

20 Spring Cloud Gateway 快速实践

更新时间：2019-07-04 18:14:00



对自己不满是任何真正有才能的人的根本特征之一。

——契诃夫

本节内给大家介绍如何在项目中使用 **Spring Cloud Gateway**，学习如何使用 **Spring Cloud Gateway** 转发单个项目的请求，实践 **Spring Cloud Gateway** 和注册中心的配合使用。

快速入手

我们先来快速实现一个 **Spring Cloud Gateway** 的 **hello world**，让大家了解一下 **Gateway** 整体工作流程。

Spring Cloud Gateway 支持两种方式配置路由的使用：

- 编码方式，通过 **@Bean** 自定义 **RouteLocator**，在启动主类 **Application** 中配置。
- 配置方式，在配置文件 **yml** 中配置。

两种方式是等价的，我们先来使用第一种方式：

编码方式

首先添加 **Spring Cloud Gateway** 的依赖包 **spring-cloud-starter-gateway**

添加依赖

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

添加启动类

```
public class GatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GatewayApplication.class, args);
    }

    @Bean
    public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
        return builder.routes()
            .route("path_route", r -> r.path("/get")
                .uri("http://httpbin.org"))
            .build();
    }
}
```

上面这段配置的意思是，配置了一个 id 为 `path_route` 的路由规则，当访问地址 `http://localhost:8080/get` 时会自动转发到地址：`http://httpbin.org/get`。配置完成启动项目即可在浏览器访问进行测试，当我们访问地址 `http://localhost:8080/get` 时页面展示如下信息：

```
{
  "args": {},
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Cookie": "Hm_lvt_f0cfcccd7b1393990c78efdeebff3968=1555735735; Hm_lpv_f0cfcccd7b1393990c78efdeebff3968=1555740301",
    "Forwarded": "proto=http;host=\"localhost:8080\";for=\"0:0:0:0:0:0:1:64367\"",
    "Host": "httpbin.org",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36",
    "X-Forwarded-Host": "localhost:8080"
  },
  "origin": "0:0:0:0:0:0:1, 221.178.127.44, ::1",
  "url": "https://localhost:8080/get"
}
```

这样访问地址 `http://localhost:8080/get` 就和访问 `http://httpbin.org/get` 的效果是一致的，从而验证了路由转发的成功。

配置方式

上面给大家演示了使用编码方式实现路由转发。下面介绍使用配置方式实现路由转发。配置文件推荐使用 `YML` 格式来配置，`YML` 风格更简洁清晰。

application.yml

```
server:
  port: 8080
spring:
  cloud:
    gateway:
      routes:
        - id: path_route
          uri: http://httpbin.org
      predicates:
        - Path=/get
```

配置中的参数：

- `id`：我们自定义的路由 ID，保持唯一，代码中就是 `route()` 方法的第一个参数。

- **uri**: 需要转发的目标服务地址，`r -> r.path("/get").uri("http://httpbin.org")` 代码使用了函数时编程简化代码。
- **predicates**: 路由条件，**Predicate** 接受一个输入参数，返回一个布尔值结果。该接口包含多种默认方法来将 **Predicate** 组合成其他复杂的逻辑（比如：与，或，非）。
- **filters**: 过滤规则，本示例暂时没用。

配置完成之后我们把上启动类中的 `customRouteLocator()` 注释掉：

```
/* @Bean
public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
    return builder.routes()
        .route("path_route", r -> r.path("/get")
            .uri("http://httpbin.org"))
        .build();
}*/
```

再重新启动项目，再次访问地址 `http://localhost:8080/get`，返回信息和上述返回的结果一致，这说明了通过配置的方式也可以到达路由转发的功能。在实际项目中为了方便维护，推荐使用配置文件的方式来配置。

两个 **hello world** 版的示例都体验完之后，基本可以了解 **Spring Cloud GateWay** 的工作机制，在日常项目中我们可根据不同的需求，配置不同的路由转发策略，大部分使用场景只需要调整不同的配置信息即可实现。

网关和注册中心

上面两种转发方式只适合在单一的项目中使用，如果在微服务架构中就会存在一个致命的问题，微服务架构中服务提供者是动态变化的，所以不能直接将目标服务器地址写在配置文件中。那么在微服务架构中如何解决这个问题呢，这个时候就需要和注册中心来配置使用，本文以 **Eureka** 和 **Spring Cloud Gateway** 为例给大家讲解。

