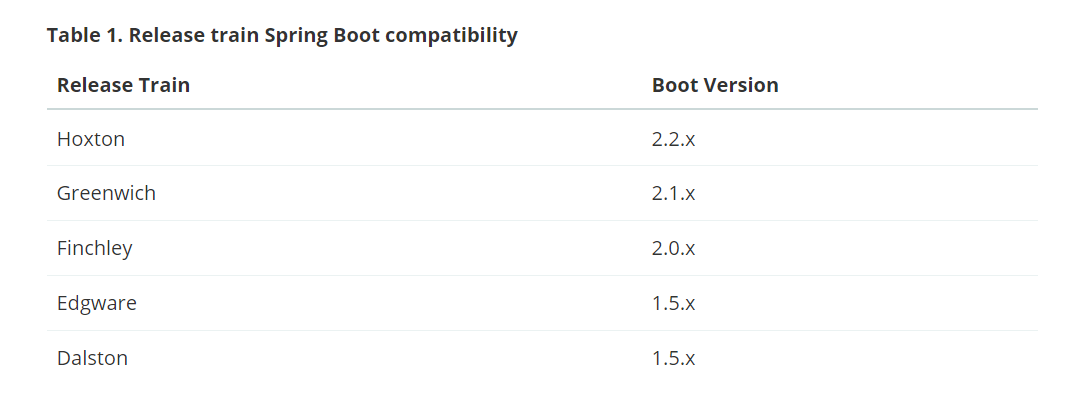
springboot版本发布信息：<https://github.com/spring-projects/spring-boot/releases>

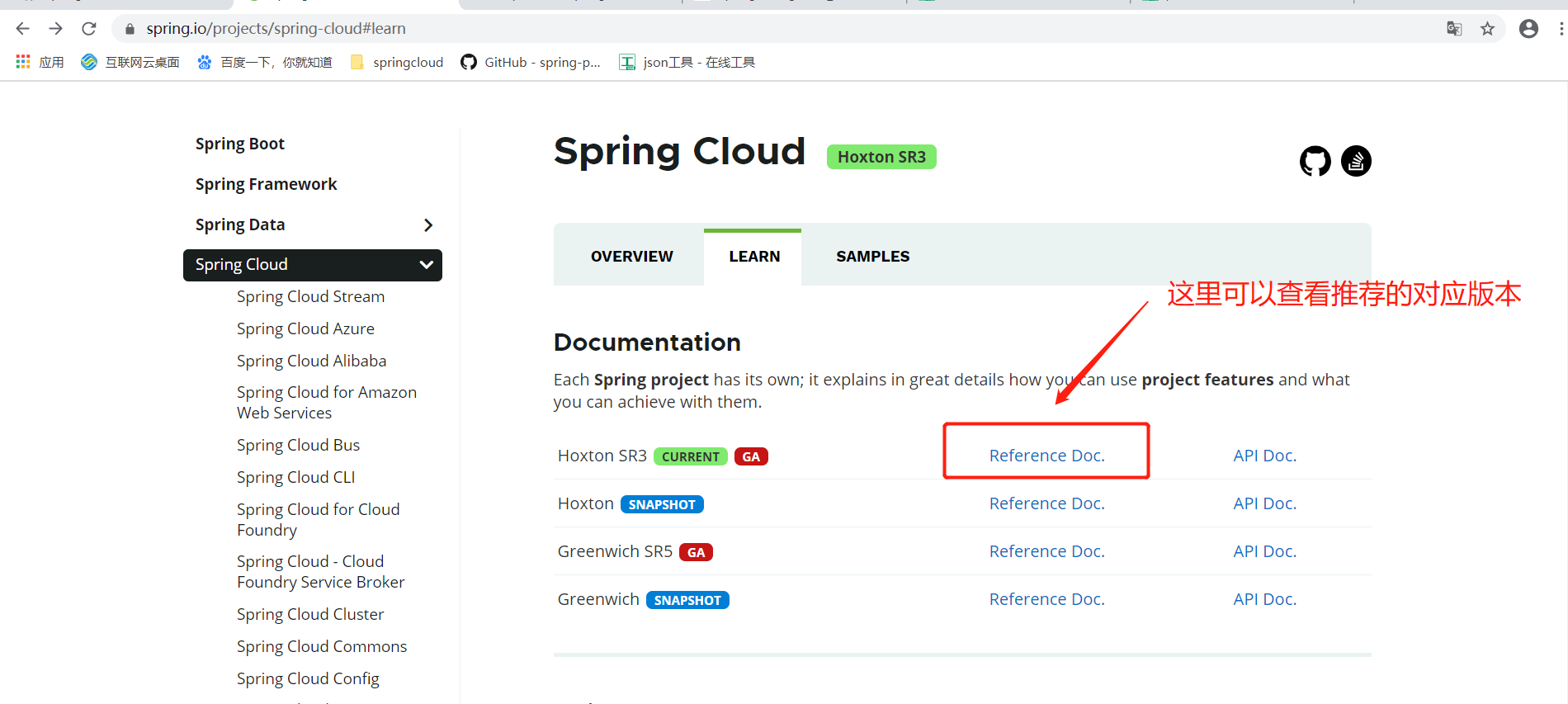
springcloud官网：<https://spring.io/projects/spring-cloud/>

