



下载APP



26 | 微服务网关：如何设置请求转发、跨域和限流规则？

2022-02-11 姚秋辰

《Spring Cloud 微服务项目实战》

课程介绍 >

**讲述：姚秋辰**

时长 21:25 大小 19.62M



你好，我是姚秋辰。

在上节课中，我们了解了如何在 Spring Cloud Gateway 中加载一个路由，以及常用的内置谓词都有哪些。今天我们就来动手实践一把，在实战项目中搭建一个 Gateway 网关，并完成三个任务：设置跨域规则、添加路由和实现网关层限流。这三个任务将以怎样的方式展开呢？

领资料

首先是跨域规则，它是一段添加在配置文件中的逻辑。我将在编写网关配置文件的同时顺便为你讲解下跨域的基本原理，以及如何设置同源访问策略。



然后，我将使用基于 Java 代码的方式来定义静态路由规则。当然了，你也可以使用配置文件来编写路由，用代码还是用配置全凭个人喜好。不过呢，如果你的路由规则比较复杂，

比如，它包含了大量谓词和过滤器，那么我还是推荐你使用代码方式，可读性高，维护起来也容易一些。

最后就是网关层限流，我们将使用内置的 Lua 脚本，并借助 Redis 组件来完成网关层限流。

闲话少叙，我们先去搭建一个微服务网关应用吧。你可以在 [Gitee 代码仓库](#) 中找到下面所有源码。

创建微服务网关

微服务网关是一个独立部署的平台化组件，我们先在 `middleware` 目录下创建一个名为 `gateway` 的子模块。接下来的工作就是按部就班地搞定依赖项、配置项和路由规则。

添加依赖项

我们要在这个模块的 `pom.xml` 文件中添加几个关键依赖项，分别是 Gateway、Nacos 和 Loadbalancer。你可以参考下面的代码。

[复制代码](#)

```
1 <dependencies>
2     <!-- Gateway正经依赖 -->
3     <dependency>
4         <groupId>org.springframework.cloud</groupId>
5         <artifactId>spring-cloud-starter-gateway</artifactId>
6     </dependency>
7
8     <!-- Nacos服务发现 -->
9     <dependency>
10        <groupId>com.alibaba.cloud</groupId>
11        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
12    </dependency>
13    <dependency>
14        <groupId>org.springframework.cloud</groupId>
15        <artifactId>spring-cloud-starter-loadbalancer</artifactId>
16    </dependency>
17
18    <!-- Redis+Lua限流 -->
19    <dependency>
20        <groupId>org.springframework.boot</groupId>
21        <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
22    </dependency>
23
```

```
24      <!-- 其它非关键注解请参考源码 -->
25    </dependencies>
```

这里我只列出了核心依赖项，还有一些辅助依赖组件我没有一一列出，你可以参考源码，查看完整的依赖项列表。

在这几个核心依赖项中，打头的 `spring-cloud-starter-gateway` 是最重要的一个，它是实现了网关功能模块的基础组件。而 `Nacos` 和 `Loadbalancer` 则扮演了“导航”的作用，让 `Gateway` 在请求转发的过程中可以通过“服务发现 + 负载均衡”定位到对应的服务节点。最后一个是 `Redis` 依赖项，待会儿我们会用它来实现网关层限流。

虽然我没有把链路追踪组件的相关依赖项添加到 `Gateway` 组件中，但是网关通常是一次服务调用的起点，我们在搭建线上应用的时候，应当把 `Gateway` 纳入到链路追踪体系当中。所以呢我们需要将 `Sleuth`、`Zipkin` 还有 `ELK` 集成进来，我把这个任务留给了你来实现，你可以从依赖项的添加开始，完整回顾一下前面学过的链路追踪知识点，温故而又知新。