当网关和注册中心结合起来使用时，把网关当作一个客户端注册到注册中心，然后网关从注册中心获取所有服务，并自动为这些服务提供路由转发功能。**Spring Cloud Gateway** 提供了此功能，在项目中只需要简单配置即可达到这样的效果，接下来进行演示。

我们将 5-1 课程的示例项目拿过来，复制 **provider** 为两个项目 **provider-1** 和 **provider-2**，将 **provider-2** 的端口修改为 4002，依次启动 **eureka**、**consumer**、**provider-1** 和 **provider-2** 项目。

全部启动完毕后，多次访问地址：`http://localhost:4003/hello?name=neo`，页面展示结果如下：

```
hello neo ; 4001
hello neo ; 4002
...
```

说明服务启动成功，并且请求时被均衡地分发到后端的两个服务。

接下来将上一节的项目复制一份，修改项目名称为 **gateway**，将网关项目也注册到注册中心的，修改配置文件的内容如下：

application.yml

```

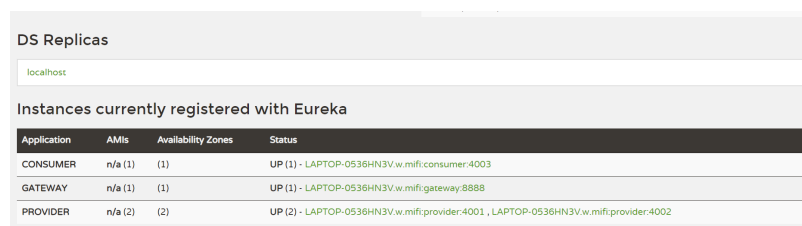
server:
  port: 8888
spring:
  application:
    name: gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1111/eureka/
logging:
  level:
    org.springframework.cloud.gateway: debug

```

配置说明：

- `spring.cloud.gateway.discovery.locator.enabled`：是否开启通过注册中心进行路由转发的功能，通过 `serviceId` 转发到服务，默认为 `false`。
- `eureka.client.service-url.defaultZone` 设置注册中心的地址，使网关项目注册到注册中心。
- `logging.level.org.springframework.cloud.gateway` 调整 `gateway` 包的 `log` 级别，以便排查问题。

修改完成后启动 `gateway` 项目，访问注册中心地址 `http://localhost:1111/` 页面效果如下图：



The screenshot shows the Eureka UI with the following content:

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CONSUMER	n/a (1)	(1)	UP (1) - LAPTOP-0536HN3V.w.mifi:consumer:4003
GATEWAY	n/a (1)	(1)	UP (1) - LAPTOP-0536HN3V.w.mifi:gateway:8888
PROVIDER	n/a (2)	(2)	UP (2) - LAPTOP-0536HN3V.w.mifi:provider:4001 , LAPTOP-0536HN3V.w.mifi:provider:4002

将 **Gateway** 注册到服务中心之后，网关会自动代理所有的在注册中心的服务，访问这些服务的语法为：

`http://网关地址: 端口/服务中心注册 serviceId/具体的 url`

比如我们的 `provider` 项目有一个 `/hello` 的服务，访问此服务的时候会返回：`hello` 参数 `name`；端口。

按照上面的语法我们通过网关来访问，浏览器输入：`http://localhost:8888/CONSUMER/hello?name=neo`。多次访问后页面依次出现如下结果：

```

hello neo ; 4001
hello neo ; 4002
...

```

说明我们通过路由转发功能调用了 `CONSUMER` 的 `hello` 服务，并且我们在配置文件中只是配置了注册中心的地址，并没有配置具体的服务提供者信息。

通过上面的实验说明 `Spring Cloud Gateway` 和 `Eureka` 已经深度融合，只需要在 `Gateway` 中配置好注册中心的地址，即可代理注册中心的所有服务提供者，省掉了中间繁琐的配置。

小结

本节课为大家演示了如何使用 **Spring Cloud Gateway**。**Spring Cloud Gateway** 默认有两种使用方式，一种是通过编码的方式来实现，一种是通过配置文件的方式来实现，推荐使用配置文件的方式来使用，便于后期修改维护。**Spring Cloud Gateway** 支持和注册中心结合起来使用，只要将 **Spring Cloud Gateway** 注册到注册中心，即可自动代理注册中心中的所有服务，简化路由配置和使用方式。

本文作者：纯洁的微笑、江南一点雨