基于Spring Cloud的微服务架构分析

智能摘要

Eureka是Spring Cloud微服务架构中的注册中心，专门负责服务的注册与发现，里面有一个注册表，保存了各个服务器的机器和端口。在客户端负载均衡中，所有客户端节点都维护着自己要访问的服务端清单，而这些服务端的清单来自于服务注册中心(比如Eureka)。微服务网关更多是在前后端分离，或者说涉及到独立的类似手机APP等前端应用的时候使用的最多，即把内部各个微服务组件模块的API接口能力统一注册和接入到网关，对于APP也只需要访问网关暴露的接口即可，同时通过网关还可以进一步的实现安全隔离。

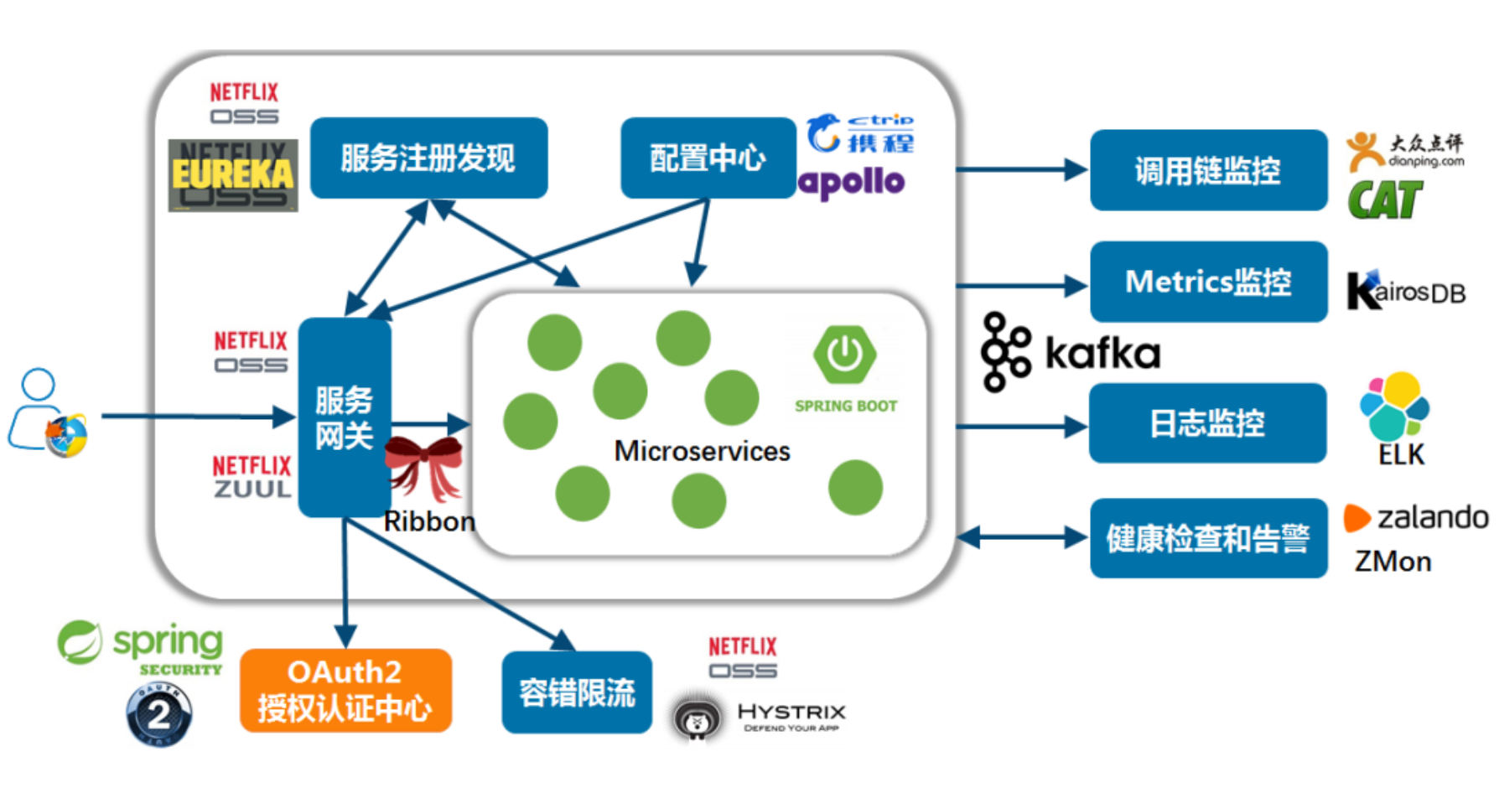
基于Spring Cloud的微服务架构分析

Alex 分布式实验室

Spring Cloud是一个相对比较新的微服务框架，2016年才推出1.0的release版本。虽然Spring Cloud时间最短，但是相比Dubbo等RPC框架，Spring Cloud提供的全套的分布式系统解决方案。

Spring Cloud是一系列框架的有序集合。它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。

Spring并没有重复制造轮子，它只是将目前各家公司（主要是Netflix）开发的比较成熟、经得起实际考验的服务框架组合起来，通过Spring Boot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。



Eureka（注册中心）

Eureka是Spring Cloud微服务架构中的注册中心，专门负责服务的注册与发现，里面有一个注册表，保存了各个服务器的机器和端口。

- Eureka服务端：也称服务注册中心，同其他服务注册中心一样，支持高可用配置。如果Eureka以集群模式部署，当集群中有分片出现故障时，那么Eureka就转入自我保护模式。它允许在分片故障期间继续提供服务的发现和注册，当故障分片恢复运行时，集群中其他分片会把它们的状态再次同步回来。
