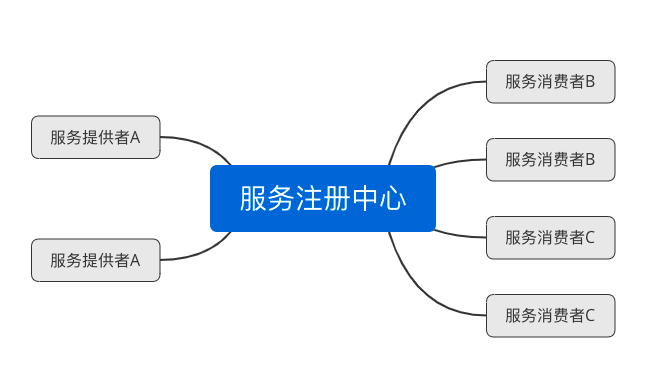
服务注册中心

# 介绍

微服务的注册中心可以说是微服务架构中的“通信录”，它记录了服务和服务地址的映射关系。在分布式架构中，服务会注册到这里，当服务需要调用其他服务时，就到这里找到服务的地址，进行调用。

服务注册中心主要分为三个部分，一个部分是服务的提供者，一个部分是服务注册中心一个部分则是服务的消费者。



在数据流程上，服务提供者启动服务之后，会将服务注册到服务注册中心，而当服务消费者启动的时候，会从服务注册中心拉取相关的配置，放到缓存中。从本质上来说，其实也就是一个名称解析服务，因为对于服务消费者来说，首先，从服务注册中心根据服务名找到服务提供者的ip和端口，然后根据内部的调度机制，找到一个服务提供者访问，得到请求结果。

从而，注册中心解偶了服务提供者和服务消费者之间的关系，并且支持弹性的扩容和缩容，当你扩容的时候，只要将你的服务再次扩展一个，也就会自动注册到注册中心了。

服务发现，是站在服务消费者的角度来说，能发现服务提供者的ip增多或者减少，其实这是因为服务注册中心的主动推送的作用。

服务注册，是站在服务生产者的角度来说，也就是服务生产者将服务注册到服务注册中心。

服务路由，主要是将服务生产者的ip缓存在本地，然后在其中使用调度算法选择一个合适的服务提供者。

# 解决问题

服务注册后，如何被及时发现

服务宕机后，如何及时下线

服务如何有效的水平扩展

服务发现时，如何进行路由

服务异常时，如何进行降级

注册中心如何实现自身的高可用

功能类似Nginx负载均衡，但是负载均衡是静态的，服务注册中心是动态的。

服务发现：Consul的客户端可以向Consul注册服务，例如api服务或者mysql服务，其他客户端可以使用Consul来发现服务的提供者。Consul支持使用DNS或HTTP来注册和发现服务。

运行时健康检查：Consul客户端可以提供任意数量的运行状况检查机制，这些检查机制可以是给定服务（“是Web服务器返回200 OK”）或本地节点（“内存利用率低于90％”）相关联。这些信息可以用来监控群集的运行状况，服务发现组件可以使用这些监控信息来路由流量，可以使流量远离不健康的服务。

KV存储：应用程序可以将Consul的键/值存储用于任何需求，包括动态配置，功能标记，协调，领导者选举等。它采用HTTP API使其易于使用。

安全服务通信：Consul可以为服务生成和分发TLS证书，以建立相互的TLS连接。

多数据中心：Consul支持多个数据中心。这意味着Consul的用户不必担心构建额外的抽象层以扩展到多个区域

consul支持的KV存储的Value值不能超过512KB

Consul的dev模式，所有数据都存储在内存中，重启Consul的时候会导致所有数据丢失，在正式的环境中，Consul的数据会持久化，数据不会丢失。

Consul与Eureka对比

1.Consul有着更丰富的服务健康检查服务状态，内存，硬盘等。

2.Consul支持kv存储服务。

3.Consul更高的一致性，一致性算法raft。

# 对比

常用的注册中心有Nacos,Consul,Eureka,Zookeeper,Etcd。

## Nacos

### 1.简介

Nacos 是阿里巴巴2018年推出来的一个新开源项目，这是一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。

