Spring Cloud 入门之 Ribbon 篇（二）

一、前言

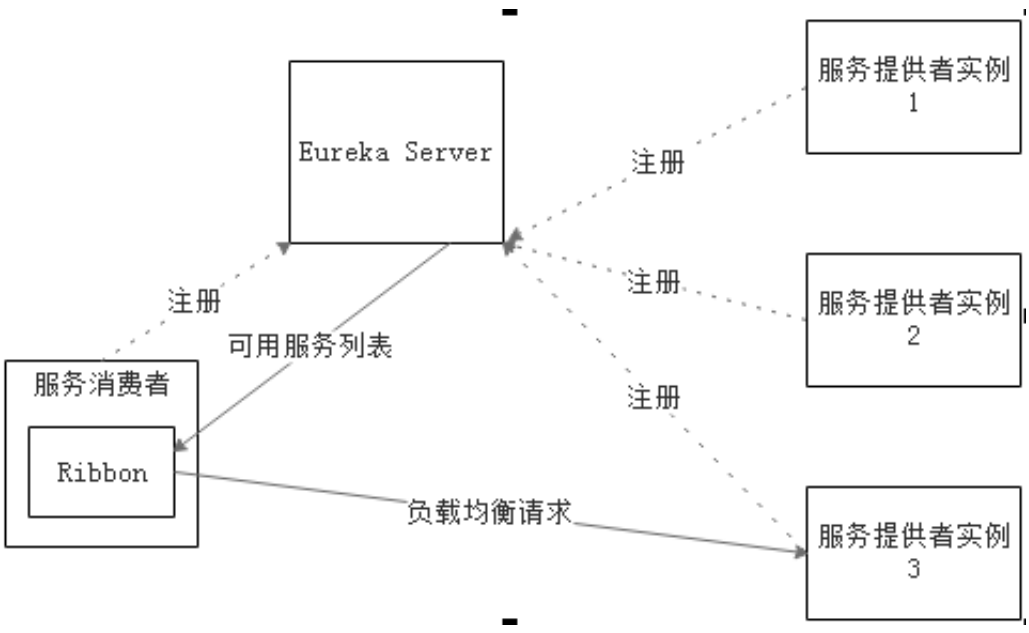
上一篇 介绍了微服务的搭建，服务注册与发现。但在文章中留了一个小尾巴--如何正确使用 Eureka 进行服务发现并调用服务。

本篇文章将介绍如何使用 Ribbon 完成发现服务的调用以及其负载均衡的规则的使用。

二、简单介绍

Spring Cloud Ribbon 是基于 Netflix Ribbon 实现的一套客户端负载均衡工具，其主要功能是提供客户端的软件负载均衡算法，将 Netflix 的中间层服务连接在一起。

其运行原理如下图：



Ribbon 运行时分成 2 个步骤：

1）先选择在同一个区域负载较少的 EurekaServer;
2）再根据用户指定的策略，在从 EurekaServer 中获取注册列表中的服务信息进行调用。

其中，Ribbon 提供多种负载均衡策略：如轮询、随机、响应时间加权等。

三、实战演练

我们在 **user-web** 项目的基础上进行修改。不清楚的读者请先转移至 《**Spring Cloud** 入门 之 **Eureka** 篇（一）》 进行浏览。

此外，笔者额外的创建 2 个 user-provider 项目，即现在有 3 个 user-provider 项目给 user-consumer 进行消费。

1. 添加依赖：

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

其实，添加 Eureka 包时，会自动添加 Ribbon 依赖包。

1. 修改请求类：

```
@Configuration
public class RestConfiguration {

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

正如上文介绍的，Ribbon 是客户端负载均衡工具，所以在 getRestTemplate 方法上添加 **@LoadBalanced** 注解实现负载均衡。

1. 修改请求地址：

```
@RestController
@RequestMapping("/user")
public class UserController {

        @Autowired
        private RestTemplate restTemplate;

//      @RequestMapping("get/{id}")
//      public User get(@PathVariable("id") Integer id) throws Exception {
//              // 没有使用 Eureka 时，uri 为消息提供者的地址，需要指定 ip 和
端口
//              return restTemplate.getForObject(new
URI("http://localhost:8081/provider/user/get/" + id), User.class);
//      }

