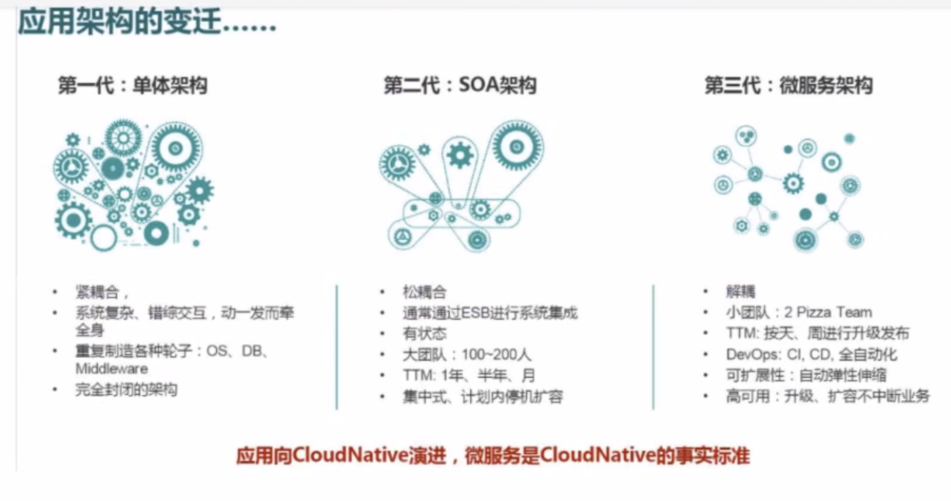
# 网站的架构演变过程

传统架构(单点应用)---分布式架构(以项目进行拆分)—SOA架构(面向服务架构)—微服务架构

无论是SOA架构、微服务架构统称为分布式架构。



## 传统架构

**传统架构(SSH或SSM)属于单体应用，把整个业务模块放入一个项目中进行开发。一个应用搞定全部功能，在单体应中通过包进行区分每个模块，采用MVC架构模式，拆分为控制层、业务逻辑层、数据库访问层。**

**部署简单、维护方便、成本低。当项目规模大，用户访问频率高，并发量大、数据量大时，会大大降低程序的执行效率，甚至出现服务器宕机等情况或单点故障。**

**适合传统管理项目，小型互联网项目。**

## 传统架构中产生的问题：

多个开发人员同时开发一个项目时，可能会产生一些问题。如代码冲突。

**分布式架构是基于传统架构演变过来的。将传统项目已项目模块进行拆分N个子项目，如会员项目、订单项目、支付项目等。每个项目中都拥有独立的数据库，独立reids等。项目与项目之间使用RPC远程通讯。**

项目：包含业务逻辑层和视图层;

服务：只包含业务逻辑层，没有视图层。

## 分布式架构

**分布式架构会把一个项目按照特定的要求（多按照模块或功能）拆分成多个项目，每个项目分别部署到不同的服务器上去。**

**增加了系统可用性，减少单点障碍，导致整个应用不可以；增加了系统重用性，模块化，重用性更高；增加了系统拓展性，有新的模块增加新的项目即可；增加了系统中每个模块的负载能力，因为每个模块是一个项目，所以每个模块的负载能力更强；**

**成本更高，架构更加复杂，整体响应时间变长，业务需要多个项目通信后得到结果。**

**吞吐量更大，吞吐量=请求数/秒**

## 分布式中待解决的问题

1. 分布式架构中各个模块（项目），如何进行通讯？

可以使用http协议也可以使用RPC协议通信，也可以使用其他的通信方式。

1. 项目与项目之间通信过程中如果响应时间过长怎么处理？
2. 负载超过整个分布式项目负载，如何处理
3. 是否有路由转发？统一入口？

## 分布式架构和传统项目的区别：

1.项目粒度更加的细致、慢慢开始适用于互联网公司开发。

2.耦合度降低了。

Maven聚合项目不一定是分布式项目。也可以将传统项目分为web,service,dao三个项目，最终达成一个war包。

## SOA架构

**SOA架构是面向服务架构。可理解为面向于业务逻辑层，将共同的业务逻辑抽取出来，提供其他接口调用。 服务与服务之间使用RPC远程调用技术。**

服务概念:将共同的业务逻辑进行拆分，拆分为独立的一个项目进行部署，没有视图层。

RPC远程调用技术框架：httpclient，springcloud，dubbo，grpc底层核心socket技术或netty实现。

Webservice:底层采用http+xml(SOAP)

