1. 摘要
2. Spring Cloud Ribbon 是Spring Cloud Netflix 子项目的核心组件之一，主要给服务间调用及API网关转发提供负载均衡的功能。
3. Ribbon简介
4. 在微服务架构中，很多服务都会部署多个，其他服务去调用该服务的时候，如何保证负载均衡是个不得不去考虑的问题。负载均衡可以增加系统的可用性和扩展性，当我们使用RestTemplate来调用其他服务时，Ribbon可以很方便的实现负载均衡功能。

三、RestTemplate的使用

1、RestTemplate介绍：RestTemplate是一个HTTP客户端，使用它我们可以方便的调用HTTP接口，支持GET、POST、PUT、DELETE等方法。

2、GET请求方法：

@Nullable  
<T> T getForObject(String var1, Class<T> var2, Object... var3) throws RestClientException;  
  
@Nullable  
<T> T getForObject(String var1, Class<T> var2, Map<String, ?> var3) throws RestClientException;  
  
@Nullable  
<T> T getForObject(URI var1, Class<T> var2) throws RestClientException;  
  
<T> ResponseEntity<T> getForEntity(String var1, Class<T> var2, Object... var3) throws RestClientException;  
  
<T> ResponseEntity<T> getForEntity(String var1, Class<T> var2, Map<String, ?> var3) throws RestClientException;  
  
<T> ResponseEntity<T> getForEntity(URI var1, Class<T> var2) throws RestClientException;

3、getForObject方法

①返回对象为响应体中数据转化成的对象，举例如下：

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

4、getForEntity方法

①返回对象为ResponseEntity对象，包含了响应中的一些重要信息，比如响应头、响应状态码、响应体等，举例如下：

@GetMapping("/getEntityByUsername")  
public CommonResult getEntityByUsername(@RequestParam String username) {  
 ResponseEntity<CommonResult> entity = restTemplate.getForEntity(userServiceUrl + "/user/getByUsername?username={1}", CommonResult.class, username);  
 if (entity.getStatusCode().is2xxSuccessful()) {  
 return entity.getBody();  
 } else {  
 return new CommonResult("操作失败", 500);  
 }  
}

1. POST请求方法

@Nullable  
URI postForLocation(String var1, @Nullable Object var2, Object... var3) throws RestClientException;  
  
@Nullable  
URI postForLocation(String var1, @Nullable Object var2, Map<String, ?> var3) throws RestClientException;  
  
@Nullable  
URI postForLocation(URI var1, @Nullable Object var2) throws RestClientException;  
  
@Nullable  
<T> T postForObject(String var1, @Nullable Object var2, Class<T> var3, Object... var4) throws RestClientException;  
  
@Nullable  
<T> T postForObject(String var1, @Nullable Object var2, Class<T> var3, Map<String, ?> var4) throws RestClientException;  
  
@Nullable  
<T> T postForObject(URI var1, @Nullable Object var2, Class<T> var3) throws RestClientException;  
  
<T> ResponseEntity<T> postForEntity(String var1, @Nullable Object var2, Class<T> var3, Object... var4) throws RestClientException;  
  
<T> ResponseEntity<T> postForEntity(String var1, @Nullable Object var2, Class<T> var3, Map<String, ?> var4) throws RestClientException;  
  
<T> ResponseEntity<T> postForEntity(URI var1, @Nullable Object var2, Class<T> var3) throws RestClientException;

6、postForObject示例

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

7、postForEntity示例

@PostMapping("/create")  
public CommonResult create(@RequestBody User user) {  
 return restTemplate.postForEntity(userServiceUrl + "/user/create", user, CommonResult.class).getBody();  
}

8、PUT请求方法

void put(String var1, @Nullable Object var2, Object... var3) throws RestClientException;  
  
void put(String var1, @Nullable Object var2, Map<String, ?> var3) throws RestClientException;  
  
void put(URI var1, @Nullable Object var2) throws RestClientException;

