Spring Cloud微服务架构

版本：1.0.0

第4章 Ribbon负载均衡

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

Neuedu

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

负载均衡是什么

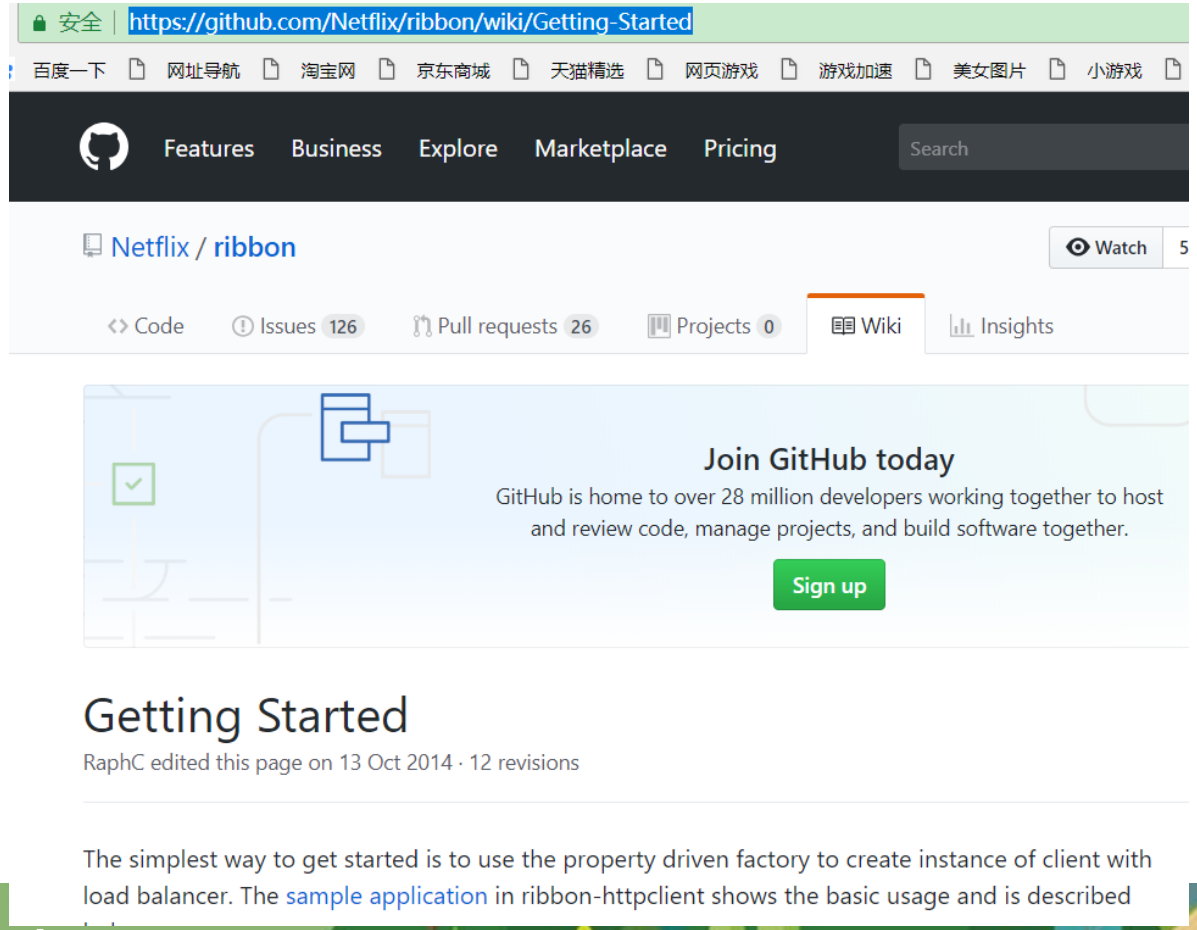
- 负载均衡（ Load Balance ），简单的说就是将用户的请求平摊的分配到多个服务上，从而达到系统的HA(High Available)。
- 负载均衡在微服务或分布式集群中经常用的一种应用。
- 常见的负载均衡有软件Nginx，LVS，硬件 F5等。
- 相应的中间件，例如：dubbo和Spring Cloud中均给我们提供了负载均衡
- Spring Cloud的负载均衡算法可以自定义。

