1. 概述

1.1. 是什么

Spring Cloud Ribbon是基于 Netflix Ribbon实现的一套客户端负载均衡的工具

简单的说，Ribbon是 Netfliix发布的开源项目，主要功能是提供客户端的软件负载均衡算法和服务调用。

Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出 Load balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们很容易使用 Ribbon实现自定义的负载均衡算法。

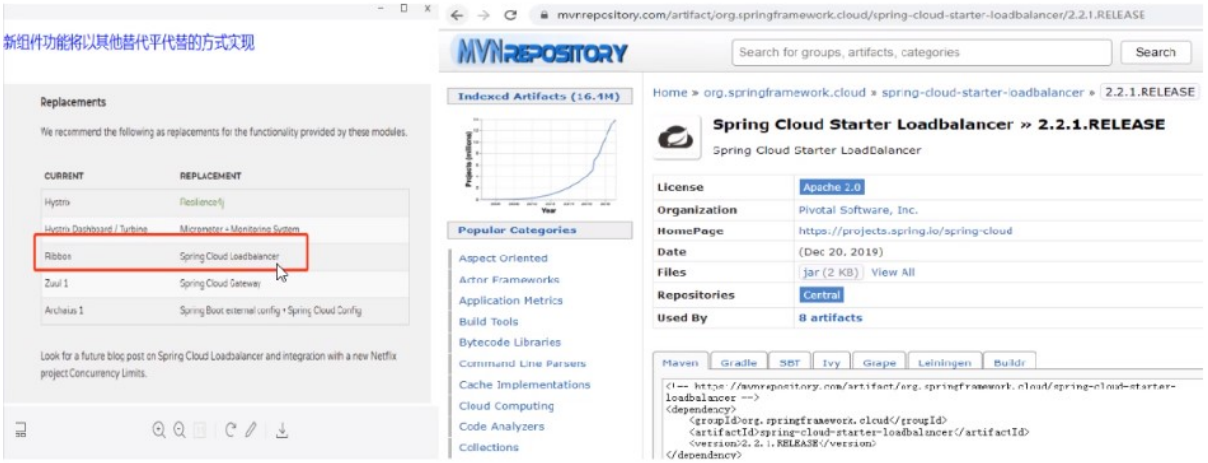
1.2. 官网资料

官网地址: https://github.com/Netflix/ribbon/wiki/Getting-Started

Ribbon目前也进入维护模式

> ## Project Status: On Maintenance
>
> Ribbon comprises of multiple components some of which are used in production internally and some of which were replaced by non-OSS solutions over time. This is because Netflix started moving into a more componentized architecture for RPC with a focus on single-responsibility modules. So each Ribbon component gets a different level of attention at this moment.
>
> More specifically, here are the components of Ribbon and their level of attention by our teams:
>
> - ribbon-core: **deployed at scale in production**
> - ribbon-eureka: **deployed at scale in production**
> - ribbon-evcache: **not used**
> - ribbon-guice: **not used**
> - ribbon-httpclient: **we use everything not under com.netflix.http4.ssl. Instead, we use an internal solution developed by our cloud security team**
> - ribbon-loadbalancer: **deployed at scale in production**
> - ribbon-test: **this is just an internal integration test suite**
> - ribbon-transport: **not used**
> - ribbon: **not used**

未来替代方案



1.3. 能干啥

1.3.1. LB(负载均衡)

LB负载均衡（Load Balance)是什么

- 简单的说就是将用户的请求平摊的分配到多个服务上，从而达到系统的HA（高可用）。
- 常见的负载均衡有软件 Nginx，LVS，硬件F5等

Ribbon本地负载均客户端 VS Nginx服务端负载均衡区别

- Nginx是服务器负载均衡，客户端所有请求都会交给 nginx，然后由 nginx实现转发请求。即负载均衡是由服务端实现的。
- Ribbon本地负载均衡，在调用微服务接口时候，会在注册中心上获取注册信息服务列表之后缓存到JVM本地，从而在本地实现RPC远服务调用技术

1.3.1.1. 集中式LB

集中式LB

- 即在服务的消费方和提供方之间使用独立的LB设施(可以是硬件，如F5，也可以是软件，如Nginx），由该设施负责把访问请求通过某种策略转发至服务的提供方；

1.3.1.2. 进程内LB

进程内LB

- 将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器
- Ribbon就属于进程内LB，它只是个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址

1.3.2. 前面我们讲解过了80通过轮询负载访问8001/8002

```java
package cn.sitedev.springcloud.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class ApplicationContextConfig {

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

```java
package cn.sitedev.springcloud.controller;

import cn.sitedev.springcloud.entities.CommonResult;
import cn.sitedev.springcloud.entities.Payment;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;
```

```java

import javax.annotation.Resource;

@RestController
@RequestMapping("/consumer/payment")
@Slf4j
public class OrderController {

//    public static final String PAYMENT_URL = "http://localhost:8001";

    // 通过在 eureka上注册过的微服务名称调用
    public static final String PAYMENT_URL = "http://CLOUD-PAYMENT-SERVICE";

    @Resource
    private RestTemplate restTemplate;

    @PostMapping(value = "/create")
    public CommonResult<Payment> create(Payment payment) {
        return restTemplate.postForObject(PAYMENT_URL + "/payment/create", payment,
    }

