# 微服务专题-一站式微服务架构SpringCloud

主讲：Cat 老师

**北京动力节点教育科技有限公司**  
**2019 • 北京**

**动力节点版权所有，禁止私自传播**

# 快速回顾

1. 分布式与微服务架构的理论梳理；
2. 什么是Spring Cloud？
3. Spring Cloud的整体架构
4. 服务消费者Controller直连调用服务提供者Controller
5. Spring Cloud的注册中心Eureka
6. Spring Cloud Eureka与Zookeeper比较
7. Spring Cloud Eureka高可用集群

# 2、Spring Cloud Eureka集群注册中心打包发布

在实际项目中，需要将Eureka发布到具体服务器上进行部署，打包部署其实和SpringBoot里面的一样，我们可以把其达成jar包，启动时对于properties文件，不同的环境激活不同的配置文件；

运行：

java -jar springcloud-service-eureka.jar --spring.profiles.active=eureka8761

java -jar springcloud-service-eureka.jar --spring.profiles.active=eureka8762

java -jar springcloud-service-eureka.jar --spring.profiles.active=eureka8763

**可以写个shell脚本实现三个注册中心的启动：**

#!/bin/sh

nohup java -jar 34-springcloud-service-eureka-1.0.0.jar --spring.profiles.active=eureka8761 > ./logs/eureka8761.log &

nohup java -jar 34-springcloud-service-eureka-1.0.0.jar --spring.profiles.active=eureka8762 > ./logs/eureka8762.log &

nohup java -jar 34-springcloud-service-eureka-1.0.0.jar --spring.profiles.active=eureka8763 > ./logs/eureka8763.log &

**修改linux的hosts文件：**

vim /etc/hosts

192.168.10.128 eureka8761

192.168.10.128 eureka8762

192.168.10.128 eureka8763

# **3、Eureka服务注册中心自我保护机制**

自我保护机制是 Eureka 注册中心的重要特性，当 Eureka 注册中心进入自我保护模式时，在 Eureka Server 首页会输出如下警告信息：

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT.

RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED

JUST TO BE SAFE.

小写是：

emergency! eureka may be incorrectly claiming instances are up when they're not. renewals are

lesser than threshold and hence the instances are not being expired just to be safe.

在没有 Eureka 自我保护的情况下，如果 Eureka Server 在一定时间内没有接收到某个微服务实例的心跳，Eureka Server 将会注销该实例，但是当发生网络分区故障时，那么微服务与 Eureka Server 之间将无法正常通信，以上行为可能变得非常危险了，因为微服务本身其实是正常的，此时不应该注销这个微服务，如果没有自我保护机制，那么 Eureka Server 就会将此服务注销掉。

Eureka 通过“自我保护模式”来解决这个问题——当 Eureka Server 节点在短时间内丢失过多客户端时（可能发生了网络分区故障），那么就会把这个微服务节点进行保护。一旦进入自我保护模式，Eureka Server 就会保护服务注册表中的信息，不删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该 Eureka Server 节点会再自动退出自我保护模式。

所以，自我保护模式是一种应对网络异常的安全保护措施，它的架构哲学是宁可

同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注

销任何健康的微服务，使用自我保护模式，可以让 Eureka 集群更加的健壮、稳定。

当然也可以使用配置项：**eureka.server.enable-self-preservation = false** 禁用自我保护模式。

关闭自我保护模式后会出现红色：

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

但是 Eureka Server 自我保护模式也会给我们带来一些困扰，如果在保护期内

某个服务提供者刚好非正常下线了，此时服务消费者就会拿到一个无效的服务实

例，此时会调用失败，对于这个问题需要服务消费者端具有一些容错机制，如重

试，断路器等。

Eureka 的自我保护模式是有意义的，该模式被激活后，它不会从注册列表中剔

除因长时间没收到心跳导致注册过期的服务，而是等待修复，直到心跳恢复正常

之后，它自动退出自我保护模式。这种模式旨在避免因网络分区故障导致服务不

可用的问题。

例如，两个微服务客户端实例 A 和 B 之间有调用的关系，A 是消费者，B 是提供者，但是由于网络故障，B 未能及时向 Eureka 发送心跳续约，这时候 Eureka 不能简单的将 B 从注册表中剔除，因为如果剔除了，A 就无法从 Eureka 服务器中获取 B 注册的服务，但是这时候 B 服务是可用的；

