

东软睿道内部公开

文件编号: D000-

Spring Cloud微服务架构

版本: 1.0.0

第4章 Ribbon负载均衡

东软睿道教育信息技术有限公司
(版权所有, 翻版必究)

Copyright © Neusoft Educational Information Technology Co., Ltd
All Rights Reserved



本章教学目标

- ✓ 了解负载均衡简介；
- ✓ 了解Ribbon简介；
- ✓ 了解Ribbon负载均衡策略；
- ✓ 理解负载均衡实现架构；
- ✓ 掌握使用Ribbon实现负载均衡；
- ✓ 掌握Irule算法修改；

本章教学内容

节	知识点	掌握程度	难易程度	教学形式	对应在线微课
Ribbon简介	负载均衡简介	了解		线下	
	Ribbon简介	了解		线下	
	实现架构	理解		线下	
使用Ribbon实现负载均衡	编写Ribbon服务消费者	掌握	难	线下	
	编写服务提供者	掌握		线下	
	测试负载均衡	掌握		线下	
Ribbon负载均衡策略	Ribbon负载均衡策略	了解		线下	
	修改Irule算法	掌握		线下	
	测试	掌握		线下	



CONTENTS 目录

01

Ribbon简介

02

使用Ribbon实现负载均衡

03

Ribbon负载均衡策略

负载均衡是什么

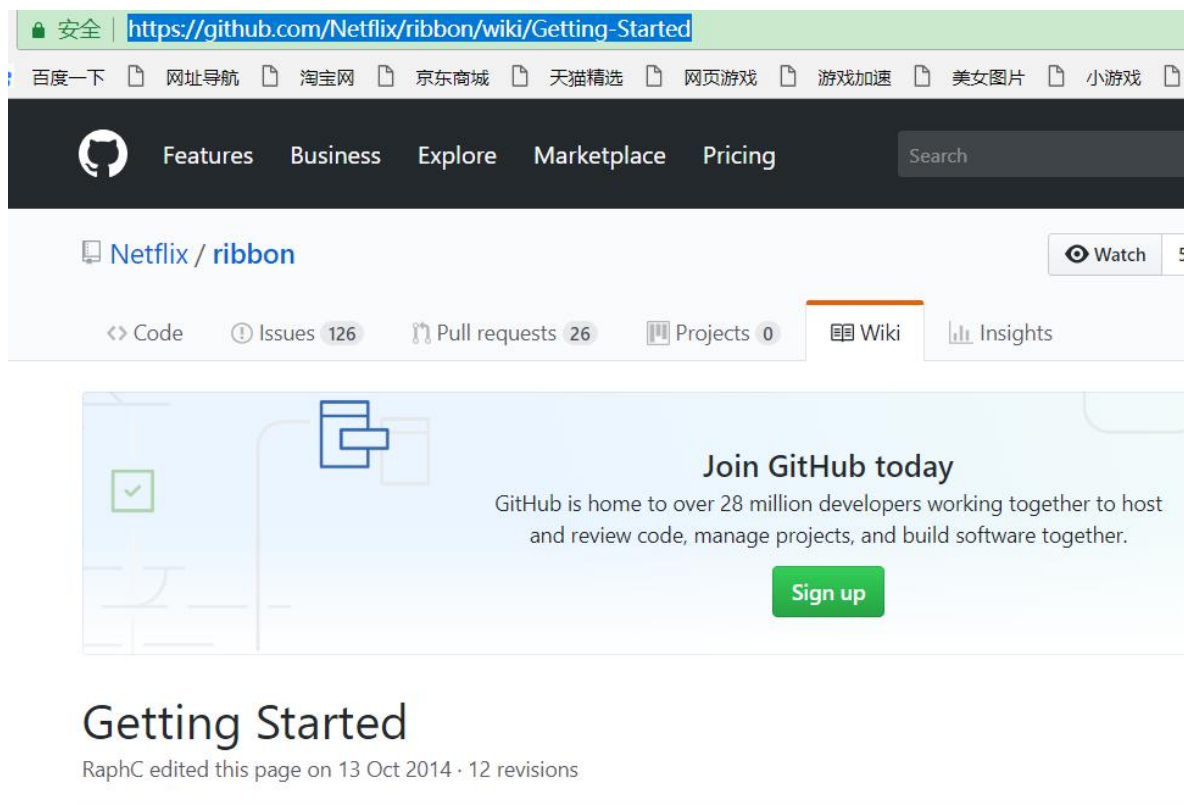
- ❖ 负载均衡（ Load Balance ），简单的说就是将用户的请求平摊的分配到多个服务上，从而达到系统的HA(High Available)。
- ❖ 负载均衡在微服务或分布式集群中经常用的一种应用。
- ❖ 常见的负载均衡有软件Nginx，LVS，硬件 F5等。
- ❖ 相应的中间件，例如：dubbo和Spring Cloud中均给我们提供了负载均衡
- ❖ Spring Cloud的负载均衡算法可以自定义。

