovery.locator.enabled=true** ayarlayın ve bir **DiscoveryClient** uygulamasının (Netflix Eureka, Consul veya Zookeeper gibi) sınıf yolunda ve etkin olduğundan emin olun.

12.4.1. Configuring Predicates and Filters For DiscoveryClient Routes

Varsayılan olarak ağ geçidi, **DiscoveryClient** ile oluşturulan rotalar için tek bir predicate ve filter tanımlar.
Varsayılan predicate, **/serviceId/**** patterns(kalıbıyla) ile tanımlanan bir yol predicate'idir(yüklemidir); burada **serviceId**, **DiscoveryClient**'ten alınan hizmet ID'sidir.
Varsayılan filter, normal ifade(regex) **/serviceId/?(?<remaining>.*)** ve yerine **/${remaining}** olan bir yeniden yazma yolu filtresidir. Bu, istek aşağı akışa gönderilmeden önce hizmet ID'sini yoldan çıkarır.
DiscoveryClient yolları tarafından kullanılan tahminleri veya filtreleri özelleştirmek istiyorsanız, **spring.cloud.gateway.discovery.locator.predicates[x]** ve **spring.cloud.gateway.discovery.locator.filters[y]** ayarlayın.
Bunu yaparken, bu işlevi korumak istiyorsanız, daha önce gösterilen varsayılan predicate'yi ve filtreyi eklediğinizden emin olmanız gerekir. Aşağıdaki örnek bunun nasıl göründüğünü gösterir:

Örnek 70. application.properties

```
spring.cloud.gateway.discovery.locator.predicates[0].name: Path
spring.cloud.gateway.discovery.locator.predicates[0].args[pattern]: "'/'+serviceId+'/**'"
spring.cloud.gateway.discovery.locator.predicates[1].name: Host
spring.cloud.gateway.discovery.locator.predicates[1].args[pattern]: "'**.foo.com'"
spring.cloud.gateway.discovery.locator.filters[0].name: CircuitBreaker
spring.cloud.gateway.discovery.locator.filters[0].args[name]: serviceId
spring.cloud.gateway.discovery.locator.filters[1].name: RewritePath
spring.cloud.gateway.discovery.locator.filters[1].args[regexp]: "'/' + serviceId + '/?(?<remaining>.*)'"
spring.cloud.gateway.discovery.locator.filters[1].args[replacement]: "'/${remaining}'"
```

13. Reactor Netty Access Logs

Reactor Netty erişim günlüklerini etkinleştirmek için **-Dreactor.netty.http.server.accessLogEnabled=true** ayarlayın.

Bir Spring Boot özelliği değil, bir Java Sistem Özelliği olmalıdır.

Günlük sistemini ayrı bir erişim günlüğü dosyasına sahip olacak şekilde yapılandırabilirsiniz. Aşağıdaki örnek bir Logback konfigürasyonu oluşturur:

Örnek 71. logback.xml

```
<appender name="accessLog" class="ch.qos.logback.core.FileAppender">
    <file>access_log.log</file>
    <encoder>
      <pattern>%msg%n</pattern>
    </encoder>
  </appender>
  <appender name="async" class="ch.qos.logback.classic.AsyncAppender">
    <appender-ref ref="accessLog" />
  </appender>

  <logger name="reactor.netty.http.server.AccessLog" level="INFO" additivity="false">
    <appender-ref ref="async"/>
  </logger>
```

14. CORS Configuration

CORS davranışını kontrol etmek için ağ geçidini yapılandırabilirsiniz. " global" CORS yapılandırması, Spring Framework **CorsConfiguration**'a yönelik URL kalıplarının bir haritasıdır. Aşağıdaki örnek, CORS'u yapılandırır:

Örnek 72. application.yml

```
spring:
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowedOrigins: "https://docs.spring.io"
            allowedMethods:
            - GET
```

Önceki örnekte, GET tarafından istenen tüm yollar için **docs.spring.io**'dan kaynaklanan isteklerden CORS isteklerine izin verilir.

Bazı ağ geçidi rota predicate tarafından işlenmeyen isteklere aynı CORS yapılandırmasını sağlamak için, **spring.cloud.gateway.globalcors.add-to-simple-url-handler-mapping** özelliğini **true** olarak ayarlayın. Bu, CORS ön kontrol isteklerini desteklemeye çalıştığınızda kullanışlıdır ve HTTP yöntemi **options** olduğundan rota predicate'iniz **true** olarak değerlendirilmez.

15. Actuator API

/gateway aktüatör uç noktası, bir Spring Cloud Gateway uygulamasını izlemenize ve bunlarla etkileşim kurmanıza olanak tanır. Uzaktan erişilebilir olması için, uç noktanın etkinleştirilmesi ve uygulama özelliklerinde HTTP veya JMX üzerinden gösterilmesi gerekir. Aşağıdaki liste bunun nasıl yapılacağını gösterir:

Örnek 73. application.properties

```
management.endpoint.gateway.enabled=true # default value
management.endpoints.web.exposure.include=gateway
```