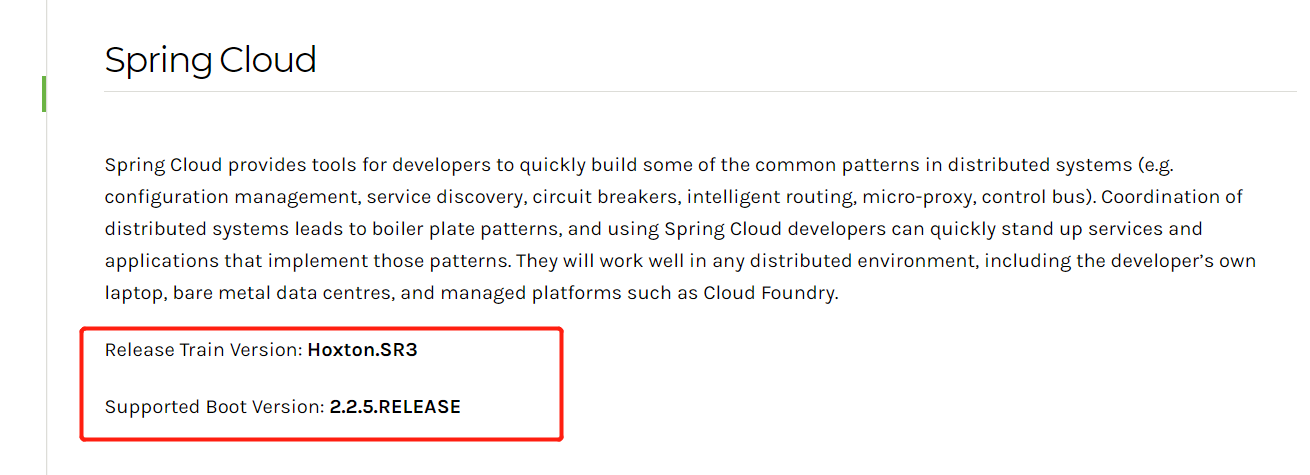
springcloud与Springboot兼容性



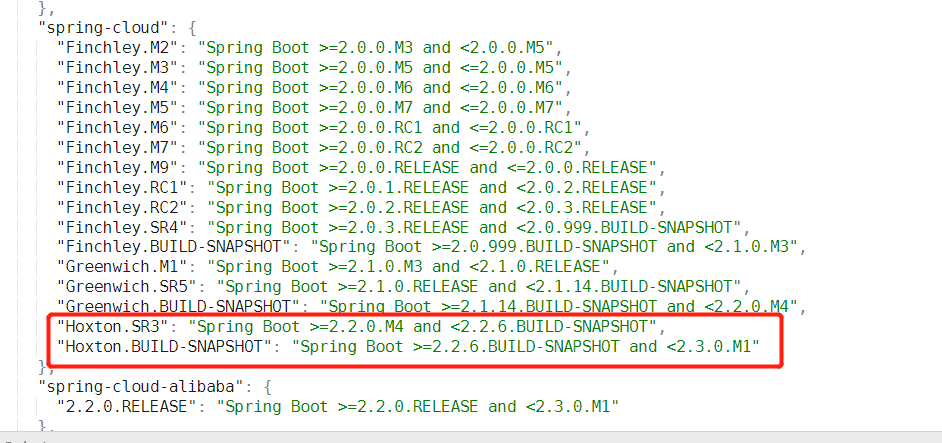
Greenwich可以构建并与Spring Boot 2.1.x一起使用，并且不能与Spring Boot 1.5.x一起使用。

