

Blog Bitix

[Java](#)[GNU/Linux](#)[JavaScript](#)[Tapestry](#)[Archivo y hemeroteca](#)[Enlaces](#)[Acerca de...](#)

Proxy para microservicios con Spring Cloud Netflix y Zuul

Escrito por [picodotdev](#) el 13/10/2018.

[java](#) [planeta-codigo](#) [programacion](#) [spring](#)

[Enlace permanente](#) [Comentarios](#)

Teniendo una buen número de microservicios con múltiples instancias ofreciendo cada uno una API y en una ubicación diferente para simplificar la visión de los que actúen clientes de los microservicios se puede utilizar un *proxy*. Con un *proxy* es posible centralizar todas las peticiones, que sea éste el encargado de conocer la ubicación de todas las instancias de los microservicios y de hacer la llamada allí donde se encuentre cada una de ellas.

Entre las varias funcionalidades que proporcionar el proyecto [Spring Cloud Netflix](#) es esta de *proxy* mediante [Zuul](#). Para hacer de *proxy* Zuul necesita tener una correspondencia entre URLs y servicios que realmente proporcionan la funcionalidad, una forma que tiene Zuul de conocer la ubicación de las instancias es utilizando el servicio de registro y descubrimiento [Eureka](#). Además, Zuul como cliente de los microservicios posee la funcionalidad de [Hystrix](#) que implementa el patrón *circuit breaker* para tolerancia a fallos, [Ribbon](#) para hacer balanceo de carga entre varias instancias de los microservicios a nivel de servidor además de reintentos cuando una instancia falla.

En el ejemplo que he utilizado para esta [serie de artículos sobre Spring Cloud](#) hay un servicio que por defecto se inicia en el puerto 8080 y ofrece un *endpoint* / que devuelve un mensaje. Para crear un microservicio *proxy* con Zuul hay que crear una aplicación [Spring Boot](#) anotar la clase principal con la anotación `@EnableZuulProxy` y proporcionar la configuración para la correspondencia de rutas y microservicios, además de las propiedades para hacer reintentos en caso de que un microservicio falle y de *timeouts* en caso de que se comporte no como se espera en cuanto tiempos de respuesta.



18/12/2019Proxy para microservicios con Spring Cloud Netflix y Zuul

```
1 package io.github.picodotdev.blogbitix.springcloud.proxy;
2
3 ...
4
5 @SpringBootApplication
6 @EnableDiscoveryClient
7 @EnableZuulProxy
8 public class Main {
9
10     public static void main(String[] args) throws Exception {
11         SpringApplication application = new SpringApplication(Main.class);
12         application.setApplicationContextClass(AnnotationConfigApplicationContext.class);
13         SpringApplication.run(Main.class, args);
14     }
15 }
```

Main (zuul).javaview raw

```
1 plugins {
2     id 'application'
3     id 'org.springframework.boot' version '2.0.7.RELEASE'
4     id 'io.spring.dependency-management' version '1.0.6.RELEASE'
5 }
6
7 mainClassName = 'io.github.picodotdev.blogbitix.springcloud.proxy.Main'
8
9 dependencyManagement {
10     imports {
11         mavenBom 'org.springframework.cloud:spring-cloud-dependencies:Finchley.SR2'
12     }
13 }
14
15 dependencies {
16     // Spring
17     def excludeSpringBootStarterLogging = { exclude(group: 'org.springframework.boot', module: 'spring-boot-starter-logging') }
18     compile('org.springframework.boot:spring-boot-starter', excludeSpringBootStarterLogging)
19     compile('org.springframework.boot:spring-boot-starter-web', excludeSpringBootStarterLogging)
20     compile('org.springframework.boot:spring-boot-starter-log4j2', excludeSpringBootStarterLogging)
21     compile('org.springframework.boot:spring-boot-starter-actuator', excludeSpringBootStarterLogging)
22     compile('org.springframework.cloud:spring-cloud-starter-config', excludeSpringBootStarterLogging)
23     compile('org.springframework.cloud:spring-cloud-starter-bus-amqp', excludeSpringBootStarterLogging)
24     compile('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client', excludeSpringBootStarterLogging)
25     compile('org.springframework.cloud:spring-cloud-starter-netflix-zuul', excludeSpringBootStarterLogging)
26     compile('org.springframework.retry:spring-retry:1.2.2.RELEASE', excludeSpringBootStarterLogging)
27
28     runtime('com.google.code.gson:gson:2.8.5')
29     runtime('com.fasterxml.jackson.core:jackson-databind:2.9.6')
30     runtime('com.fasterxml.jackson.dataformat:jackson-dataformat-yaml:2.9.6')
31 }
```