15.1. Verbose Actuator Format

Spring Cloud Gateway'e yeni, daha ayrıntılı bir format eklendi. Her bir rotaya daha fazla ayrıntı ekleyerek, mevcut herhangi bir konfigürasyonla birlikte her bir rota ile ilişkili tahminleri ve filtreleri görüntülemenize izin verir. Aşağıdaki örnek, **/actuator/gateway/routes**'ı yapılandırır:

```
[
 {
  "predicate": "(Hosts: [**.addrequestheader.org] && Paths: [/headers], match trailing slash: true)",
  "route_id": "add_request_header_test",
  "filters": [
   "[[AddResponseHeader X-Response-Default-Foo = 'Default-Bar'], order = 1]",
   "[[AddRequestHeader X-Request-Foo = 'Bar'], order = 1]",
   "[[PrefixPath prefix = '/httpbin'], order = 2]"
  ],
  "uri": "lb://testservice",
  "order": 0
 }
]
```

Bu özellik varsayılan olarak etkindir. Devre dışı bırakmak için aşağıdaki özelliği ayarlayın:

Örnek 74. application.properties

```
spring.cloud.gateway.actuator.verbose.enabled=false
```

Bu, gelecekteki bir sürümde varsayılan olarak **true** olacaktır.

15.2. Retrieving Route Filters

Bu bölüm, aşağıdakiler dahil olmak üzere rota filtrelerinin nasıl alınacağını ayrıntılı olarak açıklar:

15.2.1 Global Filters

16.2.2 [gateway-route-filters]

15.2.1. Global Filters

Tüm rotalara uygulanan genel filtreleri almak için **/actuator/gateway/globalfilters**'a bir **GET** isteği yapın. Ortaya çıkan yanıt aşağıdakine benzer:

```
{
  "org.springframework.cloud.gateway.filter.ReactiveLoadBalancerClientFilter@77856cc5": 10100,
  "org.springframework.cloud.gateway.filter.RouteToRequestUrlFilter@4f6fd101": 10000,
  "org.springframework.cloud.gateway.filter.NettyWriteResponseFilter@32d22650": -1,
  "org.springframework.cloud.gateway.filter.ForwardRoutingFilter@106459d9": 2147483647,
  "org.springframework.cloud.gateway.filter.NettyRoutingFilter@1fbd5e0": 2147483647,
  "org.springframework.cloud.gateway.filter.ForwardPathFilter@33a71d23": 0,
  "org.springframework.cloud.gateway.filter.AdaptCachedBodyGlobalFilter@135064ea": 2147483637,
  "org.springframework.cloud.gateway.filter.WebsocketRoutingFilter@23c05889": 2147483646
}
```

Yanıt(Response), yürürlükte olan genel filtrelerin ayrıntılarını içerir. Her genel filtre için, filtre nesnesinin bir dize temsili (örneğin, **org.springframework.cloud.gateway.filter.ReactiveLoadBalancerClientFilter@77856cc5**) ve filtre zincirindeki karşılık gelen sıra vardır.}

15.2.2. Route Filters

Rotalara uygulanan **GatewayFilter** fabrikalarını almak için **/actuator/gateway/routefilters** öğesine bir **GET** isteği yapın. Ortaya çıkan yanıt aşağıdakine benzer:

```
{
  "[AddRequestHeaderGatewayFilterFactory@570ed9c configClass =
AbstractNameValueGatewayFilterFactory.NameValueConfig]": null,
  "[SecureHeadersGatewayFilterFactory@fceab5d configClass = Object]": null,
  "[SaveSessionGatewayFilterFactory@4449b273 configClass = Object]": null
}
```

Yanıt(Response), belirli bir rotaya uygulanan **GatewayFilter** fabrikalarının ayrıntılarını içerir. Her fabrika için karşılık gelen nesnenin bir dize temsili vardır (örneğin, **[SecureHeadersGatewayFilterFactory@fceab5d configClass = Object]**). **null** değerin, bir **GatewayFilter** fabrika nesnesi için geçerli olmayan filtre zincirindeki nesnenin sırasını ayarlamaya çalıştığından, uç nokta denetleyicisinin eksik uygulanmasından kaynaklandığını unutmayın.

15.3. Refreshing the Route Cache

Rota önbelleğini temizlemek için **/actuator/gateway/refresh** için bir **POST** isteği yapın. İstek, yanıt gövdesi olmayan bir 200 döndürür.

15.4. Retrieving the Routes Defined in the Gateway

Ağ geçidinde tanımlanan rotaları almak için **/actuator/gateway/routes**'a bir **GET** isteği yapın.

Ortaya çıkan yanıt aşağıdakine benzer:


```
[{
  "route_id": "first_route",
  "route_object": {
    "predicate": "org.springframework.cloud.gateway.handler.predicate.PathRoutePredicateFactory$$Lambda$432/1736826640@1e9d7e7d",
    "filters": [
      "OrderedGatewayFilter{delegate=org.springframework.cloud.gateway.filter.factory.PreserveHostHeaderGatewayFilterFactory$$Lambda$436/674480275@6631ef72, order=0}"
    ]
  },
  "order": 0
},
{
  "route_id": "second_route",
  "route_object": {
    "predicate": "org.springframework.cloud.gateway.handler.predicate.PathRoutePredicateFactory$$Lambda$432/1736826640@cd8d298",
    "filters": []
  },
  "order": 0
}]
```


Yanıt, ağ geçidinde tanımlanan tüm rotaların ayrıntılarını içerir. Aşağıdaki tablo, yanıtın her bir öğesinin (her biri bir rotadır) yapısını açıklar:

Path(Yol)	Type(Tip)	Description(Tanım)
route_id	String	Rota ID'si.
route_object.predicate	Object	Rota yüklemi(predicate).
route_object.filters	Array	Rotaya uygulanan GatewayFilter fabrikaları.
order	Number	Rota sırası.

