15 Feign 中的继承、日志与数据压缩

更新时间：2019-06-21 09:37:43

> 理想的书籍是智慧的钥匙。
>
> ——列夫·托尔斯泰

上篇文章和大家分享了声明式微服务调用组件 Feign 的基本用法，相信大家已经了解到使用 Feign 的好处了，使用 Feign 有效地解决了使用 RestTemplate 时的代码模板化的问题，使服务之间的调用更加简单方便，同时也不易出错。不过，细心的读者可能也发现，上篇文章中我们学的 Feign 还是有一些明显的缺陷，例如，当我们在 provider 中定义接口时，可能是下面这样：

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello(String name) {
        return "hello " + name + " !";
    }
}
```

然后在 feign-consumer 中定义接口的调用，又是下面这样：

```
@FeignClient("provider")
public interface HelloService {
    @GetMapping("/hello")
    String hello(@RequestParam("name") String name);
}
```

这两段代码其实也有部分重复了，例如接口的定义、请求参数绑定、方法返回值等，都是一样的，只是一个有接口的具体实现，一个没有具体实现而已。这些相同的代码如果写错了，还有可能导致调用失败，例如 provider 中写了 `@GetMapping("/hello")` ，而 feign-consumer 中写了 `@GetMapping("/hello2")` ，此时就会调用失败，那么如何避免这些问题呢？这就要使用到我们本文介绍到的 Feign 的继承特性。另外，Feign 中的操作日志可以帮助我们快速定位问题，数据压缩特性又能够提高数据传输效率，这些知识点，我将在本文和大家分享。

准备工作

和前面的文章一样，我们需要先做一些准备工作。首先创建一个名为 FeignAdvanced 的父工程，然后在父工程中创建一个子模块 eureka 服务注册中心并启动，具体的操作步骤我这里就不再赘述，大家要是忘记了，可以参考上一章的第一小节。

然后再在 FeignAdvanced 工程中创建一个子模块 commons。注意，这个子模块是一个 Maven 工程，而不是 Spring Boot 工程，因为这个模块我只是用它来提供公共接口。

commons 模块创建成功后，因为要在 commons 模块中使用 SpringMVC 的一整套东西，方便起见，在 commons 模块的 pom.xml 文件中添加如下依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>
```

添加成功后，再在 commons 中添加一个 HelloService 接口：

```
public interface HelloService {
    @GetMapping("/hello")
    String hello(@RequestParam("name") String name);
}
```

在这个 HelloService 接口中，我们会将前面提到的 provider 和 feign-consumer 中公共的部分抽取出来定义在这里，然后在 provider 中调用这个接口，在 feign-consumer 中实现这个接口。

好了， HelloService 接口定义完成后，我们的准备工作就算是OK啦，接下来我们就来看看具体的继承特性要如何去实现。

继承特性

Feign 中继承，我们整体可以分两步来实现：

1. 在 provider 中实现公共接口；
2. 在 feign-consumer 中去调用接口。

我们分别来看。

在 provider 中实现接口

首先创建一个 provider 微服务项目，创建成功后，将刚刚创建的 commons 项目的依赖添加进来，完整的 pom.xml 文件如下：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>com.justdojava</groupId>
    <artifactId>commons</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

