**Пример Zuul и Фильтры**

Это руководство проведет вас через процесс маршрутизации и фильтрации запросов к приложению микросервисов с помощью библиотеки пограничных сервисов Netflix Zuul.

Что ты будешь строить

Вы напишете простое приложение для микросервисов, а затем создадите приложение обратного прокси, которое использует [Netflix Zuul](https://github.com/spring-cloud/spring-cloud-netflix) для пересылки запросов в приложение-службу. Вы также увидите, как использовать Zuul для фильтрации запросов, отправляемых через прокси-службу.

Что вам нужно

* Любимый текстовый редактор или IDE
* [JDK 1.8](http://www.oracle.com/technetwork/java/javase/downloads/index.html) или новее
* [Gradle 4+](http://www.gradle.org/downloads) или [Maven 3.2+](https://maven.apache.org/download.cgi)
* Вы также можете импортировать код прямо в свою среду IDE:
  + [Набор инструментов Spring (STS)](https://spring.io/guides/gs/sts)
  + [IntelliJ IDEA](https://spring.io/guides/gs/intellij-idea/)

Как заполнить это руководство

Как и в большинстве [руководств по началу работы с](https://spring.io/guides) Spring , вы можете начать с нуля и выполнить каждый шаг или обойти основные шаги настройки, которые вам уже знакомы. В любом случае вы получите рабочий код.

Чтобы **начать с нуля** , перейдите к [разделу «Запуск с Spring Initializr»](https://spring.io/guides/gs/routing-and-filtering/#scratch) .

Чтобы **пропустить основы** , сделайте следующее:

* [Загрузите](https://github.com/spring-guides/gs-routing-and-filtering/archive/master.zip) и распакуйте исходный репозиторий для этого руководства или клонируйте его с помощью [Git](https://spring.io/understanding/Git) :git clone <https://github.com/spring-guides/gs-routing-and-filtering.git>
* cd в gs-routing-and-filtering/initial
* Забегая вперед, чтобы [настроить микросервис](https://spring.io/guides/gs/routing-and-filtering/#initial) .

**Когда вы закончите** , вы можете сравнить свои результаты с кодом в gs-routing-and-filtering/complete.

Начиная с Spring Initializr

Для всех приложений Spring вы должны начать с [Spring Initializr](https://start.spring.io/) . Initializr предлагает быстрый способ получить все зависимости, необходимые для приложения, и выполняет большую часть настройки за вас.

Это руководство требует двух приложений.

**1. Первое приложение - микросервис RESTfuul** (приложение-**book**).

В следующем листинге показан pom.xmlфайл (для приложения-**book**), который создается при выборе Maven:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.7.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>routing-and-filtering-book</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>routing-and-filtering-book</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

В следующем листинге показан build.gradleфайл (для книжного приложения), который создается при выборе Gradle:

plugins {

id 'org.springframework.boot' version '2.3.7.RELEASE'

id 'io.spring.dependency-management' version '1.0.10.RELEASE'

id 'java'

}

group = 'com.example'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '11'

repositories {

mavenCentral()

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-web'

testImplementation('org.springframework.boot:spring-boot-starter-test') {

exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'

}

}

test {

useJUnitPlatform()

}

**Настроить микросервис**

МикроСервис **Book** будет настолько простым, насколько может быть приложение Spring. Создайте файл RoutingAndFilteringBookApplicationBookApplication.javaи расположите его по следующему адресу  book/src/main/java/com/example/routingandfilteringbook/RoutingAndFilteringBookApplication.java:

package com.example.routingandfilteringbook;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@SpringBootApplication

public class RoutingAndFilteringBookApplication {

@RequestMapping(value = "/available") //метод Get

public String available() {

return "Spring in Action";

}

@RequestMapping(value = "/checked-out") //метод Get

public String checkedOut() {

return "Spring Boot in Action";

}

public static void main(String[] args) {

SpringApplication.run(RoutingAndFilteringBookApplication.class, args);

}

}

Теперь RoutingAndFilteringBookApplicationBookApplicationкласс является контроллером REST. В @RestControllerаннотации исполняется класс как класс контроллера и гарантирует , что возвращаемые значения из @RequestMappingметодов в этом классе автоматически и надлежащим образом преобразованы и записываются непосредственно в ответ HTTP.

Говоря о @RequestMappingметодах, мы добавили два: available()и checkedOut(). Они обрабатывают запросы к /availableи /checked-outпутям, каждый из которых возвращает Stringназвание книги.

Создайте файл свойств для приложения ( book) src/main/resources/application.properties, как показано в следующем листинге:

spring.application.name=book

server.port=8090

Также установите server.portздесь, чтобы он не конфликтовал с вашей пограничной службой, когда вы запускаете обе службы локально.

**2. Второе приложение – микросервис надстройка над RESTfuul** (приложение маршрутизации и фильтрации) .

В следующем листинге показан pom.xmlфайл (для приложения маршрутизации и фильтрации), который создается при выборе Maven:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.7.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>routing-and-filtering-gateway</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>routing-and-filtering-gateway</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>Hoxton.SR9</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-zuul</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

</repository>

</repositories>

</project>

В следующем листинге показан build.gradleфайл (для приложения маршрутизации и фильтрации), который создается, когда вы выбираете Gradle:

plugins {

id 'org.springframework.boot' version '2.3.7.RELEASE'

id 'io.spring.dependency-management' version '1.0.10.RELEASE'

id 'java'

}

group = 'com.example'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '11'

repositories {

mavenCentral()

}

ext {

set('springCloudVersion', "Hoxton.SR9")

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.springframework.cloud:spring-cloud-starter-netflix-zuul'

testImplementation('org.springframework.boot:spring-boot-starter-test') {

exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'

}

}

dependencyManagement {

imports {

mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"

}

}

test {

useJUnitPlatform()

}

|  |  |
| --- | --- |
|  |  |

**Создаем пограничную службу**

Spring Cloud Netflix включает встроенный прокси-сервер Zuul, который можно включить с помощью @EnableZuulProxyаннотации. Это превратит приложение шлюза в обратный прокси-сервер, который пересылает соответствующие вызовы другим службам, например, нашему книжному приложению.