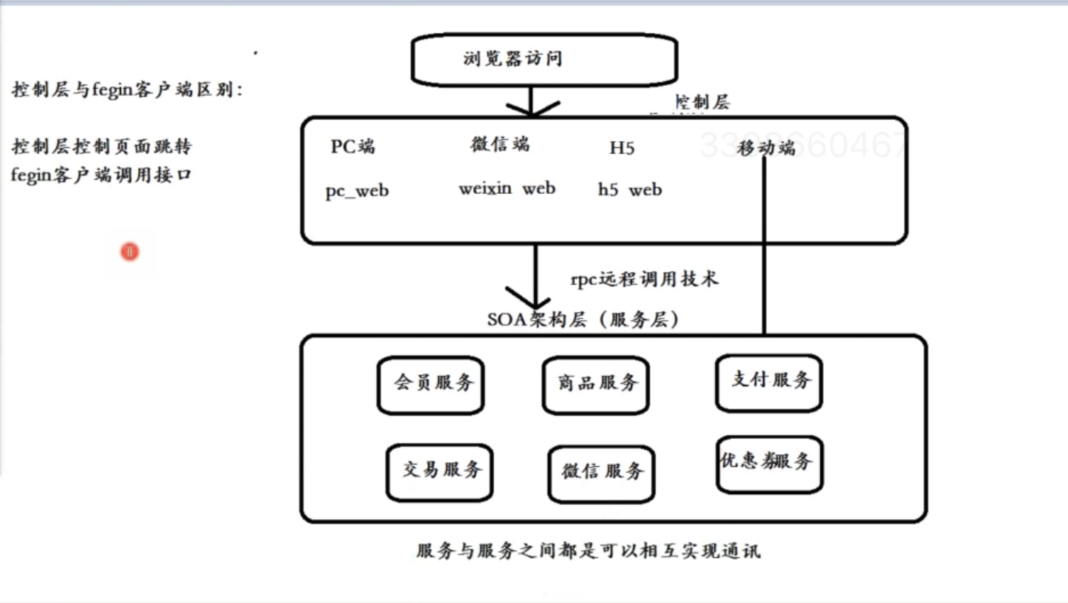
RPC是远程调用技术，两个或多个应用之间实现远程调用。

## SOA架构特点：

1. 底层基于SOAP或ESB（消息总线）实现，底层使用http或https协议+重量级的xml数据格式进行通讯。微服务中以json格式代替XML。

## SOA架构缺点：

1. 依赖与中心化服务发现机制
2. 因为SOA架构采用SOAP协议(http+XML),因为XML传输协议比较占用宽带，整个XML报文中有非常大的冗余数据，所有在微服务中以json轻量级方式代替XML格式报文传输。
3. 服务管理非常混乱，缺少服务管理，服务治理设施不完善。