- Eureka客户端：主要处理服务的注册与发现。客户端服务通过注解和参数配置的方式，嵌入在客户端应用程序的代码中，在应用程序运行时，Eureka客户端想注册中心注册自身提供的服务并周期性地发送心跳来更新它的服务租约。同时，它也能从服务端查询当前注册的服务信息并把它们缓存到本地并周期性地刷新服务状态。

Eureka Server的高可用实际上就是将自己作为服务向其他注册中心注册自己，这样就可以形成一组互相注册的服务注册中心，以实现服务清单的互相同步，达到高可用效果。

Zuul（服务网关）

Zuul网关负责转发请求给对应的服务，这个组件是负责网络路由的。

Spring Cloud Zuul通过与Spring Cloud Eureka进行整合，将自身注册为Eureka服务治理下的应用，同时从Eureka中获得了所有其他微服务的实例信息。

对于路由规则的维护，Zuul默认会将通过以服务名作为ContextPath的方式来创建路由映射。

Zuul提供了一套过滤器机制，可以支持在API网关无附上进行统一调用来对微服务接口做前置过滤，已实现对微服务接口的拦截和校验。

Ribbon（负载均衡）

提供云端负载均衡，有多种负载均衡策略可供选择，可配合服务发现和断路器使用。

Ribbon是一个基于HTTP和TCP的客户端负载均衡器，它可以在通过客户端中配置的ribbonServerList服务端列表去轮询访问以达到服务均衡的作用。

当Ribbon和Eureka联合使用时，Ribbon的服务实例清单RibbonServerList会被DiscoveryEnabledNIWSServerList重写，扩展成从Eureka注册中心中获取服务端列表。同时它也会用NIWSDiscoveryPing来取代IPing，它将职责委托给Eureka来去定服务端是否已经启动。

在客户端负载均衡中，所有客户端节点都维护着自己要访问的服务端清单，而这些服务端的清单来自于服务注册中心（比如Eureka）。在客户端负载均衡中也需要心跳去维护服务端清单的健康性，只是这个步骤需要与服务注册中心配合完成。