所以，Eureka 的自我保护模式最好还是开启它。

关于自我保护常用几个配置如下：

服务器端配置：

#测试时关闭自我保护机制，保证不可用服务及时踢出

**eureka.server.enable-self-preservation=false**

客户配置：

#每间隔 2s，向服务端发送一次心跳，证明自己依然"存活"

**eureka.instance.lease-renewal-interval-in-seconds**=**2**

#告诉服务端，如果我 10s 之内没有给你发心跳，就代表我故障了，将我踢出掉

**eureka.instance.lease-expiration-duration-in-seconds**=**10**

# 4、Spring Cloud Ribbon

## Ribbon是什么?

（Spring Cloud Netflix） -->大量使用了Netflix公司的开源项目

（Spring Cloud Alibaba）-->大量使用了Alibaba公司的开源项目

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡器；

我们通常说的负载均衡是指将一个请求均匀地分摊到不同的节点单元上执行，负

载均和分为硬件负载均衡和软件负载均衡：

**硬件负载均衡**：比如 F5、深信服、Array 等；

**软件负载均衡**：比如 Nginx、LVS、HAProxy 等；（是一个服务器实现的）

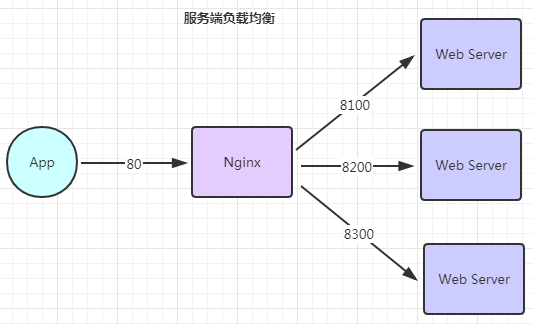
Ribbon是Netflix公司发布的开源项目（组件、框架、jar包），主要功能是提供客户端的软件负载均衡算法，它会从eureka中获取一个可用的服务端清单，通过心跳检测来剔除故障的服务端节点以保证清单中都是可以正常访问的服务端节点。

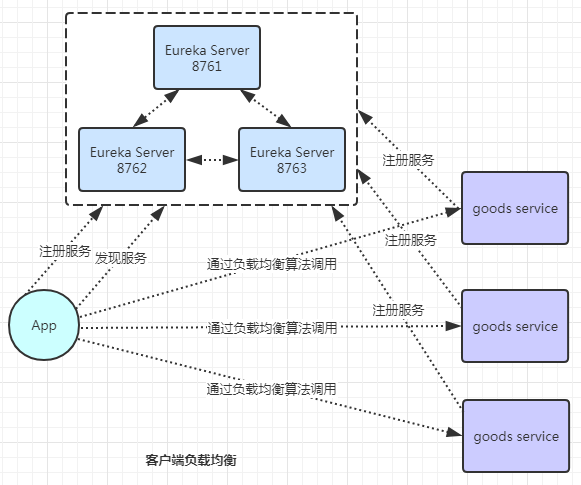
当客户端发送请求，则ribbon负载均衡器按某种算法（比如轮询、权重、 最小连接数等）从维护的可用服务端清单中取出一台服务端的地址，然后进行请求；

Ribbon非常简单，可以说就是一个jar包，这个jar包实现了负载均衡算法，Spring Cloud 对 Ribbon 做了二次封装，可以让我们使用 RestTemplate 的服务请求，自动转换成客户端负载均衡的服务调用。