依赖项添加完成后，我们接下来去编写 `bootstrap.yml` 和 `application.yml` 配置文件。

添加配置文件

首先，我们创建一个 `bootstrap.yml`，将“`coupon-gateway`”定义为当前项目的名称。使用 `bootstrap.yml` 的目的之一是优先加载 `Nacos Config` 配置项，我们要借助 `Nacos` 来完成动态路由表的加载，这部分的内容我将放到下一课再讲。


 复制代码

```
1 spring:
2   application:
3     name: coupon-gateway
```

接下来，我们创建一个 `application.yml`。这个配置文件里的内容主要就两部分，一部分是 `Nacos` 服务发现的配置项，这段是老生常谈了咱就不再展开讲了。另一部分是 `Gateway` 特有的配置项，我们来看一下。

我会通过 `Java` 代码来落地各种路由规则，所以你看到的配置文件并不包含任何路由规则，非常干净清爽。如果你比较喜欢用配置项来定义路由规则，那你可以在

spring.cloud.gateway.routes 节点下尽情发挥，定义各种路由、谓词断言和过滤器规则。我在 [🔗 上节课](#) 开头写了几个在 yaml 中定义路由规则的例子，你可以参考一下。

 复制代码

```
1 server:
2   port: 30000
3 spring:
4   # 分布式限流的Redis连接
5   redis:
6     host: localhost
7     port: 6379
8   cloud:
9     nacos:
10      # Nacos配置项
11      discovery:
12        server-addr: localhost:8848
13        heart-beat-interval: 5000
14        heart-beat-timeout: 15000
15        cluster-name: Cluster-A
16        namespace: dev
17        group: myGroup
18        register-enabled: true
19      gateway:
20        discovery:
21          locator:
22            # 创建默认路由，以"/服务名称/接口地址"的格式规则进行转发
23            # Nacos服务名称本来就是小写，但Eureka默认大写
24            enabled: true
25            lower-case-service-id: true
26      # 跨域配置
27      globalcors:
28        cors-configurations:
29          '[/**]':
30            # 授信地址列表
31            allowed-origins:
32              - "http://localhost:10000"
33              - "https://www.geekbang.com"
34            # cookie, authorization认证信息
35            expose-headers: "*"
36            allowed-methods: "*"
37            allow-credentials: true
38            allowed-headers: "*"
39            # 浏览器缓存时间
40            max-age: 1000
```

上面这段配置代码的重点是**全局跨域规则**，我在 spring.cloud.gateway.globalcors 下添加了一段跨域规则的相关配置，这里我们就来展开说道说道。

什么是跨域规则

在了解如何配置跨域规则之前，我需要先为你讲一讲什么是浏览器的“**同源保护策略**”。

如果前后端是分离部署的，大部分情况下，前端系统和后端 API 都在同一个根域名下，但也有少数情况下，前后端位于不同域名。比如前端页面的地址是 `geekbang.com`，后端 API 的访问地址是 `infoq.com`。如果一个应用请求发生了跨域名访问，比如位于 `geekbang.com` 的页面通过 Node.js 访问了 `infoq.com` 的后端 API，这种情况就叫“跨域请求”。

我们的浏览器对跨域访问的请求设置了一道保护措施，在跨域调用发起之前，浏览器会尝试发起一个 `OPTIONS` 类型的请求到目标地址，探测一下你的后台服务是否支持跨域调用。如果你的后端 Say NO，那么前端浏览器就会阻止这次非同源访问。通过这种方式，一些美女聊天类的钓鱼网站就无法对你实施跨站脚本攻击了，这就是浏览器端的同源保护策略。



不过也有一种例外，比如你的前端网站和后端接口确实部署在了两个域名，而这两个域名背后都是正经应用，这时候为了让浏览器可以通过同源保护策略的检查，你就必须在后台应用里设置跨域规则，告诉浏览器哪些跨域请求是可以被接受的。