    @GetMapping(value = "/get/{id}")
    public CommonResult<Payment> getPayment(@PathVariable("id") Long id) {
        return restTemplate.getForObject(PAYMENT_URL + "/payment/get/" + id, CommonR
    }
}
```

浏览器访问http://localhost/consumer/payment/get/1

浏览器再次访问http://localhost/consumer/payment/get/1

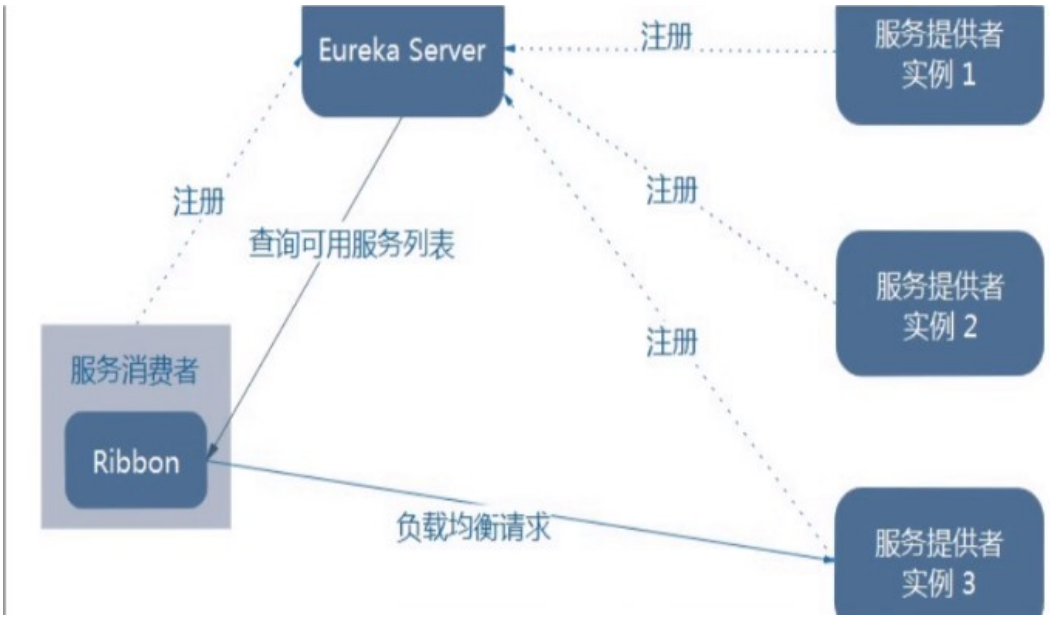
```
JSON  元数据  格式化  视图化  压缩  选中
▼ object {3}
    code : 200
    message : 查询成功, serverPort: 8002
  ▼ data {2}
      id   : 1
      serial : 测试支付模块接口
```


1.3.3. 一句话总结

负载均衡+RestTemplate调用

2. Ribbon负载均衡演示

2.1. 架构说明



Ribbon在工作时分成两步

- 第一步先选择 EurekaServer，它优先选择在同个区域内负载较少的 server.
- 第二步再根据用户指定的策略，再从 server取到的服务注册列表中选择一个地址
- 其中 Ribbon提供了多种策略：比如轮询、随机和根据响应时间加权。

总结: Ribbon其实就是一个软负载均衡的客户端组件, 他可以和其他所需请求的客户端结合使用, 和eureka结合只是其中一个实例.

2.2. POM

之前写样例时候没有引入 spring-cloud-starter-ribbon也可以使用 ribbon

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

猜测 spring-cloud-starter-netflix-eureka-client自带了 spring-cloud-starter-ribbon引用

证明如下：可以看到 spring-cloud-starter-netflix-eureka-client确实引入了 Ribbon

2.3. 二说RestTemplate的使用

2.3.1. 官网

API文档参见: https://docs.spring.io/spring-framework/docs/5.2.2.RELEASE/javadoc-api/org/springframework/web/client/RestTemplate.html

Return the error handler.

<T> ResponseEntity<T>	getForEntity(String url, Class<T> responseType, Map<String,?> uriVariables) Retrieve a representation by doing a GET on the URI template.
<T> ResponseEntity<T>	getForEntity(String url, Class<T> responseType, Object... uriVariables) Retrieve an entity by doing a GET on the specified URL.
<T> ResponseEntity<T>	getForEntity(URI url, Class<T> responseType) Retrieve a representation by doing a GET on the URL .
<T> T	getForObject(String url, Class<T> responseType, Map<String,?> uriVariables) Retrieve a representation by doing a GET on the URI template.
<T> T	getForObject(String url, Class<T> responseType, Object... uriVariables) Retrieve a representation by doing a GET on the specified URL.
<T> T	getForObject(URI url, Class<T> responseType) Retrieve a representation by doing a GET on the URL .

2.3.2. getForObject方法/getForEntity方法

2.3.2.1. getForObject

返回对象为响应体中数据转化成的对象，基本上可以理解为Json

```
@GetMapping(value = "/get/{id}")
public CommonResult<Payment> getPayment(@PathVariable("id") Long id) {
    return restTemplate.getForObject(PAYMENT_URL + "/payment/get/" + id, CommonR
}
```

浏览器访问 http://localhost/consumer/payment/get/1

JSON 元数据 格式化 视图化 压缩 选中

```
▼ object {3}
    code : 200
    message : 查询成功, serverPort: 8001
  ▼ data {2}
      id   : 1
      serial : 测试支付模块接口
```

2.3.2.2. getForEntity方法

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

```
@GetMapping(value = "/getForEntity/{id}")
public CommonResult<Payment> getPayment2(@PathVariable("id") Long id) {
    ResponseEntity<CommonResult> entity = restTemplate.getForEntity(PAYMENT_URL
            CommonResult.class);
```

```
        if (entity.getStatusCode().is2xxSuccessful()) {
            return entity.getBody();
        } else {
            return new CommonResult<>(444, "操作失败");
        }
    }
```

浏览器访问 http://localhost/consumer/payment/getForEntity/1

2.3.3. postForObject/postForEntity

2.3.3.1. postForObject

```
    @PostMapping(value = "/create")
    public CommonResult<Payment> create(Payment payment) {
        return restTemplate.postForObject(PAYMENT_URL + "/payment/create", payment,
    }
```

2.3.3.2. postForEntity

