升级Spring Cloud最新版后，有个重要的组件被弃用了！

原创 梦想de星空 macrozheng 2022-08-02 09:02 发表于江苏

收录于合集
#Spring Cloud学习教程
26个

> 前不久，我把**Mall微服务版本全面升级了**，在通过Gateway网关调用其他服务的时候，出现了 `Service Unavailable` 的问题。排查原因时发现作为负载均衡组件的Ribbon被弃用了，作为Netflix开源的一个组件，Ribbon早已进入维护状态。现在推荐使用的是Loadbalancer，今天我们就来聊聊Loadbalancer的使用！

LoadBalancer简介

LoadBalancer是Spring Cloud官方提供的负载均衡组件，可用于替代Ribbon。其使用方式与Ribbon基本兼容，可以从Ribbon进行平滑过渡。

使用

> 下面介绍下LoadBalancer的基本使用，我们将使用Nacos作为注册中心，通过 `nacos-loadbalancer-service` 和 `nacos-user-service` 两个服务间的相互调用来进行演示。

负载均衡

> 我们将使用RestTemplate来演示下LoadBalancer的负载均衡功能。

- 首先在 `nacos-loadbalancer-service` 模块的 `pom.xml` 文件中添加LoadBalancer相关依赖；

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

- 然后创建Java配置类，用于配置 `RestTemplate` ，同时使用 `@LoadBalanced` 注解赋予其负载均衡能力；

```
/**
 * RestTemplate相关配置
 * Created by macro on 2019/8/29.
 */
@Configuration
public class RestTemplateConfig {

    @Bean
    @ConfigurationProperties(prefix = "rest.template.config")
    public HttpComponentsClientHttpRequestFactory customHttpRequestFactory() {
        return new HttpComponentsClientHttpRequestFactory();
    }

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

- 在 `application.yml` 中可以使用自定义配置对RestTemplate的调用超时进行配置；

```
rest:
  template:
    config: # RestTemplate调用超时配置
      connectTimeout: 5000
      readTimeout: 5000
```

- 然后在Controller中使用RestTemplate进行远程调用；

```
/**
 * Created by macro on 2019/8/29.
 */
@RestController
@RequestMapping("/user")
public class UserLoadBalancerController {
    @Autowired
    private RestTemplate restTemplate;
```

```
    @Value("${service-url.nacos-user-service}")
    private String userServiceUrl;

    @GetMapping("/{id}")
    public CommonResult getUser(@PathVariable Long id) {
        return restTemplate.getForObject(userServiceUrl + "/user/{1}", CommonResult.class, id);
    }

    @GetMapping("/getByUsername")
    public CommonResult getByUsername(@RequestParam String username) {
        return restTemplate.getForObject(userServiceUrl + "/user/getByUsername?username={1}", Com
    }

    @GetMapping("/getEntityByUsername")
    public CommonResult getEntityByUsername(@RequestParam String username) {
        ResponseEntity<CommonResult> entity = restTemplate.getForEntity(userServiceUrl + "/user/g
        if (entity.getStatusCode().is2xxSuccessful()) {
            return entity.getBody();
        } else {
            return new CommonResult("操作失败", 500);
        }
    }

    @PostMapping("/create")
    public CommonResult create(@RequestBody User user) {
        return restTemplate.postForObject(userServiceUrl + "/user/create", user, CommonResult.cla
    }

    @PostMapping("/update")
    public CommonResult update(@RequestBody User user) {
        return restTemplate.postForObject(userServiceUrl + "/user/update", user, CommonResult.cla
    }

    @PostMapping("/delete/{id}")
    public CommonResult delete(@PathVariable Long id) {
        return restTemplate.postForObject(userServiceUrl + "/user/delete/{1}", null, CommonResult
    }
}
```

- 在 `nacos-user-service` 中我们已经实现了这些接口，可以提供给 `nacos-loadbalancer-service` 服务进行远程调用；

- 然后启动一个 `nacos-loadbalancer-service` ，和两个 `nacos-user-service` ，此时Nacos中会显示如下服务；

- 此时通过 `nacos-loadbalancer-service` 调用接口进行测试，会发现两个 `nacos-user-service` 交替打印日志信息，使用的是轮询策略，访问地址：http://localhost:8308/user/1

声明式服务调用

> 当然LoadBalancer除了使用RestTemplate来进行远程调用，还可以使用OpenFeign来进行声明式服务调用，下面我们就来介绍下。

- 首先 `nacos-loadbalancer-service` 模块的 `pom.xml` 文件中添加OpenFeign的相关依赖；

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

- 然后在OpenFeign的客户端接口中声明好需要调用的服务接口以及调用方式；

```
/**
 * Created by macro on 2019/9/5.
 */
@FeignClient(value = "nacos-user-service")
public interface UserService {
    @PostMapping("/user/create")
    CommonResult create(@RequestBody User user);

    @GetMapping("/user/{id}")
    CommonResult<User> getUser(@PathVariable Long id);

    @GetMapping("/user/getByUsername")
    CommonResult<User> getByUsername(@RequestParam String username);

    @PostMapping("/user/update")
    CommonResult update(@RequestBody User user);

    @PostMapping("/user/delete/{id}")
    CommonResult delete(@PathVariable Long id);
}
```

- 再在Controller中使用OpenFeign的客户端接口来调用远程服务；

```
/**
 * Created by macro on 2019/8/29.
```

```
 */
@RestController
@RequestMapping("/userFeign")
public class UserFeignController {
    @Autowired
    private UserService userService;