Откройте RoutingAndFilteringGatewayApplicationGatewayApplicationкласс приложения Gateway и добавьте @EnableZuulProxyаннотацию, как gateway/src/main/java/com/example/routingandfilteringgateway/RoutingAndFilteringGatewayApplication.javaпоказано в следующем листинге:

package com.example.routingandfilteringgateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

import org.springframework.context.annotation.Bean;

import com.example.routingandfilteringgateway.filters.pre.SimpleFilter;

@EnableZuulProxy

@SpringBootApplication

public class RoutingAndFilteringGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(RoutingAndFilteringGatewayApplication.class, args);

}

@Bean

public SimpleFilter simpleFilter() {

return new SimpleFilter();

}

}

Чтобы пересылать запросы от приложения *Gateway*, вам необходимо указать Zuul:

- маршруты, которые он должен отслеживать,

- службы, которым следует пересылать запросы, сделанные по этим маршрутам.

Мы указываем маршруты, устанавливая свойства в zuul.routes. Каждый из наших микросервисов может иметь запись zuul.routes.NAME, где NAME- имя приложения (сохраненное в spring.application.name) в файле ***application.properties*** смого микросервиса.

Добавьте application.propertiesфайл в новый каталог ( gateway/src/main/resources) в приложении пограничной службы. Он должен соответствовать следующему адресу (из gateway/src/main/resources/application.properties):

zuul.routes.**books**.url=http://localhost:8090

ribbon.eureka.enabled=false //отключаем использование Eureka

server.port=8080

Spring Cloud Zuul автоматически устанавливает путь к имени приложения. В этом примере настройте zuul.routes.books.urlтак, чтобы Zuul отправлял запросы /booksна этот URL-адрес.

Обратите внимание на второе свойство application.propertiesфайла: Spring Cloud Netflix Zuul использует Ribbon для балансировки нагрузки на стороне клиента. По умолчанию Ribbon будет использовать Netflix Eureka для обнаружения сервисов. В этом простом примере мы пропускаем Eureka, поэтому установите ribbon.eureka.enabledзначение false. Поскольку Ribbon теперь не может использовать Eureka для поиска сервисов, мы должны указывать urlдля микросервиса **book**.

**Добавить фильтр**

Мы напишем preфильтр. Spring Cloud Netflix подбирает в качестве фильтра все @Beanчто расширяются com.netflix.zuul.ZuulFilterи доступны в контексте приложения. В следующем листинге (из gateway/src/main/java/com/example/routingandfilteringgateway/filters/pre/SimpleFilter.java) показан нужный нам фильтр:

package com.example.routingandfilteringgateway.filters.pre;

import javax.servlet.http.HttpServletRequest;

import com.netflix.zuul.context.RequestContext;

import com.netflix.zuul.ZuulFilter;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class SimpleFilter extends ZuulFilter {

private static Logger log = LoggerFactory.getLogger(SimpleFilter.class);

@Override

public String filterType() {

return "pre";

}

@Override

public int filterOrder() {

return 1;

}

@Override

public boolean shouldFilter() {

return true;

}

@Override

public Object run() {

RequestContext ctx = RequestContext.getCurrentContext();

HttpServletRequest request = ctx.getRequest();

log.info(String.format("%s request to %s", request.getMethod(), request.getRequestURL().toString()));

return null;

}

}

Zuul фильтрует запросы на сохранение и информацию о состоянии в (и делится ею с помощью) RequestContext. Вы можете использовать это, чтобы получить, HttpServletRequestа затем зарегистрировать HTTP-метод и URL-адрес запроса, прежде чем он будет отправлен по пути.

GatewayApplicationКласс с аннотацией @SpringBootApplication, которая включает в себя (среди прочих) в @Configurationаннотации , что говорит Spring посмотреть в данном классе для @Beanопределений. Поместите фильтр в класс приложения, как показано в следующем листинге (из gateway/src/main/java/com/example/routingandfilteringgateway/RoutingAndFilteringGatewayApplication.java):

package com.example.routingandfilteringgateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

import org.springframework.context.annotation.Bean;

import com.example.routingandfilteringgateway.filters.pre.SimpleFilter;

@EnableZuulProxy

@SpringBootApplication

public class RoutingAndFilteringGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(RoutingAndFilteringGatewayApplication.class, args);

}

@Bean

public SimpleFilter simpleFilter() {

return new SimpleFilter();

}

}

Тестирование вашего приложения

Убедитесь, что оба приложения запущены. В браузере посетите одну из конечных точек приложения-книги через приложение Gateway. Если вы использовали конфигурацию, показанную в этом руководстве, вы можете получить доступ к приложению **book** напрямую по адресу localhost:8090/available или через службу шлюза по адресу localhost:8080/books/available.

Посетите одну из конечных точек службы Book ( localhost:8080/books/availableили localhost:8080/books/checked-out), и вы должны увидеть метод вашего запроса, зарегистрированный приложением Gateway, прежде чем он будет передан в приложение Book, как показано в следующем примере выходных данных журнала:

2019-10-02 10:58:34.694 INFO 11608 --- [nio-8080-exec-4] c.e.r.filters.pre.SimpleFilter : GET request to http://localhost:8080/books/available

Резюме

Поздравляю! Вы использовали Spring для разработки приложения пограничного сервиса, которое может проксировать и фильтровать запросы для ваших микросервисов.