1. PUT请求示例

@PutMapping("/update")  
public CommonResult update(@RequestBody User user) {  
 restTemplate.put(userServiceUrl + "/user/update", user);  
 return new CommonResult("操作成功",200);  
}

10、DELETE请求方法

void delete(String var1, Object... var2) throws RestClientException;  
  
void delete(String var1, Map<String, ?> var2) throws RestClientException;  
  
void delete(URI var1) throws RestClientException;

11、DELETE请求示例

@DeleteMapping("/delete/{id}")  
public CommonResult delete(@PathVariable Long id) {  
 restTemplate.delete(userServiceUrl + "/user/delete/{1}", null, id);  
 return new CommonResult("操作成功",200);  
}

四、创建一个user-service模块

注意：首先我们创建一个user-service，用于给Ribbon提供服务调用。

1、在pom.xml中添加相关依赖

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

2、在application.yml进行配置

注意：主要是配置了端口和注册中心地址。

server:  
 port: 8201 *#运行端口*spring:  
 application:  
 name: user-service  
eureka:  
 client:  
 register-with-eureka: true *#注册到Eureka注册中心* fetch-registry: true *#获取注册服务列表* service-url:  
 defaultZone: http://localhost:8001/eureka/ *#指定注册中心地址*

3、添加UserController用于提供调用接口

注意：UserController类定义了对User对象常见的CRUD接口。

1. 创建一个ribbon-service模块

注意：这里我们创建一个ribbon-service模块来调用user-service模块演示负载均衡的服务调用。

1、在pom.xml中添加相关依赖

<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-netflix-ribbon</artifactId>  
</dependency>  
  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>

2、在application.yml进行配置

①主要是配置了端口、注册中心地址及user-service的调用路径。

server:  
 port: 8301  
spring:  
 application:  
 name: ribbon-service  
eureka:  
 client:  
 register-with-eureka: true *#将自己注册到Eureka注册中心* fetch-registry: true *#获取注册服务列表* service-url:  
 defaultZone: http://localhost:8001/eureka/ *#指定注册中心地址*service-url:  
 user-service: http://user-service

3、使用@LoadBalanced注解赋予RestTemplate负载均衡的能力

①可以看出使用Ribbon的负载均衡功能非常简单，和直接使用RestTemplate没什么两样，只需给RestTemplate添加一个@LoadBalanced即可。

