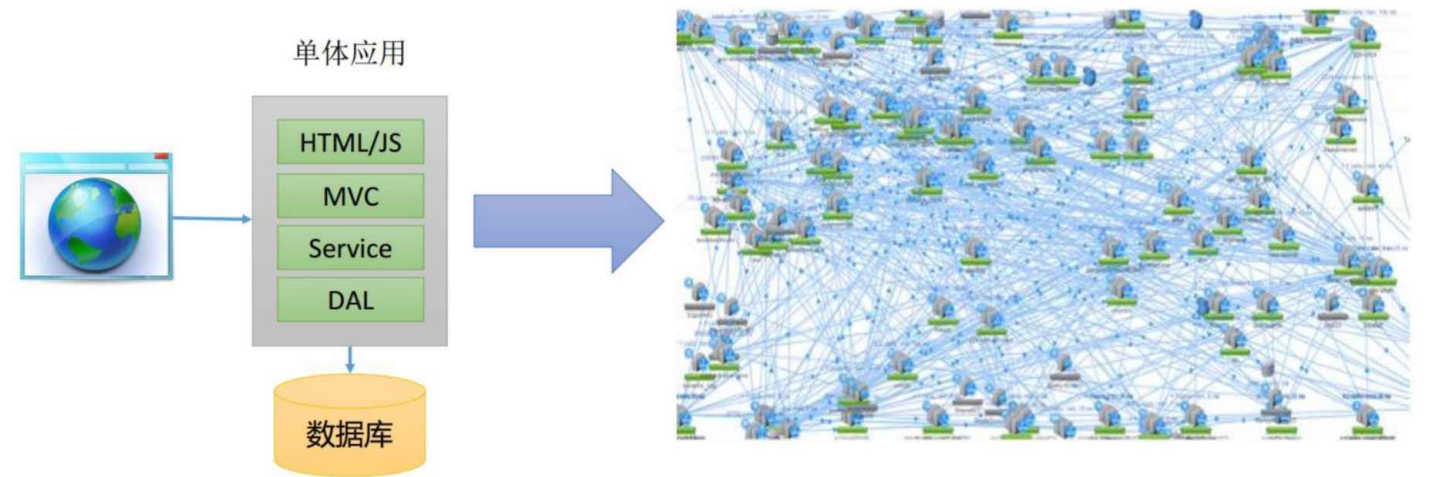


在学习Spring Cloud 之前, 我们先来了解下什么是微服务, 以及微服务的发展史. 在架构发展的过程中, 项目开发遇到了哪些问题, 以及Spring Cloud是用来解决什么问题的. 这将对咱们后面课程的学习有一定的帮助.

1. 认识微服务

下图表示了服务架构从单体应用逐渐转变为微服务应用的过程



1.1 单体架构

很多创业公司早期或者传统企业会把业务的所有功能实现都打包在一个项目, 这就是单体架构.

业务的所有功能实现都打包在一个war包或者Jar包中, 这种方式就称为单体架构

比如Spring课程中的博客系统,前端+后端+数据库实现, 都在一个项目中, 这种架构就称为单体架构.

以大家都很熟悉的电商系统为例, 电商系统包括: 用户管理, 商品管理, 订单管理, 支付管理, 库存管理, 物流管理等等, 项目早期我们会把这些模块都写在一个web项目中, 然后统一部署到一个Web服务器中



这种架构开发简单, 部署简单, 一个项目就包含了所有的功能, 省去了多个项目之间的交互和调用消耗. 直接部署在一个服务器即可.