通过Spring Cloud Ribbon的封装，我们在微服务架构中使用客户端负载均衡调用只需要如下两步：

- 服务提供者只需要启动多个服务实例并且注册到一个注册中心或是多个相关联的服务注册中心。
- 服务消费者直接通过调用被@LoadBalanced注解修饰过的RestTemplate来实现面向服务的接口调用。

Hystrix（熔断保护器）

熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。

提供线程池不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务器雪崩的问题。

在微服务架构中，存在着那么多的服务单元，若一个单元出现故障，就很容易因依赖关系而引发故障的蔓延，最终导致整个系统的瘫痪，这样的架构相较传统架构更加不稳定。为了解决这样的问题，产生了断路器等一系列的服务保护机制。

在分布式架构中，当某个服务单元发生故障之后，通过断路器的故障监控，向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

Hystrix具备服务降级、服务熔断、线程和信号隔离、请求缓存、请求合并以及服务监控等强大功能。

Hystrix使用舱壁模式实现线程池的隔离，它会为每一个依赖服务创建一个独立的线程池，这样就算某个依赖服务出现延迟过高的情况，也只是对该依赖服务的调用产生影响，而不会拖慢其他的依赖服务。

Feign（REST转换器）

基于动态代理机制，根据注解和选择的机器，拼接请求url地址，发起请求。Feign的关键机制是使用了动态代理：

- 首先，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；
- 接着调用接口的时候，本质就是调用Feign创建的动态代理；
- Feign的动态代理会根据在接口上的@RequestMapping等注解，来动态构造要请求的服务的地址；
- 针对这个地址，发起请求、解析响应。

Feign是和Ribbon以及Eureka紧密协作的：

- 首先Ribbon会从Eureka Client里获取到对应的服务注册表，也就知道了所有的服务都部署在了哪些机器上，在监听哪些端口；
- 然后Ribbon就可以使用默认的Round Robin算法，从中选择一台机器；
- Feign就会针对这台机器，构造并发起请求。