我们接下来就来了解一下，如何通过跨域配置的参数来控制跨域访问。这些参数都定义在的 `spring.cloud.gateway.globalcors.cors-configurations` 节点的 `[/]**` 路径下，`[/]**` 这串通配符可以匹配所有请求路径。当然了，你也可以为某个特定的路径设置跨域规则（比如 `[/order/]`）。


allowed-origins	你可以在allowed-origins中设置可被信任的来源地址List（可使用通配符*表示ALL），如果后台服务接收到跨域请求，它会拿请求Header中的Origin值和List中的值做比较。如果没找到匹配的地址，则禁止跨域访问。
expose-headers	在Response Header中，除了几个基本字段（如Content-Type、Cache-Control等）以外，定义了可以被暴露出去的Header属性。使用通配符*表示允许所有响应头。
allowed-methods	支持跨域的HTTP Method，使用通配符*表示允许所有方法。
allow-credentials	跨域请求默认不带Cookie，若要包含Cookie则要设置成true。
allowed-headers	允许接收的Request Header属性，使用通配符*表示允许所有请求头。

在这上面的几个配置项中，allowed-origins 是最重要的，你需要将受信任的域名添加到这个列表当中。从安全性角度考虑，非特殊情况下我并不建议你使用 * 通配符，因为这意味着后台服务可以接收任何跨域发来的请求。

到这里，所有配置都已经 Ready 了，我们可以去代码中定义路由规则了。

定义路由规则

我推荐你使用一个独立的配置类来管理路由规则，这样代码看起来比较清晰。比如我这里就在 com.geekbang.gateway 下面创建了 RoutesConfiguration 类，为三个微服务分别定义了简单明了的规则。你可以参考一下这段代码。

 复制代码

```
1 @Configuration
2 public class RoutesConfiguration {
3
4     @Bean
5     public RouteLocator declare(RouteLocatorBuilder builder) {
6         return builder.routes()
7             .route(route -> route
8                 .path("/gateway/coupon-customer/**")
9                 .filters(f -> f.stripPrefix(1))
10                .uri("lb://coupon-customer-serv")
11            ).route(route -> route
12                .order(1)
13                .path("/gateway/template/**")
14                .filters(f -> f.stripPrefix(1))
15                .uri("lb://coupon-template-serv")
16            ).route(route -> route
17                .path("/gateway/calculator/**")
```

```
18         .filters(f -> f.stripPrefix(1))
19         .uri("lb://coupon-calculation-serv")
20     ).build();
21 }
22 }
```

这三个路由规则都是大同小异的。我们就以第二个路由规则为例，你可以看出，路由设置遵循了一套三连的风格。

首先，我使用 path 谓词约定了路由的匹配规则为 path= “/template/**” 。这里你要注意的是，如果某一个请求匹配上了多个路由，但你想让各个路由之间有个先后匹配顺序，这时你就可以使用 order(n) 方法设定路由优先级，n 数字越小则优先级越高。

接下来，我使用了一个 stripPrefix 过滤器，将 path 访问路径中的第一个前置子路径删除掉。这样一来，/gateway/template/xxx 的访问请求经由过滤器处理后就变成了 /template/xxx。同理，如果你想去除 path 中打头的前两个路径，那就使用 stripPrefix(2)，参数里传入几它就能吞掉几个 prefix path。

最后，我使用 uri 方法指定了当前路由的目标转发地址，这里的 “lb://coupon-template-serv” 表示使用本地负载均衡将请求转发到名为 “coupon-template-serv” 的服务。

在这一套三连里，谓词和 uri 你是再熟悉不过了，但这个 filter 想必还是第一次见到。我来带你简单了解一下 Gateway Filter 的使用方式，再用一个简单的小案例教你借助过滤器来实现基于 Lua + Redis 的网关层限流。