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。

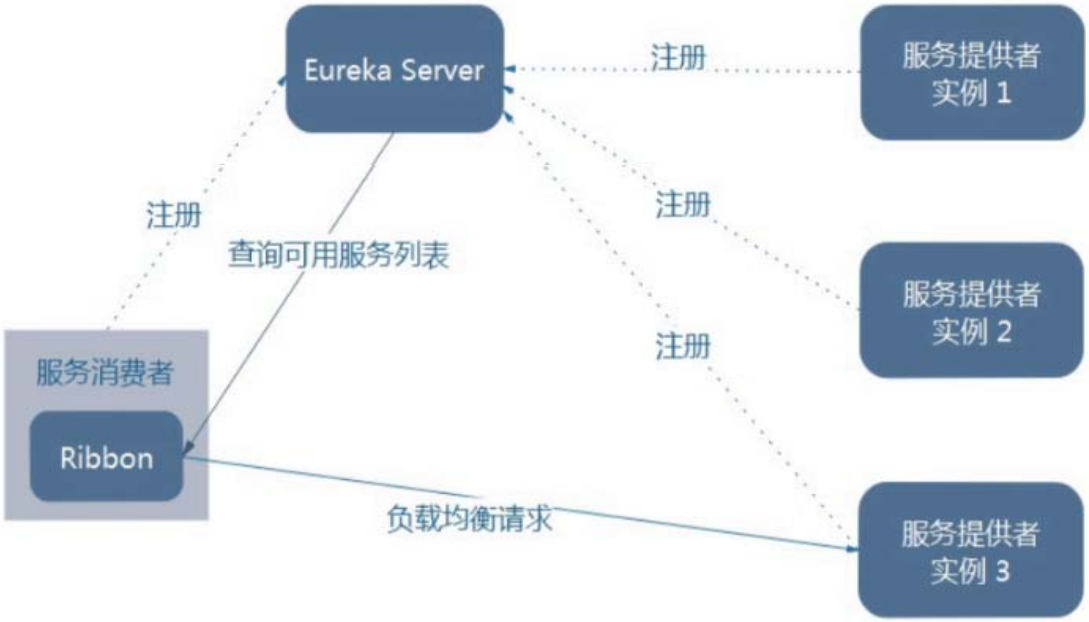
Ribbon简介

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



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。



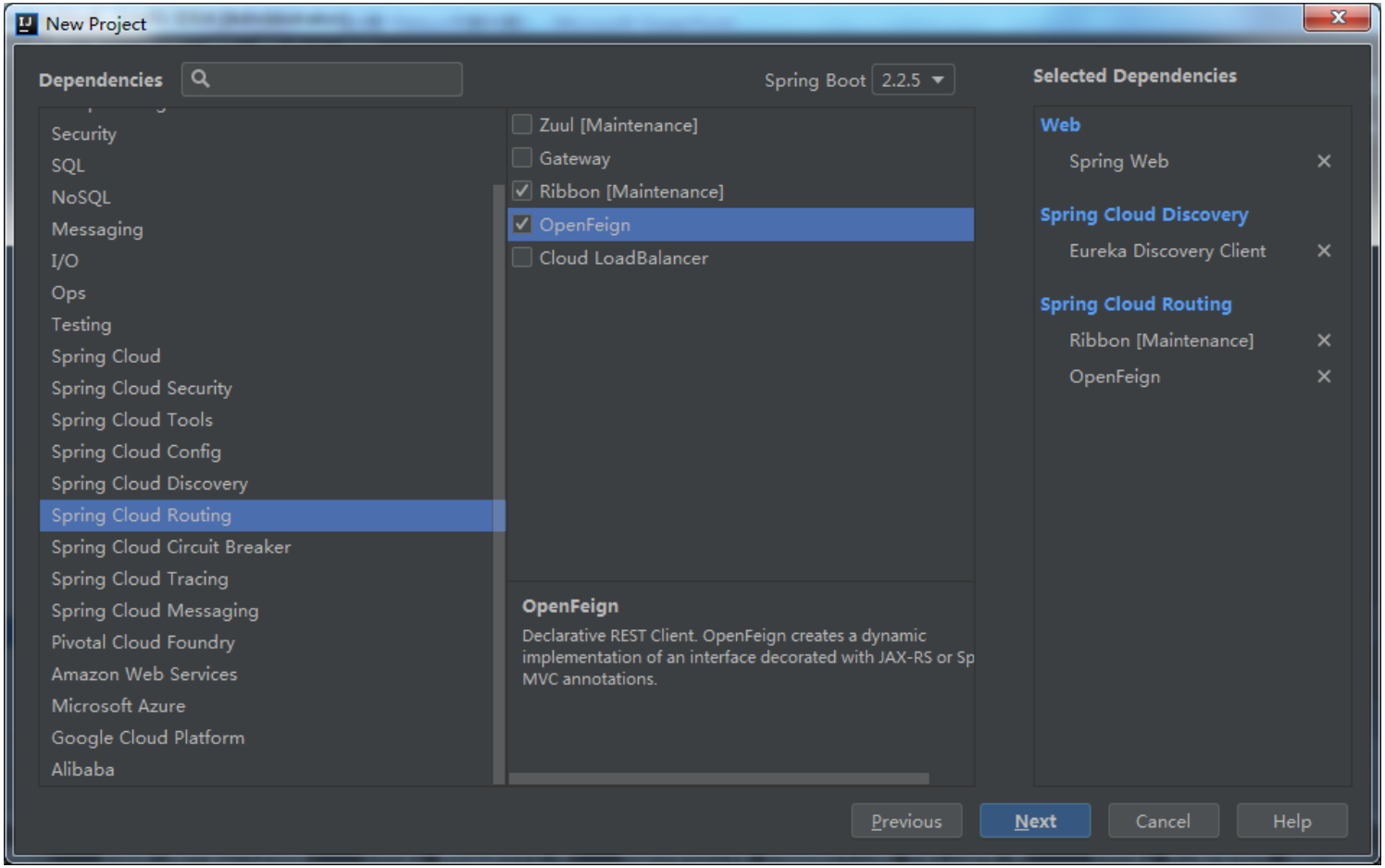
CONTENTS 目录

编写Ribbon服务消费者

- 参考上章consumer模块，创建Ribbon服务消费者
  - 新建工程ribbon
  - 修改application.yml
  - 对ConfigBean添加新注解@LoadBalanced
  - 修改主启动类，添加@EnableEurekaClient
  - 修改Controller客户端访问类
  - 测试

编写Ribbon服务消费者

- 新建工程ribbon



编写Ribbon服务消费者

- 在c:\windows\system32\drivers\etc\hosts追加域名解析信息

```
127.0.0.1 eureka1
127.0.0.1 eureka2
127.0.0.1 eureka3
```