```
    @PostMapping(value = "/createForEntity")
    public CommonResult<Payment> create2(Payment payment) {
        return restTemplate.postForEntity(PAYMENT_URL + "/payment/create", payment,
    }
```

2.3.4. GET请求方法

在RestTemplate中，GET请求可以通过如下两类方法来发起：

```
Search for: get                    RestTemplate.class
☑ Inherited members (Ctrl+F3)  ☐ Anonymous Classes (Ctrl+I)  ☐ Lambdas (Ctrl+L)
RestTemplate
    getClass(): Class<?> →Object
    getClientHttpRequestInitializers(): List<ClientHttpRequestInitializer> →HttpAccessor
    getErrorHandler(): ResponseErrorHandler
    getForEntity(String, Class<T>, Map<String, ?>): ResponseEntity<T> ↑RestOperations
    getForEntity(String, Class<T>, Object...): ResponseEntity<T> ↑RestOperations
    getForEntity(URI, Class<T>): ResponseEntity<T> ↑RestOperations
    getForObject(String, Class<T>, Map<String, ?>): T ↑RestOperations
    getForObject(String, Class<T>, Object...): T ↑RestOperations
    getForObject(URI, Class<T>): T ↑RestOperations
    getInterceptors(): List<ClientHttpRequestInterceptor> →InterceptingHttpAccessor
    getMessageConverters(): List<HttpMessageConverter<?>>
    getRequestFactory(): ClientHttpRequestFactory →InterceptingHttpAccessor
    getUriTemplateHandler(): UriTemplateHandler
    AcceptHeaderRequestCallback
        getSupportedMediaTypes(HttpMessageConverter<?>): Stream<MediaType>
```

2.3.5. POST请求方法

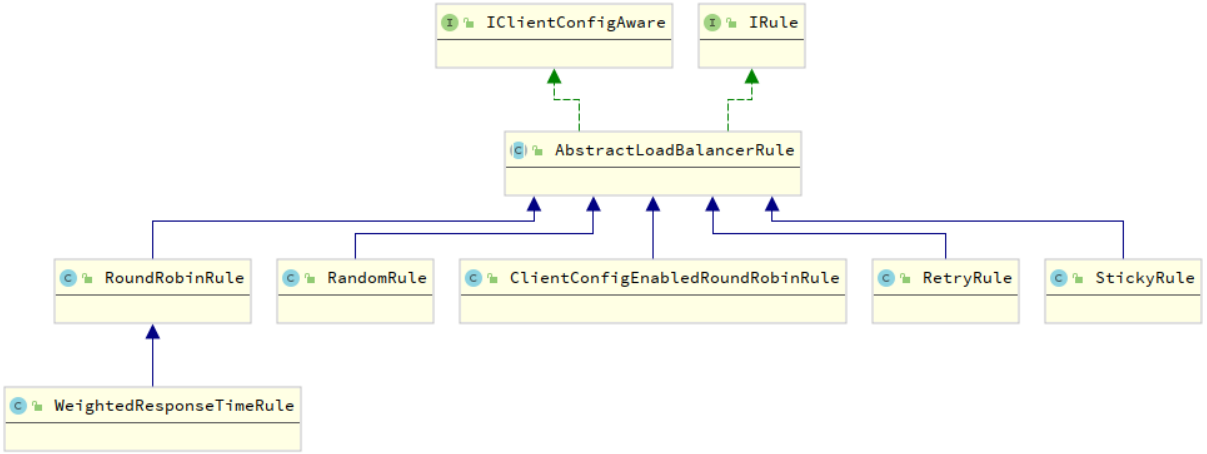
在RestTemplate中，POST请求可以通过如下三类方法来发起：