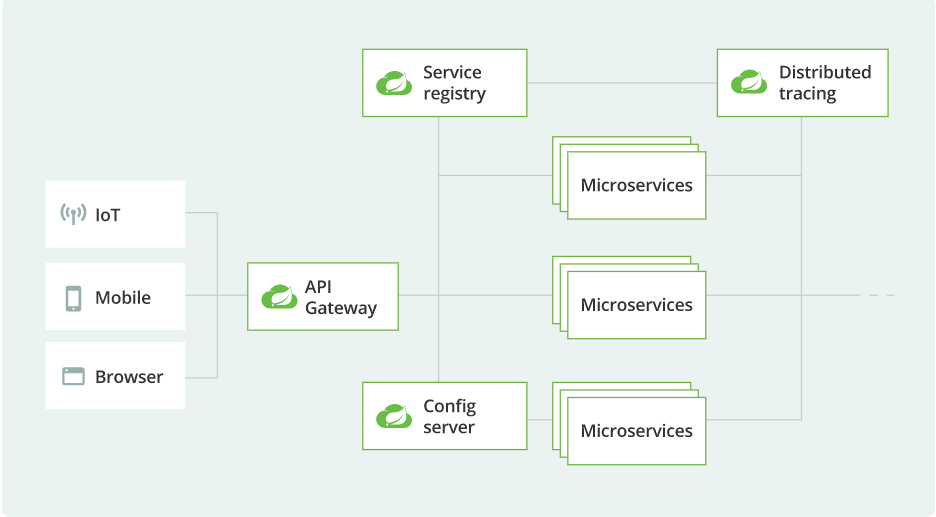
Dalston和Edgware发行列车建立在Spring Boot 1.5.x上，并且不能与Spring Boot 2.0.x一起使用。