官网：<https://nacos.io>

### 2.特性：

Dynamic Configuration Service（配置中心）

Service Discovery and Management（服务发现和管理）

Dynamic DNS Service（路由和负载均衡）

### 3.应用



## Consul

### 1.简介

Consul是HashiCorp公司推出的开源软件，使用GO语言编写，提供了分布式系统的服务注册和发现、配置等功能，这些功能中的每一个都可以根据需要单独使用，也可以一起使用以构建全方位的服务网格。Consul不仅具有服务治理的功能，而且使用分布式一致协议RAFT算法实现，有多数据中心的高可用方案，并且很容易和Spring Cloud等微服务框架集成，使用起来非常的简单，具有简单、易用、可插拔等特点。使用简而言之，Consul提供了一种完整的服务网格解决方案 。

### 2.特性

Service Discovery:

Distributed Configuration: using the Consul Key/Value store

Supports Ribbon

Supports Zuul, a dynamic router and filter via Spring Cloud Netflix

Control Bus: Distributed control events using Consul Events

Circuit Breaker with Hystrix

### 3.应用



## EureKa

### 1.简介

Eureka是一个服务注册和发现的组件,最初开始用于亚马逊公司旗下的云计算服务平台(AWS),Eureka分为Eureka Client和Eureka Server,专门用于给其他服务注册的注册中心称之为Eureka Server,注册到Eureka Server上的服务称之为Eureka Client。

Eureka完全开源,是netflix公司的开源产品,经过了生产环境的检验和历时三年的不断迭代,在功能上和性能上相对而言都比较稳定.

Eureka是springColud官方推荐首选的服务治理组件,其可与springCloud架构中其他组件无缝对接。

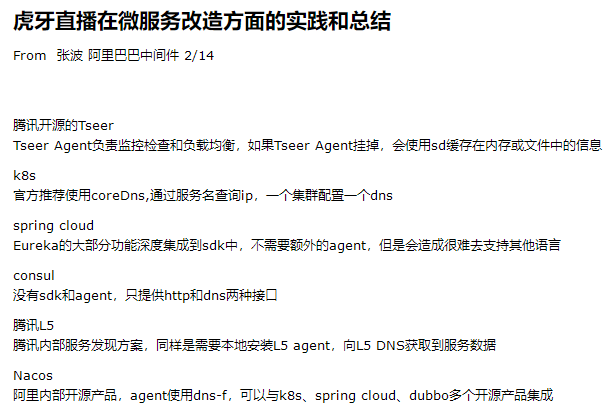
Eureka与客户端负载均衡Ribbo,熔断器Hystix,服务网关Zuul等组件,很容易实现负载均衡/熔断/智能路由等功能,形成Netflix OSS组件,这些组件是SpringCloud的基础组件和核心组件。

Eureka 2.x的开源分支已关闭，如果要使用，请自己承担风险。1.x版本仍然继续维护。

### 2.特性

### 3.应用

## 对比



从设计角度来看，Eureka可以说是无懈可击，注册中心、提供者、调用者边界清晰，通过去中心化的集群支持保证了注册中心的整体可用性，但缺点是Eureka属于应用内的注册方式，对应用的侵入性太强，且只支持Java应用。

Consul本质上属于应用外的注册方式，但可以通过SDK简化注册流程。而服务发现恰好相反，默认依赖于SDK，但可以通过Consul Template（下文会提到）去除SDK依赖。但是CAP中Consul支持CP。

阿里的nacos : 性能最好

他同时支持AP和CP模式,他根据服务注册选择临时和永久来决定走AP模式还是CP模式，他这里支持CP模式对于我的理解来说,应该是为了配置中心集群,因为nacos可以同时作为注册中心和配置中心,因为他的配置中心信息是保存在nacos里面的,假如因为nacos其中一台挂掉后,还没有同步配置信息,就可能发生配置不一致的情况., 配置中心的配置变更是服务端有监听器,配置中心发生配置变化，然后服务端会监听到配置发生变化,从而做出改变。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Consul** | **Zookeeper** | **Etcd** | **Eureka** |
| **服务健康检查** | 服务状态，内存，硬盘等 | (弱)长连接，keepalive | 连接心跳 | 可配支持 |
| **多数据中心** | 支持 | — | — | — |
| **kv存储服务** | 支持 | 支持 | 支持 | — |
| **一致性** | raft | paxos | raft | — |
| **CAP定理** | CP | CP | CP | AP |
| **使用接口(多语言能力)** | 支持http和dns | 客户端 | http/grpc | http（sidecar） |
| **watch支持** | 全量/支持long polling | 支持 | 支持 long polling | 支持 long polling/大部分增量 |
| **自身监控** | metrics | — | metrics | metrics |
| **安全** | acl /https | acl | https支持（弱） | — |
| **spring cloud集成** | 已支持 | 已支持 | 已支持 | 已支持 |

