Spring Cloud 面试题（Beta）

以下面试题，基于网络整理，和自己编辑。具体参考的文章，会在文末给出所有的链接。

如果胖友有自己的疑问，欢迎在星球提问，我们一起整理吊吊的 Spring Cloud 面试题的大保健。

而题目的难度，尽量按照从容易到困难的顺序，逐步下去。

> 为什么标题会带有 Beta 呢?
>
> 因为自己生产并未使用 Spring Cloud ，而是使用 Dubbo 。所以，在实战经验方面，会相对匮乏。所以，这篇面试题，献丑了~

另外，本文不会详细解答每个问题，而是提供很多问题回答的文章。等后续详细的研究了 Spring Cloud Alibaba 后，补充好这篇文章~

什么是 Spring Cloud ?

Spring Cloud 是构建在 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。

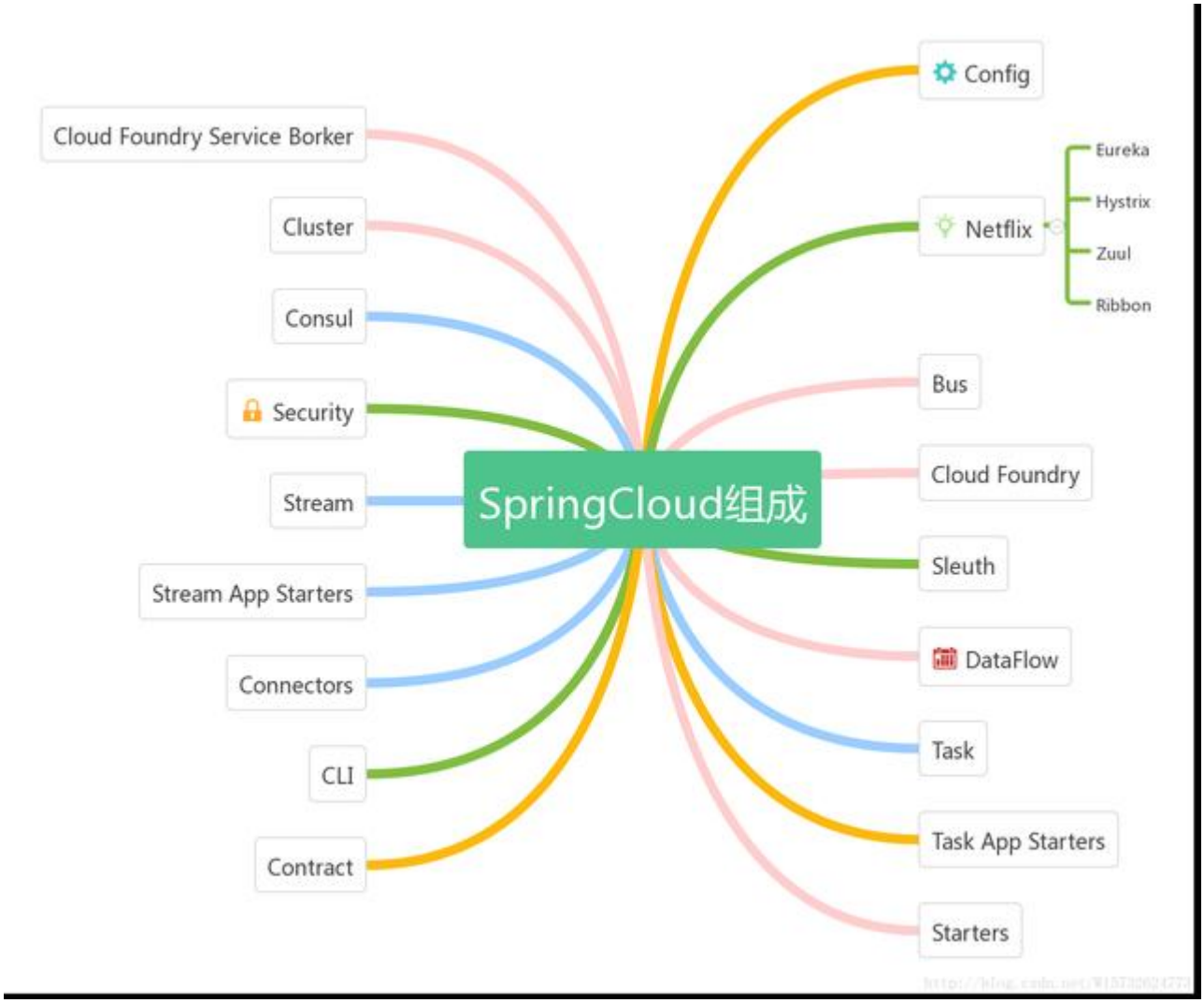
Spring Cloud 核心功能是什么?

毫无疑问，Spring Cloud 可以说是目前微服务架构的最好的选择，涵盖了基本我们需要的所有组件，所以也被称为全家桶。Spring Cloud 主要提供了如下核心的功能：

- 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

Spring Cloud 有哪些组件?

Spring Cloud的 组件相当繁杂，拥有诸多子项目。如下脑图所示：



我们最为熟知的，可能就是 Spring Cloud Netflix ，它是 Netflix 公司基于它们自己的 Eureka、Hystrix、Zuul、Ribbon 等组件，构建的一个 Spring Cloud 实现技术栈。

当然，可能关心 Spring Cloud 体系的胖友，已经知道 Spring Cloud Netflix 要进入维护模式，可能会略感担心。实际上，目前已经开始有新的基于 Spring Cloud 实现，可以作为新的选择。如下表格：

> 《Spring Cloud Netflix 项目进入维护模式》，感兴趣的胖友，可以看看新闻。

	Netflix	阿里	其它
注册中心	Eureka	Nacos	Zookeeper、Consul、Etcd
熔断器	Hystrix	Sentinel	Resilience4j
网关	Zuul1	暂无	Spring Cloud Gateway
负载均衡	Ribbon	Dubbo(未来)	`spring-cloud-loadbalancer`

其它组件，例如配置中心、链路追踪、服务引用等等，都有相应其它的实现。妥妥的~

Spring Cloud 和 Spring Boot 的区别和关系?

1. Spring Boot 专注于快速方便的开发单个个体微服务。

2. Spring Cloud 是关注全局的微服务协调整理治理框架以及一整套的落地解决方案，它将 Spring Boot 开发的一个个单体微服务整合并管理起来，为各个微服务之间提供：配置管理，服务发现，断路器，路由，微代理，事件总线等的集成服务。
3. Spring Boot 可以离开 Spring Cloud 独立使用，但是 Spring Cloud 离不开 Spring Boot ，属于依赖的关系。

总结：

- 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。

Spring Cloud 和 Dubbo 的区别?

参见 《精尽 Dubbo 面试题》 文章的 「Spring Cloud 与 Dubbo 怎么选择？」 问题的解答。

什么是微服务?