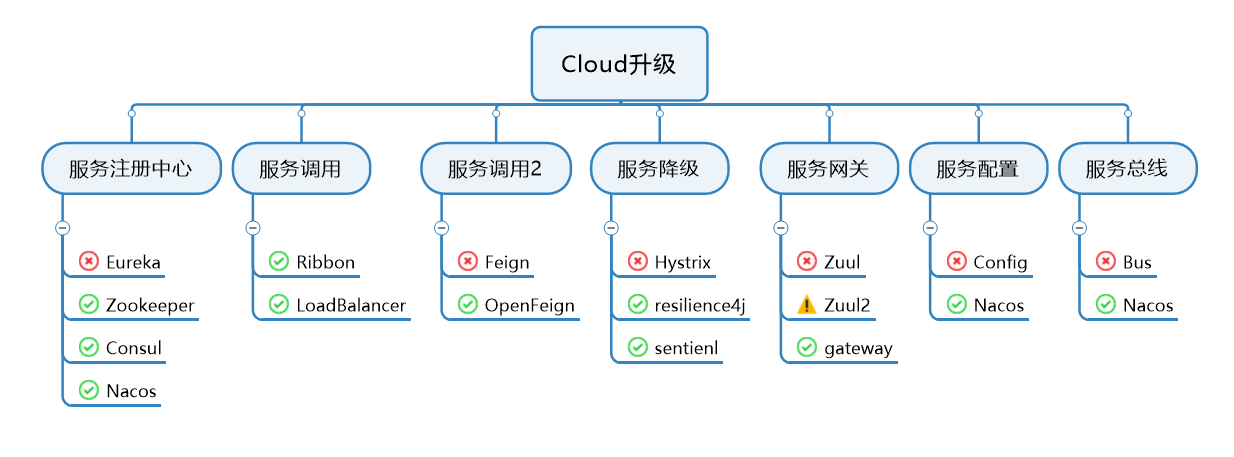




Spring官网更详细的版本对应信息：<https://start.spring.io/actuator/info>







## Eureka

### Eureka 原理

Eureka 是 Netflix 开发的，一个基于 REST 服务的，服务注册与发现的组件，以实现中间层服务器的负载平衡和故障转移。

它主要包括两个组件：Eureka Server 和 Eureka Client

* Eureka Client：一个Java客户端，用于简化与 Eureka Server 的交互（通常就是微服务中的客户端和服务端）
* Eureka Server：提供服务注册和发现的能力（通常就是微服务中的注册中心）

服务在Eureka上注册，然后每隔30秒发送心跳来更新它们的租约。如果客户端不能多次续订租约，那么它将在大约90秒内从服务器注册表中剔除。注册信息和更新被复制到集群中的所有eureka节点。来自任何区域的客户端都可以查找注册表信息（每30秒发生一次）来定位它们的服务（可能在任何区域）并进行远程调用

服务发现有两种模式：一种是客户端发现模式，一种是服务端发现模式。**Eureka采用的是客户端发现模式。**

Eureka客户端从服务器获取注册表信息并在**本地缓存**。之后，客户端使用这些信息来查找其他服务。通过在上一个获取周期和当前获取周期之间获取增量更新，这些信息会定期更新(每30秒更新一次)。获取的时候可能返回相同的实例。Eureka客户端自动处理重复信息。

Eureka客户端从服务器获取注册表信息并在本地缓存。之后，客户端使用这些信息来查找其他服务。通过在上一个获取周期和当前获取周期之间获取增量更新，这些信息会定期更新(每30秒更新一次)。获取的时候可能返回相同的实例。Eureka客户端自动处理重复信息。

### Eureka 基本搭建

服务端(Server)环境配置

客户端环境配置

BUG

### Eureka 集群环境搭建

修改系统配置

C:\Windows\System32\drivers\etc

127.0.0.1 eureka7001.com

127.0.0.1 eureka7002.com

YML 修改

|  |
| --- |
| **server:  port:** 7001  **eureka:  instance:  hostname:** eureka7001.com *#eureka服务端的实例名称* **client:** *#false表示不向注册中心注册自己* **register-with-eureka:** false  *#false表示自己端就是注册中心，我的职责就是维护服务实例，并不需要去检索服务* **fetch-registry:** false  **service-url:** *#设置与Eureka Server交互的地址查询服务和注册服务都需要依赖这个地址* **defaultZone:** http://eureka7002.com:7002/eureka/  *#defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/* **server:** *#关闭自我保护机制，保证不可用服务被及时剔除* **enable-self-preservation:** false  **eviction-interval-timer-in-ms:** 2000 |

启动参数

服务注册到 Eureka集群

1、YML 注册

|  |
| --- |
| **eureka:  client:** *#表示是否将自己注册进EurekaServer默认为true* **register-with-eureka:** true  *#是否从EurekaServer抓取已有的注册信息，默认为true。单节点无所谓，集群必须设置为true才能配合ribbon使用负载均衡* **fetch-registry:** true  **service-url:  defaultZone:** http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/  **instance:  instance-id:** payment8001  **prefer-ip-address:** true  *#Eureka客户端向服务端发送心跳的时间间隔，单位为秒（默认是30秒）* **lease-renewal-interval-in-seconds:** 1  *#Eureka服务端在收到最后一次心跳后等待时间上限，单位为秒（默认90秒），超时将剔除服务* **lease-expiration-duration-in-seconds:** 2 |