```
Search for: post                   RestTemplate.class
☑ Inherited members (Ctrl+F3)  ☐ Anonymous Classes (Ctrl+I)  ☐ Lambdas
RestTemplate
    postForEntity(String, Object, Class<T>, Map<String, ?>): ResponseEn
    postForEntity(String, Object, Class<T>, Object...): ResponseEntity
    postForEntity(URI, Object, Class<T>): ResponseEntity<T> ↑RestOpera
    postForLocation(String, Object, Map<String, ?>): URI ↑RestOperatio
    postForLocation(String, Object, Object...): URI ↑RestOperations
    postForLocation(URI, Object): URI ↑RestOperations
    postForObject(String, Object, Class<T>, Map<String, ?>): T ↑RestOp
    postForObject(String, Object, Class<T>, Object...): T ↑RestOperati
    postForObject(URI, Object, Class<T>): T ↑RestOperations
```

3. Ribbon核心组件IRule

3.1. IRule

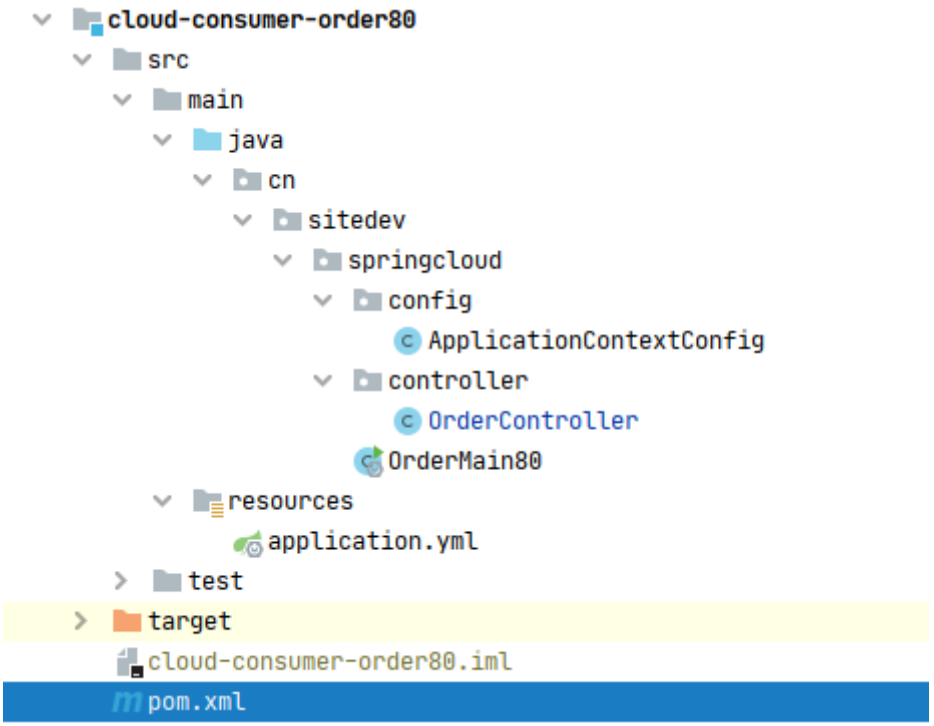
IRule:根据特定算法从服务列表中选取一个要访问的服务



- `com.netflix.loadbalancer.RoundRobinRule`：轮询
- `com.netflix.loadbalancer.RandomRule`：随机
- `com.netflix.loadbalancer.RetryRule`：先按照RoundRobinRule的策略获取服务,如果获取服务失败则在指定时间内进行重试,获取可用的服务
- `WeightedResponseTimeRule`：对RoundRobinRule的扩展,响应速度越快的实例选择权重越多大,越容易被选择
- `BestAvailableRule`：会先过滤掉由于多次访问故障而处于断路器跳闸状态的服务,然后选择一个并发量最小的服务
- `AvailabilityFilteringRule`：先过滤掉故障实例,再选择并发较小的实例
- `ZoneAvoidanceRule`：默认规则,复合判断server所在区域的性能和server的可用性选择服务器

3.2. 如何替换

3.2.1. 修改cloud-consumer-order80



3.2.2. 注意配置细节

官方文档明确给出了警告

这个自定义配置类不能放在@ComponentScan所扫描的当前包下以及子包下

否则我们自定义的这个配置类就会被所有的 Ribbon客户端所共享，达不到特殊化定制的目的了

> https://cloud.spring.io/spring-cloud-static/spring-cloud-netflix/2.2.0.RELEASE/reference/html/#customizing-the-ribbon-client

7.2. Customizing the Ribbon Client

You can configure some bits of a Ribbon client by using external properties in `<client>.ribbon.*`, which is similar to using the Netflix APIs natively, except that you can use Spring Boot configuration files. The native options can be inspected as static fields in `CommonClientConfigKey` (part of ribbon-core).

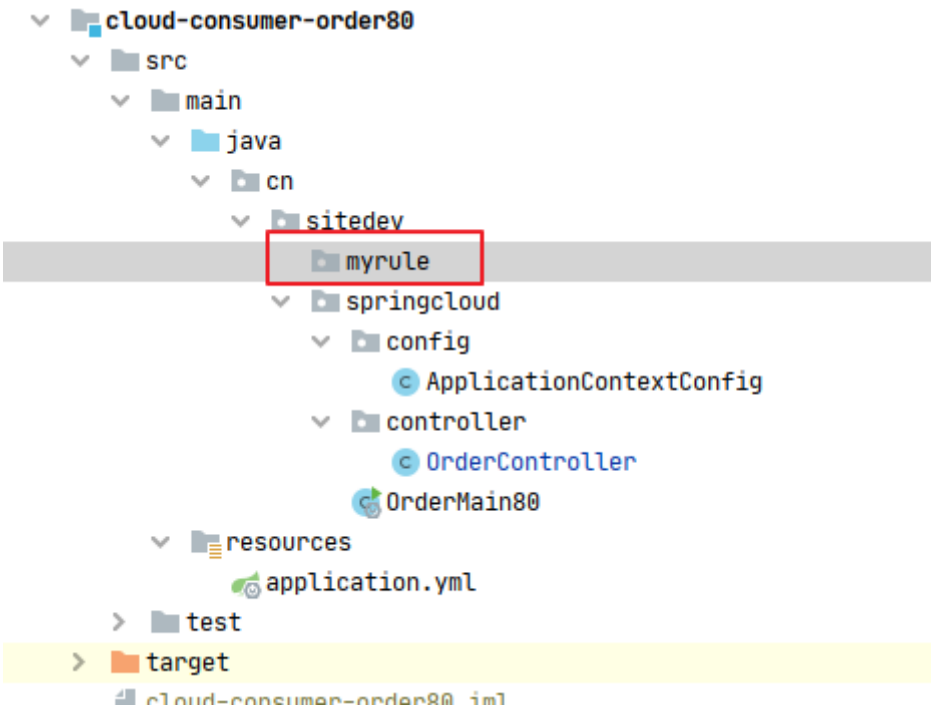
Spring Cloud also lets you take full control of the client by declaring additional configuration (on top of the `RibbonClientConfiguration`) using `@RibbonClient`, as shown in the following example:

```java
@Configuration
@RibbonClient(name = "custom", configuration = CustomConfiguration.class)
public class TestConfiguration {
}
```

In this case, the client is composed from the components already in `RibbonClientConfiguration`, together with any in `CustomConfiguration` (where the latter generally overrides the former).

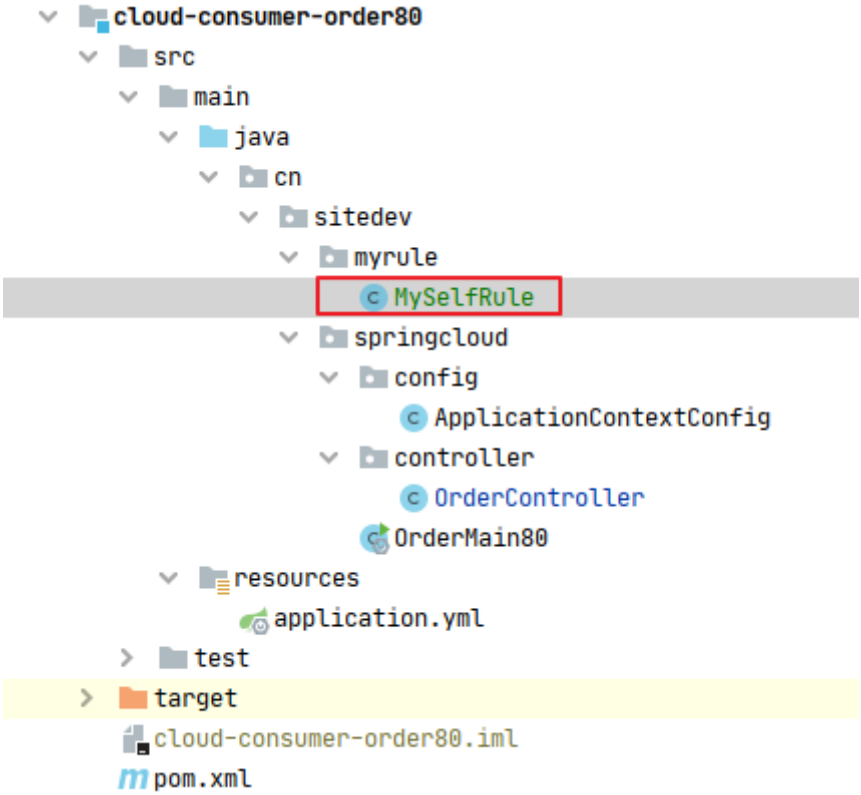
> The `CustomConfiguration` clas must be a `@Configuration` class, but take care that it is not in a `@ComponentScan` for the main application context. Otherwise, it is shared by all the `@RibbonClients`. If you use `@ComponentScan` (or `@SpringBootApplication`), you need to take steps to avoid it being included (for instance, you can put it in a separate, non-overlapping package or specify the packages to scan explicitly in the `@ComponentScan`).

3.2.3. 新建自定义rule专用的package



3.2.4. 在该包下新建MySelfRule规则类

```
package cn.sitedev.myrule;

import com.netflix.loadbalancer.IRule;
import com.netflix.loadbalancer.RandomRule;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * 自定义负载均衡路由规则类
 **/
@Configuration
public class MySelfRule {
    @Bean
    public IRule myRule() {
        // 定义为随机
        return new RandomRule();
    }
}
```



3.2.5. 主启动类添加@RibbonClient注解

```
package cn.sitedev.springcloud;

import cn.sitedev.myrule.MySelfRule;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;

@SpringBootApplication
@EnableEurekaClient
@RibbonClient(name = "CLOUD-PAYMENT-SERVICE", configuration = MySelfRule.class)
public class OrderMain80 {
    public static void main(String[] args) {
        SpringApplication.run(OrderMain80.class, args);
    }
}
```

3.2.6. 测试

浏览器访问 http://localhost/consumer/payment/get/1

JSON 元数据 格式化 视图化 压缩 选中

```
1 {"code":200,"message":"查询成功, serverPort: 8001","data":{"id":1,"serial":"测试支付模块接口"}}
```

JSON 元数据 格式化 视图化 压缩 选中

```
1 {"code":200,"message":"查询成功, serverPort: 8001","data":{"id":1,"serial":"测试支付模块接口"}}
```

JSON 元数据 格式化 视图化 压缩 选中

```
1 {"code":200,"message":"查询成功, serverPort: 8001","data":{"id":1,"serial":"测试支付模块接口"}}
```

JSON 元数据 格式化 视图化 压缩 选中

```
1 {"code":200,"message":"查询成功, serverPort: 8002","data":{"id":1,"serial":"测试支付模块接口"}}
```

JSON 元数据 格式化 视图化 压缩 选中

```
1 {"code":200,"message":"查询成功, serverPort: 8002","data":{"id":1,"serial":"测试支付模块接口"}}
```

JSON 元数据 格式化 视图化 压缩 选中

```
1 {"code":200,"message":"查询成功, serverPort: 8002","data":{"id":1,"serial":"测试支付模块接口"}}
```

4. Ribbon负载均衡算法

4.1. 原理

负载均衡算法：rest接口第几次请求数%服务器集群总数量=实际调用服务器位置下标，每次服务重启动后rest接口计数从1开始。

```
1 List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-PAYMENT-SERVIC
```

如：

```
1 List[0] instances = 127.0.0.1:8002
2 List[1] instances = 127.0.0.1:8001
```

8001 + 8002合成为集群，它们共计2台机器，集群总数为2，按轮询算法原理:

当总请求数为1时：1%2=1对应下标位置为1，则获得服务地址为127.0.0.1:8001

当总请求数位2时：2%2=0对应下标位置为0，则获得服务地址为127.0.0.1:8002

当总请求数位3时：3%2=1对应下标位置为1，则获得服务地址为127.0.0.1:8001

当总请求数位4时：4%2=0对应下标位置为0，则获得服务地址为127.0.0.1:8002

如此类推...

负载均衡算法：rest接口第几次请求数 % 服务器集群总数量 = 实际调用服务器位置下标 ，每次服务重启动后rest接口计数从1开始。