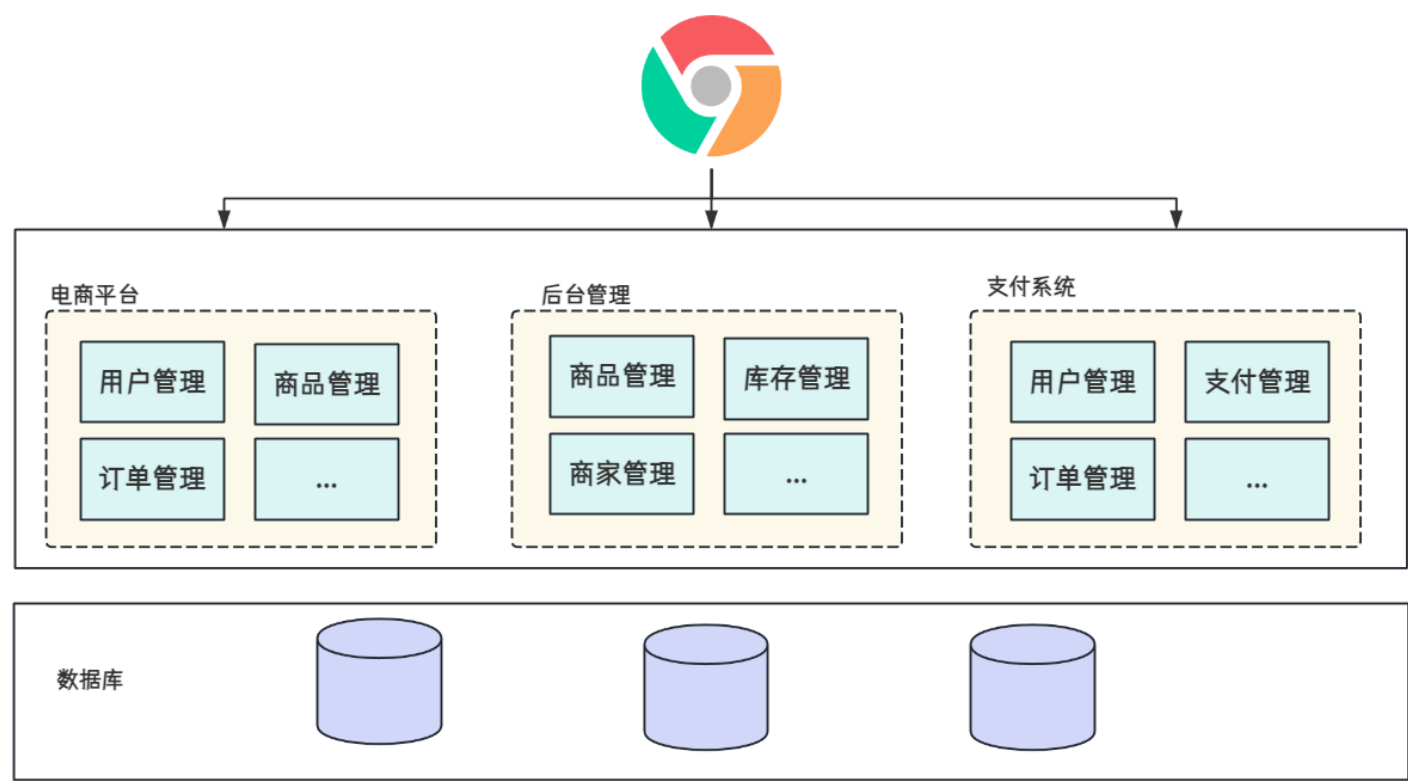
1.2 集群和分布式架构

当网站的用户量越来越大, 需求也会越来越多, 流量也会越来越大, 服务可能就会面临以下问题:

- 后端服务器的压力就会越来越大, 负载越来越高, 甚至出现无法访问的情况
- 业务场景逐渐复杂. 为了满足用户的需求, 单体应用也会越来越大. 各个业务代码之间的耦合度也会越来越高. 任何一个问题, 都需要整个项目重新构建, 发布.
- 一个微小的问题, 可能会导致整个应用挂掉

我们从两个方面进行优化:

- 横向: 添加服务器, 把单台机器变成多台机器的集群.
- 纵向: 把一个应用, 按照业务进行拆分, 拆分为多个项目. 此架构也称为垂直架构.



以单体结构规模的项目为单位进行垂直划分. 也就是将一个大项目拆分成一个一个单体结构项目. 项目和项目之间相对比较独立, 接口多为数据同步功能.

