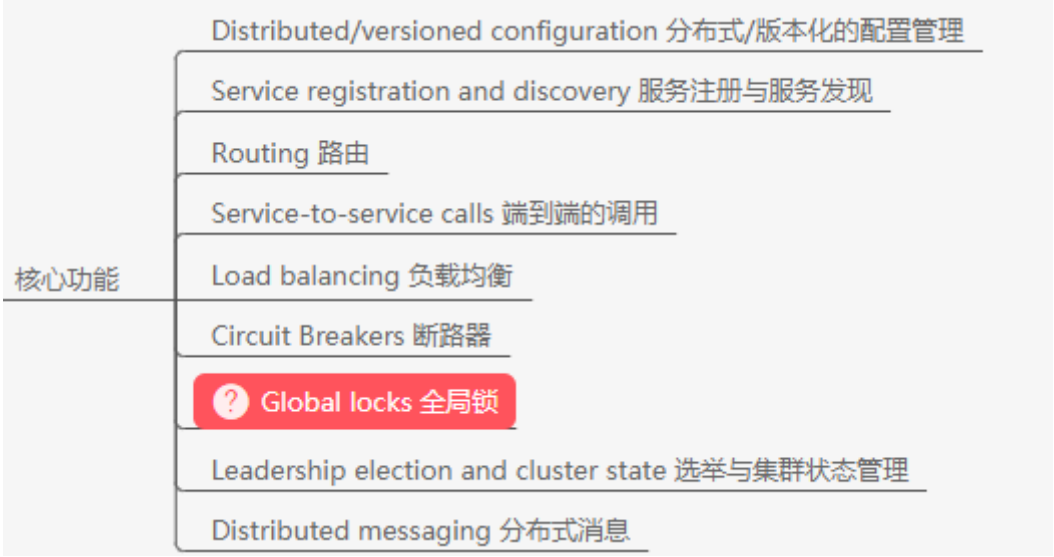
1、什么是 Spring Cloud

构建在 Spring Boot 基础之上，用于快速构建分布式系统的通用模式的工具集。

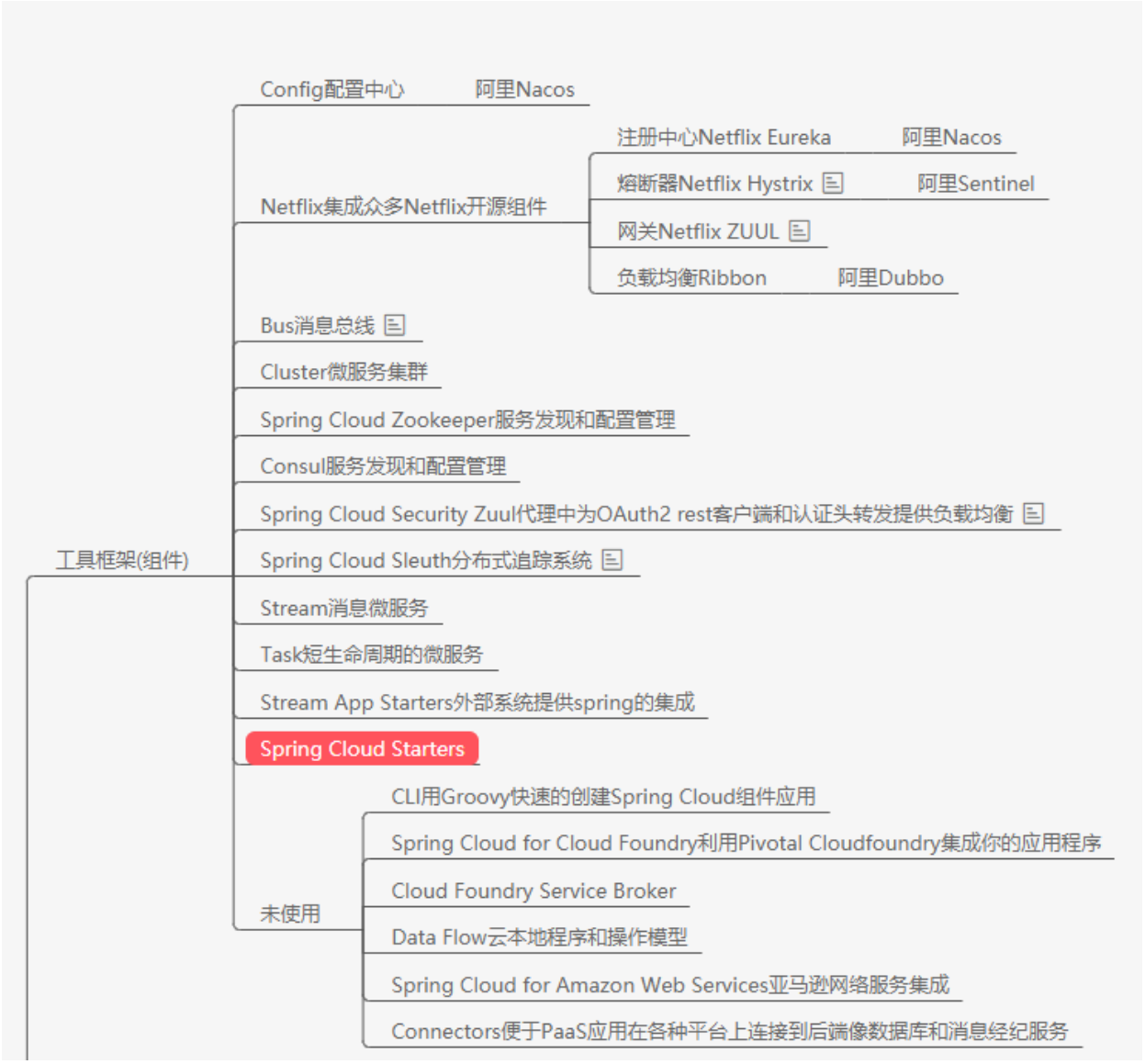
或者说，换成大家更为熟知的，用于构建微服务的技术栈。

2、Spring Cloud 核心功能是什么

- Distributed/versioned configuration 分布式/版本化的配置管理
- Service registration and discovery 服务注册与服务发现
- Routing 路由
- Service-to-service calls 端到端的调用
- Load balancing 负载均衡
- Circuit Breakers 断路器
- Global locks 全局锁
- Leadership election and cluster state 选举与集群状态管理
- Distributed messaging 分布式消息



3、Spring Cloud 有哪些组件



4、Spring Cloud 和 Spring Boot 的区别和关系

- Spring Boot，专注于快速，方便的开发单个微服务个体。
- Spring Cloud，关注全局的服务治理框架。

5、Spring Cloud 与 Dubbo 怎么选择

国内外

- 对于国外，Spring Cloud 基本已经统一国外的微服务体系。