//      @Autowired
//      private DiscoveryClient client;
//
//      @RequestMapping("get/{id}")
//      public User get(@PathVariable("id") Integer id) throws Exception {
//
//              List<ServiceInstance> list =
this.client.getInstances("USER-API");
//              String uri = "";
//          for (ServiceInstance instance : list) {
//              if (instance.getUri() != null &&
!"".equals(instance.getUri().toString())) {
//                      uri = instance.getUri().toString();
//                      break;
//              }
//          }
//              return restTemplate.getForObject(uri +
"/provider/user/get/" + id, User.class);
//      }

        @RequestMapping("get/{id}")
        public User get(@PathVariable("id") Integer id) throws Exception {
            // 使用 Eureka + Ribbon 后，uri 填写服务名称即可
                return restTemplate.getForObject("http://USER-
API/provider/user/get/" + id, User.class);
```

```
        }

}
```

修改 DiscoveryClient 相关代码，使用 **USER-API** 服务名称作为请求 URL。

1. 在启动类上将 **@EnableDiscoveryClient** 替换成 **@EnableEurekaClient** 注解。

完成上边 4 个操作后，启动 user-consumer 项目使用浏览器访问接口，运行结果如下：

```
@RequestMapping("get/{id}")
public User get(@PathVariable("id") Integer id) throws Exception {
    // 使用 Eureka + Ribbon 后，uri 填写服务名称即可
    return restTemplate.getForObject("http://USER-API/provider/user/get/" + id, User.cl
}
```

由图可知，Ribbon 默认使用负载均衡的策略是轮询，对服务进行调用。

四、负载均衡策略

4.1 策略规则

Ribbon 提供 IRule 接口，该接口定义了如何访问服务的策略，以下是该接口的实现类：

1) RoundRobinRule：轮询，默认使用的规则；
2) RandomRule：随机；
3) AvailabilityFilteringRule：先过滤由于多次访问故障而处于断路器跳闸状态以及并发连接数量超过阀值得服务，然后从剩余服务列表中按照轮询策略进行访问；
4) WeightedResponseTimeRule：根据平均响应时间计算所有的权重，响应时间越快服务权重越有可能被选中；
5) RetryRule：先按照 RoundRobinRule 策略获取服务，如果获取服务失败则在指定时间内进行重试，获取可用服务；
6) BestAvailableRule：先过滤由于多次访问故障而处于断路器跳闸状态的服务，然后选择并发量最小的服务；
7) ZoneAvoidanceRule：判断 server 所在区域的性能和 server 的可用性来选择服务器。

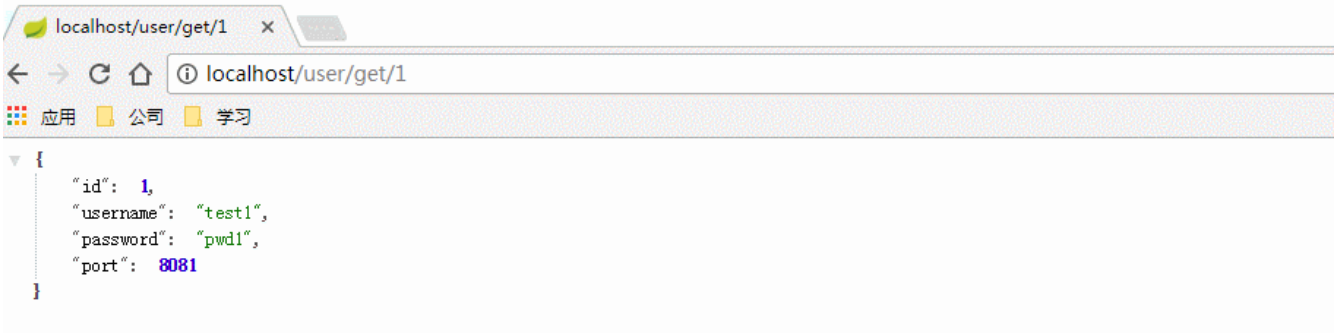
4.2 策略使用

```
@Configuration
public class RestConfiguration {

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

        @Bean
        public IRule testRule() {
            return new RandomRule();
        }
}
```

手动创建负载均衡规则对象，本次测试使用的策略是随机。

启动 user-consumer 项目使用浏览器访问接口，运行结果如下：

```
@RequestMapping("get/{id}")
public User get(@PathVariable("id") Integer id) throws Exception {
    // 使用 Eureka + Ribbon 后，uri 填写服务名称即可
    return restTemplate.getForObject("http://USER-API/provider/user/get/" + id, User.cl
}
```



由图可知，随机策略已生效，负载均衡的策略由轮询变成了随机。

五、案例源码