集群和分布式

- **集群(cluster)**是将一个系统完整的部署到多个服务器上, 每个服务器都能提供系统的所有服务, 多个服务器通过负载均衡调度完成任务. 每个服务器称为集群的节点(node)

- **分布式**是将一个系统拆分为多个子系统，多个子系统部署在多个服务器上，多个服务器上的子系统协同合作完成一个特定任务。

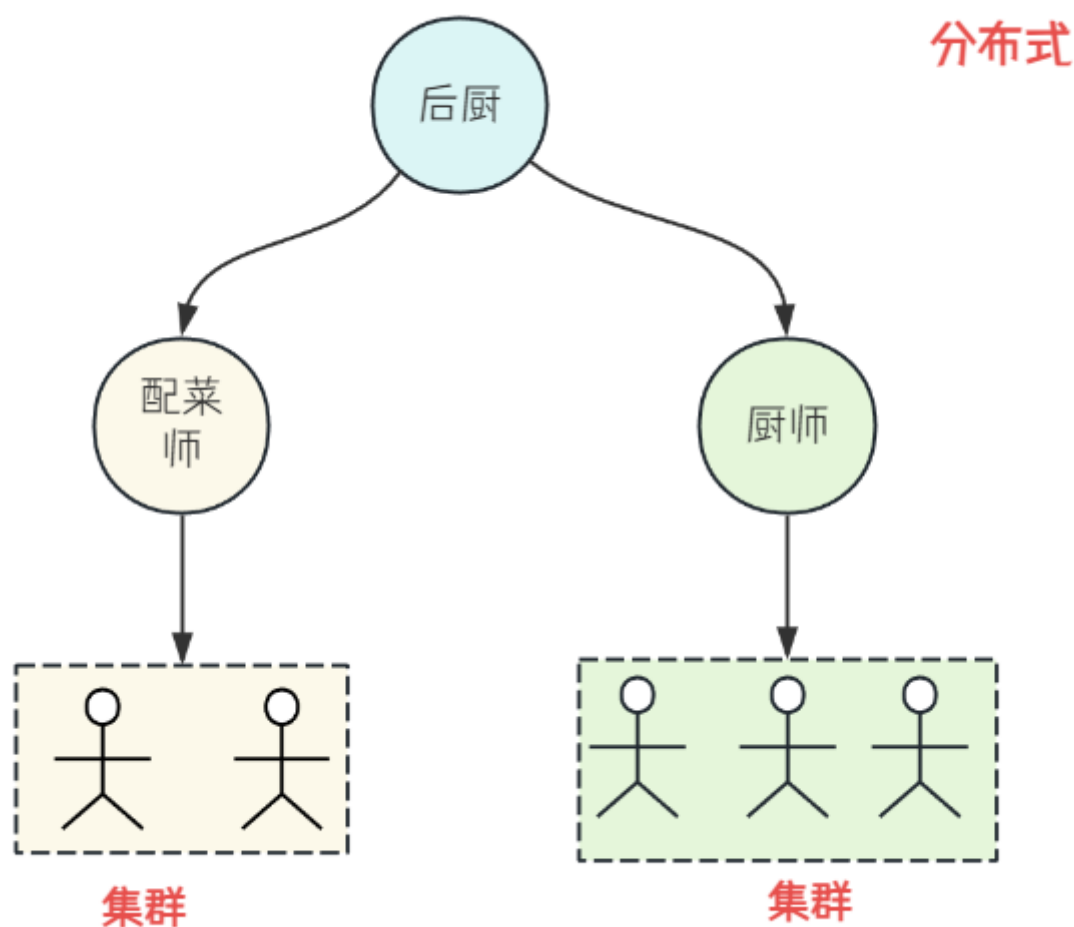
比如:

一个饭店只有一个厨师, 这个厨师负责备菜, 洗菜, 切菜, 炒菜.

随着这个饭店的生意越来越好, 这个厨师忙不过来了, 饭店又请了一个厨师, 新厨师和老厨师做一样的事情, 也是洗菜, 切菜, 炒菜. 这两个厨师的关系就是集群.

为了让厨师专心炒菜, 饭店又请了一个配菜师, 负责备菜, 洗菜, 切菜. 厨师和配菜师的关系就是分布式.

后来一个配菜师也忙不过来了, 又请了一个配菜师, 这两个配菜师的关系就是集群.