Ribbon是什么

- ❖ Ribbon是Netflix发布的负载均衡器，它有助于控制HTTP和TCP客户端的行为。
- ❖ 为Ribbon配置服务提供者地址列表后，Ribbon就可基于某种负载均衡算法，自动的帮助服务消费者去请求。
- ❖ Ribbon默认为我们提供了很多的负载均衡算法，例如轮询、随机等。
- ❖ 当然，我们也可以为Ribbon实现自定义的负载均衡算法。

Ribbon简介

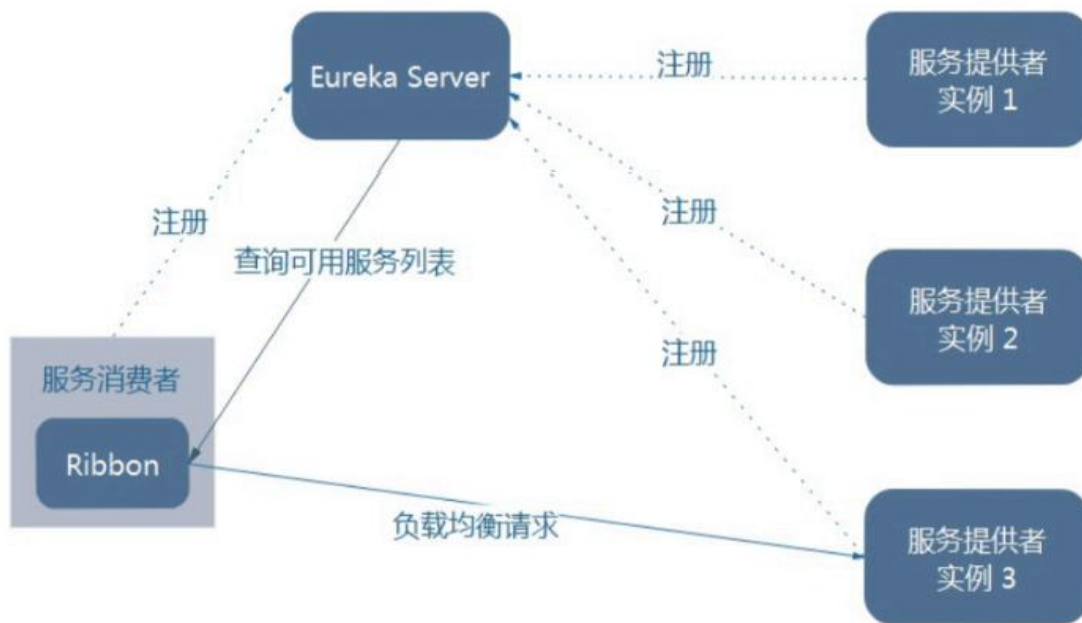
- ❖ Ribbon项目在github上托管: <https://github.com/Netflix/ribbon>
 - ▶ 可以参考学习



The simplest way to get started is to use the property driven factory to create instance of client with load balancer. The [sample application](#) in ribbon-httpclient shows the basic usage and is described

实现架构

- ❖ Ribbon在工作时分成两步
 - ▶ 第一步先选择 EurekaServer，它优先选择在同一个区域内负载较少的server.
 - ▶ 第二步再根据用户指定的策略，在从server取到的服务注册列表选择一个地址。
- ❖ 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。