直接看 什么是微服务？ 文章。

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

注册中心

在 Spring Cloud 中，能够使用的注册中心，还是比较多的，如下：

- `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。
- ... 等等

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。

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。

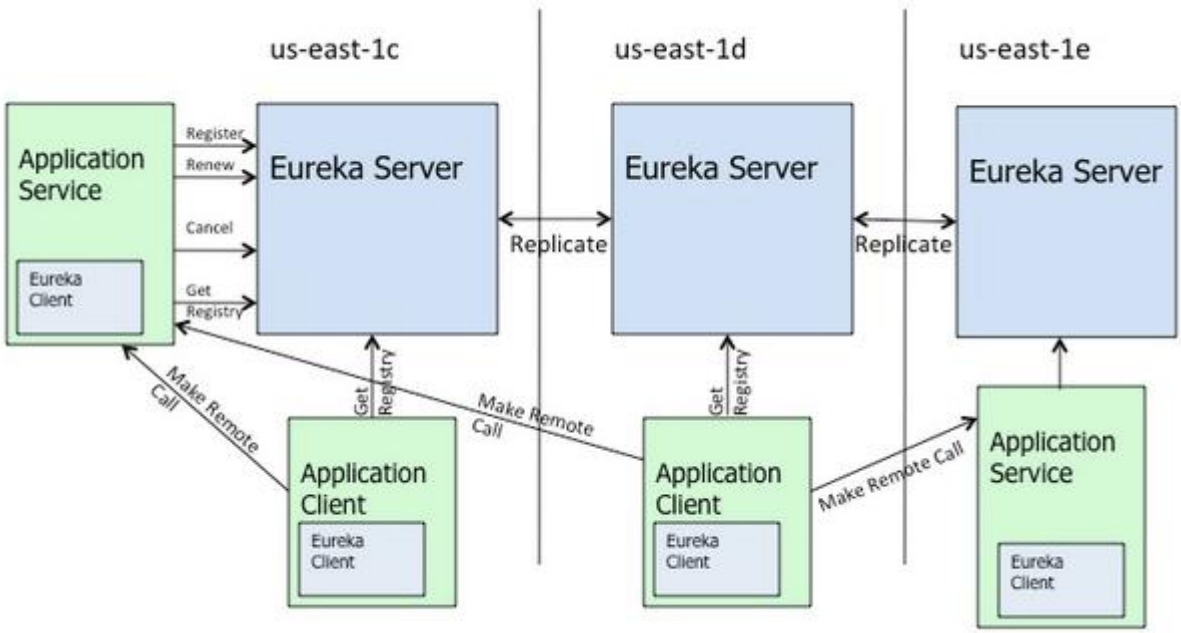
当然，实际情况下，会分成两种注册中心的发现模式：

1. 客户端发现模式
2. 服务端发现模式

在 Spring Cloud 中，我们使用前者，即客户端发现模式。

详细的内容，可以看看 《为什么要使用服务发现》 。

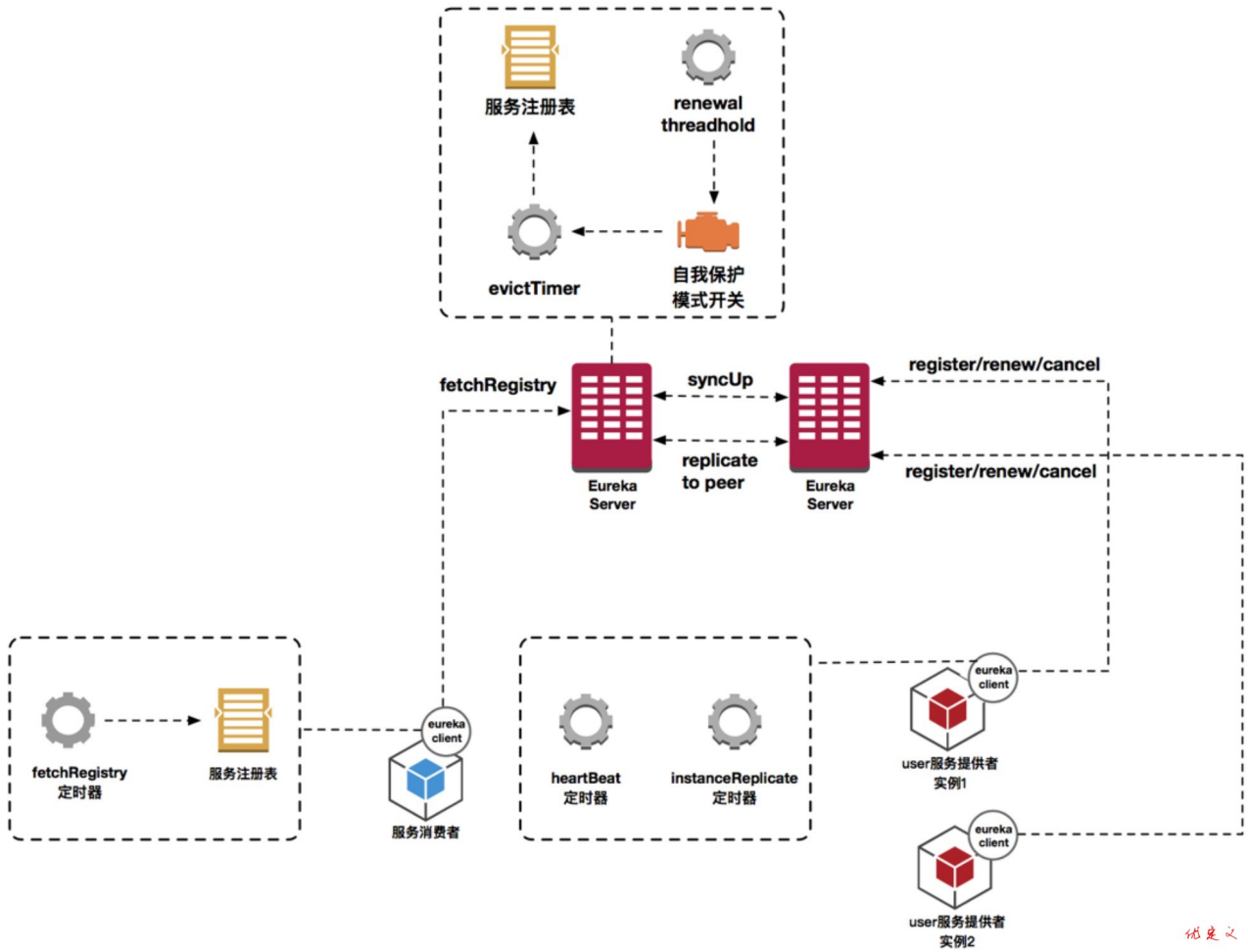
Eureka



- 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。

Eureka 原理，整体如下图：



关于 Eureka 的源码解析，可以看看写的 《Eureka 源码解析系列》 。

Eureka 如何实现集群?

《配置 Eureka Server 集群》

此处，也很容易引申出一个问题，为什么 Eureka 被设计成 AP 的系统，答案可以看看 《为什么不应该使用 ZooKeeper 做服务发现》 。

聊聊 Eureka 缓存机制?

《Eureka 缓存细节以及生产环境的最佳配置》

什么是 Eureka 自我保护机制?

[《Spring Cloud] Eureka 的自我保护模式及相关问题》

负载均衡

在 Spring Cloud 中，能够使用的负载均衡，如下：

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

以上的实现，都是基于 `spring-cloud-commons` 的 `loadbalancer` 的 ServiceInstanceChooser 接口，实现统一的服务的选择。并且，负载均衡组件在选择需要调用的服务之后，还提供调用该服务的功能，具体方法见 LoadBalancerClient 接口的 `#execute(...)` 方法。