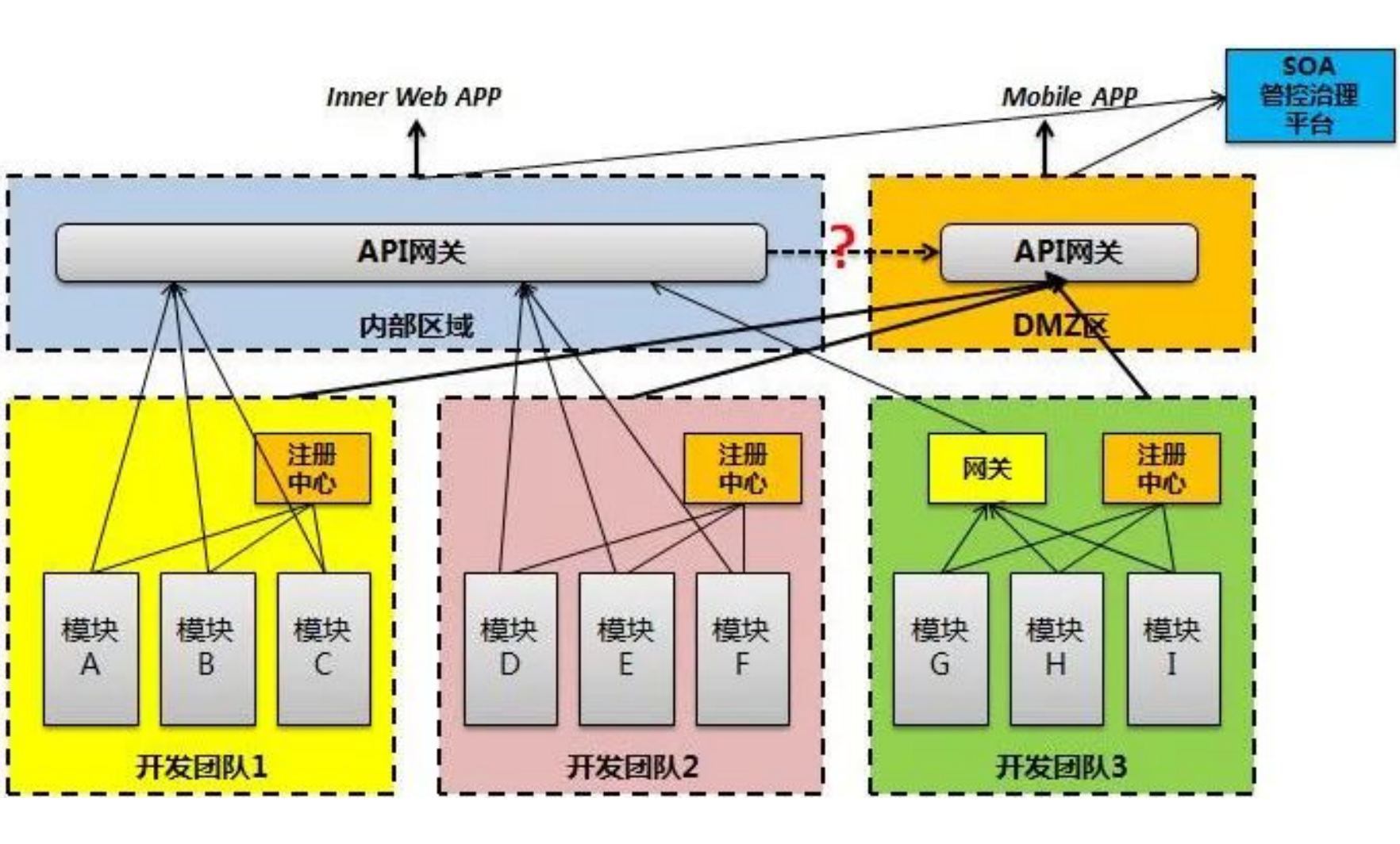
Config（分布式配置）

配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git以及Subversion。

微服务网关更多是在前后端分离，或者说涉及到独立的类似手机APP等前端应用的时候使用的最多，即把内部各个微服务组件模块的API接口能力统一注册和接入到网关，对于APP也只需要访问网关暴露的接口即可，同时通过网关还可以进一步的实现安全隔离。

也就是说在这种场景下，网关更多的是实现了接口服务的代理和路由转发能力，更多的是向外的一种能力发布。

1. 一个独立的开发团队，为保证独立自治，以及内部多个微服务模块间的交互集成，最好启用独立的服务注册中心实现服务注册，发现能力。即开发团队内部多个微服务模块间的集成，不需要通过网关，只需要通过服务注册中心进行集成即可。
2. 开发团队需要暴露能力给外部，包括暴露能力给其它的开发团队，需要考虑将该API接口注册到外部的网关上。在这里建议是拆分两个独立网关，一个是内部API网关，一个是放置到DMZ区面对公网访问的API网关。对于服务如果同时涉及到内部和外部使用，则两边注册。建议不要通过两次网关去路由，一个是影响性能，一个是不方便后续问题排查。
3. 构建在开发团队之外的API网关必须具备负载均衡能力，可以配置多个IP地址。通过该API网关也最好具备和Docker容器扩展后的服务自动注册和地址加入扩展能力。



服务发现是一个古老的话题，当应用开始脱离单机运行和访问时，服务发现就诞生了。目前的网络架构是每个主机都有一个独立的IP地址，那么服务发现基本上都是通过某种方式获取到服务所部署的IP地址。DNS协议是最早将一个网络名称翻译为网络IP的协议，在最初的架构选型中，DNS+LVS+Nginx基本可以满足所有的RESTful服务的发现，此时服务的IP列表通常配置在Nginx或者LVS。后来出现了RPC服务，服务的上下线更加频繁，人们开始寻求一种能够支持动态上下线并且推送IP列表变化的注册中心产品。