15.5. Retrieving Information about a Particular Route

Tek bir rota hakkında bilgi almak için **/actuator/gateway/routes/{id}** adresine bir **GET** isteği yapın (örneğin, **/actuator/gateway/routes/first_route**). Ortaya çıkan yanıt aşağıdakine benzer:

```
{
  "id": "first_route",
  "predicates": [{
    "name": "Path",
    "args": {"_genkey_0":"/first"}
  }],
  "filters": [],
  "uri": "https://www.uri-destination.org",
  "order": 0
}
```

Aşağıdaki tabloda yanıtın yapısı açıklanmaktadır:

Path(Yol)	Type(Tipi)	Description(Tanım)
id	String	Rota ID'si.
predicates	Array	Rota tahminlerinin toplanması. Her öğe, belirli bir yüklemin(predicate'ın) adını ve argümanlarını tanımlar.
filters	Array	Rotaya uygulanan filtrelerin koleksiyonu.
uri	String	Rotanın hedef URI'si.
order	Number	Rota sırası.

15.6. Creating and Deleting a Particular Route

Bir rota oluşturmak için, rotanın alanlarını belirten bir JSON gövdesiyle **/gateway/routes/{id_route_to_create}** adresine bir **POST** isteği yapın (bkz. Belirli Bir Rota Hakkında Bilgi Alma).
Bir rotayı silmek için **/gateway/routes/{id_route_to_delete}** için bir **DELETE** isteği yapın.

15.7. Recap: The List of All endpoints

Aşağıdaki tablo, Spring Cloud Gateway aktüatör uç noktalarını özetlemektedir (her uç noktanın temel yol olarak **/actuator/gateway**'e sahip olduğunu unutmayın):

ID	HTTP Method	Description(Tanım)
globalfilters	GET	Rotalara uygulanan global filtrelerin listesini görüntüler.
routefilters	GET	Belirli bir rotaya uygulanan **GatewayFilter** fabrikalarının listesini görüntüler.
refresh	POST	Rota önbelleğini temizler.
routes	GET	Ağ geçidinde tanımlanan rotaların listesini görüntüler.
routes/{id}	GET	Belirli bir rota hakkında bilgi görüntüler.
routes/{id}	POST	Ağ geçidine yeni bir rota ekler.
routes/{id}	DELETE	Mevcut bir rotayı gateway'dan(ağ geçidinden) kaldırır.

15.8. Sharing Routes between multiple Gateway instances

Spring Cloud Gateway, iki **RouteDefinitionRepository** uygulaması sunar. Birincisi, yalnızca bir Gateway örneğinin belleğinde yaşayan **InMemoryRouteDefinitionRepository**'dir. Bu repository türü, birden çok Ağ Geçidi örneğinde Rotaları doldurmak için uygun değildir.

Bir Spring Cloud Gateway örneği kümesinde Rotaları paylaşmak için **RedisRouteDefinitionRepository** kullanılabilir. Bu tür bir repository'i etkinleştirmek için aşağıdaki özelliğin **true** olarak ayarlanması gerekir: **spring.cloud.gateway.redis-route-definition-repository.enabled** RedisRateLimiter Filtre Fabrikası'na benzer şekilde, spring-boot-starter-data-redis-reactive Spring Boot başlatıcısının kullanılmasını gerektirir.

16. Troubleshooting (Sorun Giderme)

Bu bölüm, Spring Cloud Gateway kullandığınızda ortaya çıkabilecek genel sorunları kapsar.

16.1. Log Levels

Aşağıdaki günlükçüler, **DEBUG** ve **TRACE** düzeylerinde değerli sorun giderme bilgileri içerebilir:

- ❑ **org.springframework.cloud.gateway**
- ❑ **org.springframework.http.server.reactive**
- ❑ **org.springframework.web.reactive**
- ❑ **org.springframework.boot.autoconfigure.web**
- ❑ **reactor.netty**
- ❑ **redisratelimiter**

16.2. Wiretap

Reactor Netty **HttpClient** ve **HttpServer**, telefon dinlemeyi etkinleştirebilir. **reactor.netty** günlük düzeyinin **DEBUG** veya **TRACE** olarak ayarlanmasıyla birleştirildiğinde, kablo üzerinden gönderilen ve alınan başlıklar ve gövdeler gibi bilgilerin günlüğe kaydedilmesini sağlar. Dinlemeyi etkinleştirmek için **HttpServer** ve **HttpClient** için sırasıyla **spring.cloud.gateway.httpserver.wiretap=true** veya **spring.cloud.gateway.httpclient.wiretap=true** ayarlayın.

17. Developer Guide (Geliştirici Rehberi-Kılavuzu)

Bunlar, ağ geçidinin bazı özel bileşenlerini yazmaya yönelik temel kılavuzlardır.

17.1. Writing Custom Route Predicate Factories

Bir Route Predicate yazmak için **RoutePredicateFactory**'yi bean olarak uygulamanız gerekecek.
Genişletebileceğiniz **AbstractRoutePredicateFactory** adında soyut bir sınıf var.