    @GetMapping("/{id}")
    public CommonResult getUser(@PathVariable Long id) {
        return userService.getUser(id);
    }

    @GetMapping("/getByUsername")
    public CommonResult getByUsername(@RequestParam String username) {
        return userService.getByUsername(username);
    }

    @PostMapping("/create")
    public CommonResult create(@RequestBody User user) {
        return userService.create(user);
    }

    @PostMapping("/update")
    public CommonResult update(@RequestBody User user) {
        return userService.update(user);
    }

    @PostMapping("/delete/{id}")
    public CommonResult delete(@PathVariable Long id) {
        return userService.delete(id);
    }
}
```

- 如果你想设置下OpenFeign的超时配置的话，可以在 `application.yml` 中添加如下内容；

```
feign:
  client:
    config:
      default: # Feign调用超时配置
        connectTimeout: 5000
        readTimeout: 5000
```

- 接下来通过测试接口调用远程服务，发现可以正常调用，访问地址：http://localhost:8308/userFeign/1

服务实例缓存

> LoadBalancer为了提高性能，不会在每次请求时去获取实例列表，而是将服务实例列表进行了本地缓存。

默认的缓存时间为 `35s` ，为了减少服务不可用还会被选择的可能性，我们可以进行如下配置。

```
spring:
  cloud:
    loadbalancer:
      cache: # 负载均衡缓存配置
        enabled: true # 开启缓存
        ttl: 5s # 设置缓存时间
        capacity: 256 # 设置缓存大小
```

HTTP请求转换

> 如果你想在每次远程调用中传入自定义的请求头的话，可以试试 `LoadBalancerRequestTransformer` ，通过它可以对原始请求进行一定的转换。

- 首先我们需要配置好 `LoadBalancerRequestTransformer` 的Bean实例，这里我们将 `ServiceInstance` 的 `instanceId` 放入到请求头 `X-InstanceId` 中；

```
/**
```

```
/**
 * LoadBalancer相关配置
 * Created by macro on 2022/7/26.
 */
@Configuration
public class LoadBalancerConfig {
    @Bean
    public LoadBalancerRequestTransformer transformer() {
        return new LoadBalancerRequestTransformer() {
            @Override
            public HttpRequest transformRequest(HttpRequest request, ServiceInstance instance) {
                return new HttpRequestWrapper(request) {
                    @Override
                    public HttpHeaders getHeaders() {
                        HttpHeaders headers = new HttpHeaders();
                        headers.putAll(super.getHeaders());
                        headers.add("X-InstanceId", instance.getInstanceId());
                        return headers;
                    }
                };
            }
        };
    }
}
```

- 然后修改 `nacos-user-service` 中的代码，打印获取到的请求头 `X-InstanceId` 的信息；

```
/**
 * Created by macro on 2019/8/29.
 */
@RestController
@RequestMapping("/user")
public class UserController {
    @GetMapping("/{id}")
    public CommonResult<User> getUser(@PathVariable Long id) {
        User user = userService.getUser(id);
        LOGGER.info("根据id获取用户信息，用户名称为：{}", user.getUsername());
        ServletRequestAttributes servletRequestAttributes = (ServletRequestAttributes) RequestCont
        HttpServletRequest request = servletRequestAttributes.getRequest();
        String instanceId = request.getHeader("X-InstanceId");
        if (StrUtil.isNotEmpty(instanceId)) {
            LOGGER.info("获取到自定义请求头:X-InstanceId={}", instanceId);
        }
```

```
        return new CommonResult<>(user);
    }
}
```

- 接下来访问接口进行测试， `nacos-user-service` 控制台将打印如下日志，发现自定义请求头已经成功传递了，访问地址：http://localhost:8308/user/1

```
2022-07-26 15:05:19.920  INFO 14344 --- [nio-8206-exec-5] c.macro.cloud.controller.UserController
2022-07-26 15:05:19.921  INFO 14344 --- [nio-8206-exec-5] c.macro.cloud.controller.UserController
```

总结

今天通过对LoadBalancer的一波实践我们可以发现，使用LoadBalancer和Ribbon的区别其实并不大，主要是一些配置方式的相同。如果你之前使用过Ribbon的话，基本上可以无缝切换到LoadBalancer。

参考资料

> 官方文档：https://docs.spring.io/spring-cloud-commons/docs/current/reference/html/#spring-cloud-loadbalancer

项目源码地址

https://github.com/macrozheng/springcloud-learning/tree/master/nacos-loadbalancer-service

---

推荐阅读

- 有没有不用加班的程序员？
- 新一代开源免费的终端工具，太酷了！
- 新来个技术总监，使用流程引擎优化复杂的业务代码，老板惊呆了！
- **Mall**微服务版本全面升级！支持最新版**SpringCloud**，权限解决方案升级**...**
- 仅需一个依赖给**Swagger**换上新皮肤，既简单又炫酷！
- **Mall**电商实战项目全面升级！支持最新版**SpringBoot**，干掉循环依赖**...**
- 重磅更新！**Mall**实战教程全面升级，瞬间高大上了！
- **40K+Star**！**Mall**电商实战项目开源回忆录！

macrozheng

专注Java技术分享，解析优质开源项目。涵盖SpringBoot、SpringCloud、Docker...

240篇原创内容

公众号

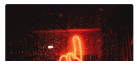
阅读原文

喜欢此内容的人还喜欢

介绍 Preact Signals

前端小馆

学习笔记-SQLSERVER的LATCH



白鳝的洞穴

探索组件在线预览和调试

政采云前端