## 微服务架构

**微服务架构:**

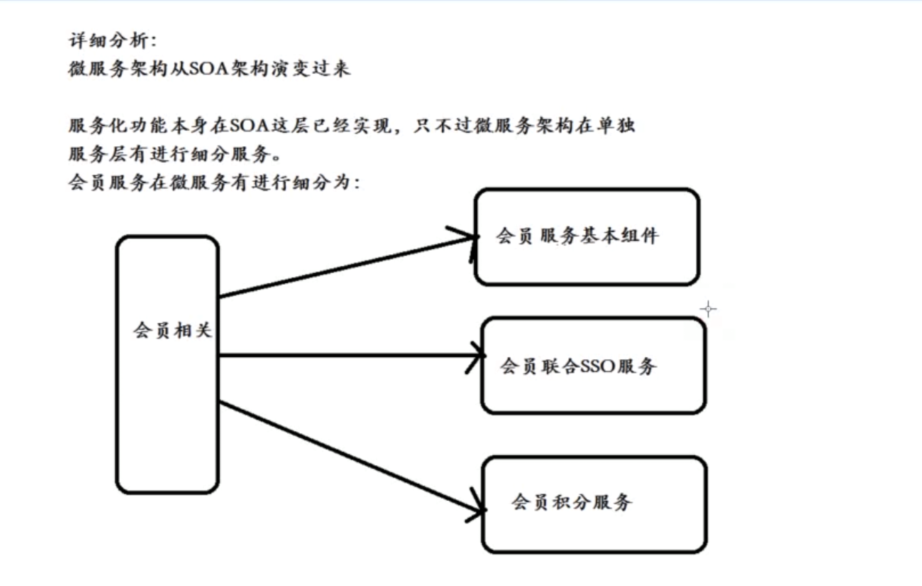
**微服务架构是分布式架构中的一种。微服务是由单一应用程序构成的小服务，拥有自己的进程与轻量化处理，服务按业务功能设计(微服务一个业务一个项目)，以全自动方式部署，与其他服务使用特定的通讯协议（如http或rpc等）（每个项目都是一个标准的web项目），同时，服务会使用最小规模的集中管理（如docker）技术，服务可以用不同编程语言与数据库等。**

## 微服务的核心点

1. 微服务是架构，发展到现在微服务可以说是综合平台(架构以外还包含服务治理、注册中心、服务容灾等相关特性)
2. 微服务中的项目都成为服务。
3. 微服务拆分的颗粒度为业务或模块。
4. 微服务中服务与服务之间使用特定协议通信(如http、RPC)
5. 微服务和Docker结合使用比较方便
6. 微服务是分布式架构中的一种。

**微服务架构是在SOA架构的基础上演化过来的，比SOA架构上的粒度上更加精细，目的是提高效率。每个服务与服务之间互不影响。每个服务必须独立部署。微服务架构根据轻量级，采用restful风格提供API，即http+json格式，更加轻巧，更加适合互联网公司敏捷开发，快速迭代产品。**

**在微服务架构中，服务化的功能本身已经在SOA这层实现，只不过微服务架构在单个服务中又进行细分服务。如会员服务在微服务中可以进一步细分为：会员服务基本组件、会员联合SSO（单点登录）服务、会员积分服务、会员授权服务等。**



在项目中要保证同时支持移动端、PC端、微信端、小程序端、H5端，需要使用SOA架构或微服务架构（包含了SOA架构）。SOAP架构层（服务处）如会员服务、支付服务、订单服务、微信服务、微信服务、优惠券服务等。当浏览器访问web控制层，web层通过RPC远程调用技术调用服务。