build.gradleview raw

Se puede establecer un tiempo máximo para establecer la conexión, de tiempo de petición, el número de reintentos en la misma instancia si falla o en otro número de instancias, el número máximo de conexiones y el número máximo de conexiones al mismo *host*. Todas ellas definibles en cada servicio de forma individual bajo las propiedades *hystrix.command.service* y *service.ribbon* donde *service* es el identificativo del servicio. Las rutas se indican bajo la propiedad *zuul.routes* con la relación identificativo del servicio y *path*.

```
1 server:
2   port: ${port:8095}
3
4 zuul:
5   routes:
6     service: '/service/**'
7   retryable: true
8
9 hystrix:
10  command:
11    service:
12      execution:
13        isolation:
14          thread:
15            timeoutInMilliseconds: 20000
16  circuitBreaker:
17    requestVolumeThreshold: 4
18    errorThresholdPercentage: 50
19
20 service:
21   ribbon:
22     ConnectTimeout: 1000
23     ReadTimeout: 2000
24     MaxTotalHttpConnections: 100
25     MaxConnectionsPerHost: 100
26     MaxAutoRetries: 0
27     MaxAutoRetriesNextServer: 2
28     ServerListRefreshInterval: 2000
29     OkToRetryOnAllOperations: true
30     retryableStatusCodes: 500,404
```

proxy.yml

view raw

Dado que Zuul es un *proxy* para múltiples instancias de microservicios a cada microservicio hay que darle una ruta, cuando Zuul realiza la llamada a una instancia del microservicio se encarga de omitirla. En el ejemplo, la ruta en Zuul */service/*** está asociada al microservicio *service* pero el servicio *service* ofrece su *endpoint* en */*, Zuul se encarga de omitir la parte de la ruta para el *proxy* y hace la llamada a la ruta */* como espera el microservicio.

Lógicamente los clientes deben contactar con el *proxy* en vez de con el microservicio directamente. Arrancado el servicio de descubrimiento y registro Eureka, el servidor de configuración de Spring Cloud, dos instancias del servicio y el *proxy* con Zuul haciendo las llamadas al *proxy* se observa que se obtiene el resultado del microservicio. Como en el ejemplo hay varias instancias del servicio Zuul realiza balanceo de carga entre ellas con Ribbon utilizando la política *round-robin* y el mensaje es diferente en cada una de las respuestas según la instancia invocada. Con Zuul además se consigue balanceo de carga a nivel de servidor que Ribbon solo ofrece a nivel de cliente.



```
1 package io.github.picodotdev.blogbitix.springcloud.client;
2
3 ...
4
5 @SpringBootApplication
6 @EnableDiscoveryClient
7 @EnableCircuitBreaker
8 @EnableHystrixDashboard
9 public class Main implements CommandLineRunner {
10
11     @Autowired
12     private DefaultConfiguration configuration;
13
14     @Autowired
15     private ClientService service;
16
17     @Autowired
18     private ProxyService proxy;
19
20     @Override
21     public void run(String... args) throws Exception {
22         System.out.printf("Valor de propiedad de configuración (%s): %s\n", "config.key", configuration.getKey());
23         System.out.printf("Valor de propiedad de configuración (%s): %s\n", "config.password", configuration.getPassword());
24         System.out.printf("Valor de propiedad de configuración (%s): %s\n", "config.service", configuration.getService());
25
26         for (int i = 0; i < 20000; ++i) {
27             String response = (configuration.getService().equals("service")) ? service.get() : proxy.get();
28             System.out.printf("Service response: %s\n", response);
29             Thread.sleep(100);
30         }
31     }
32
33     public static void main(String[] args) throws Exception {
34         SpringApplication application = new SpringApplication(Main.class);
35         application.setApplicationContextClass(AnnotationConfigApplicationContext.class);
36         SpringApplication.run(Main.class, args);
37     }
38 }
```

Main (client).java

[view raw](#)

```
1 package io.github.picodotdev.blogbitix.springcloud.client;
2
3 ...
4
5 @Component
6 public class ProxyService {
7
8     @Autowired
9     private LoadBalancerClient loadBalancer;
10
11     @HystrixCommand(fallbackMethod = "getFallback", commandProperties = {
12         @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "4"),
13         @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "50"),
14         @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "25000")
15     })
16     public String get() {
17         ServiceInstance instance = loadBalancer.choose("proxy");
18         URI uri = instance.getUri();
19         String resource = String.format("%s%s", uri.toString(), "/service");
20         return Client.create().resource(resource).get(String.class);
21     }
22
23     private String getFallback() {
24         return "Fallback";
25     }
26 }
```