MyRoutePredicateFactory.java

```
@Component
public class MyRoutePredicateFactory extends AbstractRoutePredicateFactory<MyRoutePredicateFactory.Config> {

   public MyRoutePredicateFactory() {
      super(Config.class);
   }

   @Override
   public Predicate<ServerWebExchange> apply(Config config) {
      // grab configuration from Config object
      return exchange -> {
         //grab the request
         ServerHttpRequest request = exchange.getRequest();
         //take information from the request to see if it
         //matches configuration.
         return matches(config, request);
      };
   }

   public static class Config {
      //Put the configuration properties for your filter here
   }

}
```

17.2. Writing Custom GatewayFilter Factories

GatewayFilter yazmak için, **GatewayFilterFactory**'yi bean olarak uygulamanız gerekir. **AbstractGatewayFilterFactory** adlı soyut bir sınıfı genişletebilirsiniz. Aşağıdaki örnekler bunun nasıl yapılacağını göstermektedir:

Örnek 75. PreGatewayFilterFactory.java

```
@Component
public class PreGatewayFilterFactory extends AbstractGatewayFilterFactory<PreGatewayFilterFactory.Config> {

   public PreGatewayFilterFactory() {
      super(Config.class);
   }

   @Override
   public GatewayFilter apply(Config config) {
      // grab configuration from Config object
      return (exchange, chain) -> {
         //If you want to build a "pre" filter you need to manipulate the
         //request before calling chain.filter
         ServerHttpRequest.Builder builder = exchange.getRequest().mutate();
         //use builder to manipulate the request
         return chain.filter(exchange.mutate().request(builder.build()).build());
      };
   }

   public static class Config {
      //Put the configuration properties for your filter here
   }

}
```

PostGatewayFilterFactory.java

```
@Component
public class PostGatewayFilterFactory extends AbstractGatewayFilterFactory<PostGatewayFilterFactory.Config>
{

   public PostGatewayFilterFactory() {
      super(Config.class);
   }

   @Override
   public GatewayFilter apply(Config config) {
      // grab configuration from Config object
      return (exchange, chain) -> {
         return chain.filter(exchange).then(Mono.fromRunnable(() -> {
            ServerHttpResponse response = exchange.getResponse();
            //Manipulate the response in some way
         }));
      };
   }

   public static class Config {
      //Put the configuration properties for your filter here
   }

}
```

17.2.1. Naming Custom Filters And References In Configuration

Özel filtreler sınıf adları **GatewayFilterFactory** ile bitmelidir.
Örneğin, yapılandırma dosyalarında **Something** adlı bir filtreye başvurmak için filtrenin **SomethingGatewayFilterFactory** adlı bir sınıfta olması gerekir.

AnotherThing class gibi **GatewayFilterFactory** son eki olmadan adlandırılmış bir ağ geçidi filtresi oluşturmak mümkündür. Bu filtre, yapılandırma dosyalarında **AnotherThing** olarak adlandırılabilir. Bu, desteklenen bir adlandırma kuralı değildir ve bu sözdizimi gelecekteki sürümlerde kaldırılabilir. Lütfen filtre adını uyumlu olacak şekilde güncelleyin.

17.3. Writing Custom Global Filters

Özel bir global filtre yazmak için **GlobalFilter** arabirimini bean olarak uygulamanız gerekir. Bu, filtreyi tüm isteklere uygular.

Aşağıdaki örnekler, sırasıyla genel pre ve post filtrelerin nasıl ayarlanacağını gösterir:

```
@Bean
public GlobalFilter customGlobalFilter() {
   return (exchange, chain) -> exchange.getPrincipal()
      .map(Principal::getName)
      .defaultIfEmpty("Default User")
      .map(userName -> {
        //adds header to proxied request
        exchange.getRequest().mutate().header("CUSTOM-REQUEST-HEADER", userName).build();
        return exchange;
      })
      .flatMap(chain::filter);
}

@Bean
public GlobalFilter customGlobalPostFilter() {
   return (exchange, chain) -> chain.filter(exchange)
      .then(Mono.just(exchange))
      .map(serverWebExchange -> {
        //adds header to response
        serverWebExchange.getResponse().getHeaders().set("CUSTOM-RESPONSE-HEADER",
            HttpStatus.OK.equals(serverWebExchange.getResponse().getStatusCode()) ? "It worked": "It did not work");
        return serverWebExchange;
      })
      .then();
}
```

18. Building a Simple Gateway by Using Spring MVC or Webflux

Aşağıda alternatif bir stil ağ geçidi açıklanmaktadır. Önceki belgelerin hiçbiri aşağıdakiler için geçerli değildir.

Spring Cloud Gateway, **ProxyExchange** adlı bir yardımcı program nesnesi sağlar. Bunu, bir yöntem parametresi olarak normal bir Spring web işleyicisi içinde kullanabilirsiniz. HTTP fiillerini yansıtan yöntemlerle temel aşağı akış HTTP alışverişlerini destekler. MVC ile, **forward()** yöntemi aracılığıyla yerel bir işleyiciye iletmeyi de destekler. **ProxyExchange**'i kullanmak için sınıf yolunuza doğru modülü ekleyin (ya **spring-cloud-gateway-mvc** ya da **spring-cloud-gateway-webflux**).