Nacos在经过内部多年生产经验后提炼出的数据模型，则是一种服务-集群-实例的三层模型。如上文所说，这样基本可以满足服务在所有场景下的数据存储和管理。Nacos的数据模型虽然相对复杂，但是它并不强制你使用它里面的所有数据，在大多数场景下，你可以选择忽略这些数据属性，此时可以降维成和Eureka和Consul一样的数据模型。同时支持可选的AP和CP。

### 1.数据模型

Zookeeper没有针对服务发现设计数据模型，它的数据是以一种更加抽象的树形K-V组织的，因此理论上可以存储任何语义的数据。而Eureka或者Consul都是做到了实例级别的数据扩展，这可以满足大部分的场景，不过无法满足大规模和多环境的服务数据存储。

Nacos在经过内部多年生产经验后提炼出的数据模型，则是一种服务-集群-实例的三层模型。如上文所说，这样基本可以满足服务在所有场景下的数据存储和管理。Nacos的数据模型虽然相对复杂，但是它并不强制你使用它里面的所有数据，在大多数场景下，你可以选择忽略这些数据属性，此时可以降维成和Eureka和Consul一样的数据模型。

### 2.数据一致性

Zookeeper使用ZAB协议虽然保证了数据的强一致，但是它的机房容灾能力的缺乏，无法适应一些大型场景。

Nacos因为要支持多种服务类型的注册，并能够具有机房容灾、集群扩展等必不可少的能力，在1.0.0正式支持AP和CP两种一致性协议并存。

### 3.负载均衡

目前的负载均衡有基于权重、服务提供者负载、响应时间、标签等策略。

Eureka、Zookeeper包括Consul，本身都没有去实现可配置及可扩展的负载均衡机制，Eureka的负载均衡是由ribbon来完成的，而Consul则是由Fabio做负载均衡。

其中Ribbon设计的客户端负载均衡机制，主要是选择合适现有的IRule、ServerListFilter等接口实现，或者自己继承这些接口，实现自己的过滤逻辑。这里Ribbon采用的是两步负载均衡，第一步是先过滤掉不会采用的服务提供者实例，第二步是在过滤后的服务提供者实例里，实施负载均衡策略。Ribbon内置的几种负载均衡策略功能还是比较强大的，同时又因为允许用户去扩展，这可以说是一种比较好的设计。

基于标签的负载均衡策略可以做到非常灵活，Kubernetes和Fabio都已经将标签运用到了对资源的过滤中，使用标签几乎可以实现任意比例和权重的服务流量调配。但是标签本身需要单独的存储以及读写功能，不管是放在注册中心本身或者对接第三方的CMDB。

在Nacos 0.7.0版本中，除了提供基于健康检查和权重的负载均衡方式外，还新提供了基于第三方CMDB的标签负载均衡器。

### 4.健康检查

Zookeeper和Eureka都实现了一种TTL的机制，就是如果客户端在一定时间内没有向注册中心发送心跳，则会将这个客户端摘除。Eureka做的更好的一点在于它允许在注册服务的时候，自定义检查自身状态的健康检查方法。

Nacos既支持客户端的健康检查，也支持服务端的健康检查，同一个服务可以切换健康检查模式。

### 5.性能与容量

虽然大部分用户用到的性能不高，但是他们仍然希望选用的产品的性能越高越好。影响读写性能的因素很多：一致性协议、机器的配置、集群的规模、存量数据的规模、数据结构及读写逻辑的设计等等。

Paxos协议本身就限制了Zookeeper集群的规模，3、5个节点是不能应对大规模的服务订阅和查询的

服务规模在1000上下，几乎目前所有的注册中心都可以满足。

Paoxs协议的单点瓶颈。

Eureka耦合度太高，而且只支持java。

Consul对高可用支持不是很好。

结论：推荐使用Nacos，支持继承dubbo。

# 实战Demo

Nacos配置持久化目前只支持mysql

https://nacos.io/zh-cn/docs/deployment.html

# 参考资料

<https://blog.csdn.net/chaoyuehu/article/details/90513674>

主流微服务注册中心浅析和对比：<https://www.jianshu.com/p/d1b5c8b76194>

<https://luyiisme.github.io/2017/04/22/spring-cloud-service-discovery-products/?utm_source=tuicool&utm_medium=referral>

https://nacos.io/zh-cn/docs/deployment.html