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

```
server:
  port: 80
eureka:
  client:
    service-url:
      defaultZone:
http://eureka1:7001/eureka/,http://eureka2:7002/eureka/,http://eureka3:
7003/eureka/
    register-with-eureka: false
spring:
  application:
    name: ribbon-comsumer
```

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类上使用了@ 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

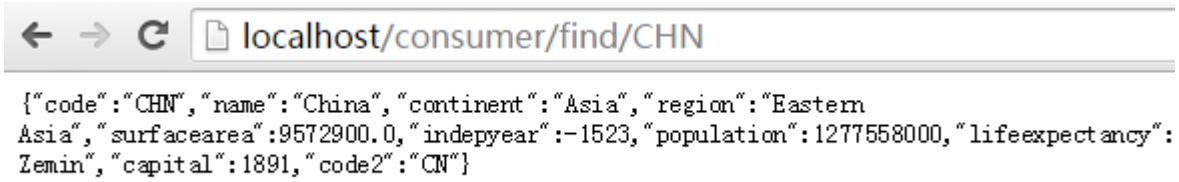
编写Ribbon服务消费者

- 修改Controller访问的地址
  - 将地址和端口号修改为服务名

```
//    private static final String REST_URL_PREFIX = "http://localhost:8003";
      private static final String REST_URL_PREFIX = "http://provider";
```