Aşağıdaki MVC örneği, bir uzak sunucuya akış yönünde **/test** etme isteğinin proxy'sini oluşturur:

```
@RestController
@SpringBootApplication
public class GatewaySampleApplication {

   @Value("${remote.home}")
   private URI home;

   @GetMapping("/test")
   public ResponseEntity<?> proxy(ProxyExchange<byte[]> proxy) throws Exception {
      return proxy.uri(home.toString() + "/image/png").get();
   }

}
```

Aşağıdaki örnek Webflux ile aynı şeyi yapıyor:

```
@RestController
@SpringBootApplication
public class GatewaySampleApplication {

   @Value("${remote.home}")
   private URI home;

   @GetMapping("/test")
   public Mono<ResponseEntity<?>> proxy(ProxyExchange<byte[]> proxy) throws
Exception {
      return proxy.uri(home.toString() + "/image/png").get();
   }

}
```

ProxyExchange üzerindeki kolaylık yöntemleri, işleyici yönteminin gelen isteğin URI yolunu keşfetmesini ve geliştirmesini sağlar. Örneğin, onları aşağı akışa geçirmek için bir yolun sondaki öğelerini çıkarmak isteyebilirsiniz:

```
@GetMapping("/proxy/path/**")
public ResponseEntity<?> proxyPath(ProxyExchange<byte[]> proxy) throws Exception {
  String path = proxy.path("/proxy/path/");
  return proxy.uri(home.toString() + "/foos/" + path).get();
}
```

Spring MVC ve Webflux'un tüm özellikleri, ağ geçidi işleyici yöntemlerinde kullanılabilir. Sonuç olarak, örneğin istek başlıklarını ve sorgu parametrelerini enjekte edebilir ve gelen istekleri eşleme ek açıklamasındaki bildirimlerle sınırlayabilirsiniz. Bu özelliklerle ilgili daha fazla ayrıntı için Spring MVC'de **@RequestMapping** belgelerine bakın.

ProxyExchange'te **header()** yöntemlerini kullanarak aşağı akış yanıtına üstbilgiler ekleyebilirsiniz.

Ayrıca get() methoduna (ve diğer methods) bir mapper(eşleştrici) ekleyerek yanıt başlıklarını (ve yanıtta beğendiğiniz herhangi bir şeyi) değiştirebilirsiniz. Mapper(Eşleştirici), gelen **ResponseEntity**'yi alıp gidene dönüştüren bir **function**'dur.

" Sensitive" başlıklar (varsayılan olarak, **cookie** ve **authorization**) için alt akışa geçirilmeyen ve "proxy" (**x-forwarded-***) başlıkları için birinci sınıf destek sağlanır.

19. Configuration properties

Spring Cloud Gateway ile ilgili tüm yapılandırma özelliklerinin listesini görmek için eke bakın.

19. Configuration properties

EK: Common application properties

application.properties dosyanızın içinde, **application.yml** dosyanızın içinde veya komut satırı anahtarları olarak çeşitli özellikler belirtilebilir. Bu ek, yaygın Spring Cloud Gateway özelliklerinin bir listesini ve bunları tüketen temel sınıflara referanslar sağlar. Özellik katkıları, sınıf yolunuzdaki ekjar dosyalarından gelebilir, bu nedenle bunu kapsamlı bir liste olarak düşünmemelisiniz. Ayrıca, kendi özelliklerinizi tanımlayabilirsiniz.

Name	Default	Description
spring.cloud.gateway.default-filters		Her rotaya uygulanan filtre tanımlarının listesi.
spring.cloud.gateway.discovery.locator.enabled	**false**	DiscoveryClient ağ geçidi entegrasyonunu sağlayan bayrak.
spring.cloud.gateway.discovery.locator.filters		
spring.cloud.gateway.discovery.locator.include-expression	**true**	Bir hizmetin ağ geçidi entegrasyonuna dahil edilip edilmeyeceğini değerlendirecek olan SpEL ifadesi, varsayılan olarak şu şekildedir: true.
spring.cloud.gateway.discovery.locator.lower-case-service-id	**false**	Tahminlerde ve filtrelerde serviceId'yi küçük harf seçeneği, varsayılan olarak false olur. ServiceId'yi otomatik olarak büyük harfle yazdığında eureka ile kullanışlıdır. yani MYSERIVCE, /myservice/** ile eşleşir
spring.cloud.gateway.discovery.locator.predicates		
spring.cloud.gateway.discovery.locator.route-id-prefix		routeId öneki, varsayılan olarak discoveryClient.getClass().getSimpleName() + "_" şeklindedir. RouteId'yi oluşturmak için Hizmet Kimliği eklenecektir.
spring.cloud.gateway.discovery.locator.url-expression	**'lb://'+serviceId**	Her yol için uri'yi oluşturan SpEL ifadesi, varsayılan olarak 'lb://'+serviceId'dir.
spring.cloud.gateway.enabled	**true**	Ağ geçidi(gateway) işlevselliğini etkinleştirir.

spring.cloud.gateway.fail-on-route-definition-error	**true**	Rota tanımlama hatalarında başarısız olma seçeneği, varsayılan olarak true olur. Aksi takdirde, bir uyarı günlüğe kaydedilir.
spring.cloud.gateway.filter.add-request-header.enabled	**true**	Add-request-header filtresini etkinleştirir.
spring.cloud.gateway.filter.add-request-parameter.enabled	**true**	Add-request-parametre filtresini etkinleştirir.
spring.cloud.gateway.filter.add-response-header.enabled	**true**	add-response-header filtresini etkinleştir
spring.cloud.gateway.filter.circuit-breaker.enabled	**true**	circuit-breaker filtresini etkinleştir
spring.cloud.gateway.filter.dedupe-response-header.enabled	**true**	dedupe-response-header filtresini etkinleştir
spring.cloud.gateway.filter.fallback-headers.enabled	**true**	fallback-headers filtresini etkinleştir
spring.cloud.gateway.filter.hystrix.enabled	**true**	Hystrix filtresini etkinleştir
spring.cloud.gateway.filter.map-request-header.enabled	**true**	map-request-header filtresini etkinleştir
spring.cloud.gateway.filter.modify-request-body.enabled	**true**	modify-request-body filtresini etkinleştir
spring.cloud.gateway.filter.modify-response-body.enabled	**true**	modify-response-body filtresini etkinleştir
spring.cloud.gateway.filter.prefix-path.enabled	**true**	prefix-path filtresini etkinleştir

