参考：

Spring Cloud构建微服务架构：服务消费（Feign）【Dalston版】（翟永超）

前言

前文讲述的ribbon虽说已经帮我们开发人员省去了很多不必要的麻烦，但还是觉得怪怪的，有想法的同学应该注意到了，这很不符合我们这个时代力求的面向接口编程的思想，这样做的话我们的逻辑代码就和其它系统耦合了。我们实际开发中肯定想要的是一个系统只暴露接口层。

那么本文涉及的Spring cloud feign就是要解决这个问题的。

demo

我们创建了一个Springboot工程，我们看看这里的pom和之前有啥不一样的。

```
<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
http://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>1.5.10.RELEASE</version>
	</parent>

	<groupId>com.xyz</groupId>
	<artifactId>eureka-consumer-feign</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>eureka-consumer-feign</name>
	<url>http://maven.apache.org</url>
```

```
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <jdk.version>1.8</jdk.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-feign</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Dalston.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <!-- 指定source和target的版本 -->
                    <source>${jdk.version}</source>
                    <target>${jdk.version}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

他这边引入了这个新的依赖。

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

我们来看看启动类有啥变化。

```
@EnableFeignClients("com.xyz.controller")
@EnableDiscoveryClient
@SpringBootApplication
@ComponentScan("com.xyz.controller")
public class Application {

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

}
```

这里比之前多了个@EnableFeignClients注解，并且不再用在这里初始化RestTemplate了。

正因为使用了feign，我们这里就可以声明一个接口去操作了，我们看看这个接口。

```
@FeignClient("eureka-client")
public interface DemoClient {

	@GetMapping("/test")
	String consumer();
}
```

很简单，通过@FeignClient声明服务提供者的实例名，@GetMapping指定调用了映射地址。上面提到的@EnableFeignClients就会扫描这个接口并使其生效。

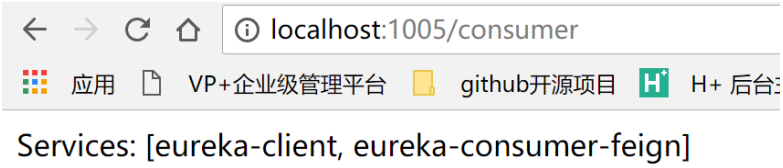
这时候我们再写Controller，你终于会有种熟悉的感觉。

```
@RestController
public class DemoController {

	@Autowired
	private DemoClient demoClient;

	@GetMapping("/consumer")
	public String dc() {
		return demoClient.consumer();
	}

}
```

这种controller的结构是不是很熟悉了，果然还是大爱面向接口编程啊。

最后附上配置文件和结果。

```
spring.application.name=eureka-consumer-feign
server.port=1005

eureka.client.serviceUrl.defaultZone=http://localhost:1001/eureka/
```



原理分析

其实也没啥，学会了ribbon，有想法的程序猿们肯定都想自己动手把那复杂的东西抽取出来，feign就是帮我们做了这一步，当然feign也不是说只是简单的封装了ribbon，feign还整合的Hystrix来实现服务的容错保护，这个等后面再行整理。