2、启动类加上注解@EnableEurekaClient

服务的负载均衡

RestTemplate Bean 声明添加 @LoadBalanced

代码实例

修改请求地址

原始地址：http://localhost:8001

修改后地址：http://PAYMENT-SERVCIE

@LoadBalanced 默认采用轮询算法

actuator 微服务信息完善

显示主机名称

YML 配置

IP 配置

YML 配置

### 服务发现 Discovery

### Eureka 自我保护

自我保护的理论:

任何客户端，如果连续3次心跳更新失败，那么它将被视为非正常终止，病句将被剔除。当超过当前注册实例15%的客户端都处于这种状态，那么自我保护将被开启。

当自我保护开启以后，eureka服务器将停止剔除所有实例，直到：它看到的心跳续借的数量回到了预期的阈值之上，或者自我保护被禁用

默认情况下，自我保护是启用的，并且，默认的阈值是要大于当前注册数量的15%

禁用自我保护

某一时刻，不会自动删除服务

配置

Eureka Server YML 配置

Eureka Client YML 配置

Eureka 停止更新

discovery

三个注册中心的异同点

| **组件名** | **语言** | **CAP** | **服务健康检查** | **对外暴露接口** | **Spring Cloud集成** |
| --- | --- | --- | --- | --- | --- |
| Eureka | Java | AP | 可配支持健康检查 | HTTP | 集成 |
| Consul | GO | CP | 支持健康检查 | HTTP、DNS | 集成 |
| Zookeeper | Java | CP | 支持健康检查 | 客户端 | 集成 |

## Zookeeper

## Consul

<springcloud.cc/spring-cloud-consul.html>

Consul是一个服务网格（微服务间的 TCP/IP，负责服务之间的网络调用、限流、熔断和监控）解决方案，它是一个一个分布式的，高度可用的系统，而且开发使用都很简便。它提供了一个功能齐全的控制平面，主要特点是：服务发现、健康检查、键值存储、安全服务通信、多数据中心。

与其它分布式服务注册与发现的方案相比，Consul 的方案更“一站式”——内置了服务注册与发现框架、分布一致性协议实现、健康检查、Key/Value 存储、多数据中心方案，不再需要依赖其它工具。Consul 本身使用 go 语言开发，具有跨平台、运行高效等特点，也非常方便和 Docker 配合使用。

安装运行consul

官网：<https://www.consul.io/intro/index.html>

下载页面：<https://www.consul.io/downloads.html>

启动命令：consul agent -dev

通过以下地址可以访问 Consul 首页

<http://localhost:8500>

pom

配置文件

主启动类

## Ribbon

github地址：<https://github.com/Netflix/ribbon/wiki/Features>

Spring Cloud Ribbon 是基于Netflix Ribbon 实现的一套客户端 负载均衡的工具。

Ribbon 是 Netflix 发布的开源项目，主要功能是提供客户端的软件负载均衡算法和服务调用。Ribbon 客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出 Load Balancer（简称LB）后面所有的机器，Ribbon 会自动的帮助你基于某种规则（如简单轮询、随机连接等）去连接这些机器。我们很容易使用Ribbon实现自定义的负载均衡算法。

**总之一句话：** Ribbon 就是 负载均衡 + RestTemplate调用，最终实现RPC的远程调用。

PS： Ribbon目前也进入维护模式，SpringCloud 想用Spring Cloud LoadBalancer 替代 Netflix 的Ribbon ，但现在 Ribbon 在生产环境中大规模部署，一时半会替不掉