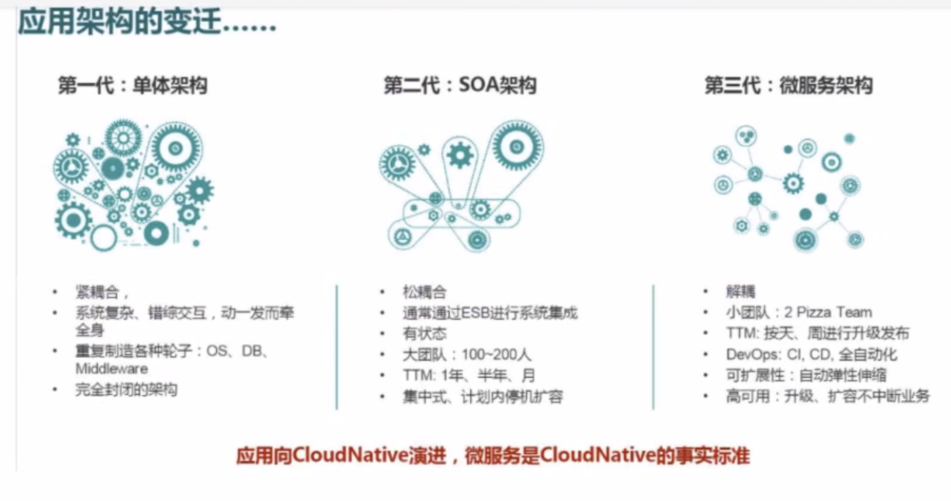
## SOA架构与微服务架构的区别：

SOA 架构：核心消息总线。消息总线过于 笨重在目前项目中已经很少使用了。在SOA

架构中每个项目都满足分布式要求，分别部署到不同的服务器上。

微服务服务架构：核心是把项目拆分成多个小服务，每个服务几天的时间就可以测试上

线，版本迭代。在微服务架构中不要求每个服务独占一台服务器。可以结合docker等技术把多个服务部署到一台服务器上。



1.微服务架构基于 SOA 架构演变过来，继承SOA架构优点，在的微服务架构中去除 SOA 架构中的 ESB 消息总线，来用httpt json (restful） 进行传输。

2.微服务架构比SOA 架构粒度会更加精细，让专业的人去做专业的事情 （专注），目的是提高效率，每个服务于服务之问互不影响，微服务架构中，每个服务必须独立部暑，微服务架构更加轻巧，轻量级。

3.S0A 架构中可能数据库存储会发生共享，微服务强调每个服务都是单独数据库，保证每

个服务于服务之间互不影响。

4.微服务架构比 sOA 架构更加造合与互联网公司敏提开发、快速选代版本，因为粒度更加精细。

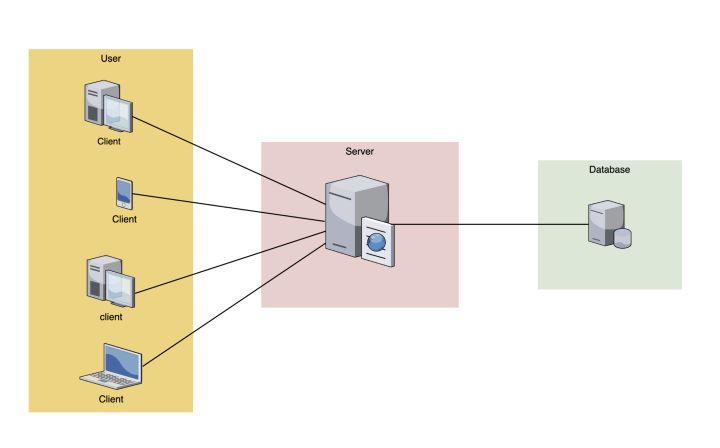
任何一个大型网站的架构都是随着用户量和数据量的不断增加不断迭代演进的结果。

在架构不断迭代演进的过程中会遇到很多问题，技术发展的本质是不断发现问题解决问题，解决问题又发现问题。

# 传统架构（单体应用）

在系统建立初期不会有特别多的客户，将所有的业务打包成一个应用放在tomacat容器中运行，与数据库共享一台服务器，这种架构一般称为单体架构。

在初期这种架构的效率非常高，根据用户的反馈可以快速迭代上线。但是随着用户数量增加，一台服务器的内存和CPU吃紧，很容易造成瓶颈。新的问题改怎么解决?