- 对于国内，老的系统使用 Dubbo 较多，新的系统使用 Spring Cloud 较多。

	SprigCloud	Alibaba	其他
注册中心	Eureka	Nacos	Zookeeper、Consul、Etcd
配置中心	Spring Cloud Config	Nacos	
熔断器	Hystrix	Sentinel	Resilience4j
负载均衡	Ribbon	Dubbo	spring-cloud-loadbalancer
消息服务	Spring Cloud Stream	RocketMQ	

消息微服务	Spring Cloud Stream	RocketMQ	
网关	Zuull		Spring Cloud Gateway

6、区别于系统，服务一个或者一组相对较小且独立的功能单元，是用户可以感知最小功能集。

7、微服务的优缺点分别是什么

1）优点

- 每一个服务足够内聚,代码容易理解
- 开发效率提高，一个服务只做一件事
- 微服务能够被小团队单独开发
- 微服务是松耦合的，是有功能意义的服务
- 可以用不同的语言开发,面向接口编程
- 易于与第三方集成
- 微服务只是业务逻辑的代码，不会和 HTML、CSS 或者其他界面组合
    - 开发中，两种开发模式
        - 前后端分离
        - 全栈工程师
- 可以灵活搭配,连接公共库/连接独立库

2）缺点

- 分布式系统的负责性
- 多服务运维难度，随着服务的增加，运维的压力也在增大
- 系统部署依赖
- 服务间通信成本
- 数据一致性
- 系统集成测试
- 性能监控