编写Ribbon服务消费者

- 测试
  - 先启动3个Eureka集群（第三章中的eureka工程）
  - 再启动3个provider并注册进Eureka （第三章中的eureka工程）
  - 最后启动本章的ribbon工程
  - 浏览器
    - http://localhost/consumer/find/CHN

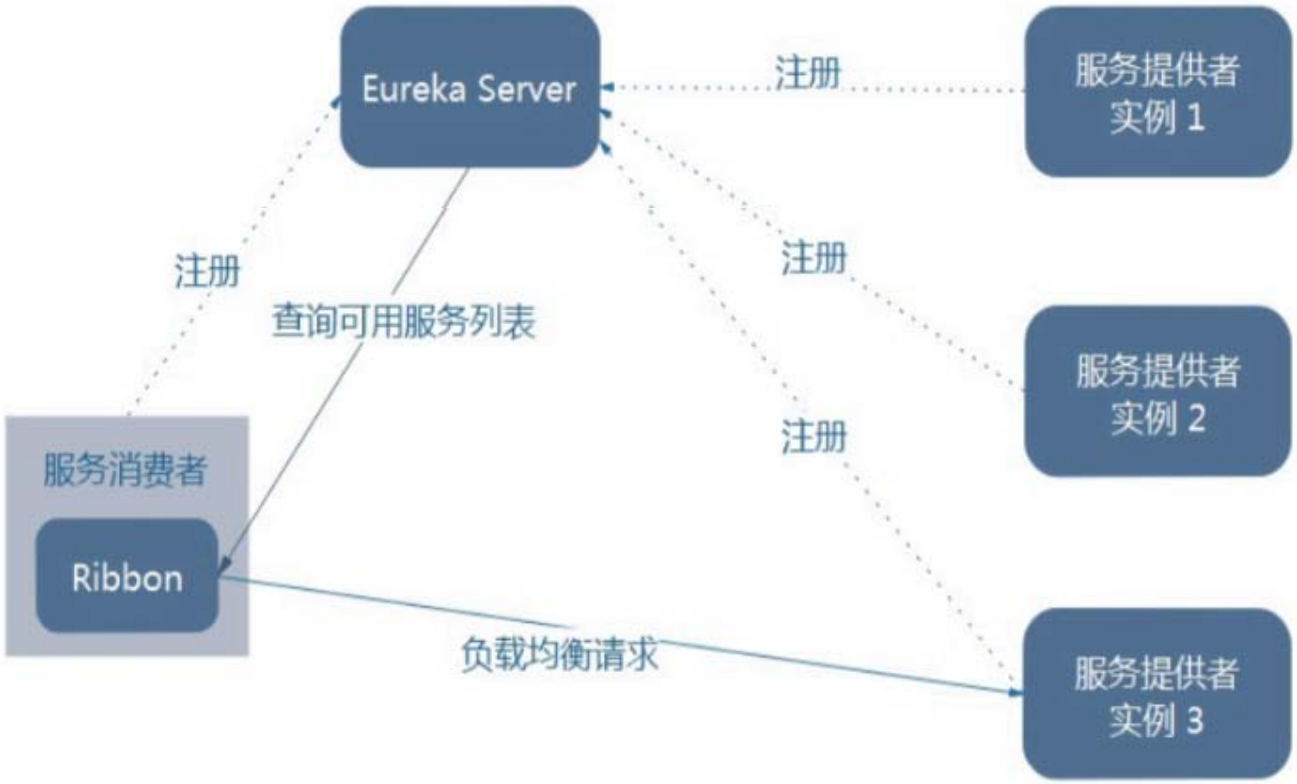


编写Ribbon服务消费者

- 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

编写服务提供者

- 架构说明



修改服务提供者代码

- 为了能够在测试的时候看到访问的是哪个微服务，需要修改第三章eureka工程中的3个provider代码
  - 在bean代码中增加微服务标识属性

```
private String identification;
public String getIdentification() {
    return identification;
}
public void setIdentification(String identification) {
    this.identification = identification;
}
```