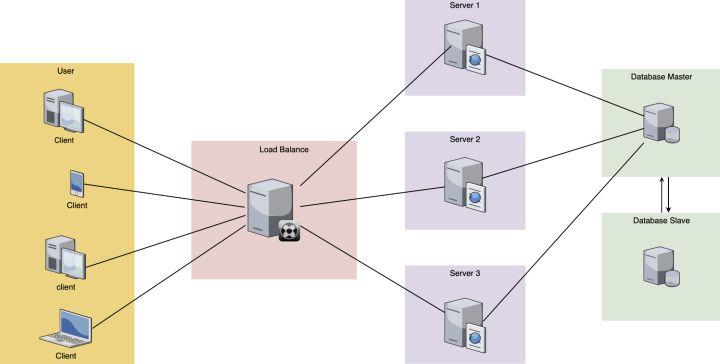


## 什么是传统架构

SSH或SSM架构,属于单点应用,把整个业务模块都放到一个项目中进行开发，分为MVC架构，拆分业务逻辑层、控制层、数据库访问层。一般只适合个人或者小团队开发;

## 传统架构的优点

1. 对于小型服务而言，传统的架构稳定且简单。适合一个人开发或小团队开发
2. 后期，随着我们的用户数渐渐变多，单台服务器的压力扛不住的时候，我们就要用到负载均衡技术，增加多台服务器来抗压，后端的数据库也可以用主从的方式来增加并发量。



## 传统架构缺点：

1. **耦合度太高，一旦某个模块收到影响可能导致服务不可用，可能会影响整个项目;**
2. 当多个开发人员开发同一个项目，可能出现代码冲突,业务不好分配。
3. 随着注册用户增加、网站应用的功能、规模在不断扩展,APP、微信、自助终端等访问渠道的增加，各种新业务，新需求不断涌入,项目工程无节制的变得臃肿庞大,项目维护成本变高。
4. 应用代码高耦合，拓展性和健壮性差。

## 单体架构-应用与数据分离

随着用户请求量增加，一台服务器的内存和CPU持续飙升，用户请求响应事件变慢。这时可以考虑将应用与数据库拆开,各自使用一台服务器。

但是其中一台服务器不能使用，用户所以的请求都会报错，应用无法使用。

## 集群部署

单实例很容易造成单点问题，比如遇到服务器故障或服务能力瓶颈。此时可以用集群来解决。

集群部署是指将应用部署在多个服务器或虚拟机上，用户通过负载均衡随机访问其中的一个实例，从而使多个实例的流量均衡，如果一个实例出现故障可以将其下线，其他实例不受影响仍然可以对外提供服务。

随着用户数量的快速增长，业务逻辑越来越复杂化，代码间耦合越来越严重。此时需要对架构重构，使用微服务架构解决。

# 分布式架构

## 什么是分布式架构

**分布式架构会把一个项目按照特定的要求（多按照模块或功能）拆分成多个项目，每个项目分别部署到不同的服务器上去。**

分布式架构是在传统架构上演变来的,将传统的项目以项目模块进行拆分成N多个子项目。  比如会员项目，订单项目，支付项目，每个项目中都有自己独立的数据库，独立的redis。

# SOA架构

SOA架构(Service Oriented Architecture)是基于分布式架构演变过来的，SOA架构代表面向服务架构（服务化），可理解为面向业务逻辑层开发，将共同的业务逻辑抽取出来形成一个服务，提供给其他服务接口进行调用，服务与服务之间使用rpc远程技术进行通信;

# 微服务架构

## 什么是微服务

**微服务架构是分布式架构中的一种。微服务是由单一应用程序构成的小服务，拥有自己的进程与轻量化处理，服务按业务功能设计(微服务一个业务一个项目)，以全自动方式部署，与其他服务使用特定的通讯协议（如http或rpc等）（每个项目都是一个标准的web项目），同时，服务会使用最小规模的集中管理（如docker）技术，服务可以用不同编程语言与数据库等。**