8、注册中心

- spring-cloud-netflix-eureka-server 和 spring-cloud-netflix-eureka-client ，基于 Eureka 实现。
- spring-cloud-alibaba-nacos-discovery ，基于 Nacos 实现。
- spring-cloud-zookeeper-discovery ，基于 Zookeeper 实现。

以上的实现，都是基于 spring-cloud-commons 的 discovery 的 DiscoveryClient 接口，实现统一的客户端的注册发现

9、为什么要使用服务发现？

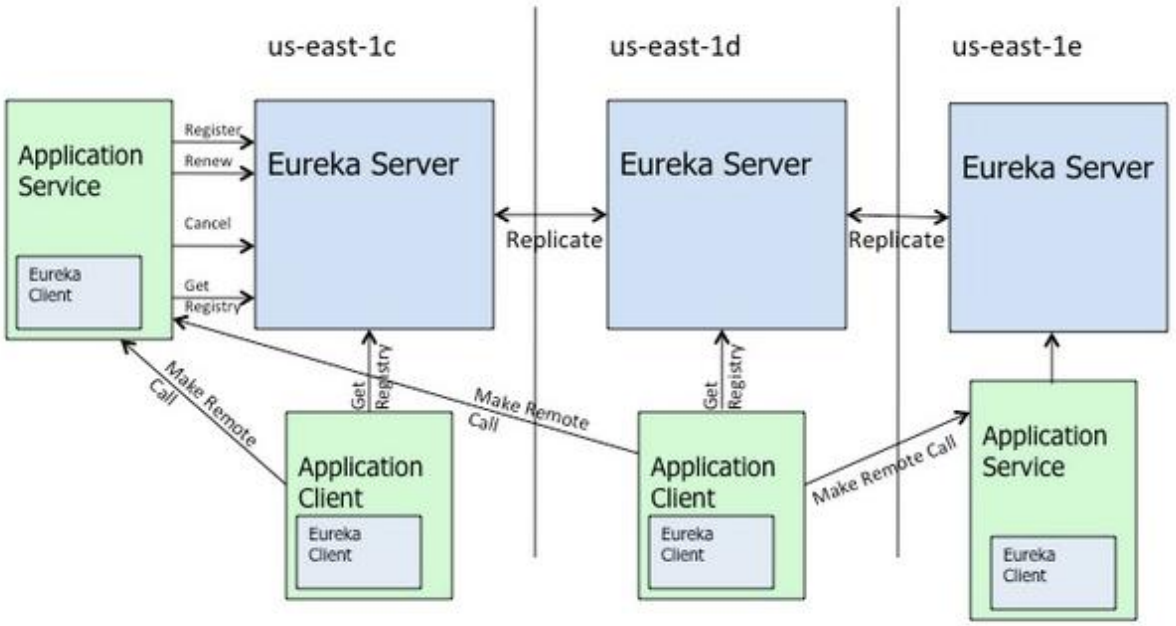
简单来说，通过注册中心，调用方(Consumer)获得服务方(Provider)的地址，从而能够调用。

当然，实际情况下，会分成两种注册中心的发现模式：

1. 客户端发现模式
2. 服务端发现模式

服务发现有两种主要模式：客户端发现和服务端发现。在使用客户端服务发现的系统中，客户端查询服务注册表，选择可用的服务实例，然后发出请求。在使用服务端发现的系统中，客户端通过路由转发请求，路由器查询服务注册表并转发请求到可用的实例。

10、Eureka



- 作用：实现服务治理（服务注册与发现）
- 简介：Spring Cloud Eureka是Spring Cloud Netflix项目下的服务治理模块。
- 由两个组件组成：Eureka 服务端和 Eureka 客户端。
    - Eureka 服务端，用作服务注册中心，支持集群部署。
    - Eureka 客户端，是一个 Java 客户端，用来处理服务注册与发现。

在应用启动时，Eureka 客户端向服务端注册自己的服务信息，同时将服务端的服务信息缓存到本地。客户端会和服务端周期性的进行心跳交互，以更新服务租约和服务信息。

11、Eureka 如何实现集群

https://www.jianshu.com/p/5d5b2cf7d476

12、什么是 Eureka 自我保护机制

当 Eureka Server 节点在短时间内丢失了过多实例的连接时（比如网络故障或频繁的启动关闭客户端），那么这个节点就会进入自我保护模式，一旦进入到该模式，Eureka server 就会保护服务注册表中的信息，不再删除服务注册表中的数据（即不会注销任何微服务），当网络故障恢复后，该 Ereaka Server 节点就会自动退出自我保护模式

