微服务架构是互联网很热门的话题，是互联网技术发展的必然结果。它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。虽然微服务架构没有公认的技术标准和规范或者草案，但业界已经有一些很有影响力的开源微服务架构框架提供了微服务的关键思路，例如 Dubbo 和 Spring Cloud。各大互联网公司也有自研的微服务框架，但其模式都于这二者相差不大。

微服务主要的优势如下：

1、降低复杂度

将原来偶合在一起的复杂业务拆分为单个服务，规避了原本复杂度无止境的积累。每一个微服务专注于单一功能，并通过定义良好的接口清晰表述服务边界。每个服务开发者只专注服务本身，通过使用缓存、DAL 等各种技术手段来提升系统的性能，而对于消费方来说完全透明。

2、可独立部署

由于微服务具备独立的运行进程，所以每个微服务可以独立部署。当业务迭代时只需要发布相关服务的迭代即可，降低了测试的工作量同时也降低了服务发布的风险。

3、容错

在微服务架构下，当某一组件发生故障时，故障会被隔离在单个服务中。 通过限流、熔断等方式降低错误导致的危害，保障核心业务正常运行。

4、扩展

单块架构应用也可以实现横向扩展，就是将整个应用完整的复制到不同的节点。当应用的不同组件在扩展需求上存在差异时，微服务架构便体现出其灵活性，因为每个服务可以根据实际需求独立进行扩展。

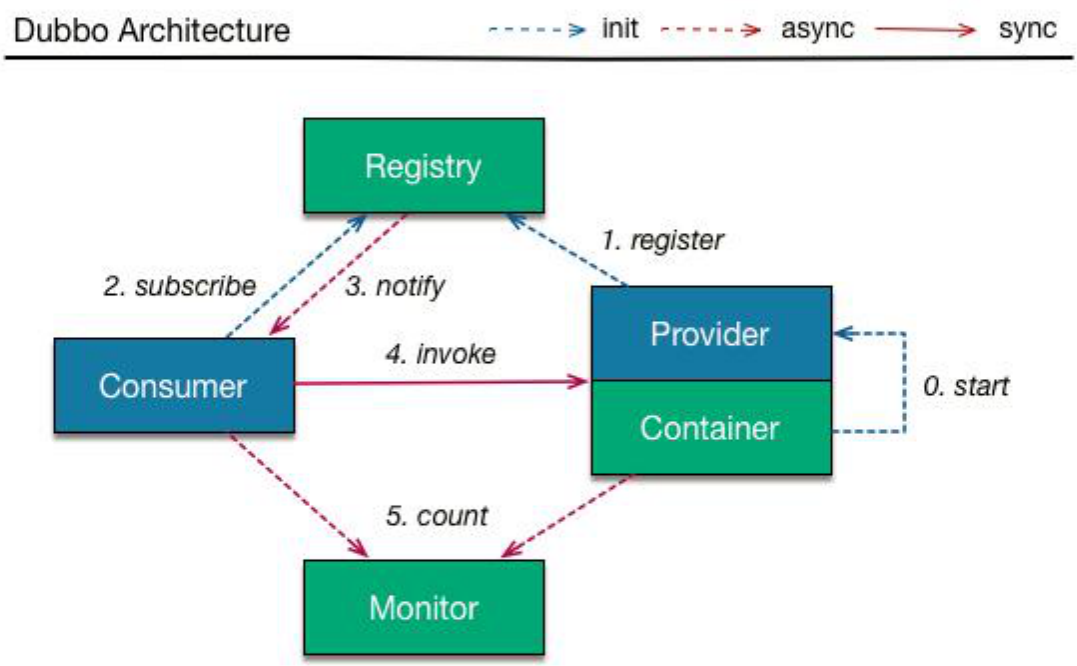
本文主要围绕微服务的技术选型、通讯协议、服务依赖模式、开始模式、运行模式等几方面来综合比较 Dubbo 和 Spring Cloud 这 2 种开发框架。架构师可以根据公司的技术实力并结合项目的特点来选择某个合适的微服务架构平台，以此稳妥地实施项目的微服务化改造或开发进程。