CONTENTS 目录

01

Ribbon简介

02

使用Ribbon实现负载均衡

03

Ribbon负载均衡策略

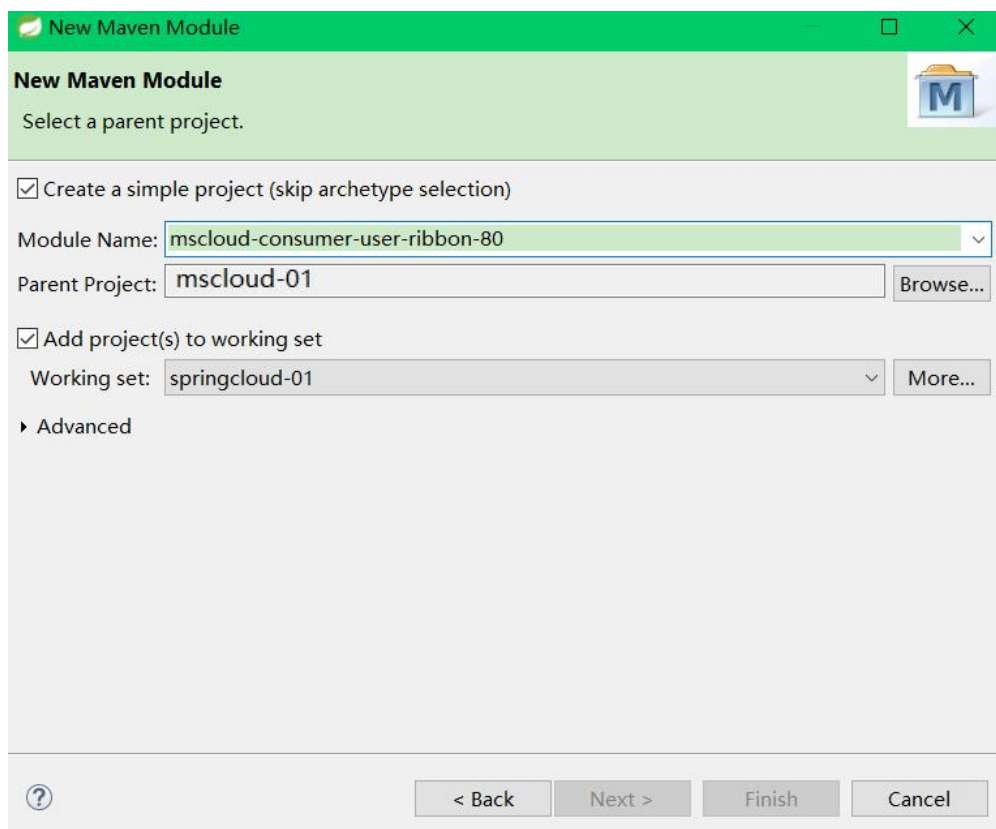
编写Ribbon服务消费者

- ❖ 根据mscloud-consumer-user-80工程，创建Ribbon服务消费者
 - ▶ 新建Module模块mscloud-consumer-user-ribbon-80
 - ▶ 修改pom.xml
 - ▶ 修改application.properties
 - ▶ 对ConfigBean添加新注解@LoadBalanced
 - ▶ 修改主启动类，添加@EnableEurekaClient
 - ▶ 修改UserController_Consumer客户端访问类
 - ▶ 测试

- ▶ 全部代码参见：[ch04-01-ribbon负载均衡-默认算法/mscloud-consumer-user-ribbon-80](#)

编写Ribbon服务消费者

- ❖ 新建Module模块mscloud-consumer-user-ribbon-80
 - ▶ 将mscloud-consumer-user-80工程内容拷贝到该工程中



编写Ribbon服务消费者

❖ 修改pom, 添加如下依赖

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

编写Ribbon服务消费者

- ❖ 修改全局配置文件
 - ▶ 追加eureka的服务注册地址

#配置服务名

```
spring.application.name=ribbon-consumer
```

#添加eureka的服务注册地址

```
eureka.client.register-with-eureka=false
```

```
eureka.client.service-url.defaultZone=http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

编写Ribbon服务消费者

- ❖ 对ConfigBean添加新注解@LoadBalanced
 - ▶ 获得Rest时加入Ribbon的配置

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

- ❖ 原理:
 - ▶ 当我们在RestTemplate该类上使用了一个@ LoadBalanced这个注解的时候，Spring Cloud会给该类生成一个代理对象，然后在进行请求发送的时候，需要先做一个处理就是根据ServiceId在注册中心中查找服务列表数据（也就是每一个服务对应的ip地址和端口号），然后再基于自身的负载均衡算法，找出一个服务，然后将ServiceId这一部分使用ip地址和对应的端口号进行替换，形成一个完整的请求路径，然后再发送请求。

编写Ribbon服务消费者

❖ 主启动类

- ▶ 添加@EnableEurekaClient