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

4、添加UserRibbonController类

①注入RestTemplate，使用其调用user-service中提供的相关接口，这里对GET和POST调用进行了演示，其他方法调用均可参考。

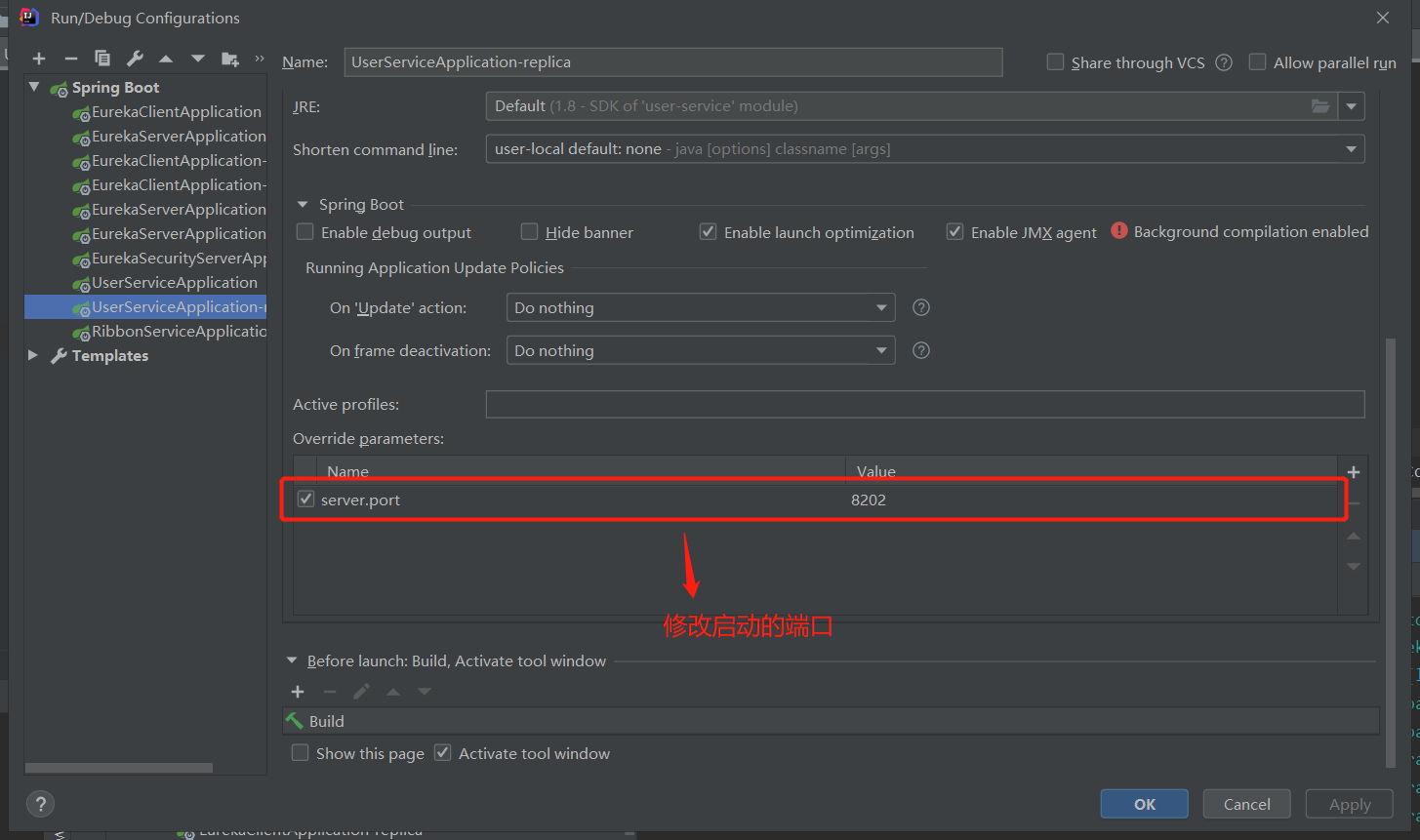
六、负载均衡功能演示

1、启动eureka-server于8001端口；

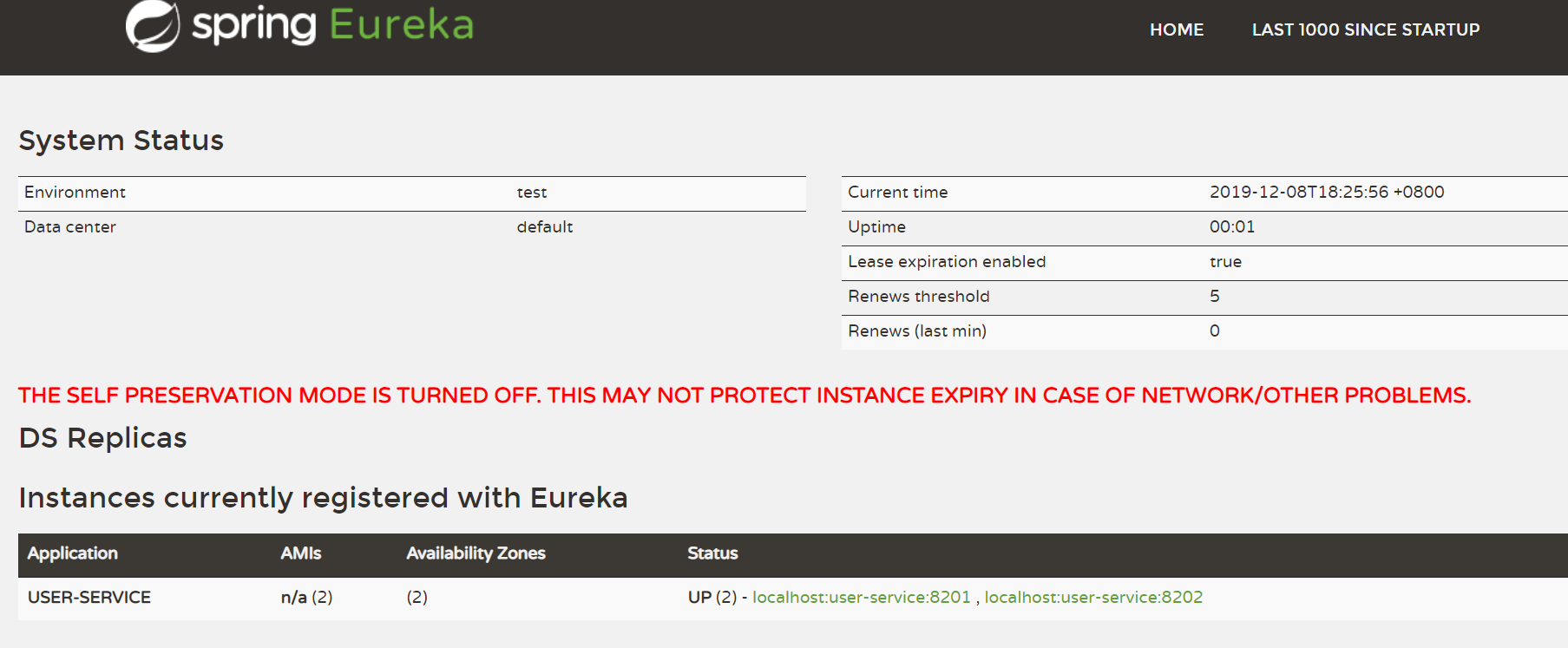
2、启动user-service于8201端口；

3、启动另一个user-service于8202端口，可以通过修改IDEA中的SpringBoot的启动配置

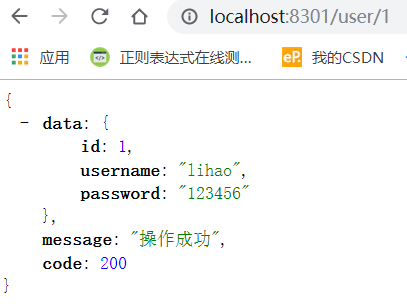
实现：



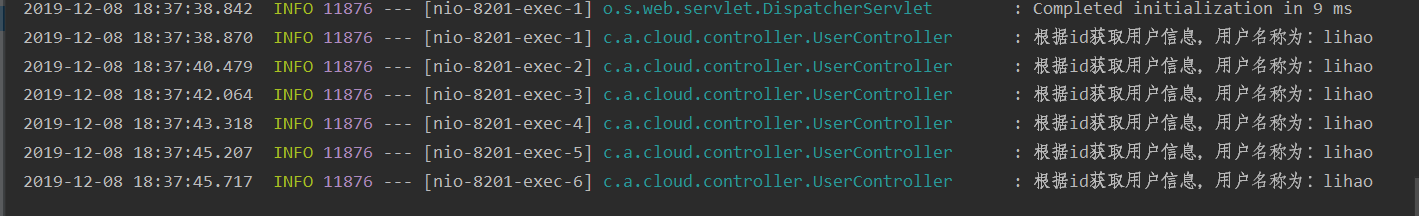
4、然后启动几个应用，访问localhost:8001此时运行中的服务如下：

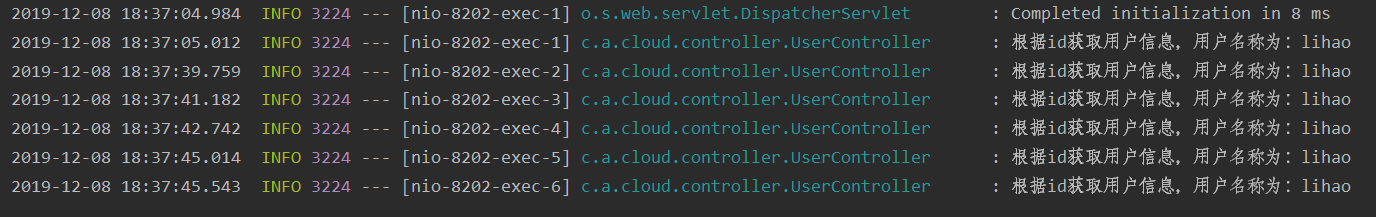


5、调用接口进行测试：[http://localhost:8301/user/1](http://localhost:8301/user/1" \t "https://juejin.im/post/_blank)



6、可以发现运行在8201和8202的user-service控制台交替打印如下信息：





七、Ribbon的常用配置

1、全局配置：

*#全局配置*ribbon:  