一、核心部件

微服务的核心要素在于服务的发现、注册、路由、熔断、降级、分布式配置，基于上述几种必要条件对 Dubbo 和 Spring Cloud 做出对比。

1、总体架构

- Dubbo 核心部件（如下图）：

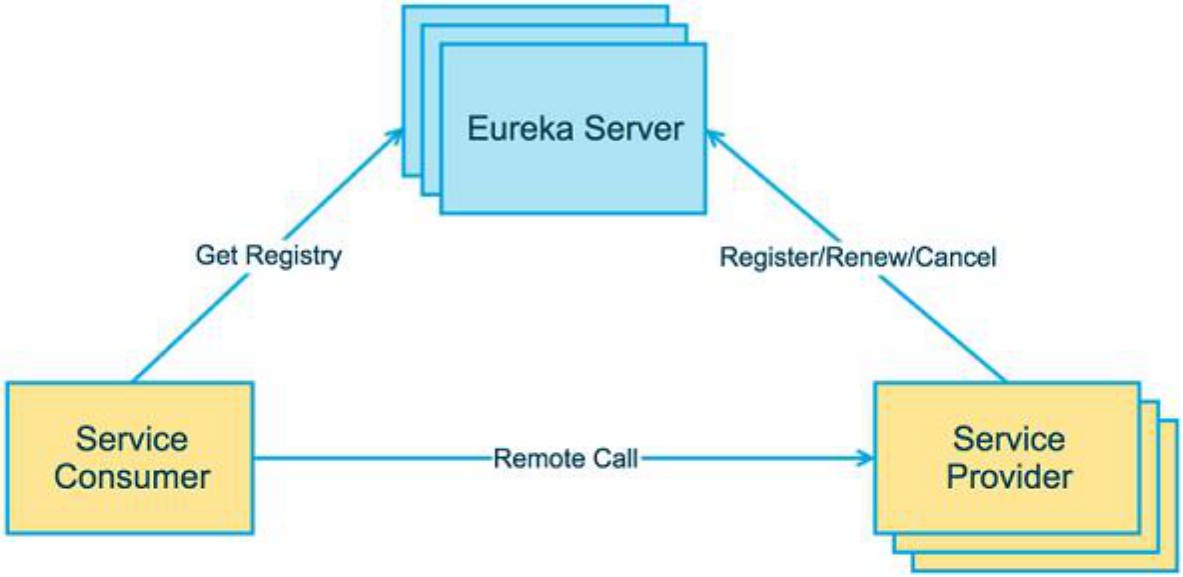


- Provider：暴露服务的提供方，可以通过 jar 或者容器的方式启动服务
- Consumer：调用远程服务的服务消费方。
- Registry：服务注册中心和发现中心。
- Monitor：统计服务和调用次数，调用时间监控中心。（dubbo 的控制台页面中可以显示，目前只有一个简单版本）
- Container：服务运行的容器。

▲Dubbo 总体架构

Spring Cloud 总体架构如下图

- Service Provider： 暴露服务的提供方。
- Service Consumer：调用远程服务的服务消费方。
- EureKa Server： 服务注册中心和服务发现中心。



▲Spring Cloud 总体架构

点评：从整体架构上来看，二者模式接近，都需要需要服务提供方，注册中心，服务消费方。

2、微服务架构核心要素

Dubbo 只是实现了服务治理，而 Spring Cloud 子项目分别覆盖了微服务架构下的众多部件，而服务治理只是其中的一个方面。Dubbo 提供了各种 Filter，对于上述中“无”的要素，可以通过扩展 Filter 来完善。

例如

1．分布式配置：可以使用淘宝的 diamond、百度的 disconf 来实现分布式配置管理

2．服务跟踪：可以使用京东开源的 Hydra，或者扩展 Filter 用 Zippin 来做服务跟踪

3．批量任务：可以使用当当开源的 Elastic-Job、tbschedule

点评：从核心要素来看，Spring Cloud 更胜一筹，在开发过程中只要整合 Spring Cloud 的子项目就可以顺利的完成各种组件的融合，而 Dubbo 缺需要通过实现各种 Filter 来做定制，开发成本以及技术难度略高。