```
@SpringBootApplication
@EnableEurekaClient
public class UserConsumer80_App {
    public static void main(String[] args)
    {
        SpringApplication.run(UserConsumer80_App.class, args);
    }
}
```

编写Ribbon服务消费者

- ❖ 修改UserController_Consumer客户端访问类
 - ▶ 将地址和端口号修改为服务名

```
@RestController
public class UserController_Consumer {
    // private static final String REST_URL_PREFIX = "http://localhost:8001";
    private static final String REST_URL_PREFIX = "EUREKA-CLIENT-USER";
```


编写Ribbon服务消费者

❖ 测试

- ▶ 先启动3个Eureka集群
- ▶ 再启动mscloud-provider-user-8001并注册进Eureka
- ▶ 最后启动mscloud-consumer-user-ribbon-80
- ▶ 浏览器
 - ★ <http://localhost/consumer/user/findUserById/1>
 - ★ <http://localhost/consumer/user/findAll>



编写Ribbon服务消费者

❖ 结论

- ▶ Ribbon和Eureka整合后，Consumer可以直接调用服务而不用再关心地址和端口号

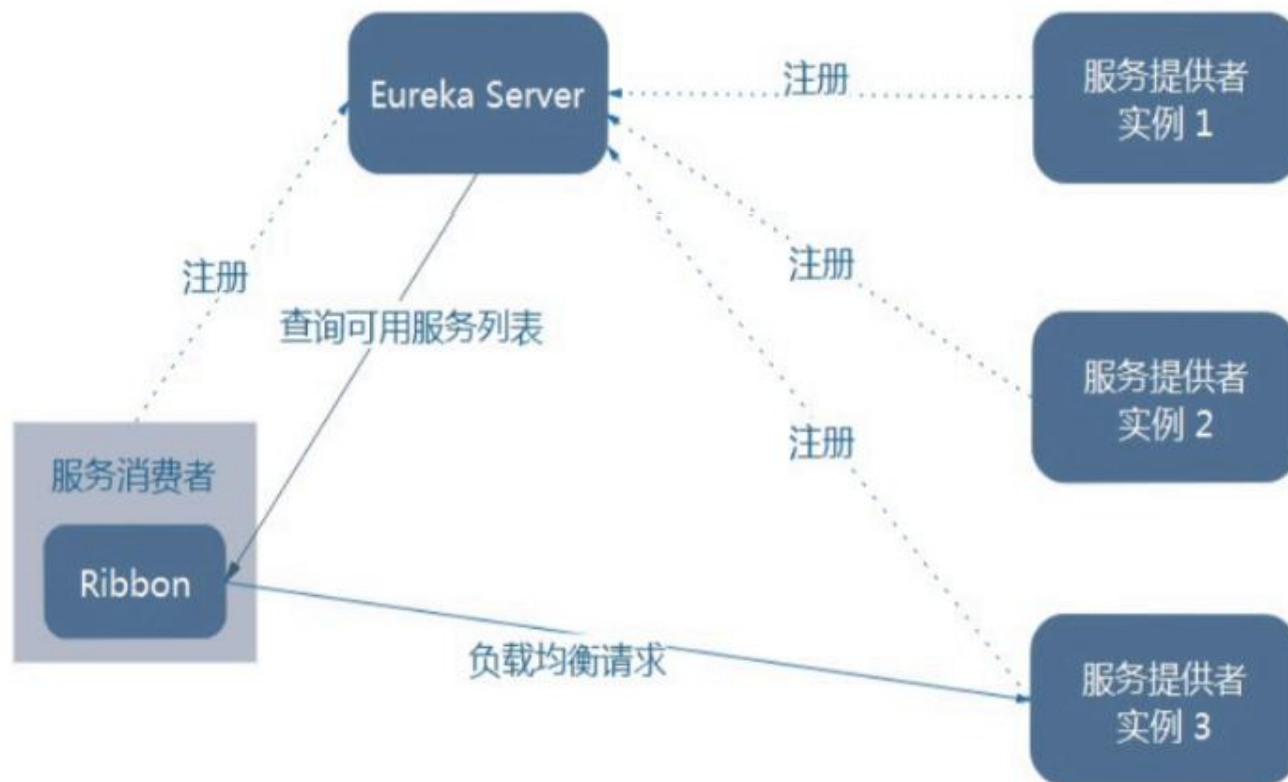
```
@RestController
public class UserController_Consumer {
    // private static final String REST_URL_PREFIX = "http://localhost:8001";
    private static final String REST_URL_PREFIX = "http://EUREKA-CLIENT-USER";