Ribbon 支持多种负载均衡算法，还支持自定义的负载均衡算法。

## 客户端负载均衡 vs 服务端负载均衡





## 采用Ribbon实现服务调用

1、首先加入ribbon的依赖，但是eureka已经依赖了ribbon，所以这里不需要再引用ribbon的依赖；

2、要使用ribbon，只需要一个注解：

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

在RestTemplate上面加入@LoadBalanced注解，这样就可以实现RestTemplate 在调用时自动负载均衡；

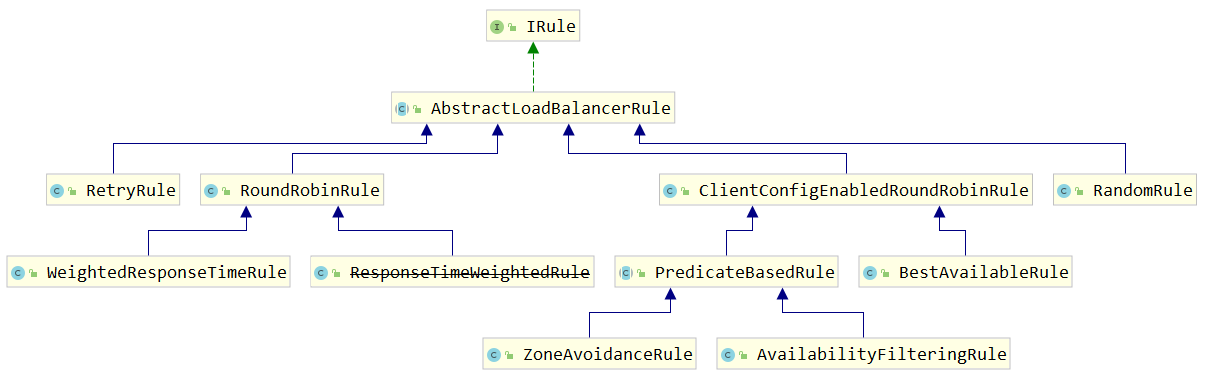
我们这里现在启动了eureka集群（3个eureka） 和服务提供者集群（2个service-goods） 和一个服务调用者（service-portal）

*#告诉服务端，服务实例的唯一ID***eureka.instance.instance-id**=**34-springcloud-service-portal**

## Ribbon 负载均衡策略

Ribbon 的负载均衡策略是由 IRule 接口定义, 该接口由如下实现：

在jar包：com.netflix.ribbon#ribbon-loadbalancer中；



要使用ribbon实现负载均衡，在Spring 的配置类里面把对应的负载均衡接口实现类作为一个Bean配置一下就行了；

负载均衡的入口：ILoadBalancer接口

如果要切换负载均衡策略：

@Bean  
public IRule iRule(){  
 return new RoundRobinRule();  
}

|  |  |
| --- | --- |
| **负载均衡实现** | **策略** |
| RandomRule | 随机 |
| RoundRobinRule | 轮询 |
| AvailabilityFilteringRule | 先过滤掉由于多次访问故障的服务，以及并  发连接数超过阈值的服务，然后对剩下的服  务按照轮询策略进行访问； |
| WeightedResponseTimeRule | 根据平均响应时间计算所有服务的权重，响  应时间越快服务权重就越大被选中的概率即  越高，如果服务刚启动时统计信息不足，则  使用RoundRobinRule策略，待统计信息足够会切换到该WeightedResponseTimeRule策  略； |
| RetryRule | 先按照RoundRobinRule策略分发，如果分发  到的服务不能访问，则在指定时间内进行重  试，然后分发其他可用的服务； |
| BestAvailableRule | 先过滤掉由于多次访问故障的服务，然后选  择一个并发量最小的服务； |
| ZoneAvoidanceRule （默认） | 综合判断服务节点所在区域的性能和服务节  点的可用性，来决定选择哪个服务； |

如果我们没有指定负载均衡策略，ribbon默认的负载均衡是ZoneAvoidanceRule；（新版本，G版本）

# 5、Spring Cloud Feign

## Feign是什么？

Feign 是 Netflix 公司开发的一个声明式的 REST 调用客户端； （调用远程的restful风格的http接口 的一个组件）

调用组件其实很多，比如：

1. Httpclient（apache）
2. Httpurlconnection （jdk）
3. restTemplate（spring）
4. OkHttp（android）
5. Feign （Netflix） --> 实现非常优雅

Spring Cloud Feign 对 Ribbon 负载均衡进行了简化，在其基础上进行了进一步的封装，在配置上大大简化了开发工作，它是一种声明式的调用方式，它的使用方法是定义一个接口，然后在接口上添加注解，使其支持了Spring MVC标准注解和HttpMessageConverters，Feign可以与Eureka和Ribbon组合使用以支持负载均衡。