spring.cloud.gateway.filter.preserve-host-header.enabled	**true**	preserve-host-header filtresini etkinleştir.
spring.cloud.gateway.filter.redirect-to.enabled	**true**	redirect-to filtresini etkinleştir
spring.cloud.gateway.filter.remove-hop-by-hop.headers		
spring.cloud.gateway.filter.remove-hop-by-hop.order		
spring.cloud.gateway.filter.remove-request-header.enabled	**true**	remove-request-header filtresini etkinleştir
spring.cloud.gateway.filter.remove-request-parameter.enabled	**true**	remove-request-parameter filtresini etkinleştir
spring.cloud.gateway.filter.remove-response-header.enabled	**true**	remove-response-header filtresini etkinleştir
spring.cloud.gateway.filter.request-header-size.enabled	**true**	request-header-size filtresini etkinleştir
spring.cloud.gateway.filter.request-header-to-request-uri.enabled	**true**	request-header-to-request-uri filtresini etkinleştir
spring.cloud.gateway.filter.request-rate-limiter.deny-empty-key	**true**	Anahtar Çözümleyici boş bir anahtar döndürürse istekleri reddetmeye geçin, varsayılan olarak true olur
spring.cloud.gateway.filter.request-rate-limiter.empty-key-status-code		DenenEmptyKey doğru olduğunda döndürülecek HttpStatus, varsayılan olarak FORBITTEN'dir.
spring.cloud.gateway.filter.request-rate-limiter.enabled	**true**	request-rate-limiter filtresini etkinleştir

spring.cloud.gateway.filter.request-size.enabled	**true**	request-size filtresini etkinleştir
spring.cloud.gateway.filter.retry.enabled	**true**	retry filtresini etkinleştir
spring.cloud.gateway.filter.rewrite-location-response-header.enabled	**true**	rewrite-location-response-header filtresini etkinleştir
spring.cloud.gateway.filter.rewrite-location.enabled	**true**	rewrite-location filtresini etkinleştir
spring.cloud.gateway.filter.rewrite-path.enabled	**true**	rewrite-path filtresini etkinleştir
spring.cloud.gateway.filter.rewrite-response-header.enabled	**true**	rewrite-response-header filtresini etkinleştir
spring.cloud.gateway.filter.save-session.enabled	**true**	save-session filtresini etkinleştir
spring.cloud.gateway.filter.secure-headers.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'**	
spring.cloud.gateway.filter.secure-headers.content-type-options	**nosniff**	
spring.cloud.gateway.filter.secure-headers.disable		
spring.cloud.gateway.filter.secure-headers.download-options	**noopen**	

spring.cloud.gateway.filter.secure-headers.enabled	**true**	secure-headers filtresini etkinleştir
spring.cloud.gateway.filter.secure-headers.frame-options	**DENY**	
spring.cloud.gateway.filter.secure-headers.permitted-cross-domain-policies	**none**	
spring.cloud.gateway.filter.secure-headers.referrer-policy	**no-referrer**	
spring.cloud.gateway.filter.secure-headers.strict-transport-security	**max-age=631138519**	
spring.cloud.gateway.filter.secure-headers.xss-protection-header	**1 ; mode=block**	
spring.cloud.gateway.filter.set-path.enabled	**true**	set-path filtresini etkinleştir
spring.cloud.gateway.filter.set-request-header.enabled	**true**	set-request-header filtresini etkinleştir
spring.cloud.gateway.filter.set-request-host-header.enabled	**true**	set-request-host-header filtresini etkinleştir
spring.cloud.gateway.filter.set-response-header.enabled	**true**	set-response-header filtresini etkinleştir
spring.cloud.gateway.filter.set-status.enabled	**true**	set-status filtresini etkinleştir
spring.cloud.gateway.filter.strip-prefix.enabled	**true**	strip-prefix filtresini etkinleştir
spring.cloud.gateway.forwarded.enabled	**true**	ForwardedHeadersFilter'ı etkinleştir
spring.cloud.gateway.global-filter.adapt-cached-body.enabled	**true**	adapt-cached-body global filtresini etkinleştir
spring.cloud.gateway.global-filter.forward-path.enabled	**true**	forward-path global filtresini etkinleştir

spring.cloud.gateway.global-filter.forward-routing.enabled	**true**	forward-routing global filtresini etkinleştir
spring.cloud.gateway.global-filter.load-balancer-client.enabled	**true**	load-balancer-client global filtresini etkinleştir
spring.cloud.gateway.global-filter.netty-routing.enabled	**true**	netty-routing global filtresini etkinleştir
spring.cloud.gateway.global-filter.netty-write-response.enabled	**true**	netty-write-response global filtresini etkinleştir
spring.cloud.gateway.global-filter.reactive-load-balancer-client.enabled	**true**	reactive-load-balancer-client global filtresini etkinleştir
spring.cloud.gateway.global-filter.remove-cached-body.enabled	**true**	remove-cached-body global filtresini etkinleştir
spring.cloud.gateway.global-filter.route-to-request-url.enabled	**true**	route-to-request-url global filtresini etkinleştir
spring.cloud.gateway.global-filter.websocket-routing.enabled	**true**	websocket-routing global filtresini etkinleştir
spring.cloud.gateway.globalcors.add-to-simple-url-handler-mapping	**false**	Global CORS yapılandırması ise URL işleyicisine eklenmelidir.
spring.cloud.gateway.globalcors.cors-configurations		