    /**
     * 使用 使用restTemplate访问restful接口非常的简单。
     */
    @Autowired
    private RestTemplate restTemplate;

    @RequestMapping("/consumer/user/findAll")
    public List<User> findAll(){
        return restTemplate.getForObject(REST_URL_PREFIX + "/user/findAll",
    }
```

编写服务提供者

❖ 架构说明

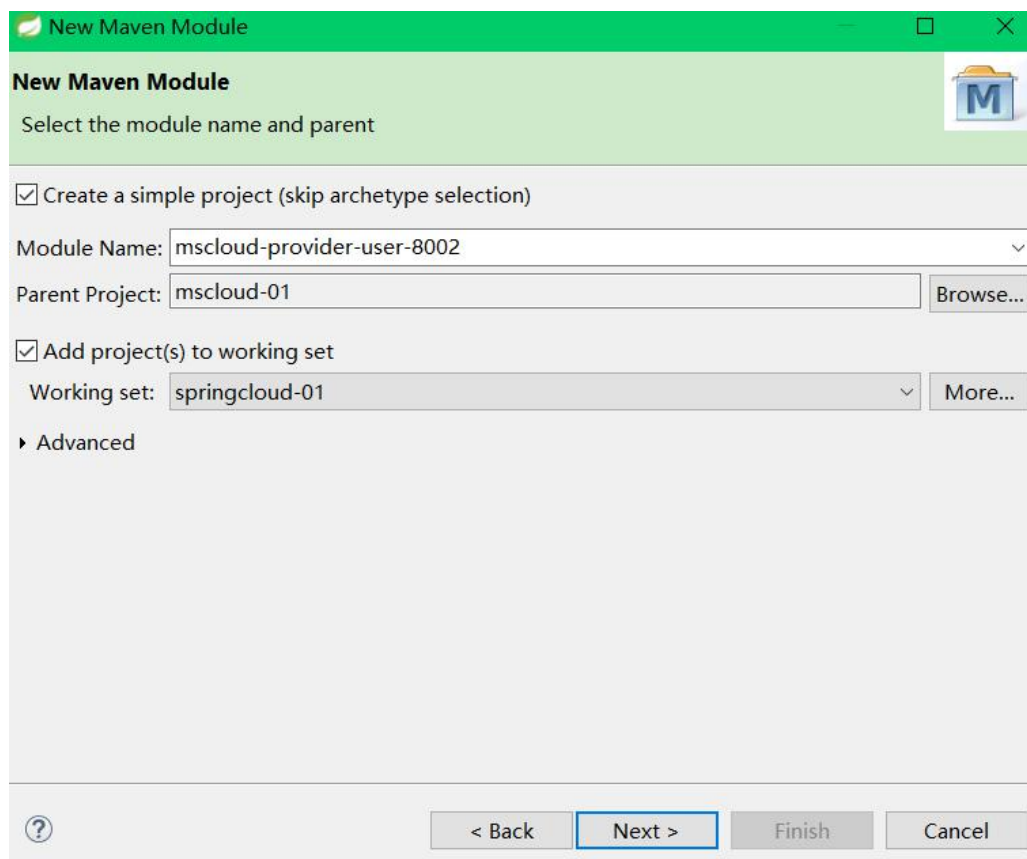


编写服务提供者

- ❖ 参考mscloud-provider-user-8001，新建两个服务提供者，服务端口号分别为8002，8003
 - ▶ 新建mscloud-provider-user-8002/8003
 - ▶ 新建2个数据库，各自微服务分别连各自的数据库
 - ▶ 修正mscloud-api中的Entity，重新编译
 - ▶ 分别修改8002/8003各自的全局配置文件
- ▶ 全部代码参见：[ch04-01-ribbon负载均衡-默认算法/mscloud-provider-user-8002、mscloud-provider-user-8003、mscloud-api](#)

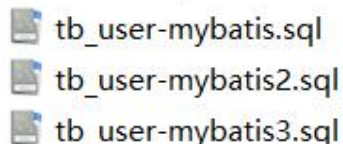
编写服务提供者

- ❖ 参考mscloud-provider-user-8001，再新建两个Module模块，分别命名为mscloud-provider-user-8002，mscloud-provider-user-8003
 - ▶ 将mscloud-provider-user-8001工程内容拷贝到两个新工程中



编写服务提供者

- ❖ 新建2个数据库，各自微服务分别连各自的数据库
 - ▶ 具体参考sql脚本



tb_user-mybatis.sql
tb_user-mybatis2.sql
tb_user-mybatis3.sql

- ❖ **特别说明**：该数据库中的tb_user表，添加了一个字段db_source，这样最终的查询结果就可以知道来自于哪个微服务（**为了便于单机测试的时候方便**）
- ❖ **全部代码参见**：[ch04-01-ribbon负载均衡-默认算法/tb_user-*.sql](#)

编写服务提供者

- ❖ 修正mscloud-api中的Entity
 - ▶ 添加dbSource属性
 - ▶ mvn clean install