相关博客

<https://blog.csdn.net/mmgithub123/article/details/124744282>

## 为什么使用微服务？

大部分的开发者都开发过单体应用，无论是传统的 Servlet + JSP，还是 SSM，还是

现在的 SpringBoot，它们都是单体应用。

那么长期陪伴我们的单体应用有什么弊端？我们是面临了什么问题，导致我们要抛弃单体应用转向微服务架构？

主要原因如下：

a) 部署成本高（无论是修改 1行代码，还是 10行代码，都要全量替换）。

b) 改动影响大，风险高（不论代码改动多少，成本都相同）。

c)因为成本高，风险高，所以导致部署频率低 （无法快速交付客户需求）。

d)无法满足快速扩容，弹性伸缩，无法适应云环境特性等问题。

注意：单体应用虽然有上面缺点，但是依然有自己的市场，如果项目规模比较小，办公

类等不需要频繁修改版本的应用使用单体架构还是很方便的。

## 微服务的特点

**微服务架构的特点**：针对特定服务发布，影响小，风险小，成本低;频繁发布版本，快

速交付需求；低成本扩容，弹性伸缩，适应云环境等特点。

微服务架构与现在企业中敏捷开发思想是匹配的，核心都是强调希望项目快速更新迭代

(当项目需要快速更新迭代时微服务架构就特别适合）（这就是为什么使用微服务）

世上没有任何事物是完美的，任何东西都有两面性，有得必有失，那么在选择微服务在解决了快速响应和弹性伸缩的问题同时，它又给我们带来了什么问题？

**微服务架构的缺点**：

分布式系统的复杂性

部署，测试和监控的成本问题

分布式事务和 CAP 的相关问题

分布式中微服务架构相对于单体架构还需要处理很多事情 例：分布式事务、团队合作等问题都需要明确的提前设计好。

微服务架构中常用实现框架

实现框架：springCloud

Spring Cloud包含三个体系：

Spring Cloud Netflix:目前市场上使用最多的。

Spring Cloud Alibaba:基于Dubbo实现的；

Spring其他：为了摆脱Netflix的限制，逐渐推出的一套组件。

## 什么是Spring Cloud

Spring Cloud是Spring下的一个顶级项目，它没有具体的内容，但里面包含了很多

二级子项目，Spring Cloud就是这此一级子项目的统称。

Spring Cloud 包含了很多二级子项目，每个二级子项目都有对应的功能，所以 Spring

Cloud 整体包含的功能是比较强大。

我们主要学习的是 Spring Cloud Netflix相关框架。

强调：**Spring Cloud 是完全基于 Spring Boot 的。**官方文档中就没有 XML配置版本。