spring.cloud.gateway.handler-mapping.order	**1**	RoutePredicateHandlerMapping'in sırası.
spring.cloud.gateway.httpclient.compression	**false**	Netty HttpClient için sıkıştırmayı etkinleştirir.
spring.cloud.gateway.httpclient.connect-timeout		Mil cinsinden bağlantı zaman aşımı, varsayılan 45s'dir.
spring.cloud.gateway.httpclient.max-header-size		Maksimum yanıt başlığı boyutu.
spring.cloud.gateway.httpclient.max-initial-line-length		Maksimum ilk satır uzunluğu.
spring.cloud.gateway.httpclient.pool.acquire-timeout		Yalnızca FIXED tipi için, edinme için beklenecek maksimum süre milis cinsindendir.
spring.cloud.gateway.httpclient.pool.eviction-interval	**0**	Belirli aralıklarla arka planda düzenli tahliye kontrolleri yapın. Varsayılan olarak devre dışıdır ({@link Duration#ZERO})
spring.cloud.gateway.httpclient.pool.max-connections		Yalnızca FIXED türü için, mevcut bağlantılarda bekleyen edinmeye başlamadan önce maksimum bağlantı sayısı.
spring.cloud.gateway.httpclient.pool.max-idle-time		Kanalın kapatılacağı süre milis cinsinden. NULL ise, maksimum boşta kalma süresi yoktur.
spring.cloud.gateway.httpclient.pool.max-life-time		Kanalın kapatılacağı süre. NULL ise, maksimum yaşam süresi yoktur.
spring.cloud.gateway.httpclient.pool.metrics	**false**	Mikrometrede toplanacak ve kaydedilecek kanal havuzları metriklerini etkinleştirir. Varsayılan olarak devre dışıdır.
spring.cloud.gateway.httpclient.pool.name	**proxy**	Kanal havuzu eşleme adı, varsayılan olarak proxy'dir.

spring.cloud.gateway.httpclient.pool.type		HttpClient'in kullanacağı havuz türü, varsayılan olarak ELASTIC'tir.
spring.cloud.gateway.httpclient.proxy.host		Netty HttpClient'in proxy yapılandırması için ana bilgisayar adı.
spring.cloud.gateway.httpclient.proxy.non-proxy-hosts-pattern		Konfigüre edilmiş bir ana bilgisayar listesi için normal ifade (Java). proxy'yi atlayarak doğrudan ulaşılması gereken
spring.cloud.gateway.httpclient.proxy.password		Netty HttpClient'in proxy yapılandırması için parola.
spring.cloud.gateway.httpclient.proxy.port		Netty HttpClient'in proxy yapılandırması için bağlantı noktası.
spring.cloud.gateway.httpclient.proxy.type		Netty HttpClient'in proxy yapılandırması için proxyType.
spring.cloud.gateway.httpclient.proxy.username		Netty HttpClient'in proxy yapılandırması için kullanıcı adı.
spring.cloud.gateway.httpclient.response-timeout		Yanıt zaman aşımı.
spring.cloud.gateway.httpclient.ssl.close-notify-flush-timeout	**3000ms**	SSL close_notify yıkama zaman aşımı. Varsayılan olarak 3000 ms'dir.
spring.cloud.gateway.httpclient.ssl.close-notify-read-timeout	**0**	SSL close_notify okuma zaman aşımı. Varsayılan olarak 0 ms'dir.
spring.cloud.gateway.httpclient.ssl.default-configuration-type		Varsayılan ssl yapılandırma türü. Varsayılanlar TCP'dir.
spring.cloud.gateway.httpclient.ssl.handshake-timeout	**10000ms**	SSL anlaşması zaman aşımı. 10000 ms'ye varsayılan
spring.cloud.gateway.httpclient.ssl.key-password		Anahtar parolası, varsayılan olarak keyStorePassword ile aynıdır.
spring.cloud.gateway.httpclient.ssl.key-store		Netty HttpClient için anahtar deposu yolu.
spring.cloud.gateway.httpclient.ssl.key-store-password		Keystore(Anahtar deposu) şifresi.
spring.cloud.gateway.httpclient.ssl.key-store-provider		Netty HttpClient için keystore sağlayıcısı, isteğe bağlı alan.
spring.cloud.gateway.httpclient.ssl.key-store-type	**JKS**	Netty HttpClient için keystore türü, varsayılan JKS'dir.
spring.cloud.gateway.httpclient.ssl.trusted-x509-certificates		Uzak uç noktanın(endpoints) sertifikasını doğrulamak için güvenilen sertifikalar.