  - 在controller代码中增加微服务标识幅值代码

```
@RequestMapping("/find/{code}")
public Country findByCode(@PathVariable String code){
    Country country = countryService.findByCode(code);
    country.setIdentification("providerIII");
    return country;
}
```

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（第三章eureka工程）
  - 启动3个provider微服务（第三章eureka工程）
  - 启动ribbon微服务
  - 客户端通过Ribbon完成负载均衡并访问：
    - http://localhost/consumer/find/CHN

测试负载均衡

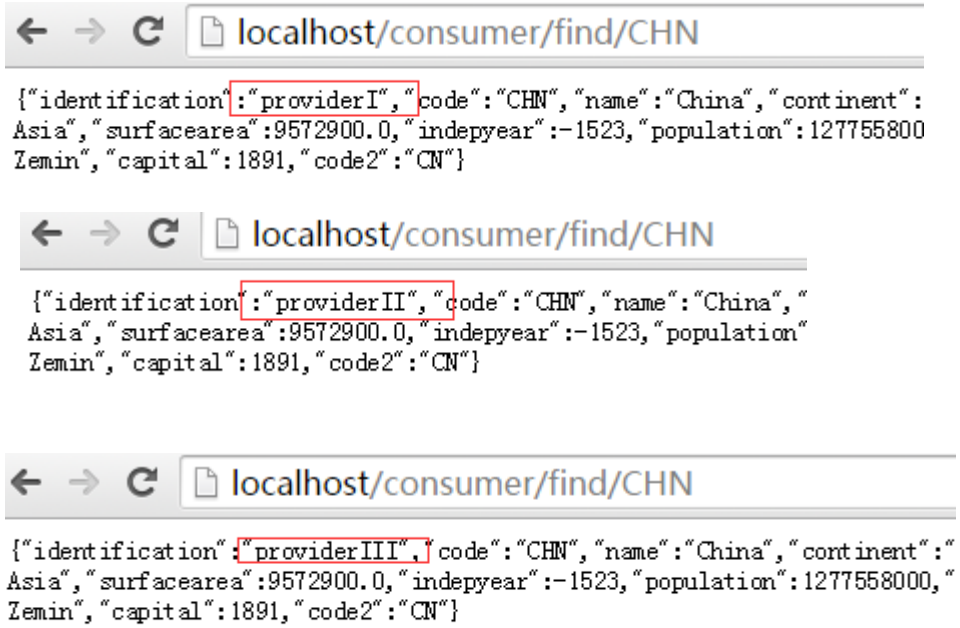
- 应该一共启动7个微服务
  - 3个eureka集群
  - 3个provider集群
  - 1个ribbon