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

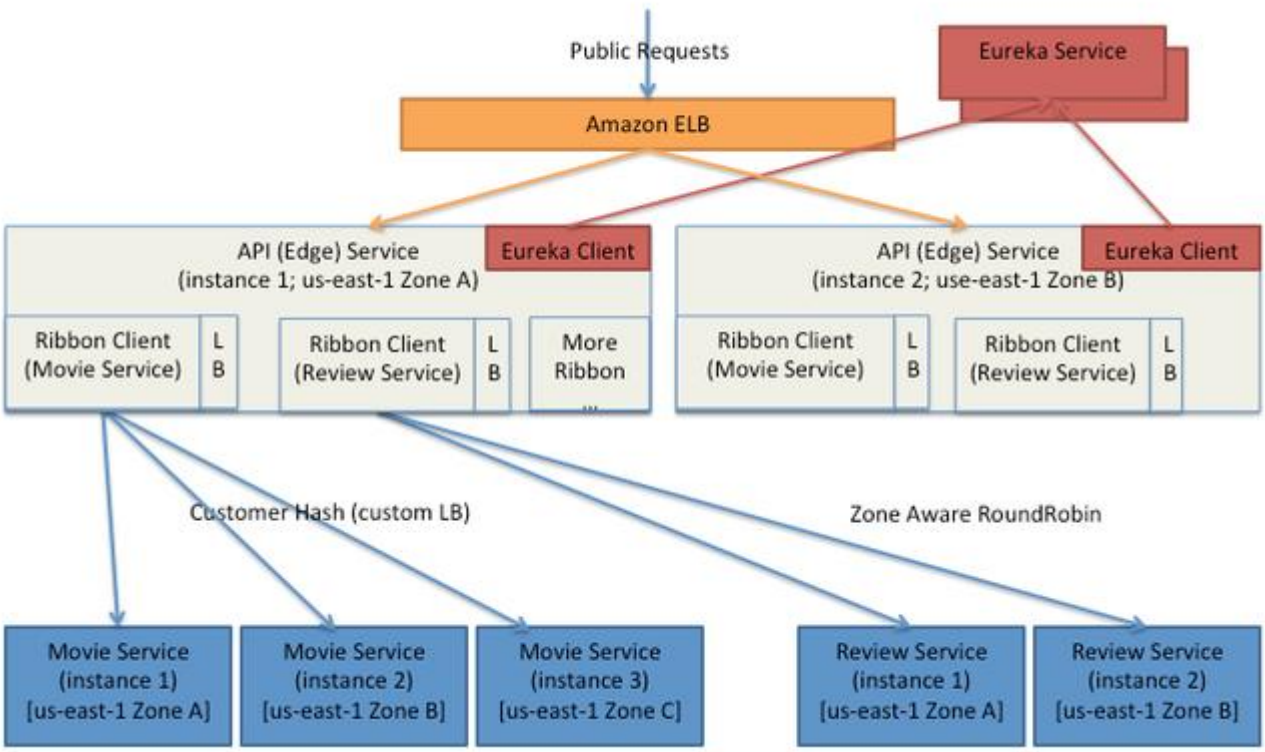
在 Spring Cloud 中，我们使用前者，即客户端模式。

详细的内容，可以看看 《客户端负载均衡与服务端负载均衡》 。

负载平衡的意义什么?

在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

Ribbon

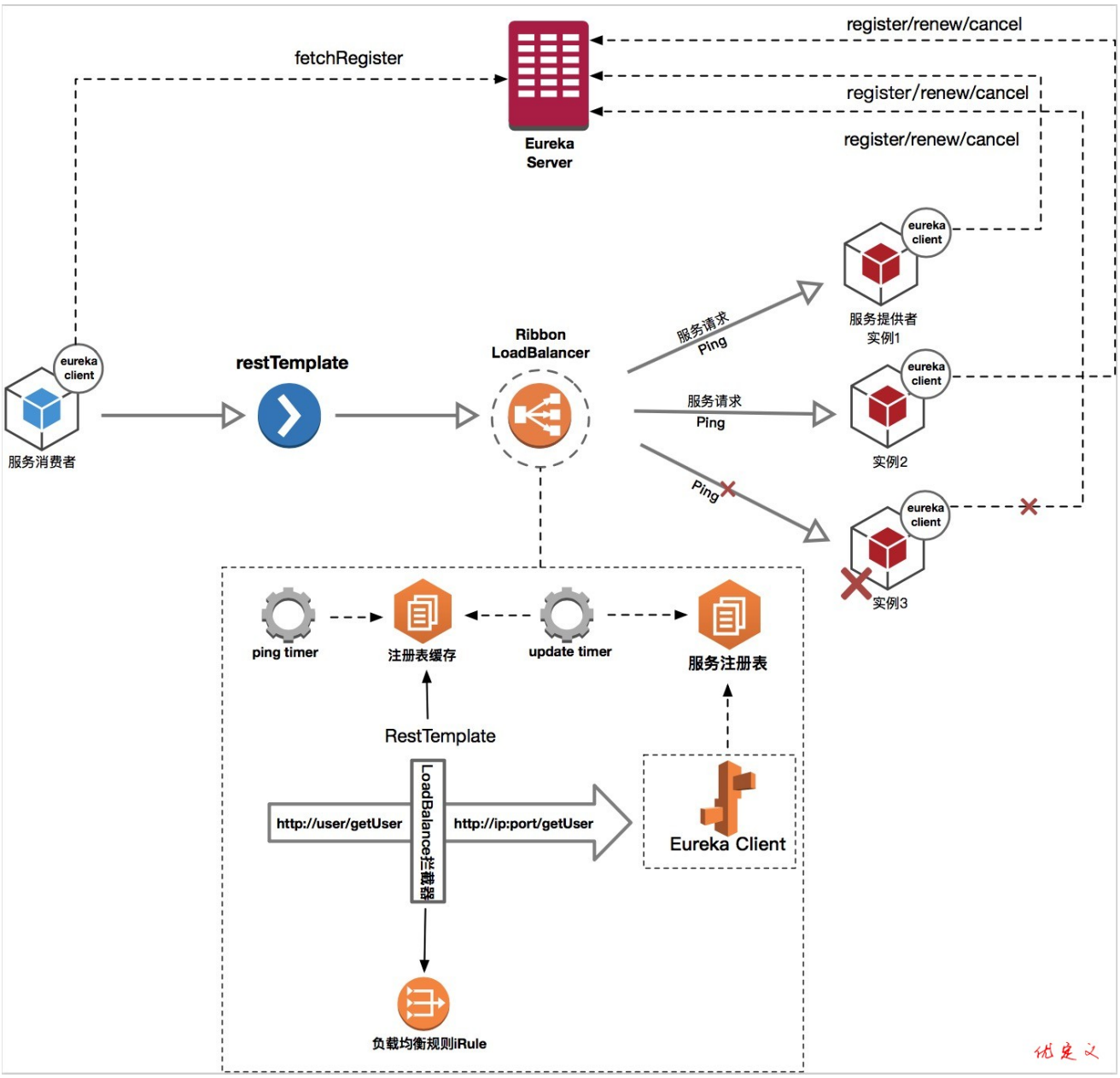


- 作用：主要提供客户侧的软件负载均衡算法。
- 简介：Spring Cloud Ribbon 是一个基于 HTTP 和 TCP 的客户端负载均衡工具，它基于 Netflix Ribbon 实现。通过 Spring Cloud 的封装，可以让我们轻松地将面向服务的 REST 模版请求自动转换成客户端负载均衡

的服务调用。

- 注意看上图，关键点就是将外界的 rest 调用，根据负载均衡策略转换为微服务调用。

Ribbon 原理，整体如下图：



关于 Ribbon 的源码解析，可以看看整理的 《Ribbon 源码解析系列》 。

Ribbon 有哪些负载均衡算法?

《Ribbon 负载均衡策略配置》

其中，默认的负载均衡算法是 Round Robin 算法，顺序向下轮询。

聊聊 Ribbon 缓存机制?