## Feign能干什么？

Feign旨在简化微服务消费方（调用者，客户端）代码的开发，前面在使用**Ribbon+RestTemplate**进行服务调用时，利用RestTemplate对http请求的封装处理，形成了一套模版化的调用方式，但是在实际开发中，由于服务提供者提供的接口非常多，一个接口也可能会被多处调用，Feign在Ribbon+RestTemplate的基础上做了进一步封装，在Feign封装之后，我们只需创建一个接口并使用注解的方式来配置，即可完成对服务提供方的接口绑定，简化了使用Ribbon + RestTemplate的调用，自动封装服务调用客户端，减少代码开发量；

## 使用Feign实现消费者

使用 Feign 实现消费者，我们通过下面步骤进行：

**第一步**：创建普通 Spring Boot 工程

把接口放在通用的接口层、常量类、model的项目中

**第二步**：添加依赖

要 添 加 的 依 赖 主 要 是 spring-cloud-starter-netflix-eureka-client 和 spring-cloud-starter-feign，如下：

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

**第三步**：声明服务

定义一个 GoodsRemoteClient 接口，通过@FeignClient 注解来指定服务名称，进而绑定服务，然后再通过 SpringMVC 中提供的注解来绑定服务提供者提供的接口， 如下：

@FeignClient("34-SPRINGCLOUD-SERVICE-GOODS")

public interface GoodsService {

@RequestMapping("/service/goods")

public String goods();

}

这相当于绑定了一个名叫 **34-SPRINGCLOUD-SERVICE-GOODS** (这里

**34-SPRINGCLOUD-SERVICE-GOODS**大小写**34-springcloud-service-goods**

都可以 ) 的服务提供者提供的/service/goods 接口；

**第四步**：添加注解

在项目入口类上添加@EnableFeignClients 注解表示开启 Spring Cloud Feign的支持功能；

**第五步**：使用 Controller 中调用服务

接着来创建一个 Controller 来调用上面的服务，如下：

**public class** GoodsController {  
  
 @Autowired  
 **private** GoodsService **goodsService**;  
  
 */\*\*  
 \* 使用feign进行调用  
 \*  
 \** ***@return*** *\*/* @RequestMapping(**"/cloud/goodsFeign"**)  
 **public** ResultObject goodsFeign() {  
 *//调用远程的一个controller, restful的调用* **return goodsService**.goods();  
 }  
}

**第六步**：属性配置

在 application.properties 中指定服务注册中心、端口号等信息，如下：

**server.port**=**8090***#打开所有的web访问端点***management.endpoints.web.exposure.include**=**\****#此实例注册到eureka服务端的name***spring.application.name**=**34-springcloud-service-feign***#不注册自己，我是一个消费者，别人如果不调用我的话，我就不用注册***eureka.client.register-with-eureka**=**false***#每间隔2s，向服务端发送一次心跳，证明自己依然"存活"***eureka.instance.lease-renewal-interval-in-seconds**=**2***#告诉服务端，如果我10s之内没有给你发心跳，就代表我故障了，将我踢出掉***eureka.instance.lease-expiration-duration-in-seconds**=**10***#告诉服务端，服务实例以IP作为链接，而不是取机器名***eureka.instance.prefer-ip-address**=**true***#告诉服务端，服务实例的唯一ID***eureka.instance.instance-id**=**34-springcloud-service-feign***#eureka注册中心的连接地址  
#eureka.client.service-url.defaultZone=http://eureka8761:8761/eureka  
#eureka.client.service-url.defaultZone=http://eureka8761:8761/eureka,http://eureka8762:8762/eureka,http://eureka8763:8763/eureka***eureka.client.service-url.defaultZone**=**http://192.168.10.128:8761/eureka,http://192.168.10.128:8762/eureka,http://192.168.10.128:8763/eureka**

**第七步**：测试

依次启动注册中心、服务提供者和 feign 实现服务消费者，然后访问如下地址：

<http://localhost:8090/cloud/goodsFeign>

## **使用Feign实现消费者的测试**

**负载均衡：**

我们知道，Spring Cloud 提供了 Ribbon 来实现负载均衡，使用 Ribbo 直接注入一个 RestTemplate 对象即可，RestTemplate 已经做好了负载均衡的配置；

在 Spring Cloud 下，使用 Feign 也是直接可以实现负载均衡的，定义一个有@FeignClient 注解的接口，然后使用@RequestMapping 注解到方法上映射远程的 REST 服务，此方法也是做好负载均衡配置的；

通过feign只需要定义服务绑定接口且以声明式的方法，优雅而简单的实现了服务调用；