测试负载均衡

- 启动3个Eureka集群配置

测试负载均衡

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



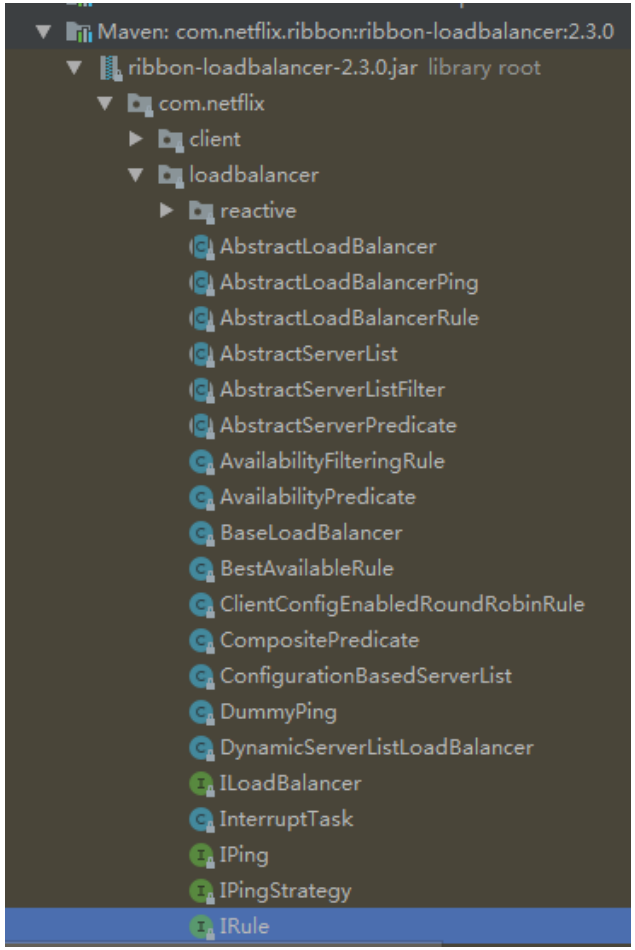
测试负载均衡

- 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

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

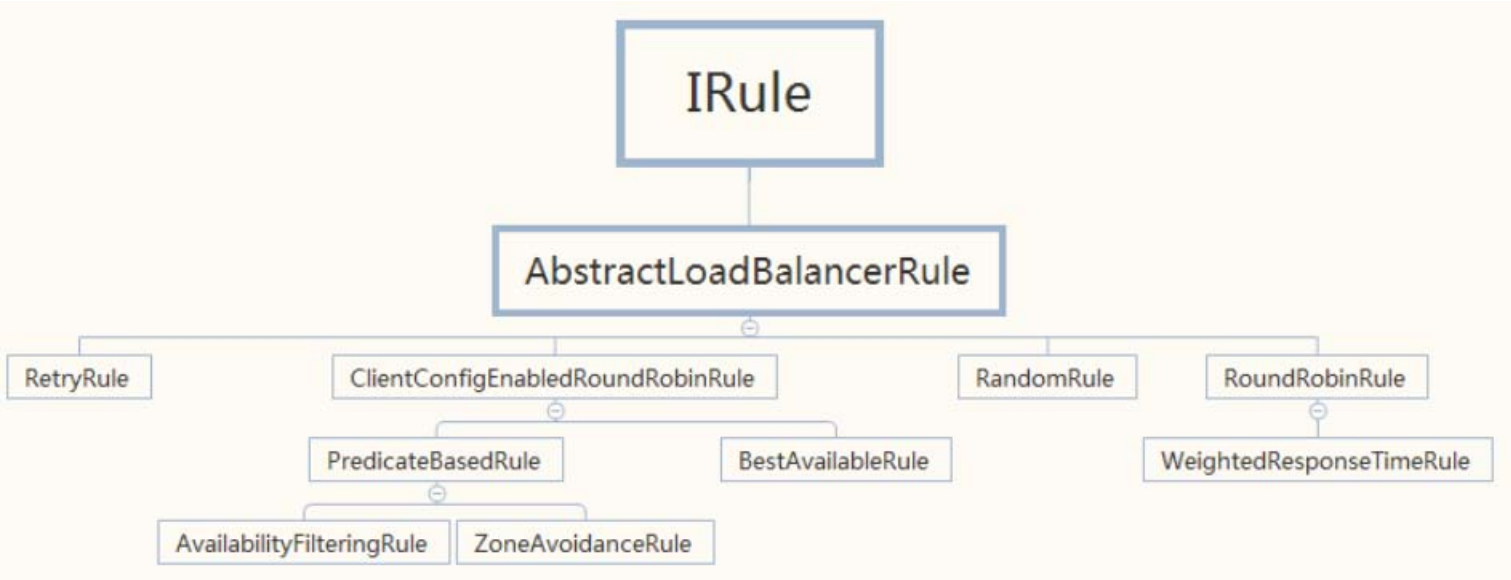
- Irule接口的源码如下：

```
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添加Irule bean

```
    @Bean
    public IRule myRule(){
//      return new RoundRobinRule();
        return new RandomRule();
    }
```

测试

- 按照如下步骤测试
  - 启动3个eureka集群配置
  - 启动3个provider微服务启动
  - 启动ribbon
  - 多次输入同一个网址，随机访问不同的微服务
    - http://localhost/consumer/find/CHN
  - 结果是每次随机访问其中一个微服务

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