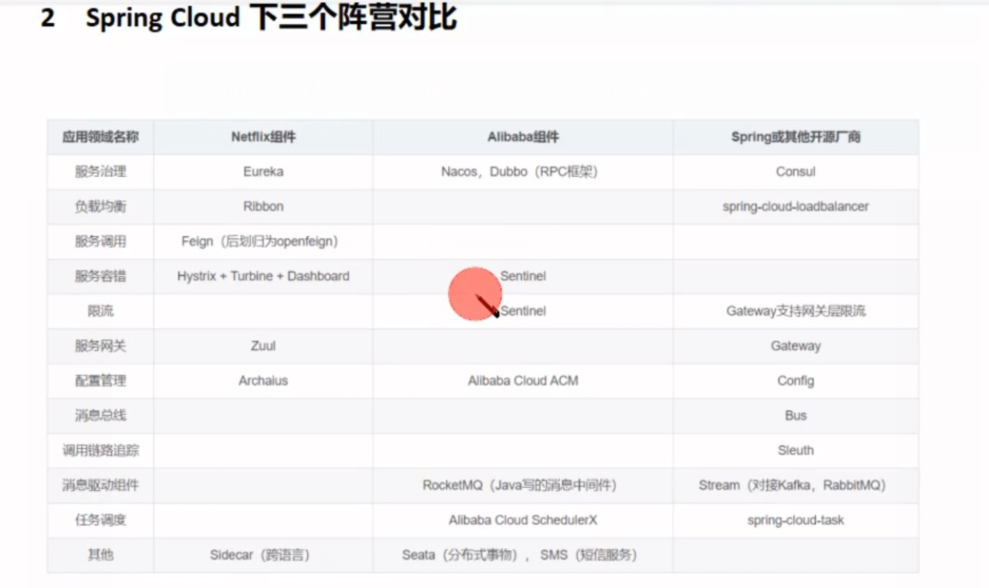
## Netflix 简介

Netflix(Nasdag NFLX) 成立于 1997 年，是一家在线影片租赁提供商，主要提供

Netflx超大数量的 DVD 并免费递送，总部位于美国加利福尼亚州洛斯盖图。

Spring Cloud 集成时会把, Netflix 的软件进行打包成一个依赖，我们项目依赖了对应  
的jar，可以直接使用这个软件，免去了软件安装的过程（这也是 Spring Cloud 非常大的优点）

SpringCloud下三大阵营对比



## SpringCloud版本说明

### 1.常见版本说明

开发中，使用的框架版本，最好是 RELEASE版本或 Final版本。

常见版本号格式为： x.y.z.stage

×-数字格式主版本号，当功能模块有较大更新或者整体架构发生变化时，主版本号

会更新。

y-数字格式次版本号，次版本表示只是局部的一些变动。

乙-数字格式修正版本号，一般是bug 的修复或者是小的变动。

stage- 希腊字母版本号，也称为阶段版本号。用于标注当前版本的软件处于哪个开发阶段。常用的阶段版本包括：BASE、ALPHA、 BATE、RELEASE/FINAL

BASR- 设计阶段。只有相应的设计没有具体的功能实现。

ALPHA- 软件的初级版本。存在较多的bug。

BATE - 表示相对 ALPHA 有了很大的进步，消除了严重的bug，还存在一些潜在的

bug.

RELEASE/FINAL一 该版本表示最终版，即正式发布版本。

### SpringCloud版本说明

Spring Cloud 是一个包含若干子框架的框架集合体，是一个完整的微服务框架体系，

如果使用场景版本号来进行标记，容易混淆主框架版本和子框架版本标记。所以 Spring

cloud 使用一种全新的版本号来对主框架进行版本标记，而子框架的版本标记大多还是使

用常用版本号标记的。

Spring Cloud 版本格式如：版本号命名,stage

版本号命名：Spring cloud 主框架版本号是使用英国伦敦地铁站名称来进行标记的，

并根据地铁站名称的首字母的英文自然升序排列来识别版本的递增。如：Angle、 Brixton.

Camden、 Dalston、 Edgware. Finchley、Greenwich、Hoxdon 等。后续饭本提升会

继续根据首字母升序排列。

stage：阶段版本号。常用的阶段版本包括：BUILD-XXX[SNAPSHOT]、GA、 PRE(M1、

M2#). RC. SR.