还是 《Eureka 缓存细节以及生产环境的最佳配置》 这篇文章，Ribbon 的缓存，可能也坑道蛮多人了。

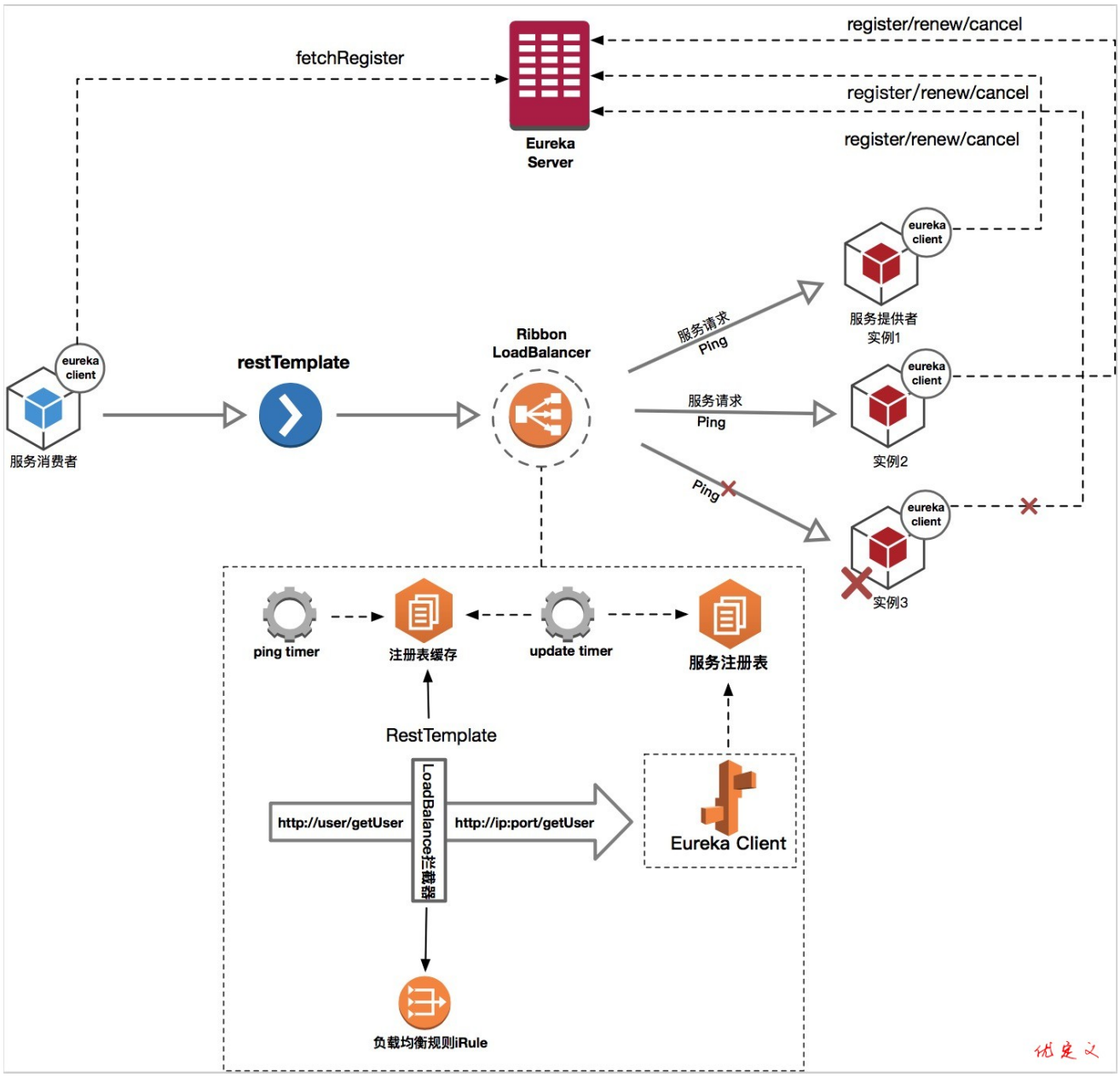
聊聊 Ribbon 重试机制?

《Spring Cloud Ribbon 重试机制》

除了重试次数，还有请求的超时可以配置。

Ribbon 是怎么和 Eureka 整合的?

对着我们看到那张 Ribbon 原理图：



- 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。

另外，此处的 Eureka 仅仅是作为注册中心的举例，也是可以配合其它的注册中心使用，例如 Zookeeper 。可参考《以 Zookeeper 为注册中心搭建 Spring Cloud 环境》文章。

声明式调用

在 Spring Cloud 中，目前使用的声明式调用组件，只有：

- `spring-cloud-openfeign` ，基于 Feign 实现。

如果熟悉 Dubbo 胖友的会知道，Dubbo 的 Service API 接口，也是一种声明式调用的提现。

> ：注意噢，Feign 并非 Netflix 团队开发的组件。所有基于 Netflix 组件都在 `spring-cloud-netflix` 项目下噢。

Feign

Feign 是受到 Retrofit、JAXRS-2.0 和 WebSocket 启发的 Java 客户端联编程序。Feign 的主要目标是将Java Http 客户端变得简单。

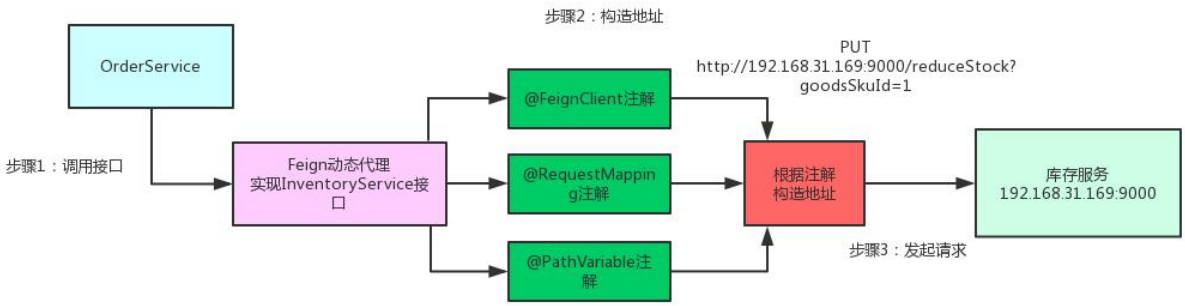
具体 Feign 如何使用，可以看看 《对于 Spring Cloud Feign 入门示例的一点思考》 。

有一点要注意，Feign 并非一定要在 Spring Cloud 下使用，单独使用也是没问题的。

Feign 实现原理?

Feign的一个关键机制就是使用了动态代理。咱们一起来看看下面的图，结合图来分析：

- 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。



Feign 和 Ribbon 的区别?

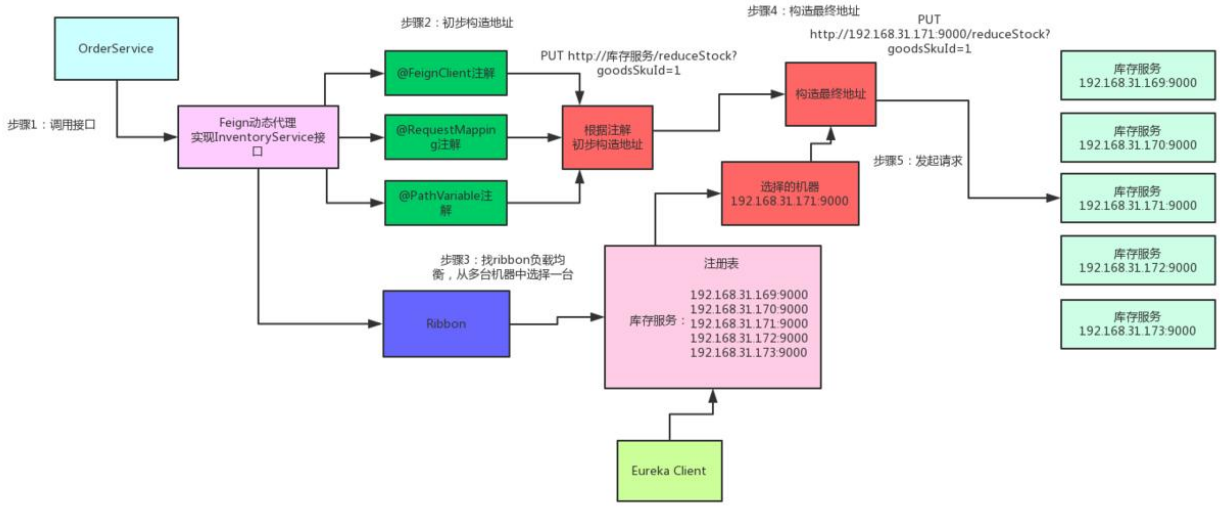
Ribbon 和 Feign 都是使用于调用用其余服务的，不过方式不同。

- 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。

Feign 是怎么和 Ribbon、Eureka 整合的?