ProxyService.java

[view raw](#)

```
1 $ ./gradlew discoveryserver:run --args="--port=8761"
2 $ ./gradlew configserver:run --args="--port=8090"
3 $ ./gradlew service:run --args="--port=8080"
4 $ ./gradlew service:run --args="--port=8081"
5 $ ./gradlew service:run --args="--port=8082"
6 $ ./gradlew proxy:run --args="--port=8085"
7 $ ./gradlew client:run --args="--service=proxy"
```

gradle-run.sh

view raw

Las URLs del servicio en el microservicio y en el *proxy* son.

```
1 # Microservicio
2 $ curl http://192.168.1.4:8080/
3
4 # Microservicio en el proxy
5 $ curl http://192.168.1.4:8085/service
```

curl.sh

view raw

El cliente de ejemplo realiza peticiones al *proxy*, en la salida se muestra el resultado del balanceo de carga cuando hay varias instancias, cuando se añade una nueva instancia entra a formar parte del balanceo de carga. Otro beneficio de Zuul es que ofrece la funcionalidad de reintentos de modo que si una instancia de un servicio falla la petición se reintenta en otra. En el artículo [Balanceo de carga y resiliencia en un microservicio con Spring Cloud Netflix y Ribbon](#) usando solo Ribbon se observaba que cuando una instancia falla se le siguen haciendo peticiones hasta que la lista de instancias del servicio en Eureka se actualiza quitando la fallida, con Hystrix se obtiene la respuesta *fallback* pero no se evita completamente el error. Zuul puede ocultar el error provocado por una instancia que falla reintentado la petición en la misma nuevamente, en otra u otras instancias según se configure. El comportamiento con Zuul cuando una instancia falla se puede comparar con el comportamiento incluido en el [artículo anterior usando en el cliente los microservicios directamente](#).

```
1 $ ./gradlew client:run --args="--service=proxy"
2 ... # initially two service instances (8080, 8081)
3 Service response: Hello world (http://192.168.1.4:8080/, value)
4 Service response: Hello world (http://192.168.1.4:8081/, value)
5 Service response: Hello world (http://192.168.1.4:8080/, value)
6 Service response: Hello world (http://192.168.1.4:8081/, value)
7 Service response: Hello world (http://192.168.1.4:8080/, value)
8 Service response: Hello world (http://192.168.1.4:8081/, value)
9 ... # new service instance, ./gradlew service:run --args="--port=8082"
10 Service response: Hello world (http://192.168.1.4:8080/, value)
11 Service response: Hello world (http://192.168.1.4:8081/, value)
12 Service response: Hello world (http://192.168.1.4:8082/, value)
13 Service response: Hello world (http://192.168.1.4:8080/, value)
14 Service response: Hello world (http://192.168.1.4:8081/, value)
15 Service response: Hello world (http://192.168.1.4:8082/, value)
16 ... # kill service instance (8082), Ctrl+C
17 Service response: Hello world (http://192.168.1.4:8080/, value)
18 Service response: Hello world (http://192.168.1.4:8081/, value)
19 Service response: Hello world (http://192.168.1.4:8080/, value)
20 Service response: Hello world (http://192.168.1.4:8081/, value)
21 Service response: Hello world (http://192.168.1.4:8080/, value)
22 Service response: Hello world (http://192.168.1.4:8081/, value)
```

System.out

view raw

Zuul además es capaz de proporciona otras muchas funcionalidades como:

- Autenticación
- Seguridad
- Recolección de métricas y monitorización
- Pruebas de carga
- Pruebas de verificación o *canary testing*
- Enrutado dinámico
- Migración de servicio
- Abandono de carga o *load shedding*
- Manejo de respuesta estática

- Gestión de tráfico active/active

El [código fuente completo del ejemplo](#) puedes descargarlo del repositorio de ejemplos de Blog Bitix alojado en [GitHub](#) y probarlo en tu equipo ejecutando el comando `./gradlew-run.sh`.