Ribbon和Nginx的区别

Nginx是服务器负载均衡，客户端所有请求都会交给nginx，然后由nginx转发请求。（集中式LB）

Ribbon是本地负载均衡，在调用微服务接口时，会在注册中心上获取注册信息服务列表之后缓存到 JVM 本地，从而实现 RPC 远程服务调用技术。（进程内LB）

RestTemplate

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html>

常用方法：

get请求

* getForEntity：包含响应体、响应头、响应状态码等其他信息
* getForObject：只返回响应体

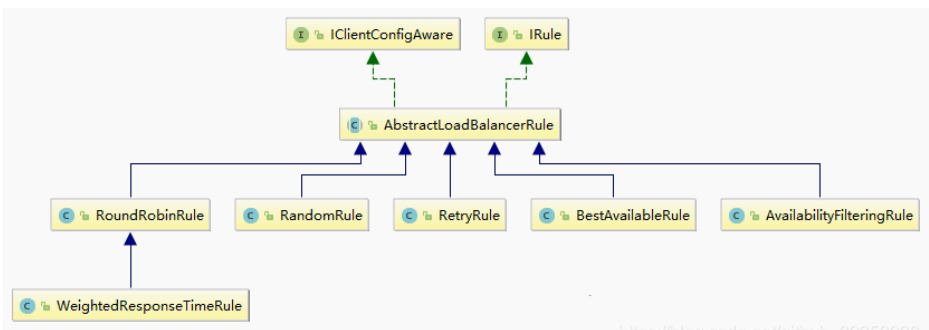
post请求

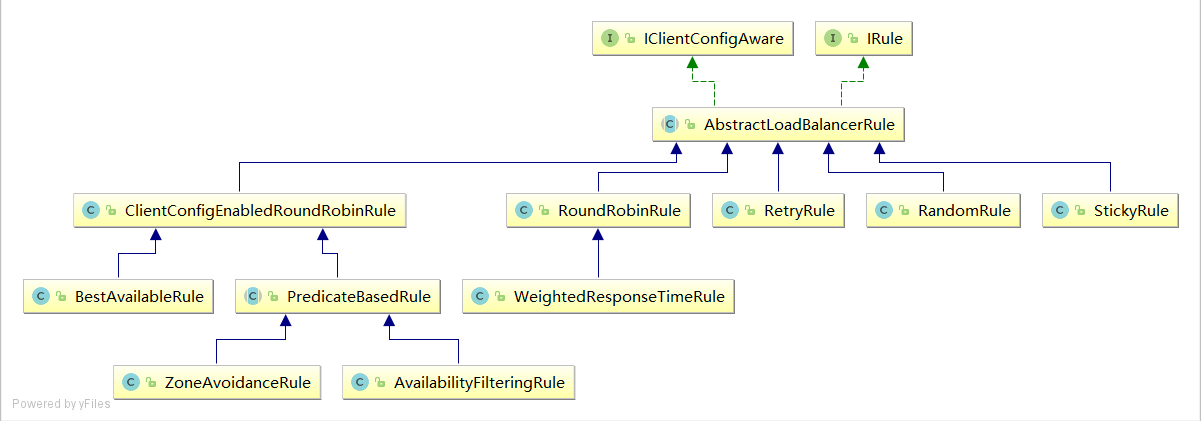
* postForEntity：包含响应体、响应头、响应状态码等其他信息
* postForObject：只返回响应体

|  |
| --- |
| @Configuration **public class** MyRule {   @Bean  **public** IRule rule() {  **return new** RandomRule();  } } |

### 核心组件IRule

IRule是一个接口，它表示根据特定算法从服务列表中选取一个要访问的服务





六种提供的算法

RoundRobinRule：轮询

RandomRule：随机

RetryRule：先按照RoundRobinRule的策略获取服务，如果获取服务失败则在指定时间内会进行重试

BestAvailableRule：先过滤掉由于多次访问故障而处于断路器跳闸状态的服务，然后选择并发量最小的服务

AvailabilityFilteringRule：先过滤掉故障的实例，再选择并发较小的实例

ZoneAvoidanceRule：默认规则，符合判断

### 修改默认负载均衡策略

## OpenFeign

<https://cloud.spring.io/spring-cloud-static/spring-cloud-openfeign/2.2.2.RELEASE/reference/html/>

github地址：<https://github.com/spring-cloud/spring-cloud-openfeign>

## Hystrix

github地址：<https://github.com/Netflix/Hystrix/wiki/How-To-Use>

## Nacos

### 服务注册中心

## 

Nacos 支持AP 和 CP模式的转换

C是所有节点在同一时间看到的数据是一致的；而A的定义是所有的请求都会受到响应。

用下面命令切换

curl -X PUT '$NACOS\_SERVER:8848/nacos/v1/ns/operator/switches?entry=serverMode&value=CP'