结合 「Ribbon 是怎么和 Eureka 整合的？」 问题，并结合如下图：



- 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，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 。

聊聊 Feign 重试机制?

可以看看 《Spring Cloud 各组件重试总结》 文章。因为 Ribbon 和 Feign 都有重试机制，在整合 Ribbon 的情况下，不使用 Feign 重试，而是使用 Ribbon 的重试。

服务保障

在 Spring Cloud 中，能够使用的服务保证，如下：

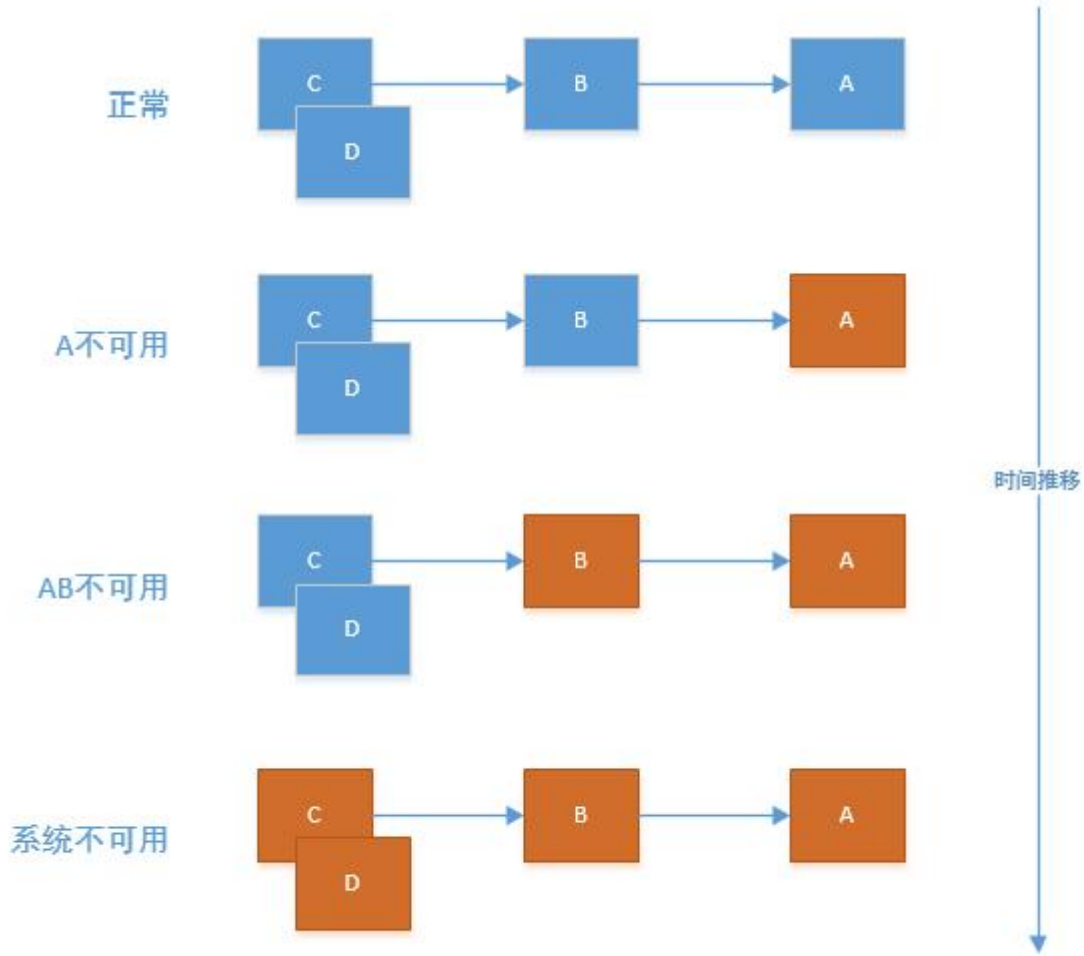
- `spring-cloud-netflix-hystrix` ，基于 Hystrix 实现。
- Resilience4j
- `spring-cloud-alibaba-sentinel` ，基于 Sentinel 实现。

为什么要使用服务保障?

在微服务架构中，我们将业务拆分成一个个的服务，服务与服务之间可以相互调用（RPC）。为了保证其高可用，单个服务又必须集群部署。由于网络原因或者自身的原因，服务并不能保证服务的 100% 可用，如果单个服务出现问题，调用这个服务就会出现网络延迟，此时若有大量的网络涌入，会形成任务累积，导致服务瘫痪，甚至导致服务“雪崩”。为了解决这个问题，就出现断路器模型。

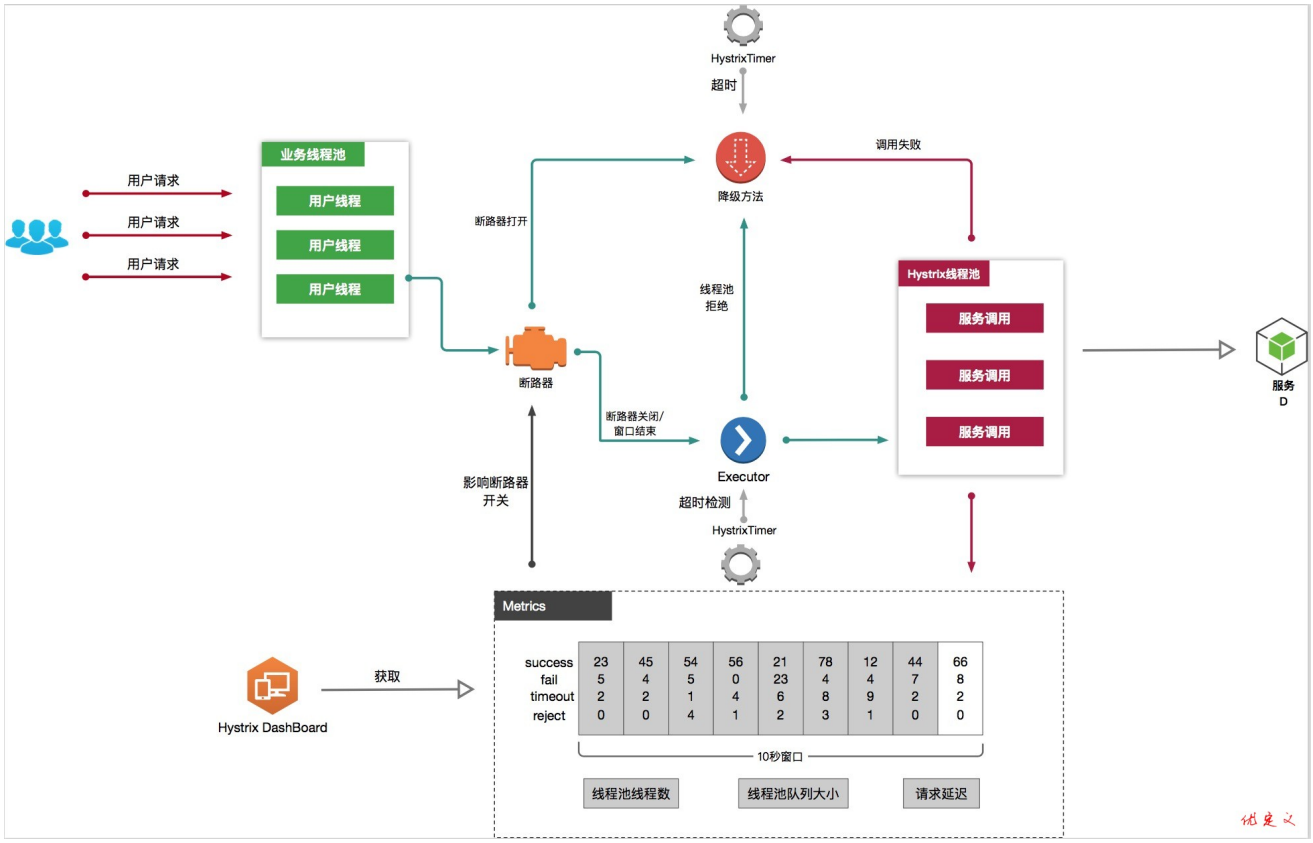
详细的内容，可以看看 《为什么要使用断路器 Hystrix？》 。

Hystrix



- 作用：断路器，保护系统，控制故障范围。
- 简介：Hystrix 是一个延迟和容错库，旨在隔离远程系统，服务和第三方库的访问点，当出现故障是不可避免的故障时，停止级联故障并在复杂的分布式系统中实现弹性。

Hystrix 原理，整体如下图：



关于 Hystrix 的源码解析，可以看看写的 《Hystrix 源码解析系列》 。

Hystrix 隔离策略?