13、负载均衡

- spring-cloud-netflix-ribbon，基于 Ribbon 实现。
- spring-cloud-loadbalancer，提供简单的负载均衡功能。

以上都基于 spring-cloud-commons 的 loadbalancer 的 ServiceInstanceChooser 接口

14、为什么要负载均衡

简单来说，随着业务的发展，单台服务无法支撑访问的需要，于是搭建多个服务形成集群。那么随之要解决的是，每次请求，调用哪个服务，也就是需要进行负载均衡。

目前负载均衡有两种模式：

1. 客户端模式
2. 服务端模式

在 Spring Cloud 中，我们使用前者，即客户端模式。

服务端负载均衡又分为两种，一种是硬件负载均衡，还有一种是软件负载均衡。

硬件负载均衡主要通过在服务器节点之间安装专门用于负载均衡的设备，常见的如F5。

软件负载均衡则主要是在服务器上安装一些具有负载均衡功能的软件来完成请求分发进而实现负载均衡，常见的就是Nginx。

客户端负载均

Ribbo是一个基于HTTP和TCP的客户端负载均衡器，当我们将Ribbon和Eureka一起使用时，Ribbon会从Eureka注册中心去获取服务端列表，然后进行轮询访问以到达负载均衡的作用

客户端负载均衡和服务端负载均衡最大的区别在于服务清单所存储的位置。在客户端负载均衡中，所有的客户端节点都有一份自己要访问的服务端清单，这些清单统统都是从Eureka服务注册中心获取的