spring.cloud.gateway.httpclient.ssl.use-insecure-trust-manager	**false**	Netty InsecureTrustManagerFactory'yi yükler. Bu güvensizdir ve üretim için uygun değildir.
spring.cloud.gateway.httpclient.websocket.max-frame-payload-length		Maksimum çerçeve yükü uzunluğu.
spring.cloud.gateway.httpclient.websocket.proxy-ping	**true**	Aşağı akış hizmetlerine proxy ping çerçeveleri, varsayılan olarak true olur.
spring.cloud.gateway.httpclient.wiretap	**false**	Netty HttpClient için dinleme hata ayıklamasını etkinleştirir.
spring.cloud.gateway.httpserver.wiretap	**false**	Netty HttpServer için dinleme hata ayıklamasını etkinleştirir.
spring.cloud.gateway.loadbalancer.use404	**false**	
spring.cloud.gateway.metrics.enabled	**false**	Metrik verilerinin toplanmasını sağlar.
spring.cloud.gateway.metrics.prefix	**spring.cloud.gateway**	Ağ geçidi(gateway) tarafından yayılan tüm metriklerin öneki.
spring.cloud.gateway.metrics.tags		Metriklere eklenen etiketler haritası.
spring.cloud.gateway.predicate.after.enabled	**true**	After yüklemini etkinleştirir.
spring.cloud.gateway.predicate.before.enabled	**true**	Before yüklemini etkinleştirir.
spring.cloud.gateway.predicate.between.enabled	**true**	Between yüklemini etkinleştirir.
spring.cloud.gateway.predicate.cloud-foundry-route-service.enabled	**true**	Bulut dökümhanesi-rota hizmeti yüklemini etkinleştirir.
spring.cloud.gateway.predicate.cookie.enabled	**true**	Çerez yüklemini etkinleştirir.
spring.cloud.gateway.predicate.header.enabled	**true**	Başlık yüklemini etkinleştirir.
spring.cloud.gateway.predicate.host.enabled	**true**	Ana bilgisayar yüklemini etkinleştirir.
spring.cloud.gateway.predicate.method.enabled	**true**	Yöntem yüklemini etkinleştirir.
spring.cloud.gateway.predicate.path.enabled	**true**	Yol yüklemini etkinleştirir.
spring.cloud.gateway.predicate.query.enabled	**true**	Sorgu yüklemini etkinleştirir.
spring.cloud.gateway.predicate.read-body.enabled	**true**	Okuma gövdesi yüklemini etkinleştirir.
spring.cloud.gateway.predicate.remote-addr.enabled	**true**	Remote-addr yüklemini etkinleştirir.

spring.cloud.gateway.predicate.weight.enabled	**true**	Ağırlık yüklemini etkinleştirir.
spring.cloud.gateway.predicate.xforwarded-remote-addr.enabled	**true**	x forwarded-remote_addr yüklemini etkinleştirir.
spring.cloud.gateway.redis-rate-limiter.burst-capacity-header	**X-RateLimit-Burst-Capacity**	Ayırma kapasitesi yapılandırmasını döndüren başlığın adı.
spring.cloud.gateway.redis-rate-limiter.config		
spring.cloud.gateway.redis-rate-limiter.include-headers	**true**	Hız sınırlayıcı bilgilerini içeren başlıkların dahil edilip edilmeyeceği, varsayılan olarak true olur.
spring.cloud.gateway.redis-rate-limiter.remaining-header	**X-RateLimit-Remaining**	Geçerli saniye boyunca kalan isteklerin sayısını döndüren başlığın adı.
spring.cloud.gateway.redis-rate-limiter.replenish-rate-header	**X-RateLimit-Replenish-Rate**	Yenileme hızı yapılandırmasını döndüren başlığın adı.
spring.cloud.gateway.redis-rate-limiter.requested-tokens-header	**X-RateLimit-Requested-Tokens**	İstenen belirteç yapılandırmasını döndüren başlığın adı.
spring.cloud.gateway.restrictive-property-accessor.enabled	**true**	SpEL'de yöntem ve özellik erişimini kısıtlar.
spring.cloud.gateway.routes		Güzergah Listesi.
spring.cloud.gateway.set-status.original-status-header-name		Routes List.Proxy isteğinin http kodunu içeren başlığın adı.
spring.cloud.gateway.streaming-media-types		
spring.cloud.gateway.x-forwarded.enabled	**true**	XForwardedHeadersFilter etkinse.
spring.cloud.gateway.x-forwarded.for-append	**true**	X-Forwarded-For öğesinin liste olarak eklenmesi etkinse.
spring.cloud.gateway.x-forwarded.for-enabled	**true**	X-Forwarded-For etkinse.
spring.cloud.gateway.x-forwarded.host-append	**true**	X-Forwarded-Host'u liste olarak ekleme etkinse.
spring.cloud.gateway.x-forwarded.host-enabled	**true**	X-Forwarded-Host etkinse.
spring.cloud.gateway.x-forwarded.order	**0**	XForwardedHeadersFilter'ın sırası.
spring.cloud.gateway.x-forwarded.port-append	**true**	X-Forwarded-Port'u liste olarak eklemek etkinse.
spring.cloud.gateway.x-forwarded.port-enabled	**true**	X-Forwarded-Port etkinse.
spring.cloud.gateway.x-forwarded.prefix-append	**true**	X-Forwarded-Prefix'in liste olarak eklenmesi etkinse.
spring.cloud.gateway.x-forwarded.prefix-enabled	**true**	X-Forwarded-Prefix etkinse.
spring.cloud.gateway.x-forwarded.proto-append	**true**	X-Forwarded-Proto'nun liste olarak eklenmesi etkinse.
spring.cloud.gateway.x-forwarded.proto-enabled	**true**	X-Forwarded-Proto etkinse.