然后修改 provider 的配置文件，将 provider 微服务注册到 eureka 服务注册中心上，这一步也比较简单，不赘述。

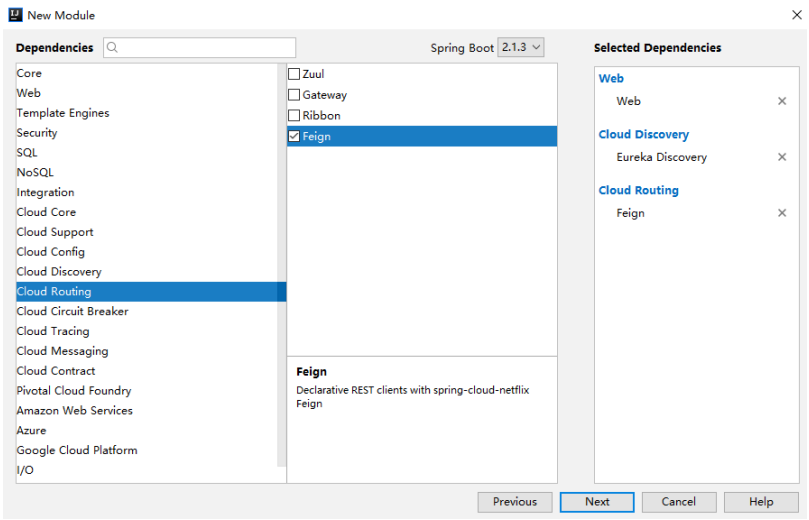
接下来，我们在 provider 中定义一个 HelloController 来实现 commons 模块中的 HelloService 接口，如下：

```
@RestController
public class HelloController implements HelloService {
    @Override
    public String hello(String name) {
        return "hello " + name + " !";
    }
}
```

做完这一切之后，就可以启动 provider 啦！ provider 启动成功后，我们来继续开发 feign-consumer 。

在 feign-consumer 中调用接口

接下来我们在 FeignAdvanced 工程中再创建一个子模块 feign-consumer ，注意创建时候除了选择 Eureka Discovering 依赖之外，还需要选择 Feign 的依赖，具体步骤和上文一致，如下图：



项目创建成功后，也将 commons 模块加入到 pom.xml 文件中，如下：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>com.justdojava</groupId>
    <artifactId>commons</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

然后修改 feign-consumer 的配置文件 application.properties ，将 feign-consumer 注册到 eureka 服务注册中心上，这一步就比较简单，我这里就不再赘述。

接下来，和前面的步骤一样，在项目的启动类上添加 @EnableFeignClients 注解，开启 Feign 的使用：

```
@SpringBootApplication
@EnableFeignClients
public class FeignConsumerApplication {

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

}
```

再接下来创建的 FeignHelloService 接口继承自 HelloSerivce 接口，如下：

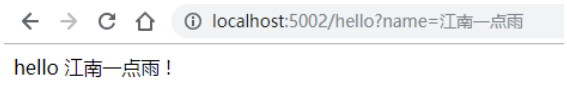
```
@FeignClient("PROVIDER")
public interface FeignHelloService extends HelloService {
}
```

注意，不同于上篇文章中的 HelloService 接口，这里的 FeignHelloService 接口直接继承自 HelloSerivce ，继承之后， FeignHelloService 自动具备了 HelloService 中的接口，因此可以在使用 @FeignClient(“PROVIDER”) 注解绑定服务之后就可以直接使用了。

最后，我们来创建一个 UseHelloController，在 UseHelloController 中来使用 FeignHelloService，如下：

```
@RestController
public class UseHelloController {
    @Autowired
    FeignHelloService feignHelloService;
    @GetMapping("/hello")
    public String hello(String name) {
        return feignHelloService.hello(name);
    }
}
```

配置完成后，启动 feign-consumer ，在浏览器中就可以访问 feign-consumer 了，通过 feign-consumer 就能调用 provider 的服务了。



优缺点分析

通过上面案例的搭建，相信大家对 Feign 的继承特性已经有了一个大致的了解，那么这种写法和上篇文章我们的写法各自有什么优缺点呢？我们来分析下：

1. 使用继承特性，代码简洁明了，不易出错，不必担心接口返回值是否写对，接口地址是否写对。如果接口地址有变化，也不用 provider 和 feign-consumer 大动干戈，只需要修改 commons 模块即可，provider 和 feign-consumer 就自然变了；
2. 前面提到的在 feign-consumer 中绑定接口时，如果是 key/value 形式的参数或者放在 header 中的参数，就必须要使用 @RequestParam 注解或者 @RequestHeader 注解，这个规则在这里一样适用。即在 commons 中定义接口时，如果涉及到相关参数，该加的@RequestParam 注解或者 @RequestHeader 注解一个都不能少；
3. 当然，使用了继承特性也不是没有缺点。继承的方式将 provider 和 feign-consumer 绑定在一起，代码耦合度变高，一变俱变，此时就需要严格的设计规范，否则会牵一发而动全身，增加项目维护的难度。

