



下载APP



27 | 微服务网关：如何借助 Nacos 实现动态路由规则？

2022-02-14 姚秋辰

《Spring Cloud 微服务项目实战》

课程介绍 >



讲述：姚秋辰

时长 13:46 大小 12.62M



你好，我是姚秋辰。

在上节课中，我们通过一系列谓词和过滤器的组合为三个微服务模块配置了路由规则，这套方案足以应对大部分线上业务的需求，但在可扩展性方面还不够完美。为什么这么说呢？因为这些路由规则是以 yml 文件或者 Java 代码配置在项目中的静态规则。随着项目启动，这些路由规则会被加载到应用上下文并生效。但在程序运行期，如果我们想要修改这些预定义的路由规则，或者创建新的路由规则，似乎只有提交改动到 Gateway 组件 -> 编译项目 -> 重新部署这一条路子。



那么，如果我们希望不重新部署网关，就能更改路由规则，可以有哪些途径呢？

有一种“临时性”的方案，是借助 Gateway 网关的 actuator endpoint 进行 CRUD。Gateway 组件内定义了一套内置的 actuator endpoints，当满足下面两个条件时，我们就可以借助 actuator 提供的能力对路由表进行修改了。

项目中存在 spring-boot-starter-actuator 依赖项；

Gateway 组件的 actuator endpoint 已对外开放。

为了满足上面这两个条件，我已经将配置项添加到了 Gateway 模块中，并且在 application.yml 文件中的 management 节点下，对外开放了所有 actuator 端点。

接下来，你就可以借助 Gateway 组件的 actuator endpoint 完成一系列 CRUD 操作了。以实战项目的源码为例，actuator endpoint 地址是 localhost:30000/actuator/gateway/routes。这套接口遵循了标准的 RESTful 规范，你可以对这个路径发起 GET 请求，获取一段 JSON 格式的路由规则全集，也可以使用 POST 请求添加一个新的路由规则，或者使用 PUT/DELETE 请求修改 / 删除指定路由规则。


好了，Gateway 的 actuator 动态路由功能我就点到即止了，actuator 方案尽管实现了动态路由管理，但这些动态路由只保存在了应用的上下文中，一重启就没了。接下来我要给你介绍个更牛的方案，它不仅能动态管理路由表，而且还能让这些规则实现持久化，无论怎么重启都不会丢失路由规则。

下面，我们就来了解一下，如何借助 Nacos Config 实现动态路由规则的持久化。

使用 Nacos Config 添加动态路由表

但凡有动态配置相关的需求，使用 Nacos Config 就对了。上节课里我已经将 Nacos Config 的依赖项添加到了 Gateway 模块，接下来我们直奔主题，看一下 Gateway 和 Nacos 是如何来集成的吧。

首先，我们需要定义一个底层的网关路由规则编辑类，它的作用是将变化后的路由信息添加到网关上下文中。我把这个类命名为 GatewayService，放置在 com.geekbang.gateway.dynamic 包路径下。

 复制代码

```
1 @Slf4j
2 @Service
```

```
3 public class GatewayService {
4
5     @Autowired
6     private RouteDefinitionWriter routeDefinitionWriter;
7
8     @Autowired
9     private ApplicationEventPublisher publisher;
10
11     public void updateRoutes(List<RouteDefinition> routes) {
12         if (CollectionUtils.isEmpty(routes)) {
13             log.info("No routes found");
14             return;
15         }
16
17         routes.forEach(r -> {
18             try {
19                 routeDefinitionWriter.save(Mono.just(r)).subscribe();
20                 publisher.publishEvent(new RefreshRoutesEvent(this));
21             } catch (Exception e) {
22                 log.error("cannot update route, id={}", r.getId());
23             }
24         });
25     }
26 }
```

这段代码接收了一个 RouteDefinition List 对象作为入参，它是 Gateway 网关组件用来封装路由规则的标准类，在里面包含了谓词、过滤器和 metadata 等一系列构造路由规则所需要的元素。在主体逻辑部分，我调用了 Gateway 内置的路由编辑类 RouteDefinitionWriter，将路由规则写入上下文，再调用 ApplicationEventPublisher 类发布一个路由刷新事件。