二、通讯协议

基于通讯协议层面对 2 种框架支持的协议类型以及运行效率方面进行比较；

（一）、支持协议

1、Dubbo：dubbo 使用 RPC 通讯协议，提供序列化方式如下：

dubbo：Dubbo 缺省协议采用单一长连接和 NIO 异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况

rmi：RMI 协议采用 JDK 标准的 java.rmi.*实现，采用阻塞式短连接和 JDK 标准序列化方式

Hessian:Hessian 协议用于集成 Hessian 的服务，Hessian 底层采用 Http 通讯，采用 Servlet 暴露服务，Dubbo 缺省内嵌 Jetty 作为服务器实现

http:采用 Spring 的 HttpInvoker 实现

Webservice:基于 CXF 的 frontend-simple 和 transports-http 实现

2、Spring Cloud：Spring Cloud 使用 HTTP 协议的 REST API

（二）、性能比较

使用一个 Pojo 对象包含 10 个属性，请求 10 万次，Dubbo 和 Spring Cloud 在不同的线程数量下，每次请求耗时（ms）如下：

线程数	Dubbo	Spring Cloud
10线程	2.75	6.52
20线程	4.18	10.03
50线程	10.3	28.14
100线程	20.13	55.23
200线程	42	110.21

说明：客户端和服务端配置均采用阿里云的 ECS 服务器，4 核 8G 配置，dubbo 采用默认的 dubbo 协议

点评：dubbo 支持各种通信协议，而且消费方和服务方使用长链接方式交互，通信速度上略胜 Spring Cloud，如果对于系统的响应时间有严格要求，长链接更合适。

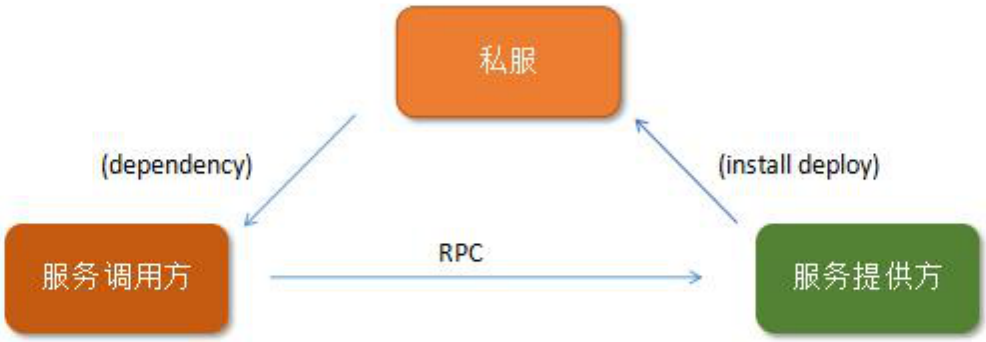
三、服务依赖方式

Dubbo：服务提供方与消费方通过接口的方式依赖，服务调用设计如下：

- interface 层：服务接口层，定义了服务对外提供的所有接口
- Molel 层：服务的 DTO 对象层，
- business 层：业务实现层，实现 interface 接口并且和 DB 交互

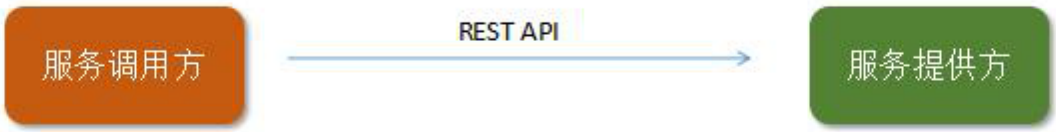
因此需要为每个微服务定义了各自的 interface 接口，并通过持续集成发布到私有仓库中，调用方应用对微服务提供的抽象接口存在强依赖关系，开发、测试、集成环境都需要严格的管理版本依赖。

通过 maven 的 install & deploy 命令把 interface 和 Model 层发布到仓库中，服务调用方只需要依赖 interface 和 model 层即可。在开发调试阶段只发布 Snapshot 版本。等到服务调试完成再发布 Release 版本，通过版本号来区分每次迭代的版本。通过 xml 配置方式即可方面接入 dubbo，对程序无入侵。