Ribbon

15、Ribbon 有哪些负载均衡算法

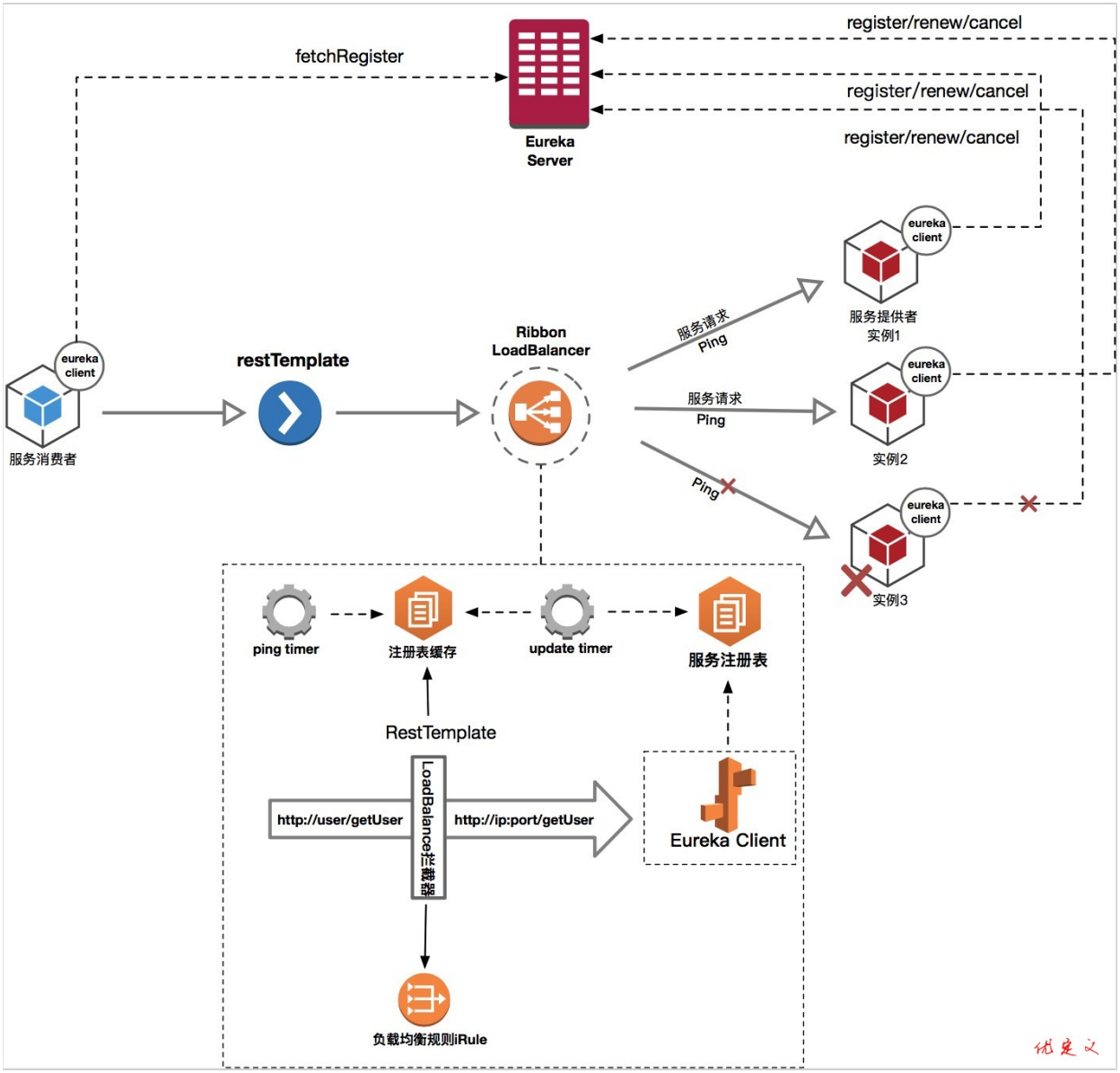
16、Ribbon 缓存机制

17、Ribbon 重试机制

重试次数，还有请求的超时可以配置

```
# 开启重试
zuul.retryable=true
spring.cloud.loadbalancer.retry.enabled=true
# 切换实例的重试次数
ribbon.maxAutoRetriesNextServer=3
# 对所有操作请求都进行重试
ribbon.okToRetryOnAllOperations=true
```

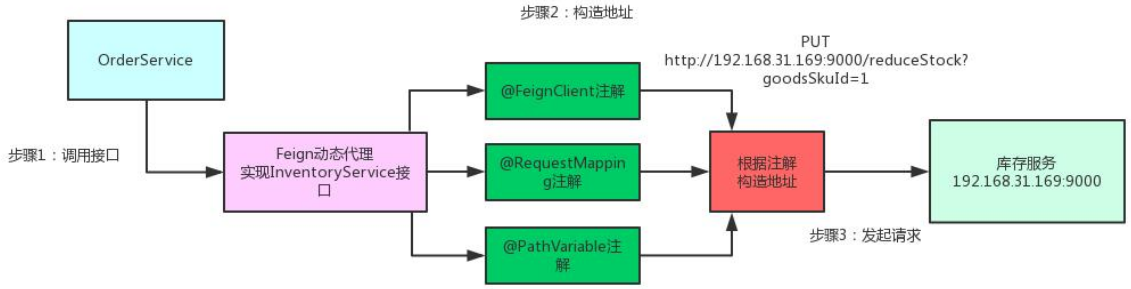
18、Ribbon 是怎么和 Eureka 整合的



- 首先，Ribbon 会从 Eureka Client 里获取到对应的服务列表。
- 然后，Ribbon 使用负载均衡算法获得使用的服务。
- 最后，Ribbon 调用对应的服

19、Feign 实现原理

- 首先，如果你对某个接口定义了 @FeignClient 注解，Feign 就会针对这个接口创建一个动态代理。
- 接着你要是调用那个接口，本质就是会调用 Feign 创建的动态代理，这是核心中的核心。
- Feig n的动态代理会根据你在接口上的 @RequestMapping 等注解，来动态构造出你要请求的服务的地址。
- 最后针对这个地址，发起请求、解析响应。