```
public class User implements Serializable{  
  
    private static final long serialVersionUID = 1L;  
  
    private int id ;  
    private String loginName ;  
    private String username ;  
    private String password;  
    private String dbSource;
```

编写服务提供者

- ❖ 分别修改mscloud-provider-user-8002/mscloud-provider-user-8003各自全局配置文件，示例如下：

- ▶ 修改数据源url

```
# url  
spring.datasource.url=jdbc:mysql://localhost:3306/mybatis3?useUnicode=true&characterEncoding=utf-8
```

- ▶ server.port=8003

```
eureka.instance.instance-id: msccloud-user8003
```

```
#配置服务名称
```

```
spring.application.name=eureka-client-user
```


测试负载均衡

❖ 按照如下步骤测试负载均衡

- ▶ 启动3个eureka集群配置
- ▶ 启动3个User微服务启动并各自测试通过
 - ★ `http://localhost:8001/user/findUserById/1`
 - ★ `http://localhost:8002/user/findUserById/1`
 - ★ `http://localhost:8003/user/findUserById/1`
- ▶ 启动mscloud-consumer-user-ribbon-80
- ▶ 客户端通过Ribbon完成负载均衡并访问上一步的User微服务
 - ★ `http://localhost/consumer/user/findUserById/1`

测试负载均衡

❖ 启动3个Eureka集群配置

← → ↻ localhost:7001

应用 百度一下 网址导航 淘宝网 京东商城 天猫精选 网页游戏 游戏加速 美女图片 小游戏 免费电影 Spring Cloud (Fin

DS Replicas

eureka7003.com

eureka7002.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-USER	n/a (3)	(3)	UP (3) - mscloud-user8003 , mscloud-user8001 , mscloud-user8002

测试负载均衡

❖ 测试3个User微服务

- ▶ <http://localhost:8001/user/findUserById/1>
- ▶ <http://localhost:8002/user/findUserById/1>
- ▶ <http://localhost:8003/user/findUserById/1>



测试负载均衡

- ❖ 启动mscloud-consumer-user-80
 - ▶ 控制台可以看到启动了7个微服务