 ConnectTimeout: 1000 *#服务请求连接超时时间（毫秒）* ReadTimeout: 3000 *#服务请求处理超时时间（毫秒）* OkToRetryOnAllOperations: true *#对超时请求启用重试机制* MaxAutoRetriesNextServer: 1 *#切换重试实例的最大个数* MaxAutoRetries: 1 *# 切换实例后重试最大次数* NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule *#修改负载均衡算法*

1. 指定服务进行配置：

①与全局配置的区别就是ribbon节点挂在服务名称下面，如下是对ribbon-service调用user-service时的单独配置：

*#指定服务进行配置*user-service: *#服务名称* ribbon:  
 ConnectTimeout: 1000 *#服务请求连接超时时间（毫秒）* ReadTimeout: 3000 *#服务请求处理超时时间（毫秒）* OkToRetryOnAllOperations: true *#对超时请求启用重试机制* MaxAutoRetriesNextServer: 1 *#切换重试实例的最大个数* MaxAutoRetries: 1 *# 切换实例后重试最大次数* NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule *#修改负载均衡算法*

八、Ribbon的负载均衡策略

注意：所谓的负载均衡策略，就是当A服务调用B服务时，此时B服务有多个实例，这时A服务以何种方式来选择调用的B实例，ribbon可以选择以下几种负载均衡策略。