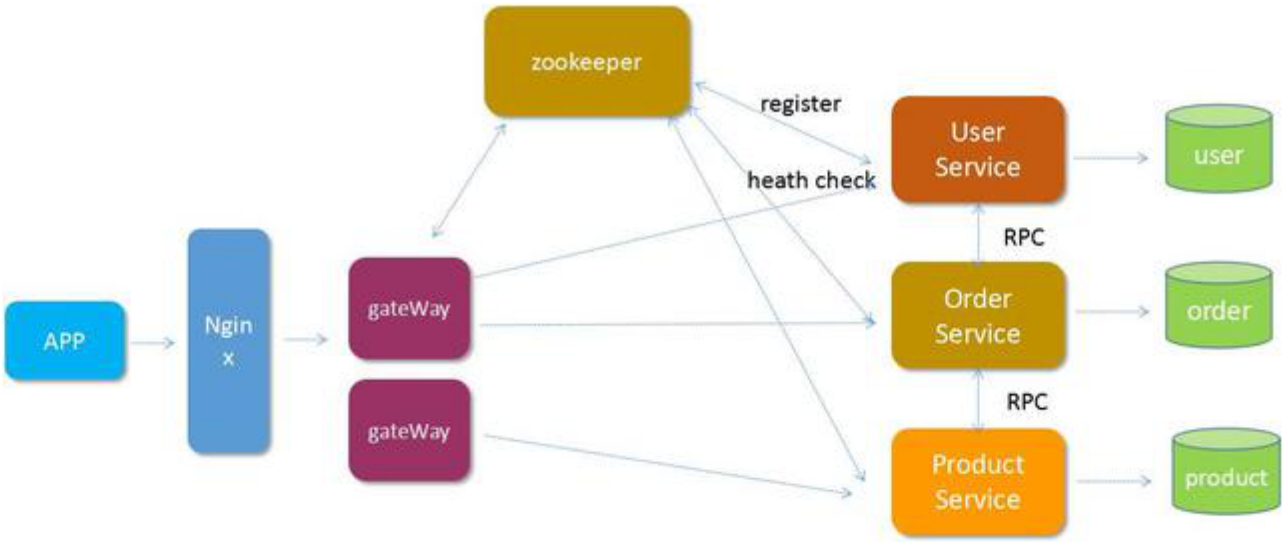
▲Dubbo 接口依赖方式

Spring Cloud：服务提供方和服务消费方通过 json 方式交互，因此只需要定义好相关 json 字段即可，消费方和提供方无接口依赖。通过注解方式来实现服务配置，对于程序有一定入侵。



点评：Dubbo 服务依赖略重，需要有完善的版本管理机制，但是程序入侵少。而 Spring Cloud 通过 Json 交互，省略了版本管理的问题，但是具体字段含义需要统一管理，自身 Rest API 方式交互，为跨平台调用奠定了基础。