Hystrix 有两种隔离策略：

- 线程池隔离
- 信号量隔离

实际场景下，使用线程池隔离居多，因为支持超时功能。

详细的，可以看看 《Hystrix 的资源隔离策略》 文章。

聊聊 Hystrix 缓存机制?

Hystrix 提供缓存功能，作用是：

- 减少重复的请求数。
- 在同一个用户请求的上下文中，相同依赖服务的返回数据始终保持一致。

详细的，可以看看 《Hystrix 缓存功能的使用》 文章。

什么是 Hystrix 断路器?

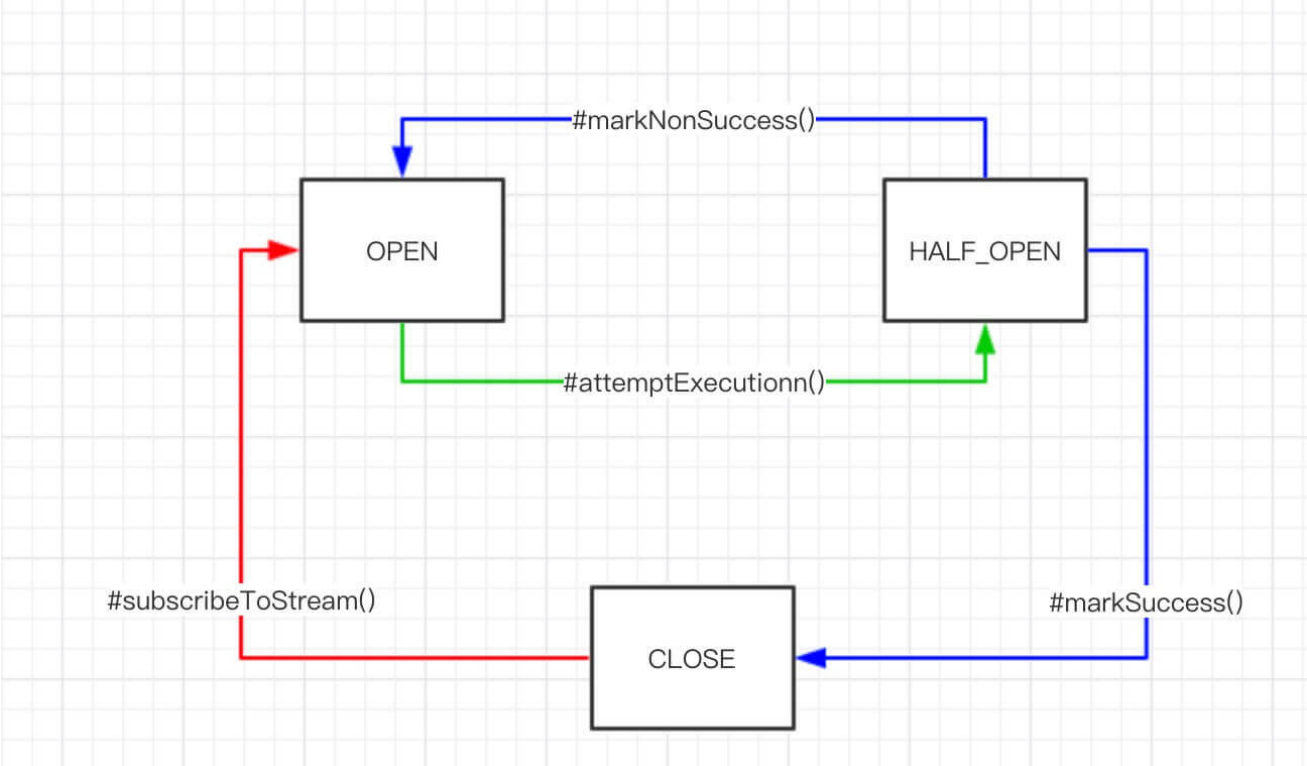
Hystrix 断路器通过 HystrixCircuitBreaker 实现。

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。

HystrixCircuitBreaker 状态变迁如下图：



- 红线

  ：初始时，断路器处于

  ```
  CLOSED
  ```

  状态，链路处于

  健康

  状态。当满足如下条件，断路器从

  ```
  CLOSED
  ```

  变成

  ```
  OPEN
  ```

  状态：

  - **周期**(可配，`HystrixCommandProperties.default_metricsRollingStatisticalWindow = 10000 ms`)内，总请求数超过一定**量**(可配，`HystrixCommandProperties.circuitBreakerRequestVolumeThreshold = 20`)。
  - **错误**请求占总请求数超过一定**比例**(可配，`HystrixCommandProperties.circuitBreakerErrorThresholdPercentage = 50%`)。

- **绿线**：断路器处于 `OPEN` 状态，命令执行时，若当前时间超过断路器**开启**时间一定时间(`HystrixCommandProperties.circuitBreakerSleepWindowInMilliseconds = 5000 ms`)，断路器变成 `HALF_OPEN` 状态，**尝试**调用**正常**逻辑，根据执行是否成功，**打开或关闭**熔断器**【蓝线】**。

什么是 Hystrix 服务降级?

在 Hystrix 断路器熔断时，可以调用一个降级方法，返回相应的结果。当然，降级方法需要配置和编码，如果胖友不需要，也可以不写，也就是不会有服务降级的功能。

具体的使用方式，可以看看 《通过 Hystrix 理解熔断和降级》 。

网关服务

在 Spring Cloud 中，能够使用的网关服务，主要是两个，如下：

- `spring-cloud-netflix-zuul` ，基于 Zuul1 实现。

  > Netflix 最新开源的网关服务是 Zuul2 ，基于响应式的网关服务。

- `spring-cloud-gateway` ，基于 Spring Webflux 实现。

  > ：比较大的可能性，是未来 Spring Cloud 网关的主流选择。考虑到目前资料的情况，建议使用 Zuul1 可能是更稳妥的选择，因为 Zuul1 已经能满足绝大数性能要求，实在不行也可以集群。

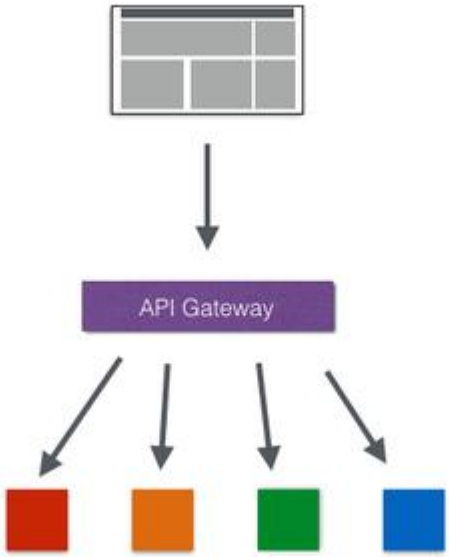
为什么要网关服务?