Este artículo forma parte de la serie **spring-cloud**:

1. [Datos de sesión externalizados con Spring Session](#)
2. [Aplicación Java autocontenida con Spring Boot](#)
3. [Configuración de una aplicación en diferentes entornos con Spring Cloud Config](#)
4. [Información y métricas de la aplicación con Spring Boot Actuator](#)
5. [Registro y descubrimiento de servicios con Spring Cloud y Consul](#)
6. [Aplicaciones basadas en microservicios](#)
7. [Registro y descubrimiento de servicios con Spring Cloud Netflix](#)
8. [Servicio de configuración para microservicios con Spring Cloud Config](#)
9. [Recargar sin reiniciar la configuración de una aplicación Spring Boot con Spring Cloud Config](#)
10. [Almacenar cifrados los valores de configuración sensibles en Spring Cloud Config](#)
11. [Tolerancia a fallos en un cliente de microservicio con Spring Cloud Netflix y Hystrix](#)
12. [Balanceo de carga y resiliencia en un microservicio con Spring Cloud Netflix y Ribbon](#)
13. [Proxy para microservicios con Spring Cloud Netflix y Zuul](#)
14. [Monitorizar una aplicación Java de Spring Boot con Micrometer, Prometheus y Grafana](#)
15. [Exponer las métricas de Hystrix en Grafana con Prometheus de una aplicación Spring Boot](#)
16. [Servidor OAuth, gateway y servicio REST utilizando tokens JWT con Spring](#)
17. [Trazabilidad en microservicios con Spring Cloud Sleuth](#)
18. [Implementar tolerancia a fallos con Resilience4j](#)
19. [Iniciar una aplicación de Spring Boot en un puerto aleatorio](#)
20. [Utilizar credenciales de conexión a la base de datos generadas por Vault en una aplicación de Spring](#)
21. [Microservicios con Spring Cloud, Consul, Nomad y Traefik](#)

181
Shares

Tweet

Share

Share

Share



El primer Máster Online que estudias como si estuvieras viendo una serie en Netflix

FORMACIÓN Y ECONOMÍA - ThePowerMBA - Septiemb

0 Comentarios

Blog Bitix

Javier Martín Alon...

Recomendar

Tweet

Compartir

Ordenar por los más antiguos




Sé el primero en comentar...

Sé el primero en comentar.

TAMBIÉN EN BLOG BITIX


Novedades de Java 10

3 comentarios • hace 2 años

 **picodotdev** — La estrategia del lenguaje Java es adoptar aquellas características que se han demostrado son útiles. Esto hace que Java parezca que avanza más lento pero creo que es una buena estrategia,


Como eliminar metainformación de las fotos en GNU/Linux

2 comentarios • hace 2 años

 **picodotdev** — A lo primero, exacto, esto es mucho más rápido. En las fotos que pongo en los artículos es lo que utilizo.A lo segundo, si supiese de algún sistema de anuncios que respetase la privacidad y estuviese


Desempaquetado Intel NUC8i5BEK (Bean Canyon), HyperX Impact (RAM) y Samsung 970 EVO NVMe (SSD)

2 comentarios • hace un año

 **picodotdev** — Con los meses que estuve decidiéndome tuve tiempo para pensarlo. Después de un tiempo con todo y usándolo a diario estoy muy contento con lo que compré. Salvo el ratón inalámbrico que al final por

Tú con tu Mac, yo con mi GNU/Linux

13 comentarios • hace 2 años

 **picodotdev** — Era este [https://elblogdepicodev.blo...](https://elblogdepicodev.blog/) y lo compre directamente online en la página de Sony pero he visto modelos en tiendas dedicadas a electrónica. El negocio de los portátiles parece que

Suscríbete

Añade Disqus a tu sitio webAñade Disqus Añadir

Política de privacidad de DisqusPolítica de privacidadPrivacidad

Blog Bitix

Blog dedicado a la distribución GNU/Linux que uso habitualmente, Arch Linux, a mis andanzas alrededor del software libre, la programación y a otros temas relacionados con la tecnología y la informática.

Publicando de uno a tres artículos únicos a la semana desde el año 2010.

Copyleft © 2019 - 

Blog Bitix by [pico.dev](#) is licensed under a [Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional License](#).

Powered by [Hugo](#) and [GitHub Pages](#). Background patterns from [Subtle Patterns](#).

- [Java](#)
- [GNU/Linux](#)
- [JavaScript](#)
- [Tapestry](#)
- [Archivo y hemeroteca](#)
- [Enlaces](#)
- [Publicidad](#)
- [Donaciones](#)
- [Acerca de...](#)