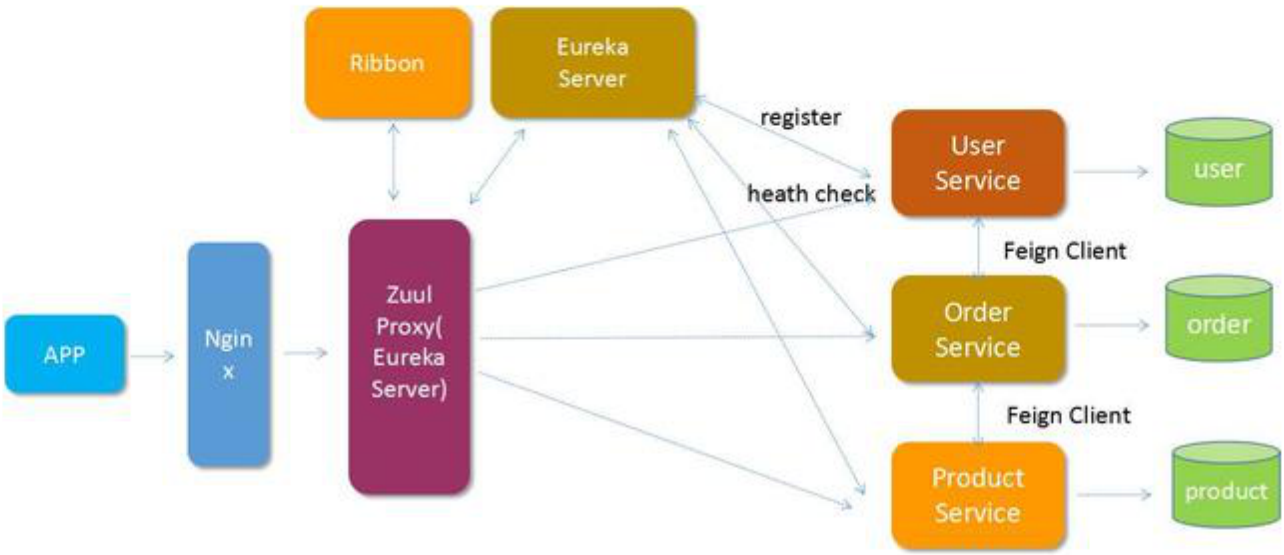
四、组件运行流程



下图中的每个组件都是需要部署在单独的服务器上，*gateway* 用来接受前端请求、聚合服务，并批量调用后台原子服务。每个 *service* 层和单独的 *DB* 交互。

▲*Dubbo* 组件运行流程

- *gateWay*:前置网关，具体业务操作，*gateWay* 通过 *dubbo* 提供的负载均衡机制自动完成
- *Service*：原子服务，只提供该业务相关的原子服务
- *Zookeeper*：原子服务注册到 *zk* 上



▲*Spring Cloud* 组件运行

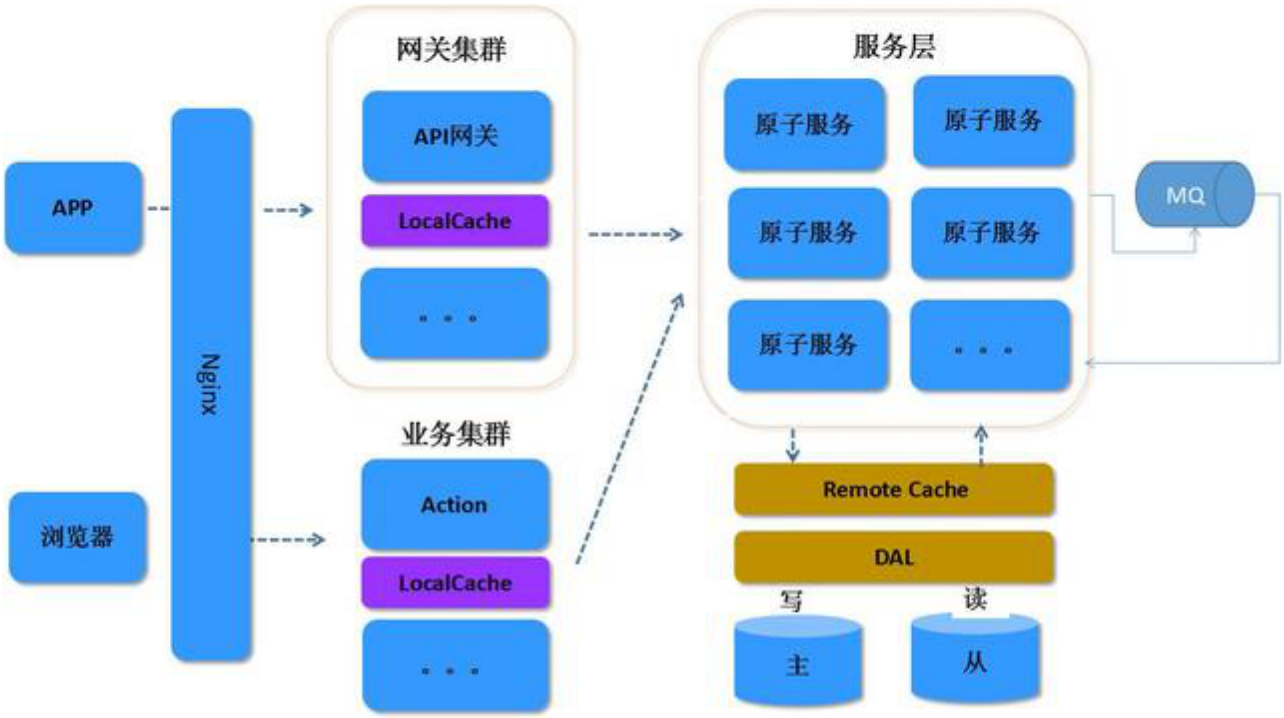
Spring Cloud

- 所有请求都统一通过 API 网关（Zuul）来访问内部服务。
- 网关接收到请求后，从注册中心（Eureka）获取可用服务。
- 由 Ribbon 进行均衡负载后，分发到后端的具体实例。
- 微服务之间通过 Feign 进行通信处理业务。