```
1 EurekaServer7001_App [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:12:43)
2 EurekaServer7002_App [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:13:09)
3 EurekaServer7003_App [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:13:31)
4 UserProvider8001_App [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:13:58)
5 UserProvider8002_App (1) [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:14:49)
6 UserProvider8003_App (2) [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:15:34)
7 UserConsumer80_App (1) [Java Application] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (2018年9月20日 下午5:17:02)
```

测试负载均衡

- ❖ 客户端通过Ribbon完成负载均衡并访问上一步的User微服务
 - ▶ 多次输入同一个网址，按照一定的规律访问不同的微服务



localhost/consumer/user/findUserById/1

应用 百度一下 网址导航 淘宝网 京东商城 天猫精选 网页游戏

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis"}
```



localhost/consumer/user/findUserById/1

应用 百度一下 网址导航 淘宝网 京东商城 天猫精选 网页游戏

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis3"}
```



localhost/consumer/user/findUserById/1

应用 百度一下 网址导航 淘宝网 京东商城 天猫精选 网页游戏

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis2"}
```

测试负载均衡

- ❖ 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结合只是其中的一个实例。

CONTENTS 目录

01

Ribbon简介

02

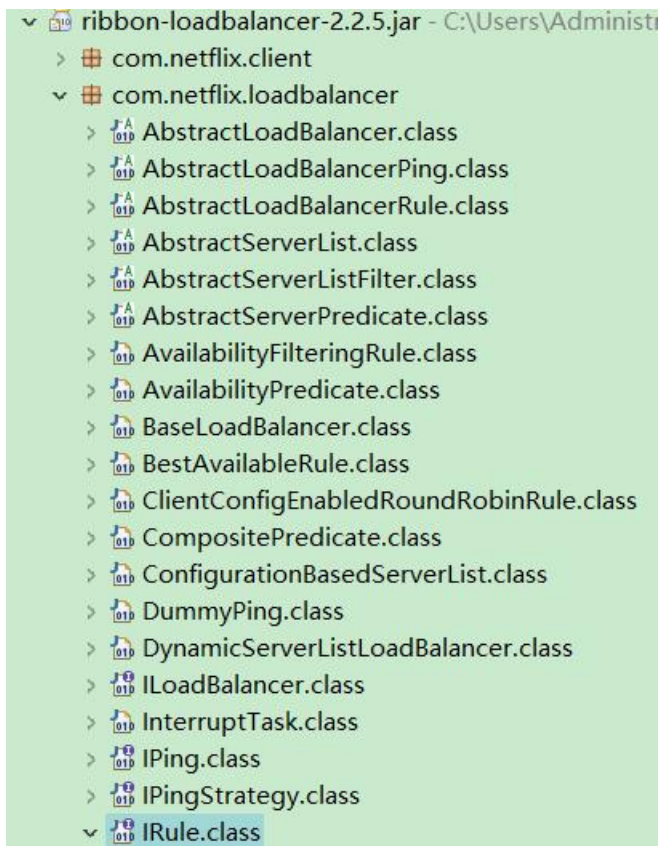
使用Ribbon实现负载均衡

03

Ribbon负载均衡策略

Ribbon负载均衡策略

- ❖ 负载均衡器Ribbon中的IRule负责选择什么样的负载均衡算法



Ribbon负载均衡策略

❖ IRule接口的源码如下：

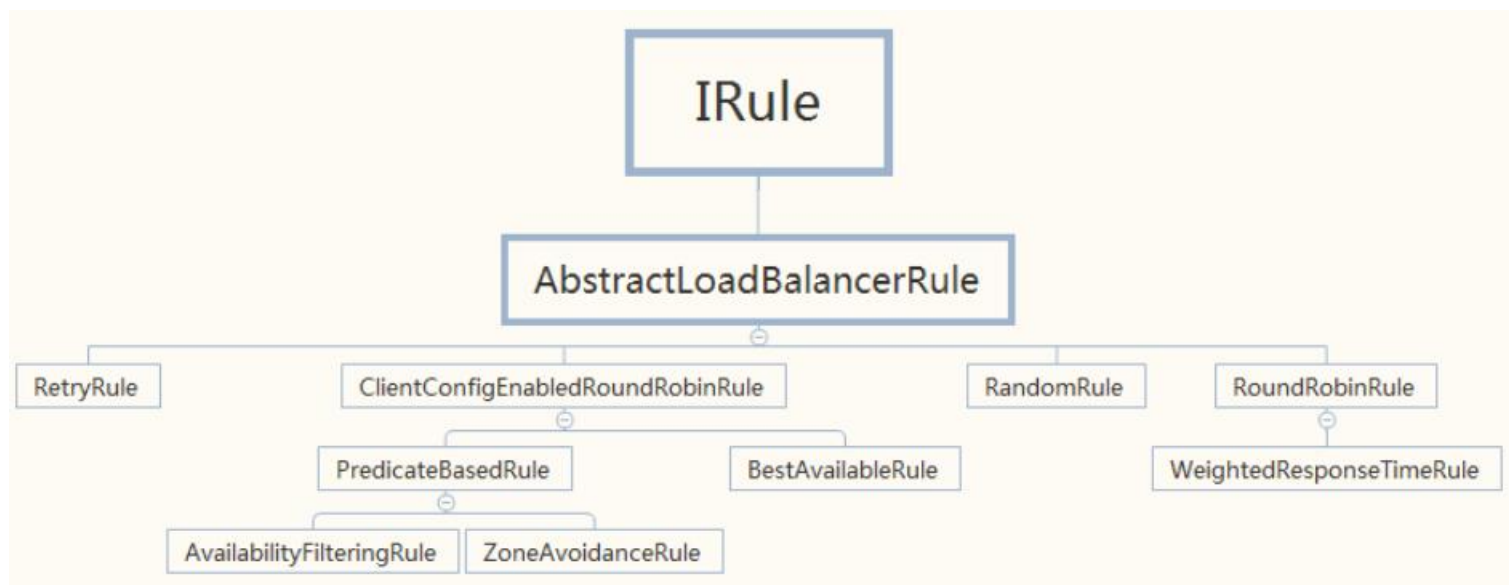
```
public interface IRule{  
    /*  