BUILD-XXX[SNAPSHOT]-开发版本、 一般是开发团队内部使用。

GA- 稳定版，内部开发到一定阶段了，各个模块集成后，经过全面测试发现没有问

题，可对外发行了。这个时候叫 GA(General Availability)。基本上可以使用了。没有严

重的 BUG 问题，但是有未测出的 BUG 隐患。不推荐商业使用。

PRE - 里程碑版，由于 GA 还不属于公开发行版，里面还有些功能不完善或者 bug,

于是就有了milestone（里程碑版）。milestone 版主要修复了一些bug。一个GA后，

一般会有多个里程本版。例如口M1 M2 M3. 不推荐商业使用。

RC-候选发布版，从 BUILD后到 GA 在到M 基本上系统就算定型了，这个时候系

统就进入 Release Candidate(候选发布版)。该阶段的软件类似于最终发行前的一个观

察期，该期间只对一些发现的等级高的 bug 进行修复。发布RC1 RC2 等版本。可以考虑

RC版本。

SR-正式发布版，公开正式发布。正式发布版一般也有多个发布，例如 SR1 SR2SR3

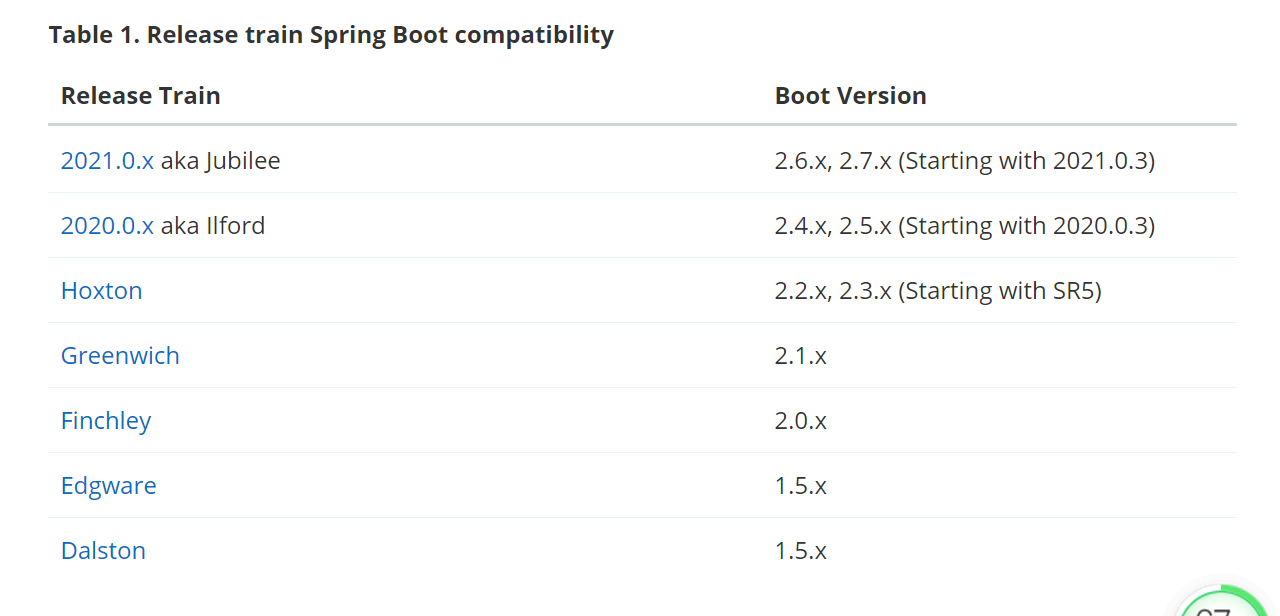
等等，

一般是用来修复大 bug或者优化。 最好使用SR版本。

要注意： Spring cloud基于springboot，不同的spring cloud要使用不同版本的springboot版本。使用spring cloud H版本一定使用的是2.2.x不能使用2.3.x版本，如果使用2.3.x版本可能出现问题。

## SpringCloud与springboot版本对应关系

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



# 微服务架构中的问题

## 服务与服务之间如何交换

### 问题

1.微服务架构下一个应用拆分为多个独立微服务。多个微服务可以使用Docker容器化进行多实例部署。

架构演化到这里遇到了一个难题，如果要查询用户所有的订单，用户服务可能会依赖订单服务，用户服务如何与订单服务交互呢？订单服务有多个实例该访问哪一个？

### 解决方案

1. **服务地址硬编码**
2. 服务的地址写死在数据库或者配置文件，通过访问DNS域名进行寻址路由。
3. 如果遇到大促活动需要对服务实例扩容，大促完需要对服务实例进行下线，运维人员要做大量的手工操作，非常容易误操作。

**2.服务注册和发现**

在单体架构中一个应用程序就是一个服务包，包内模块通过函数方式相互调用，模型足够简单，没有服务注册和发现一说。

在微服务中，将一个应用拆分为多个微服务，微服务部署在不同的服务器、不同的容器下，微服务间要相互调用。服务注册和发现成为了一种很好的解决方案。

相关博客：

<https://blog.51cto.com/u_15659694/5755282>