接下来，我们要去做一个中间层转换层来对接 Nacos 和 GatewayService，这个中间层主要完成两个任务，一是动态接收 Nacos Config 的参数，二是将配置文件的内容转换为 GatewayService 的入参。

这里我不打算使用 @RefreshScope 来获取 Nacos 动态参数了，我另辟蹊径使用了一种更为灵活的监听机制，通过注册一个“监听器”来获取 Nacos Config 的配置变化通知。我把这段逻辑封装在了 DynamicRoutesListener 类中，它位于 GatewayService 同级目录下，你可以参考下面的代码实现。

 复制代码

```
1 @Slf4j
2 @Component
```

```
3 public class DynamicRoutesListener implements Listener {
4
5     @Autowired
6     private GatewayService gatewayService;
7
8     @Override
9     public Executor getExecutor() {
10         log.info("getExecutor");
11         return null;
12     }
13
14     // 使用JSON转换，将plain text变为RouteDefinition
15     @Override
16     public void receiveConfigInfo(String configInfo) {
17         log.info("received routes changes {}", configInfo);
18
19         List<RouteDefinition> definitionList = JSON.parseArray(configInfo, Rou
20             gatewayService.updateRoutes(definitionList);
21     }
22 }
```

DynamicRoutesListener 实现了 Listener 接口，后者是 Nacos Config 提供的标准监听器接口，当被监听的 Nacos 配置文件发生变化的时候，框架会自动调用 receiveConfigInfo 方法执行自定义逻辑。在这段方法里，我将接收到的文本对象 configInfo 转换成了 List 类，并调用 GatewayService 完成路由表的更新。


这里需要你注意的一点是，你需要按照 RouteDefinition 的 JSON 格式来编写 Nacos Config 中的配置项，如果两者格式不匹配，那么这一步格式转换就会抛出异常。

定义好了监听器之后，接下来你就要考虑如何来加载 Nacos 路由配置项了。我们需要在两个场景下加载配置文件，一个是项目首次启动的时候，从 Nacos 读取文件用来初始化路由表；另一个场景是当 Nacos 的配置项发生变化的时候，动态获取配置项。

为了能够一石二鸟简化开发，我决定使用一个类来搞定这两个场景。我定义了一个叫做 DynamicRoutesLoader 的类，它实现了 InitializingBean 接口，后者是 Spring 框架提供的标准接口。它的作用是在当前类所有的属性加载完成后，执行一段定义在 afterPropertiesSet 方法中的自定义逻辑。


在 afterPropertiesSet 方法中我执行了两项任务，第一项任务是调用 Nacos 提供的 NacosConfigManager 类加载指定的路由配置文件，配置文件名是 routes-config.json；

第二项任务是将前面我们定义的 `DynamicRoutesListener` 注册到 `routes-config.json` 文件的监听列表中，这样一来，每次这个文件发生变动，监听器都能够获取到通知。

 复制代码

```
1 @Slf4j
2 @Configuration
3 public class DynamicRoutesLoader implements InitializingBean {
4
5     @Autowired
6     private NacosConfigManager configService;
7
8     @Autowired
9     private NacosConfigProperties configProps;
10
11     @Autowired
12     private DynamicRoutesListener dynamicRoutesListener;
13
14     private static final String ROUTES_CONFIG = "routes-config.json";
15
16     @Override
17     public void afterPropertiesSet() throws Exception {
18         // 首次加载配置
19         String routes = configService.getConfigService().getConfig(
20             ROUTES_CONFIG, configProps.getGroup(), 10000);
21         dynamicRoutesListener.receiveConfigInfo(routes);
22
23         // 注册监听器
24         configService.getConfigService().addListener(ROUTES_CONFIG,
25             configProps.getGroup(),
26             dynamicRoutesListener);
27     }
28
29 }
```

到这里，我们的代码任务就完成了，你只需要往项目的 `bootstrap.yml` 文件中添加 Nacos Config 的配置项就可以了。按照惯例，我仍然使用 `dev` 作为存放配置文件的 `namespace`。

 复制代码

```
1 spring:
2   application:
3     name: coupon-gateway
4   cloud:
5     nacos:
6       config:
```

```
7      server-addr: localhost:8848
8      file-extension: yml
9      namespace: dev
10     timeout: 5000
11     config-long-poll-timeout: 1000
12     config-retry-time: 100000
13     max-retry: 3
14     refresh-enabled: true
15     enable-remote-sync-config: true
```