Filter 和网关限流

在 [第 23 课](#) 中，我们了解了 Gateway 过滤器在一个 Request 生命周期中的作用阶段。其实 Filter 的一大功能无非就是对 Request 和 Response 动手动脚，为什么这么说呢？比如你想对 Request Header 和 Parameter 进行删改，又或者从 Response 里面删除某个 Header，那么你就可以使用下面这种方式，通过链式 Builder 风格构造过滤器链。

```
1 .route(route -> route
2     .order(1)
3     .path("/gateway/template/**")
4     .filters(f -> f.stripPrefix(1))
```

[复制代码](#)

```
5 // 修改Request参数
6 .removeRequestHeader("mylove")
7 .addRequestHeader("myLove", "u")
8 .removeRequestParameter("urLove")
9 .addRequestParameter("urLove", "me")
10 // response系列参数 不一一列举了
11 .removeResponseHeader("responseHeader")
12 )
13 .uri("lb://coupon-template-serv")
```

当然了，Gateway 的内置过滤器远不止上面这几个，还包括了 redirect 转发、retry 重试、修改 requestBody 等等内置 Filter。如果你对这些内容感兴趣，你可以根据 IDE 中自动弹出的代码提示来了解它们，再配几个到路由规则里把玩一下。

接下来，我们通过一个轻量级的网关层限流方案来进一步熟悉 Filter 的使用，这个限流方案所采用的底层技术是 Redis + Lua。

Redis 你一定很熟悉了，而 Lua 这个名词你可能是第一次听说，但提到愤怒的小鸟这个游戏，你一定不陌生，这个游戏就是用 Lua 语言写的。Lua 是一类很小巧的脚本语言，它和 Redis 可以无缝集成，你可以在 Lua 脚本中执行 Redis 的 CRUD 操作。在这个限流方案中，Redis 用来保存限流计数，而限流规则则是定义在 Lua 脚本中，默认使用令牌桶限流算法。如果你对 Lua 脚本的内容感兴趣，可以在 IDE 中全局搜索 request_rate_limiter.lua 这个文件。

前面我们已经添加了 Redis 的依赖和连接配置，现在你可以直接来定义限流参数了。我在 Gateway 模块里新建了一个 RedisLimitationConfig 类，专门用来定义限流参数。我们用到的主要参数有两个，一个是限流的维度，另一个是限流规则，你可以参考下面的代码。

[复制代码](#)

```
1 @Configuration
2 public class RedisLimitationConfig {
3
4     // 限流的维度
5     @Bean
6     @Primary
7     public KeyResolver remoteHostLimitationKey() {
8         return exchange -> Mono.just(
9             exchange.getRequest()
10                 .getRemoteAddress()
11                 .getAddress()
12                 .getHostAddress()
```



```
13         );
14     }
15
16     //template服务限流规则
17     @Bean("tempalteRateLimiter")
18     public RedisRateLimiter templateRateLimiter() {
19         return new RedisRateLimiter(10, 20);
20     }
21
22     // customer服务限流规则
23     @Bean("customerRateLimiter")
24     public RedisRateLimiter customerRateLimiter() {
25         return new RedisRateLimiter(20, 40);
26     }
27
28     @Bean("defaultRateLimiter")
29     @Primary
30     public RedisRateLimiter defaultRateLimiter() {
31         return new RedisRateLimiter(50, 100);
32     }
33 }
```

我在 `remoteHostLimitationKey` 这个方法中定义了一个以 `Remote Host Address` 为维度的限流规则，当然了你也可以自由发挥，改用某个请求参数或者用户 ID 为限流规则的统计维度。其它的三个方法定义了基于令牌桶算法的限流速率，`RedisRateLimiter` 类接收两个 `int` 类型的参数，第一个参数表示每秒发放的令牌数量，第二个参数表示令牌桶的容量。通常来说一个请求会消耗一张令牌，如果一段时间内令牌产生量大于令牌消耗量，那么积累的令牌数量最多不会超过令牌桶的容量。