Feature	Consul	zookeeper	etcd	euerka
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	ca	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

Eureka

- Spring Cloud Eureka所选择的是AP，采用的是去中心化结构，放弃了强一致性。也就是说Eureka集群中的各个结点都是平等的，没有主从的概念。通过互相注册的方式来进行消息同步和保证高可用。并且一个Eureka Server结点挂掉了，还有其他同等的结点来提供服务，并不会引发服务的中断。
- Eureka只能当注册中心，想搞配置中心的话，还得搭配Spring Cloud Config+Spring Cloud Bus。其中后者支持Rabbiimq和Kafka两种模式。
- 使用Java语言来开发的，并且也是Spring Cloud的子项目，所以可以直接通过引入jar包的方式来集成Eureka，这点非常方便。

ZooKeeper

这是一款经典的服务注册中心产品（虽然它最初的定位并不在于此），在很长一段时间里，它是国人在提起RPC服务注册中心时心里想到的唯一选择，这很大程度上与Dubbo在中国的普及程度有关。

- Apache Zookeeper所选择的是CP，也就是放弃了高可用性。Zookeeper集群在进行消息同步的时候，必须有一半以上结点完成了同步才会返回；而当Master结点挂了或者集群中有过半的结点不能工作了，此时就会触发故障恢复，重新进行Master选举。在这个过程中，整个Zookeeper集群无法对外提供服务，从而实去了A（可用性）。
- 为了达到C，Zookeeper采用的是自己的ZAB协议。

Nacos

Nacos是阿里巴巴旗下的开源项目，在2018年开源，携带着阿里巴巴大规模服务生产经验，试图在服务注册和配置管理这个市场上，提供给用户一个新的选择。

- Nacos一大特性是即支持CP，也支持AP。可以根据需要灵活选择。
- Nacos除了注册中心之外，也能充当配置中心的作用。且配置中心可以按照namespace，group等维度来进行数据隔离，来达到不同环境之间配置隔离的功能。

> 值得一提的是，Nacos作为配置中心的持久化机制可以依赖于MySQL来完成（默认依赖于内置数据库）。只需要将Nacos目录下的SQL脚本放到mysql中执行（会生成11个表），然后在Nacos配置文件里面配一下MySQL的账号密码即可。这样使用MySQL作为数据源的方式相比于Nacos内置数据库来说更容易管理。

Consul

Consul是HashiCorp公司推出的一个开源工具。

- Consul是用Go语言编写的，所以无法像Eureka那样直接引入jar包就能集成，它还需要去服务器中进行额外的安装。
- 除了注册中心的功能之外，Consul还能起到配置中心的作用。Consul它保证的是CP，使用Raft协议，要求必须有过半的结点都写入成功才算是注册成功了，并且它也有Master和Follower的概念，在Master挂掉后，也需要自己内部进行。

etcd（待续）

对比Spring Cloud，Kubernetes也提供完整的分布式微服务管理框架，几乎所有组件都有对应的产品，其中etcd也可以提供类似Eureka的注册中心。