完成了以上步骤之后，Gateway 组件的改造任务就算搞定了，接下来我再带你去 Nacos 里创建一个路由规则配置文件。

添加 Nacos 配置文件

在 Nacos 配置列表页中，你需要在“开发环境”的命名空间下创建一个 JSON 格式的文件，文件名要和 Gateway 代码中的名称一致，叫做“routes-config.json”，它的 Group 是默认分组，也就是 DEFAULT_GROUP。

创建好之后，你需要根据 RoutesDefinition 这个类的格式定义配置文件的内容。以 coupon-customer-serv 为例，我编写了下面的路由规则。

[复制代码](#)

```
1  [{
2      "id": "customer-dynamic-router",
3      "order": 0,
4      "predicates": [{
5          "args": {
6              "pattern": "/dynamic-routes/**"
7          },
8          "name": "Path"
9      }],
10     "filters": [{
11         "name": "StripPrefix",
12         "args": {
13             "parts": 1
14         }
15     }],
16     "uri": "lb://coupon-customer-serv"
17 }]
```

在这段配置文件中，我指定当前路由的 ID 是 `customer-dynamic-router`，并且优先级为 0。除此之外，我还定义了一段 Path 谓词作为路径匹配规则，还通过 `StripPrefix` 过滤器将 Path 中第一个前置路径删除。

创建完成后，你可以在本地启动项目，并尝试访问 `localhost:30000/dynamic-routes/coupon-customer/requestCoupon`，发起一个用户领券请求到 Gateway 组件来领取优惠券。在配置正确无误的情况下，这个请求就会被转发到 Customer 服务啦。

到这里，我们就完整搭建了一套可被持久化的动态路由方案。下面让我来带你回顾下本节重点吧。

总结

在今天的课程里，我们借助 Nacos Config 作为路由规则的数据源，完成了路由表的动态加载和持久化。我这里讲的解决方案只是一种思路，在代码中我还留了一个坑给你来填，希望你可以顺着这节课学到的技术方案向下继续探索，把这个坑填上。

我所指的坑是什么呢？我在 Nacos Config 里定义的路由表中有一个 ID，它是这个路由的全局唯一 ID，借助这个 ID 呢我们就可以完成路由的 UPDATE 操作。但是，如果我想要删除某个路由，应该怎么办呢？

我可以给你提示几个解决方案，你挑其中一种来实现就好。比如说，你可以对 Nacos 配置项做一层额外封装，添加几个新字段用来表示“删除路由”这个语义，并创建一个自定义 POJO 类接收参数；还有，你可以在路由的 metadata 里为 Nacos 的动态路由做一个特殊标记，每次当 Nacos 刷新路由表的时候，就删除上下文当中的所有 Nacos 路由表，再重新创建；又或者你可以通过 metadata 做一个逻辑删除的标记，每次更新路由表的时候只要见到这个标记就删除当前路由，否则就更新或新建路由。

思考题

我在上面提示了几种路由删除的方案，希望可以抛砖引玉，接下来就轮到你来设计并实现一个优雅方案了，欢迎在留言区分享你自己的思路。

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 2 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 26 | 微服务网关：如何设置请求转发、跨域和限流规则？

下一篇 28 | 消息驱动：谁说消息队列只能削峰填谷？

精选留言 (1)

写留言



peter

2022-02-14

请教老师四个问题啊：

Q1：Gateway内置的actuator endpoint是Gateway独有的？还是具有通用性？比如其他的组件(e.g,nacos)也可以加入actuator endpoint？

Q2：DynamicRoutesListener是观察者模式吗？

Q3：DynamicRoutesLoader是把Nacos中的文件“routes-config.json”读取过来吗？（相当...
展开

作者回复: Q1：通用规范接口，理念类似于jmx

Q2：设计模式是微观层面的实现，很难把某个功能归类到设计模式上。只能说表现方式上是observer，但底层实现复杂得多

Q3：读取文件内容

Q4：gateway-serv是gateway这个module的一个类，和template-serv->Template模块这个所属关系一样



1