好了，通过上面这样一个简单的案例，相信大家对 Feign 的继承特性已经有所了解。

日志配置

使用了 Feign 之后，如果希望能够查看微服务之间调用的日志，则可以通过开启 Feign 的日志功能实现， Feign 中的日志级别一共分为四种：

1. NONE，不开启日志记录，默认即此；
2. BASIC，记录请求方法和请求 URL，以及响应的状态码以及执行时间；
3. HEADERS，在第2条的基础上，再增加请求头和响应头；
4. FULL，在第3条的基础上再增加 body 以及元数据。

那么具体的配置是怎样的呢？很简单，首先在配置类中配置一个日志级别的 Bean ，我这里直接放在系统启动类中，如下：

```
@SpringBootApplication
@EnableFeignClients
public class FeignConsumerApplication {

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

    @Bean
    Logger.Level loggerLevel() {
        return Logger.Level.FULL;
    }
}
```

然后在 application.properties 中开启日志级别。注意， Feign 中的日志只对 DEBUG 级别的日志输出进行响应：

```
logging.level.com.justdojava.feignconsumer.FeignHelloService=debug
```

这里 logging.level 是指日志级别的前缀，com.justdojava.feignconsumer.FeignHelloService 表示该 class 以 debug 级别输出日志。当然，类路径也可以是一个 package ，这样就表示该 package 下的所有 class 以 debug 级别输出日志。配置完成后，重启 feign-consumer 项目，访问其中任意一个接口，就可以看到请求日志，如下：

数据压缩

使用 Feign 执行请求时，也可以对请求数据执行 GZIP 压缩，提高数据传输效率。具体配置如下：

```
feign.compression.request.enabled=true
feign.compression.response.enabled=true
feign.compression.request.mime-types=text/html,application/json
feign.compression.request.min-request-size=2048
```

前两行表示开启请求和响应压缩，第三行表示压缩的数据类型，默认是 `text/html,application/json,application/xml` ，第四行表示压缩数据的下限，即当要传输的数据大于2048时才需要对请求进行压缩。

请求重试

Feign 中默认也自带请求重试功能，即这里不需要添加 `spring-retry` 依赖，直接配置即可使用：

```
# 最大的重试次数，不包括第一次请求
ribbon.MaxAutoRetries=3
# 最大重试server的个数，不包括第一个 server
ribbon.MaxAutoRetriesNextServer=1
# 是否开启任何异常都重试
ribbon.OkToRetryOnAllOperations=false
```

这样的配置，请求失败重试适用于所有的请求，也可以配置专门针对某一个微服务的请求失败重试，例如专门配置针对 provider 微服务的请求失败重试，如下：

```
#最大的重试次数
provider.ribbon.MaxAutoRetries=3
#最大重试server的个数
provider.ribbon.MaxAutoRetriesNextServer=1
#是否开启任何异常都重试
provider.ribbon.OkToRetryOnAllOperations=false
```

这样，就可以针对不同的微服务配置不同的请求失败重试策略。

也可以不配置 application.properties ，而是通过提供如下一个 Bean 来实现请求重试：

```
@Bean
public Retryer feignRetryer() {
   Retryer.Default retryer = new Retryer.Default();
   return retryer;
}
```

小结

本文主要向大家介绍了声明式微服务调用工具 Feign 的一些高级特性，例如继承机制、日志配置、请求压缩、请求重试等，并对继承特性的优缺点进行了分析。在实际开发中，灵活地使用这些属性，可以使我们的微服务以一个更高的效率运行。通过对这些特性的学习，相信大家对 Feign 将会有一个更深刻的认识。

本文作者：纯洁的微笑、江南一点雨