```
List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-PAYMENT-SERVICE");
```

如：　List [0] instances = 127.0.0.1:8002
　　List [1] instances = 127.0.0.1:8001

8001+ 8002 组合成为集群，它们共计2台机器，集群总数为2， 按照轮询算法原理：

当总请求数为1时： 1 % 2 =1 对应下标位置为1 ，则获得服务地址为127.0.0.1:8001
当总请求数位2时： 2 % 2 =0 对应下标位置为0 ，则获得服务地址为127.0.0.1:8002
当总请求数位3时： 3 % 2 =1 对应下标位置为1 ，则获得服务地址为127.0.0.1:8001
当总请求数位4时： 4 % 2 =0 对应下标位置为0 ，则获得服务地址为127.0.0.1:8002
如此类推.....

4.2. 源码

以RoundRobinRule为例:

```java
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

package com.netflix.loadbalancer;

import com.netflix.client.config.IClientConfig;
import java.util.List;
import java.util.concurrent.atomic.AtomicInteger;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class RoundRobinRule extends AbstractLoadBalancerRule {
    private AtomicInteger nextServerCyclicCounter;
    private static final boolean AVAILABLE_ONLY_SERVERS = true;
    private static final boolean ALL_SERVERS = false;
    private static Logger log = LoggerFactory.getLogger(RoundRobinRule.class);

    public RoundRobinRule() {
        this.nextServerCyclicCounter = new AtomicInteger(0);
    }

    public RoundRobinRule(ILoadBalancer lb) {
        this();
```

```
26          this.setLoadBalancer(lb);
27      }
28
29      public Server choose(ILoadBalancer lb, Object key) {
30          if (lb == null) {
31              log.warn("no load balancer");
32              return null;
33          } else {
34              Server server = null;
35              int count = 0;
36
37              while(true) {
38                  if (server == null && count++ < 10) {
39                      List<Server> reachableServers = lb.getReachableServers();
40                      List<Server> allServers = lb.getAllServers();
41                      int upCount = reachableServers.size();
42                      int serverCount = allServers.size();
43                      if (upCount != 0 && serverCount != 0) {
44                          int nextServerIndex = this.incrementAndGetModulo(serverCount
45                          server = (Server)allServers.get(nextServerIndex);
46                          if (server == null) {
47                              Thread.yield();
48                          } else {
49                              if (server.isAlive() && server.isReadyToServe()) {
50                                  return server;
51                              }
52
53                              server = null;
54                          }
55                          continue;
56                      }
57
58                      log.warn("No up servers available from load balancer: " + lb);
59                      return null;
60                  }
61
62                  if (count >= 10) {
63                      log.warn("No available alive servers after 10 tries from load ba
64                  }
65
66                  return server;
67              }
68          }
```

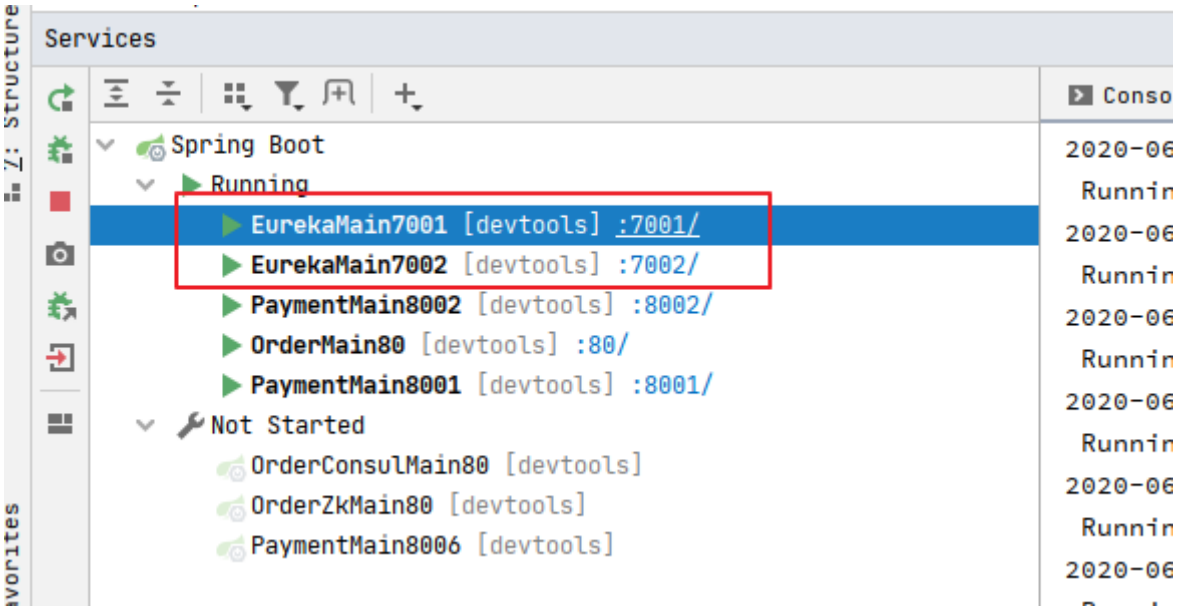
```
    }

    private int incrementAndGetModulo(int modulo) {
        int current;
        int next;
        do {
            current = this.nextServerCyclicCounter.get();
            next = (current + 1) % modulo;
        } while(!this.nextServerCyclicCounter.compareAndSet(current, next));

        return next;
    }

    public Server choose(Object key) {
        return this.choose(this.getLoadBalancer(), key);
    }

    public void initWithNiwsConfig(IClientConfig clientConfig) {
    }
}
```

4.3. 手写

自己试着手写一个本地负载均衡器

4.3.1. 7001/7002集群启动

启动7001/7002服务



4.3.2. 8001/8002微服务改造

修改8001的Controller

```
package cn.sitedev.springcloud.controller;

import cn.sitedev.springcloud.entities.CommonResult;
import cn.sitedev.springcloud.entities.Payment;
import cn.sitedev.springcloud.service.PaymentService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.List;