     * choose one alive server from lb.allServers or  
     * lb.upServers according to key  
     *  
     * @return choosen Server object. NULL is returned if none  
     * server is available  
     */  
  
    public Server choose(Object key);  
  
    public void setLoadBalancer(ILoadBalancer lb);  
  
    public ILoadBalancer getLoadBalancer();  
}
```

❖ 接口有三个方法

- ▶ 其中choose()是根据key 来获取server
- ▶ setLoadBalancer()和getLoadBalancer()是用来设置和获取ILoadBalancer的

Ribbon负载均衡策略

- ❖ IRule有很多默认的实现类，这些实现类根据不同的算法和逻辑来处理负载均衡。Ribbon实现的IRule有以下。在大多数情况下，这些默认的实现类是可以满足需求的，如果有特性的需求，还可以自定义。



Ribbon负载均衡策略

- ❖ BestAvailableRule 选择最小请求数
- ❖ ClientConfigEnabledRoundRobinRule 轮询
- ❖ RandomRule 随机选择一个server
- ❖ RoundRobinRule 轮询选择server
- ❖ RetryRule 根据轮询的方式重试
- ❖ WeightedResponseTimeRule 根据响应时间去分配一个weight，weight越低，被选择的可能性就越低
- ❖ ZoneAvoidanceRule 根据server的zone区域和可用性来轮询选择

修改Irule算法

- ❖ 默认是轮询算法，如果要修改其他算法，可以修改java配置文件
 - ▶ 添加Irule bean

```
@Bean
public IRule myRule()
{
    //return new RoundRobinRule();//默认的
    return new RandomRule();//达到的目的，用我们重新选择的随机算法替
}
```

- ▶ 全部代码参见：[ch04-02-ribbon负载均衡-修改算法/mscloud-consumer-user-ribbon-80](#)

测试

❖ 按照如下步骤测试

- ▶ 启动3个eureka集群配置
- ▶ 启动3个User微服务启动
- ▶ 启动mscloud-consumer-user-ribbon-80
- ▶ 多次输入同一个网址，随机访问不同的微服务
 - ★ <http://localhost/consumer/user/findUserById/1>

测试



本章重点总结

- ❖ 了解负载均衡简介；
- ❖ 了解Ribbon简介；
- ❖ 了解Ribbon负载均衡策略；
- ❖ 理解负载均衡实现架构；
- ❖ 掌握使用Ribbon实现负载均衡；
- ❖ 掌握Irule算法修改；



课后作业【必做任务】

- ❖ 1、独立完成课件中的示例



课后作业【线上任务】

❖ 线上任务

- ▶ 安排学员线上学习任务（安排学员到睿道实训平台进行复习和预习的任务，主要是进行微课的学习）