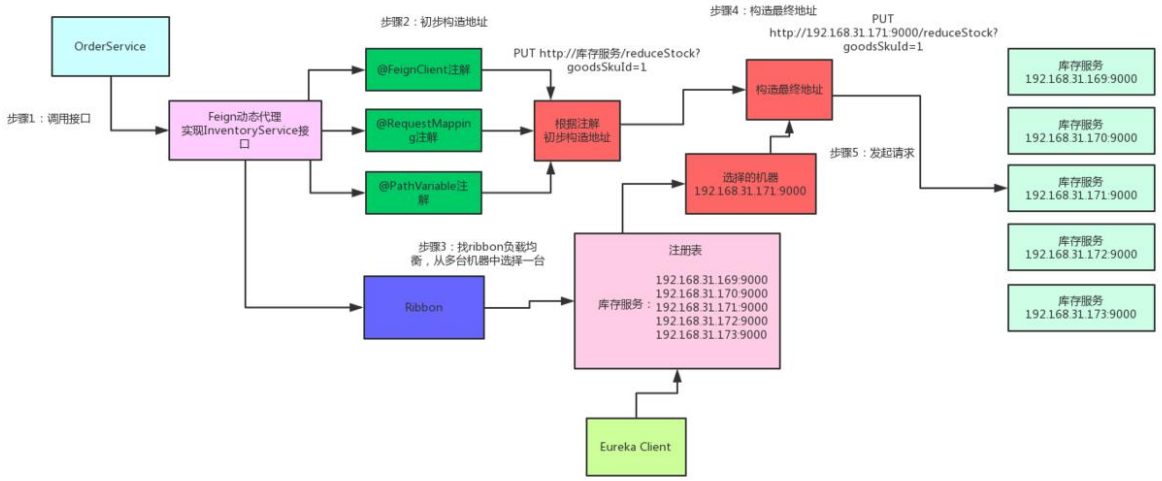


20、Feign 和 Ribbon 的区别?

Ribbon 和 Feign 都是使用于调用用其余服务的，不过方式不同。

- 启动类用的注解不同。
  - Ribbon 使用的是 @RibbonClient 。
  - Feign 使用的是 @EnableFeignClients 。
- 服务的指定位置不同。
  - Ribbon 是在 @RibbonClient 注解上设置。
  - Feign 则是在定义声明方法的接口中用 @FeignClient 注解上设置。
- 调使用方式不同。
  - Ribbon 需要自己构建 Http 请求，模拟 Http 请求而后用 RestTemplate 发送给其余服务，步骤相当繁琐。
  - Feign 采使用接口的方式，将需要调使用的其余服务的方法定义成声明方法就可，不需要自己构建 Http 请求。不过要注意的是声明方法的注解、方法签名要和提供服务的方法完全一致。

21、Feign 是怎么和 Ribbon、Eureka 整合的



- 首先，用户调用 Feign 创建的动态代理。
- 然后，Feign 调用 Ribbon 发起调用流程。
  - 首先，Ribbon 会从 Eureka Client 里获取到对应的服务列表。

- 然后，Ribbon 使用负载均衡算法获得使用的服务。
- 最后，Ribbon 调用对应的服务。最后，Ribbon 调用 Feign，而 Feign 调用 HTTP 库最终调用使用的服务。

这可能是比较绕的，芳芳自己也困惑了一下，后来去请教了下 didi 。因为 Feign 和 Ribbon 都存在使用 HTTP 库调用指定的服务，那么两者在集成之后，必然是只能保留一个。比较正常的理解，也是保留 Feign 的调用，而 Ribbon 更纯粹的只负责负载均衡的功能。

想要完全理解，建议胖友直接看如下两个类：

- LoadBalancerFeignClient，Spring Cloud 实现 Feign Client 接口的二次封装，实现对 Ribbon 的调用。
- FeignLoadBalancer，Ribbon 的集成。

集成的是 AbstractLoadBalancerAwareClient 抽象类，它会自动注入项目中所使用的负载均衡组件。

- LoadBalancerFeignClient =》调用=》 FeignLoadBalancer 。

22、什么是 Hystrix 断路器

Hystrix 断路器通过 HystrixCircuitBreaker 实现

HystrixCircuitBreaker 有三种状态 ：

- `CLOSED` ： 关闭
- `OPEN` ： 打开
- `HALF_OPEN` ： 半开

其中，断路器处于 `OPEN` 状态时，链路处于**非健康**状态，命令执行时，直接调用**回退**逻辑，跳过**正常**逻辑。

23、什么是 Hystrix 服务降级

https://blog.csdn.net/jiaobuchong/article/details/78232920