定义好了限流参数之后，我们来看一下如何将限流规则应用到路由表中。

因为 Gateway 路由规则都定义在 `RoutesConfiguration` 类中，所以你需要把刚才我们定义的限流参数类注入到 `RoutesConfiguration` 类中。考虑到不同的路由表可能会使用不同的限流参数，所以你在定义多个限流参数的时候，可以使用 `@Bean("customerRateLimiter")` 这种方式来做区分，然后在 `Autowired` 注入对象的时候，使用 `@Qualifier("customerRateLimiter")` 指定想要加载的限流参数就可以了。

[复制代码](#)

```
1 @Autowired
2 private KeyResolver hostAddrKeyResolver;
3
4 @Autowired
5
```

```
6 @Qualifier("customerRateLimiter")
7 private RateLimiter customerRateLimiter;
8
9 @Autowired
10 @Qualifier("tempalteRateLimiter")
11 private RateLimiter templateRateLimiter;
```

限流参数注入完成之后，接下来我们只需要添加一个内置的限流过滤器，分别指定限流的维度、限流速率就可以了，你可以参考下面这段 `requestRateLimiter` 过滤器配置代码。除了限流参数之外，我还额外定义了一个 Status Code，当服务请求被限流的时候，后端服务便会返回我指定的这个 Status Code。

[复制代码](#)

```
1 .route(route -> route.path("/gateway/coupon-customer/**"))
2     .filters(f -> f.stripPrefix(1)
3         .requestRateLimiter(limiter-> {
4             limiter.setKeyResolver(hostAddrKeyResolver);
5             limiter.setRateLimiter(customerRateLimiter);
6             // 限流失败后返回的HTTP status code
7             limiter.setStatusCode(HttpStatus.BANDWIDTH_LIMIT_EXCEEDED);
8         })
9     )
10 )
11 .uri("lb://coupon-customer-serv")
```

到这里，我们就完整搭建了 Gateway 组件的路由和限流规则，最后你只需要写一个普通的启动类就可以在本地测试了。接下来我来带你回顾一下这一节的重点内容吧。

总结

今天我们为三个微服务组件设置了路由规则和限流规则。尽管 Gateway 组件本身提供了丰富的内置谓词和过滤器，但在实际项目中我们大多用不到它们，因为网关层的核心用途只是简单的路由转发，**为了保证组件之间的职责隔离，我并不建议通过谓词和过滤器实现带有业务属性的逻辑。**

那什么样的逻辑可以在网关层实现呢？比如一些通用的身份鉴权、登录检测和签名验签之类的服务，你可以将这类安全检测的逻辑前置到网关层来实现，这样可以对不合法请求做快速失败处理。

思考题

结合这节课的内容，请你尝试说一说，内置 Filter 是如何实现的，它继承了哪些通用类和接口。再请你在本地用类似的方式实现一个自定义的过滤器，并配置到路由表中。你可以使用这个过滤器完成一些简单的业务，比如打印所有发到网关服务的请求和响应参数。

好啦，这节课就结束啦。欢迎你把这节课分享给更多对 Spring Cloud 感兴趣的朋友。我是姚秋辰，我们下节课再见！

分享给需要的人，Ta购买本课程，你将得 20 元

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 25 | 微服务网关：Gateway 中的路由和谓词有何应用？

精选留言 (3)

 写留言



peter

2022-02-12

请教老师几个问题：

Q1：Gateway的限流与sentinel的限流是什么关系？

Gateway的限流是替代sentinel的限流吗？或者是相互配合？从“总结”部分来看，老师你是不赞成在Gateway做限流吗？

Q2：定义路由规则中的uri用的lb，gateway怎么知道是用的哪一个loadbalancer? 需要配...
展开



inrtyx

2022-02-11

老师，能否讲讲网关如何鉴权？即鉴权时序图





黄叶

2022-02-11

老师请问下，今天试着写了gateway+vue整合。

vue请求发送给后端，预检通过了但是当真正发送跨域请求时，提示：CORS错误，也配置了gateway跨域 但是就是不成功

