参考：

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

前言

上一篇的消费者的简单模拟我们虽然连通了消费者和提供者，不过很麻烦。

```
@GetMapping("/consumer")
public String dc() {
    ServiceInstance serviceInstance = loadBalancerClient.choose("eureka-client");
    String url = "http://" + serviceInstance.getHost() + ":" + serviceInstance.getPort() + "/test";
    System.out.println(url);
    return restTemplate.getForObject(url, String.class);
}
```

注入了LoadBalancerClient操作每次开发都得手工写，显然很麻烦。该篇要讲的Spring Cloud Ribbon就是为了简化这里的操作。

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具。它是一个基于HTTP和TCP的客户端负载均衡器。它可以通过在客户端中配置ribbonServerList来设置服务端列表去轮询访问以达到均衡负载的作用。

Demo

说再多，搭框架这玩意都不如练一练，同样的，创建一个Springboot工程，其中pom。

```
<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-ribbon</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>eureka-consumer-ribbon</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-ribbon</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>
```

比之前的消费者也就是多了个这玩意。

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

然后我们看看启动类。

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

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

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

}
```

比之前的消费者多了这玩意。

```
@Bean
@LoadBalanced
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

然后看看测试的Controller。

```
@RestController
public class DemoController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/consumer")
    public String dc() {
        String url = "http://eureka-client/test";
        return restTemplate.getForObject(url, String.class);
    }
```

```
}
```

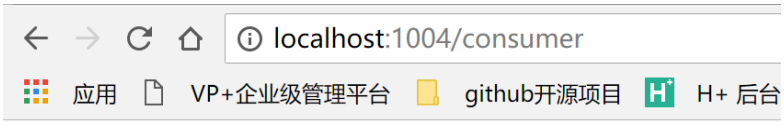
我们发现这里不再用LoadBalancerClient了我们的url就指定了服务名称以及映射路径即可。

最后附一下配置文件。

```
spring.application.name=eureka-consumer-ribbon
server.port=1004

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

基本上比之前的消费者没多啥东西，多引入了包，启动类给RestTemplate多加了个注解，Controller里反而少了很多冗杂的代码，但我们的结果还是对的。



Services: [eureka-client]

原理分析

虽说上述的demo跑通了，但是肯定是有疑问的啊，一通操作猛如虎，一看代码感觉都没用到ribbon，那既然引入了依赖，而且上述demo我们取消了LoadBalancerClient肯定得有补偿措施。我们开篇不就说了吗，ribbon就是用来简化这里的操作，它到底是在哪里简化了呢。

因为Spring Cloud Ribbon有一个拦截器，它能够在这里进行实际调用的时候，自动的去选取服务实例，并将实际要请求的IP地址和端口替换这里的服务名，从而完成服务接口的调用。