使用网关服务，我们实现统一的功能：

- 动态路由
- 灰度发布
- 健康检查
- 限流
- 熔断
- 认证: 如数支持 HMAC, JWT, Basic, OAuth 2.0 等常用协议
- 鉴权: 权限控制，IP 黑白名单，同样是 OpenResty 的特性
- 可用性
- 高性能

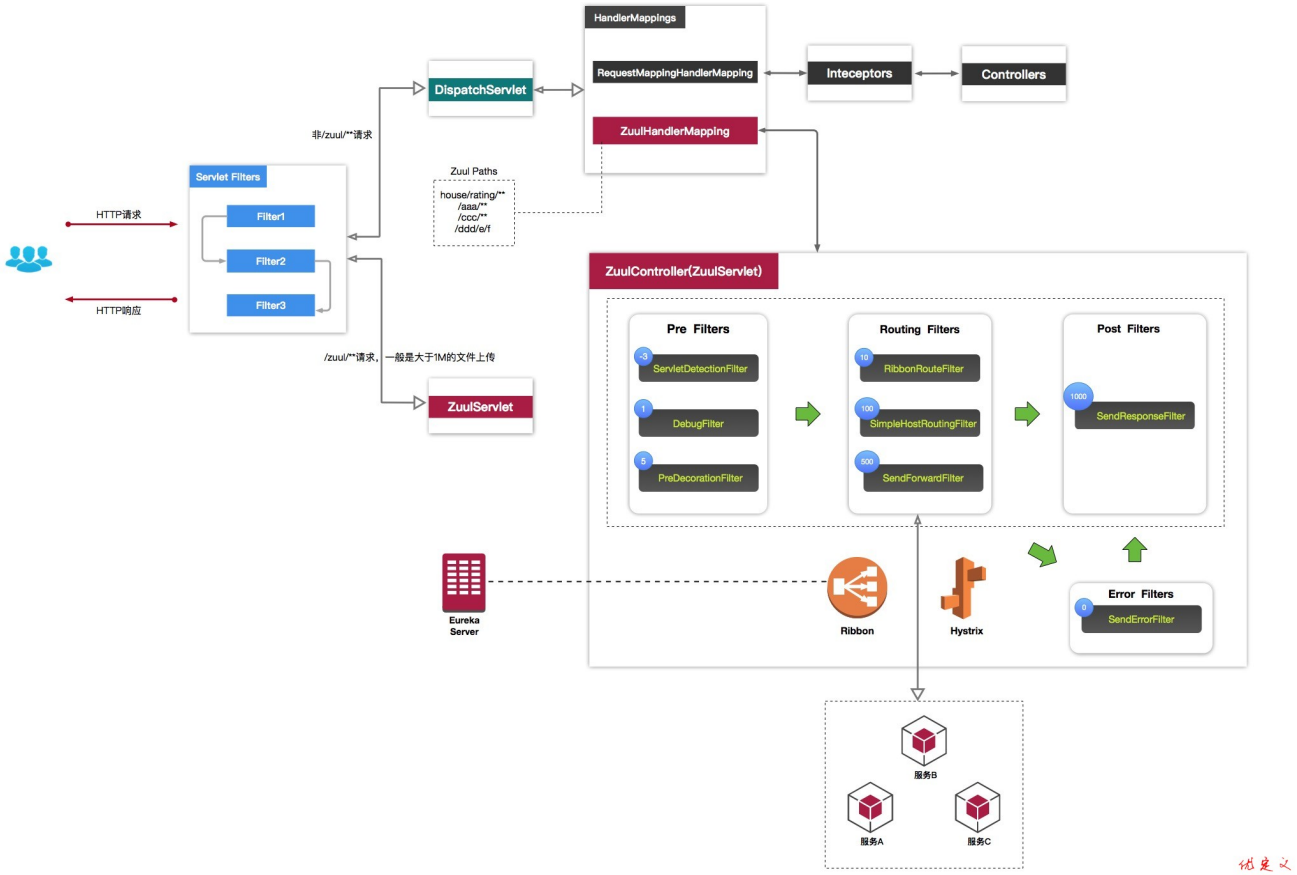
详细的，可以看看 《为什么微服务需要 API 网关？》 。

Zuul



- 作用：API 网关，路由，负载均衡等多种作用。
- 简介：类似 Nginx ，反向代理的功能，不过 Netflix 自己增加了一些配合其他组件的特性。
- 在微服务架构中，后端服务往往不直接开放给调用端，而是通过一个 API网关根据请求的 url ，路由到相应的服务。当添加API网关后，在第三方调用端和服务提供方之间就创建了一面墙，这面墙直接与调用方通信进行权限控制，后将请求均衡分发给后台服务端。

Zuul 原理，整体如下图：



关于 Zuul 的源码解析，可以看看整理的 《Zuul 源码解析系列》 。

Spring Cloud Gateway

关于 Spring Cloud Gateway 的源码解析，可以看看写的 《Spring Cloud Gateway 源码解析系列》 。

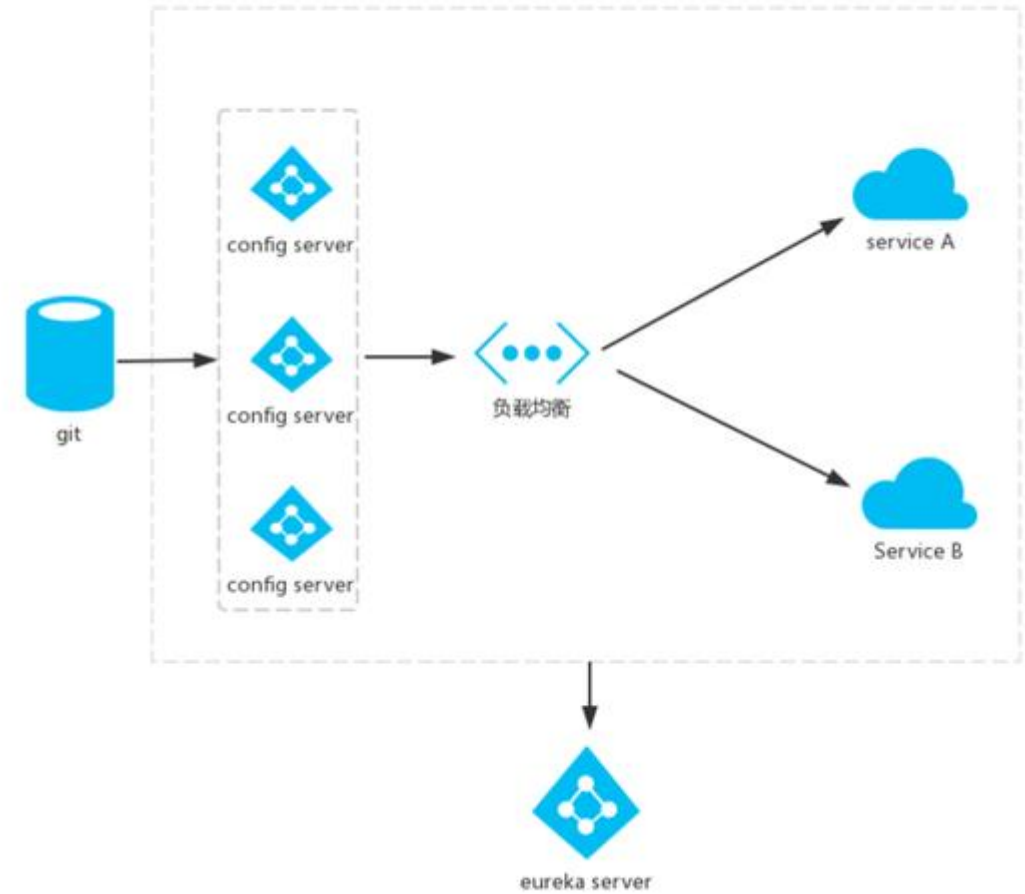
配置中心

在 Spring Cloud 中，能够使用的配置中心，如下：

- `spring-cloud-config` ，基于 Git、SVN 作为存储。
- `spring-cloud-alibaba-nacos-config` ，基于 Nacos 实现。
- Apollo ，携程开源的配置中心。