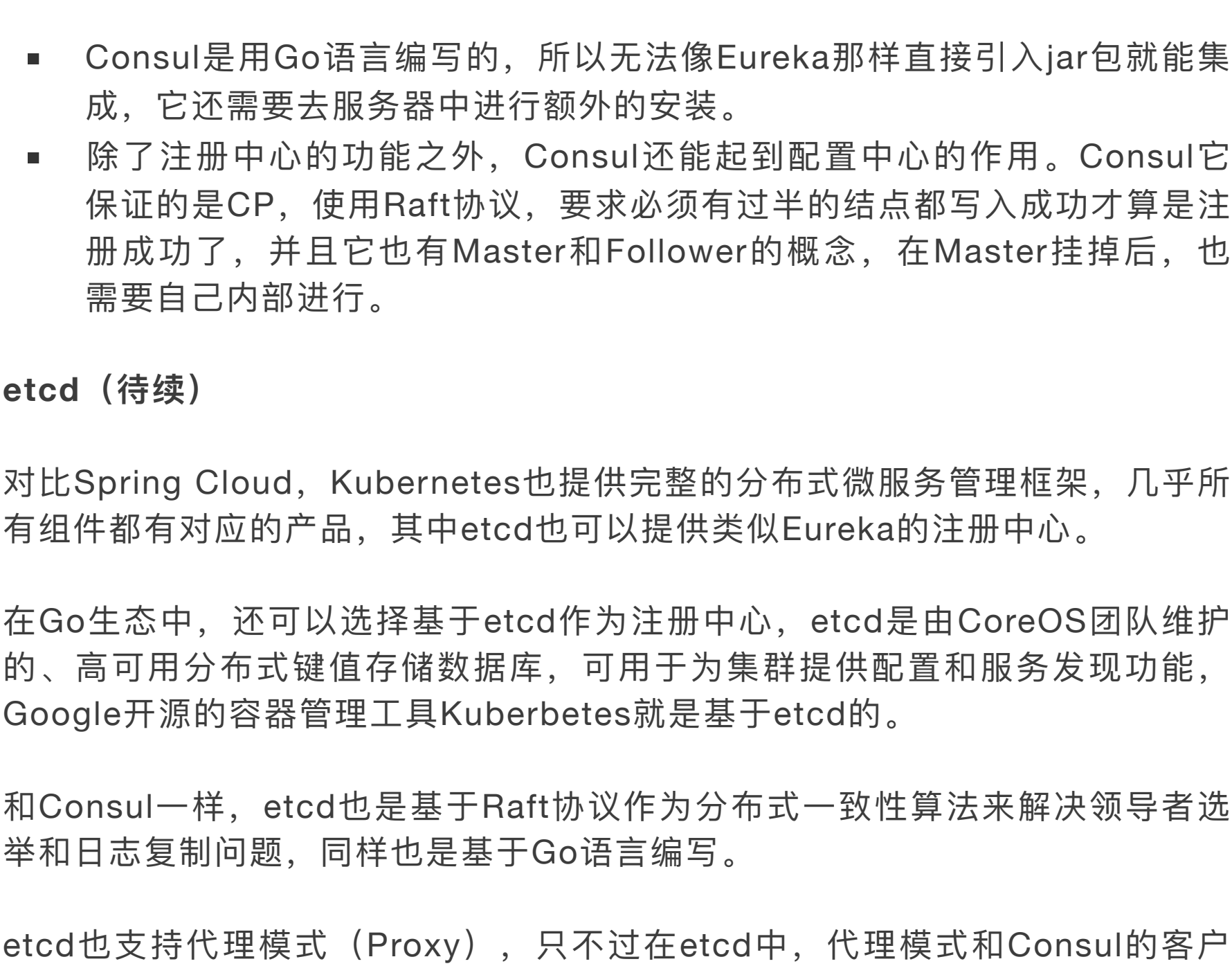
在Go生态中，还可以选择基于etcd作为注册中心，etcd是由CoreOS团队维护的、高可用分布式键值存储数据库，可用于为集群提供配置和服务发现功能，Google开源的容器管理工具Kuberbetes就是基于etcd的。

和Consul一样，etcd也是基于Raft协议作为分布式一致性算法来解决领导者选举和日志复制问题，同样也是基于Go语言编写。

etcd也支持代理模式（Proxy），只不过在etcd中，代理模式和Consul的客户端代理模式类似，安装在部署服务的节点上，用来转发请求到etcd集群，本身不存储任何数据，etcd集群相当于Consul中以服务端模式运行的Consul集群，通常要求配置三个及以上节点（不要太多，3~5就够了，以便可用性和性能上达到平衡），负责真正的请求处理——服务注册与发现。