点评：业务部署方式相同，都需要前置一个网关来隔绝外部直接调用原子服务的风险。Dubbo 需要自己开发一套 API 网关，而 Spring Cloud 则可以通过 Zuul 配置即可完成网关定制。使用方式上 Spring Cloud 略胜一筹。

五、微服务架构组成以及注意事项

到底使用是 dubbo 还是 Spring Cloud 其实并不重要，重点在于如何合理的利用微服务。下面是一张互联网通用的架构图,其中每个环节都是微服务的核心部分。



（一）、架构分解

- 网关集群：数据的聚合、实现对接入客户端的身份认证、防报文重放与防数据篡改、功能调用的业务鉴权、响应数据的脱敏、流量与并发控制等
- 业务集群：一般情况下移动端访问和浏览器访问的网关需要隔离，防止业务耦合
- *Local Cache*：由于客户端访问业务可能需要调用多个服务聚合，所以本地缓存有效的降低了服务调用的频次，同时也提示了访问速度。本地缓存一般使用自动过期方式，业务场景中允许有一定的数据延时。
- 服务层：原子服务层，实现基础的增删改查功能，如果需要依赖其他服务需要在 *Service* 层主动调用
- *Remote Cache*：访问 *DB* 前置一层分布式缓存，减少 *DB* 交互次数，提升系统的 *TPS*
- *DAL*：数据访问层，如果单表数据量过大则需要通过 *DAL* 层做数据的分库分表处理。
- *MQ*：消息队列用来解耦服务之间的依赖，异步调用可以通过 *MQ* 的方式来执行
- 数据库主从：服务化过程中毕竟的阶段，用来提升系统的 *TPS*

（二）注意事项

- 服务启动方式建议使用 *jar* 方式启动，启动速度快，更容易监控
- 缓存、缓存、缓存，系统中能使用缓存的地方尽量使用缓存，通过合理的使用缓存可以有效的提高系统的 *TPS*
- 服务拆分要合理，尽量避免因服务拆分而导致的服务循环依赖
- 合理的设置线程池，避免设置过大或者过小导致系统异常

六、总结

Dubbo 出生于阿里系，是阿里巴巴服务化治理的核心框架，并被广泛应用于中国各互联网公司；只需要通过 spring 配置的方式即可完成服务化，对于应用无入侵。设计的目的还是服务于自身的业务为主。虽然阿里内部原因 dubbo 曾经一度暂停维护版本，但是框架本身的成熟度以及文档的完善程度，完全能满足各大互联网公司的业务需求。如果我们需要使用配置中心、分布式跟踪这些内容都需要自己去集成，这样无形中增加了使用 Dubbo 的难度。

Spring Cloud 是大名鼎鼎的 Spring 家族的产品， 专注于企业级开源框架的研发。 Spring Cloud 自从发展到现在，仍然在不断的高速发展，几乎考虑了服务治理的方方面面，开发起来非常的便利和简单。

Dubbo 于 2017 年开始又重启维护，发布了更新后的 2.5.6 版本，而 Spring Cloud 更新的非常快，目前已经更新到 Finchley.M2。因此，企业需要根据自身的研发水平和所处阶段选择合适的架构来解决业务问题，不管是 Dubbo 还是 Spring Cloud 都是实现微服务有效的工具。

微信扫描二维码获取资料学习

【 一线大厂最新面试题与答案汇总 】 包含阿里，京东、百 度、腾讯、等一线大厂最新面试题与面试题答案。群里还会 讨论 Kafka、Mysql、Tomcat、Docker、Spring、MyBatis、 Nginx、Netty、Dubbo、Redis、Netty、Spring cloud、 JVM、分布式、高并发、性能调优、微服务等架构师最新技能 与问题学习——进群备注好信息即可免费领取。