@RestController
@Slf4j
@RequestMapping("/payment")
public class PaymentController {
    @Resource
    private PaymentService paymentService;

    /**
     * 端口号
     * 查看负载均衡效果
     */
    @Value("${server.port}")
    private String serverPort;

    /**
     * 服务发现，获取服务信息
     */
    @Resource
    private DiscoveryClient discoveryClient;

    @PostMapping(value = "/create")
    public CommonResult create(@RequestBody Payment payment) {
        int result = paymentService.create(payment);
        log.info("*****插入结果: " + result);
        if (result > 0) {
```

```
            return new CommonResult(200, "插入数据库成功, serverPort: " + serverPort
        } else {
            return new CommonResult(444, "插入数据库失败", null);
        }
    }

    @GetMapping(value = "/get/{id}")
    public CommonResult getPaymentById(@PathVariable("id") Long id) {
        Payment payment = paymentService.getPaymentById(id);
        log.info("*****查询结果: " + payment);
        if (payment != null) {
            return new CommonResult(200, "查询成功, serverPort: " + serverPort, paym
        } else {
            return new CommonResult(444, "没有对应记录, 查询id: " + id, null);
        }
    }

    /**
     * 服务发现
     *
     * @return
     */
    @GetMapping("/discovery")
    public Object discovery() {
        List<String> services = discoveryClient.getServices();
        for (String element : services) {
            log.info("*****element: " + element);
        }
        // 一个微服务下的全部实例
        List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-PAYMEN
        for (ServiceInstance instance : instances) {
            log.info(instance.getServiceId() + "\t" + instance.getHost() + "\t" + in
        }
        return discoveryClient;
    }

    @GetMapping(value = "/lb")
    public String getPaymentLB() {
        return serverPort;
    }
}
```

修改8002的Controller

```java
package cn.sitedev.springcloud.controller;

import cn.sitedev.springcloud.entities.CommonResult;
import cn.sitedev.springcloud.entities.Payment;
import cn.sitedev.springcloud.service.PaymentService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.List;

@RestController
@Slf4j
@RequestMapping("/payment")
public class PaymentController {
    @Resource
    private PaymentService paymentService;

    /**
     * 端口号
     * 查看负载均衡效果
     */
    @Value("${server.port}")
    private String serverPort;

    /**
     * 服务发现，获取服务信息
     */
    @Resource
    private DiscoveryClient discoveryClient;

    @PostMapping(value = "/create")
    public CommonResult create(@RequestBody Payment payment) {
        int result = paymentService.create(payment);
        log.info("*****插入结果: " + result);
        if (result > 0) {
            return new CommonResult(200, "插入数据库成功, serverPort: " + serverPort
        } else {
            return new CommonResult(444, "插入数据库失败", null);
```

```
        }
    }

    @GetMapping(value = "/get/{id}")
    public CommonResult getPaymentById(@PathVariable("id") Long id) {
        Payment payment = paymentService.getPaymentById(id);
        log.info("*****查询结果: " + payment);
        if (payment != null) {
            return new CommonResult(200, "查询成功, serverPort: " + serverPort, paym
        } else {
            return new CommonResult(444, "没有对应记录, 查询id: " + id, null);
        }
    }

    /**
     * 服务发现
     *
     * @return
     */
    @GetMapping("/discovery")
    public Object discovery() {
        List<String> services = discoveryClient.getServices();
        for (String element : services) {
            log.info("*****element: " + element);
        }
        // 一个微服务下的全部实例
        List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-PAYMEN
        for (ServiceInstance instance : instances) {
            log.info(instance.getServiceId() + "\t" + instance.getHost() + "\t" + in
        }
        return discoveryClient;
    }

    @GetMapping(value = "/lb")
    public String getPaymentLB() {
        return serverPort;
    }

}
```

4.3.3. 80订单微服务改造

4.3.3.1. ApplicationContextBean去掉注解@LoadBalanced

```java
package cn.sitedev.springcloud.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class ApplicationContextConfig {

//    @LoadBalanced // 暂时去除该注解,以测试自定义的负载均衡器
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

4.3.3.2. LoadBalancer接口

```java
package cn.sitedev.springcloud.lb;

import org.springframework.cloud.client.ServiceInstance;

import java.util.List;

public interface LoadBalancer {
    ServiceInstance instances(List<ServiceInstance> serviceInstances);
}
```

4.3.3.3. MyLB

```java
package cn.sitedev.springcloud.lb;

import org.springframework.cloud.client.ServiceInstance;
import org.springframework.stereotype.Component;

import java.util.List;
```

```java
import java.util.concurrent.atomic.AtomicInteger;

@Component
public class MyLB implements LoadBalancer {
    private AtomicInteger atomicInteger = new AtomicInteger(0);

    public final int getAndIncrement() {
        int current;
        int next;
        do {
            current = this.atomicInteger.get();
            next = current >= Integer.MAX_VALUE ? 0 : current + 1;
        } while (!this.atomicInteger.compareAndSet(current, next));
        System.out.println("*****第几次访问, 次数next: " + next);
        return next;
    }

    @Override
    public ServiceInstance instances(List<ServiceInstance> serviceInstances) {
        int index = this.getAndIncrement() % serviceInstances.size();
        return serviceInstances.get(index);
    }
}
```

4.3.3.4. OrderController

```java
package cn.sitedev.springcloud.controller;

import cn.sitedev.springcloud.entities.CommonResult;
import cn.sitedev.springcloud.entities.Payment;
import cn.sitedev.springcloud.lb.LoadBalancer;
import lombok.extern.slf4j.Slf4j;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

import javax.annotation.Resource;
import java.net.URI;
import java.util.List;
```