  > ：目前 Spring Cloud 最成熟的配置中心的选择。

Spring Cloud Config



- 作用：配置管理
- 简介：Spring Cloud Config 提供服务器端和客户端。服务器存储后端的默认实现使用 Git ， 因此它轻松支持标签版本的配置环境，以及可以访问用于管理内容的各种工具。
- 这个还是静态的，得配合 Spring Cloud Bus 实现动态的配置更新。

虽然 Spring Cloud Config 官方并未推出管理平台，我们可以考虑看看 《为 Spring Cloud Config 插上管理的翅膀》 。

Apollo

关于 Apollo 的源码解析，可以看看写的 《Spring Cloud Apollo 源码解析系列》 。

链路追踪

在 Spring Cloud 中，能够使用的链路追踪，主要是两个，如下：

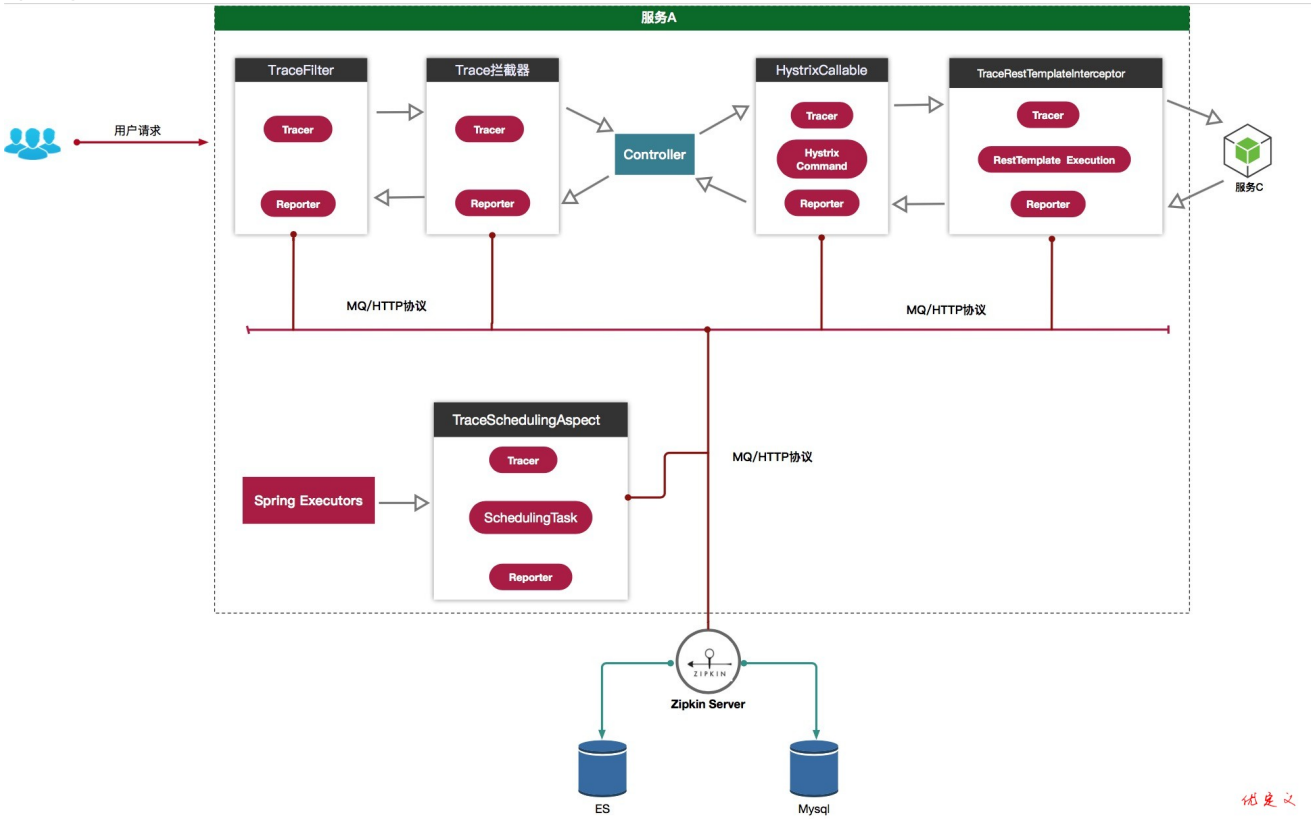
- `skywalking` ，已经进入 Apache ，不仅仅能够透明的监控链路，还可以监控 JVM 等等。
- `spring-cloud-sleuth` ，基于 Zipkin 实现。

SkyWalking

关于 SkyWalking 的源码解析，可以看看写的 《SkyWalking 源码解析系列》 。

Spring Cloud Sleuth

Spring Cloud Sleuth 原理，整体如下图：

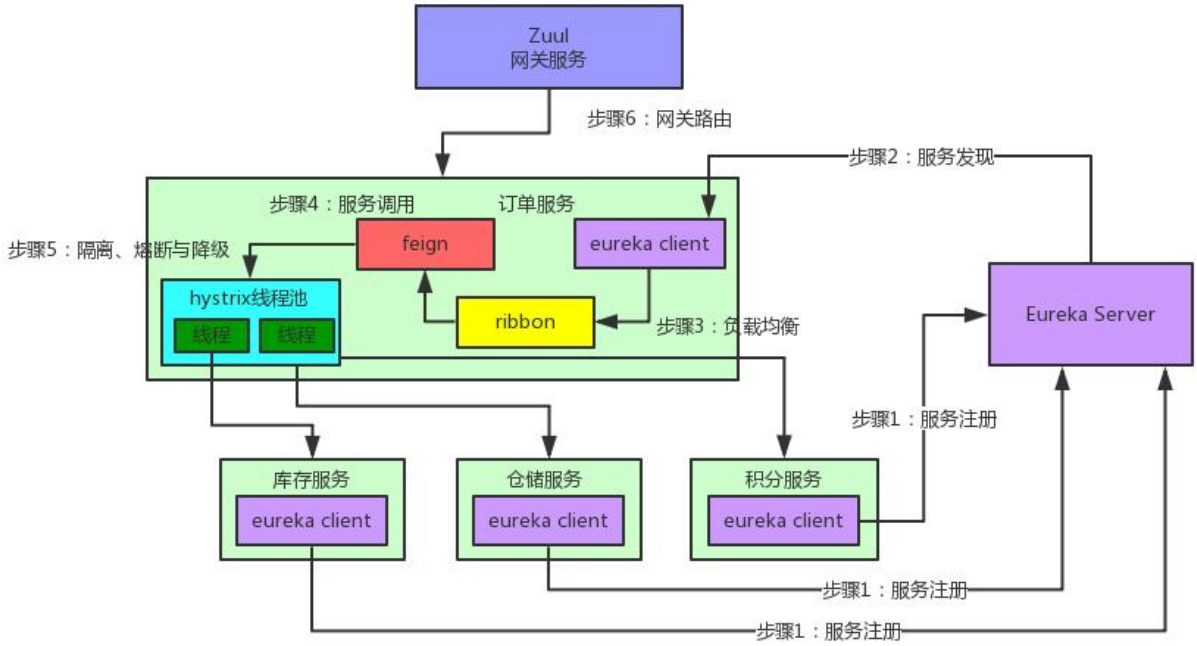


TODO 消息队列

彩蛋

第一个版本的 Spring Cloud 面试题，主要把整个文章的大体结构定了，后续在慢慢完善补充~

如下是 Eureka + Ribbon + Feign + Hystrix + Zuul 整合后的图：



参考与推荐如下文章：

- 《微服务框架之 Spring Cloud 面试题汇总》
- 《Spring Boot 和 Spring Cloud 面试题》
- 《Spring Cloud 简介与 5 大常用组件》
- 《面试必问的 Spring Cloud 实现原理图》
- 《拜托！面试请不要再问我 Spring Cloud 底层原理》