在目前最新版本的etcd v3中，通过网关模式（Gateway）取代了V2版本中的代理模式（Proxy）。

从服务发现的实现原理上来说，Consul和etcd的基本设计思路是一致的，etcd更简单，Consul则更像一个全栈的解决方案，功能比etcd要更丰富，比如支持可视化的Web UI管理界面、支持多数据库中心、安全层面除了HTTPS外还支持ACL、更加全面的健康检查功能、内置DNS Server等，这些都是etcd所不具备的，但是更全面的功能往往意味着更高的复杂性，针对微服务的服务注册和发现场景，etcd完全够用了。



- Spring Cloud Config：配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git以及Subversion。
- Spring Cloud Bus：事件、消息总线，用于在集群（例如，配置变化事件）中传播状态变化，可与Spring Cloud Config联合实现热部署。
- Eureka：云端服务发现，一个基于REST的服务，用于定位服务，以实现云端中间层服务发现和故障转移。
- Hystrix：熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。
- Zuul：Zuul是在云平台上提供动态路由，监控，弹性，安全等边缘服务的框架。Zuul相当于是设备和Netflix流应用的Web网站后端所有请求的前门。
- Archaius：配置管理API，包含一系列配置管理API，提供动态类型化属性、线程安全配置操作、轮询框架、回调机制等功能。
- Consul：封装了Consul操作，Consul是一个服务发现与配置工具，与Docker容器可以无缝集成。
- Spring Cloud for Cloud Foundry：通过Oauth2协议绑定服务到Cloud Foundry，Cloud Foundry是VMware推出的开源PaaS云平台。
- Spring Cloud Sleuth：日志收集工具包，封装了Dapper和log-based追踪以及Zipkin和HTrace操作，为Spring Cloud应用实现了一种分布式追踪解决方案。
- Spring Cloud Data Flow：大数据操作工具，作为Spring XD的替代产品，它是一个混合计算模型，结合了流数据与批量数据的处理方式。
- Spring Cloud Security：基于Spring Security的安全工具包，为你的应用程序添加安全控制。
- Spring Cloud ZooKeeper：操作ZooKeeper的工具包，用于使用ZooKeeper方式的服务发现和配置管理。
- Spring Cloud Stream：数据流操作开发包，封装了与Redis、Rabbit、Kafka等发送接收消息。
- Spring Cloud CLI：基于Spring Boot CLI，可以让你以命令行方式快速建立云组件。
- Ribbon：提供云端负载均衡，有多种负载均衡策略可供选择，可配合服务发现和断路器使用。
- Turbine：Turbine是聚合服务器发送事件流数据的一个工具，用来监控集群下Hystrix的Metrics情况。
- Feign：Feign是一种声明式、模板化的HTTP客户端。
- Spring Cloud Task：提供云端计划任务管理、任务调度。
- Spring Cloud Connectors：便于云端应用程序在各种PaaS平台连接到后端，如：数据库和消息代理服务。
- Spring Cloud Cluster：提供Leadership选举，如：ZooKeeper，Redis，Hazelcast，Consul等常见状态模式的抽象和实现。
- Spring Cloud Starters：Spring Boot式的启动项目，为Spring Cloud提供开箱即用的依赖管理。

参考文献：

1. https://zhuanlan.zhihu.com/p/63263168
2. https://toutiao.io/posts/uglos82/preview
3. https://segmentfault.com/a/1190000020849168
4. https://segmentfault.com/a/1190000039842901
5. https://www.cnblogs.com/burningmyself/p/12941807.html
6. https://perkins4j2.github.io/posts/35353/
7. https://juejin.cn/post/6844904205870694413
8. http://www.vispractice.com/yxweb/blogs/API02.html
9. https://blog.csdn.net/dyc87112/article/details/116266269
10. http://www.heartthinkdo.com/?p=1933
11. https://my.oschina.net/icebergxty/blog/3080748

原文链接：https://blog.caogo.cn/2021/06/20/基于Spring-Cloud的微服务架构分析/