1. com.netflix.loadbalancer.RandomRule：从提供服务的实例中以随机的方式；
2. com.netflix.loadbalancer.RoundRobinRule：以线性轮询的方式，就是维护一个计数器，

从提供服务的实例中按顺序选取，第一次选第一个，第二次选第二个，以此类推，到最后一个以后再从头来过；

1. com.netflix.loadbalancer.RetryRule：在RoundRobinRule的基础上添加重试机制，即在指

定的重试时间内，反复使用线性轮询策略来选择可用实例；

1. com.netflix.loadbalancer.WeightedResponseTimeRule：对RoundRobinRule的扩展，响应

速度越快的实例选择权重越大，越容易被选择；

5、com.netflix.loadbalancer.BestAvailableRule：选择并发较小的实例；

6、com.netflix.loadbalancer.AvailabilityFilteringRule：先过滤掉故障实例，再选择并发较小

的实例；

7、com.netflix.loadbalancer.ZoneAwareLoadBalancer：采用双重过滤，同时过滤不是同一区域的实例和故障实例，选择并发较小的实例。

九、使用到的模块

Spring-cloud-learning

├── eureka-server -- eureka注册中心

├── user-service -- 提供User对象CRUD接口的服务

├── ribbon-service -- ribbon服务调用测试服务