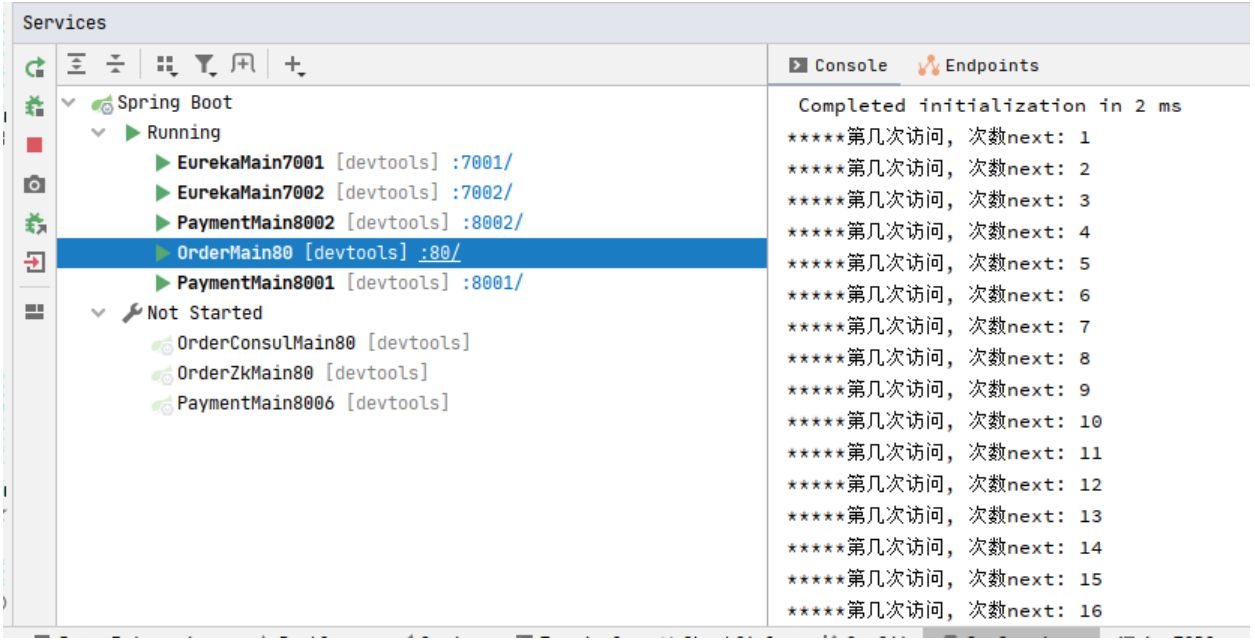
```
16
17 @RestController
18 @RequestMapping("/consumer/payment")
19 @Slf4j
20 public class OrderController {
21
22 //    public static final String PAYMENT_URL = "http://localhost:8001";
23
24     // 通过在 eureka上注册过的微服务名称调用
25     public static final String PAYMENT_URL = "http://CLOUD-PAYMENT-SERVICE";
26
27     @Resource
28     private RestTemplate restTemplate;
29
30     @Resource
31     private LoadBalancer loadBalancer;
32
33     @Resource
34     private DiscoveryClient discoveryClient;
35
36     @PostMapping(value = "/create")
37     public CommonResult<Payment> create(Payment payment) {
38         return restTemplate.postForObject(PAYMENT_URL + "/payment/create", payment,
39     }
40
41     @PostMapping(value = "/createForEntity")
42     public CommonResult<Payment> create2(Payment payment) {
43         return restTemplate.postForEntity(PAYMENT_URL + "/payment/create", payment,
44     }
45
46     @GetMapping(value = "/get/{id}")
47     public CommonResult<Payment> getPayment(@PathVariable("id") Long id) {
48         return restTemplate.getForObject(PAYMENT_URL + "/payment/get/" + id, CommonR
49     }
50
51     @GetMapping(value = "/getForEntity/{id}")
52     public CommonResult<Payment> getPayment2(@PathVariable("id") Long id) {
53         ResponseEntity<CommonResult> entity = restTemplate.getForEntity(PAYMENT_URL
54                 CommonResult.class);
55         if (entity.getStatusCode().is2xxSuccessful()) {
56             return entity.getBody();
57         } else {
58             return new CommonResult<>(444, "操作失败");
```

```
        }
    }

    @GetMapping(value = "/lb")
    public String getPaymentLB() {
        List<ServiceInstance> instances = discoveryClient.getInstances("CLOUD-PAYMEN
        if (instances == null || instances.size() <= 0) {
            return null;
        }
        ServiceInstance serviceInstance = loadBalancer.instances(instances);
        URI uri = serviceInstance.getUri();
        return restTemplate.getForObject(uri + "/payment/lb", String.class);
    }
}
```

4.3.3.5. 测试

浏览器访问: http://localhost/consumer/payment/lb

查看80服务控制台日志输出:


Services
Console
Endpoints
Spring Boot
Running
EurekaMain7001 [devtools] :7001/
EurekaMain7002 [devtools] :7002/
PaymentMain8002 [devtools] :8002/
OrderMain80 [devtools] :80/
PaymentMain8001 [devtools] :8001/
Not Started
OrderConsulMain80 [devtools]
OrderZkMain80 [devtools]
PaymentMain8006 [devtools]
Completed initialization in 2 ms
*****第几次访问, 次数next: 1
*****第几次访问, 次数next: 2
*****第几次访问, 次数next: 3
*****第几次访问, 次数next: 4
*****第几次访问, 次数next: 5
*****第几次访问, 次数next: 6
*****第几次访问, 次数next: 7
*****第几次访问, 次数next: 8
*****第几次访问, 次数next: 9
*****第几次访问, 次数next: 10
*****第几次访问, 次数next: 11
*****第几次访问, 次数next: 12
*****第几次访问, 次数next: 13
*****第几次访问, 次数next: 14
*****第几次访问, 次数next: 15
*****第几次访问, 次数next: 16