集群和分布式区别和联系

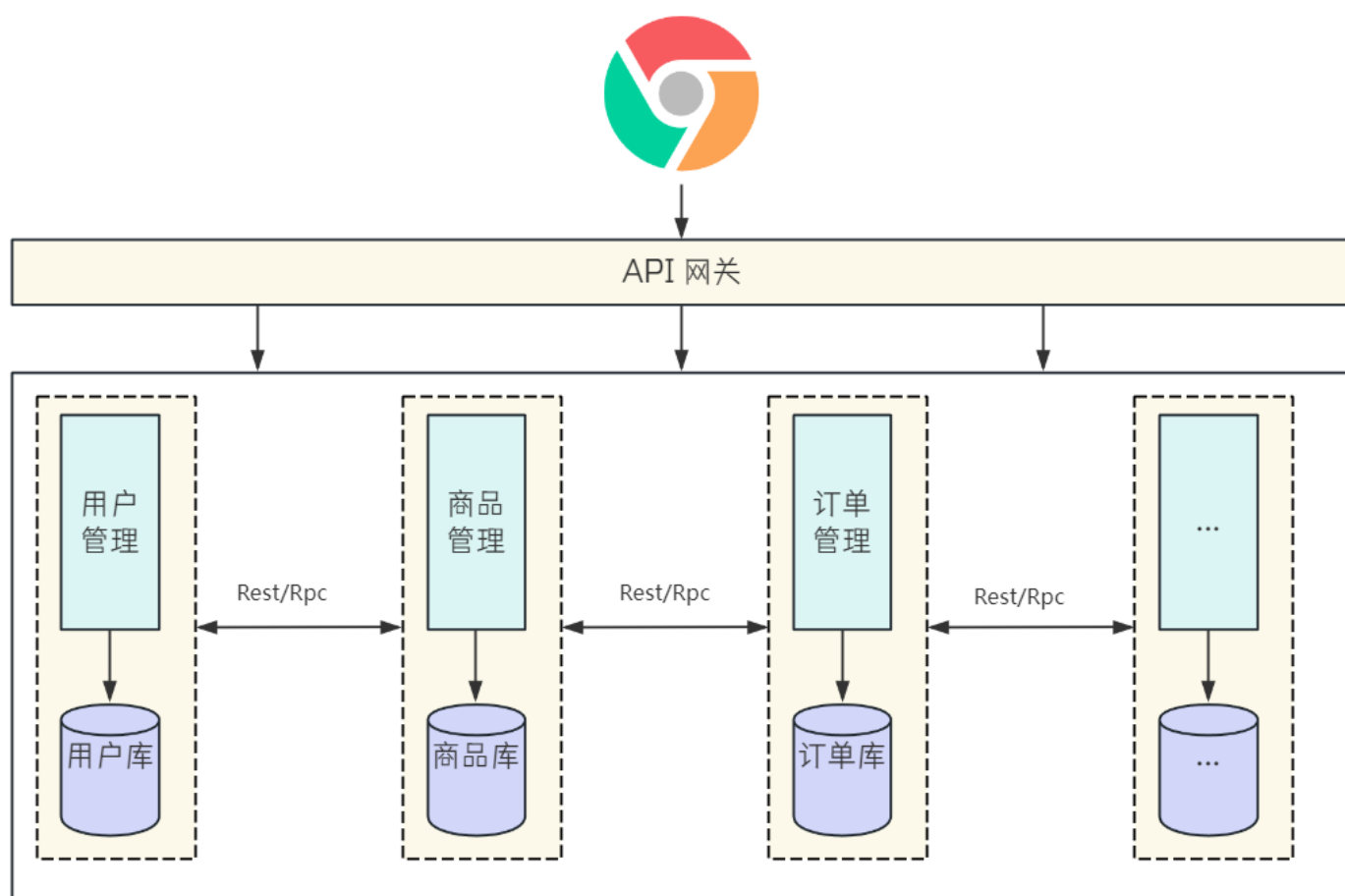
1. 从概念上. 集群是多个计算机做同样的事, 分布式是多个计算机做不同的事
2. 从功能上. 集群的每一个节点功能是相同的, 并且可以替代的. 分布式也是多个节点组成的系统, 但是每个节点完成的业务是不同的, 一个节点出现问题, 这个业务就不可访问了.

3. 从关系上, 分布式和集群在实践中, 很多时候是互相配合使用的. 比如分布式的某一个节点, 可能由一个集群来代替. 分布式架构大多是建立在集群上的. 所以实际的分布式架构设计中并不会把分布式和集群单独区分, 而是统称: 分布式架构.

1.3 微服务架构

从上图中可以看出, 按照业务进行拆分后, 会有一些重复的功能开发, 比如订单系统, 电商平台和支付系统都会涉及.

在分布式架构下, 当部署的服务越来越多, 重复的代码就会越来越多, 服务的调用关系也会越来越复杂. 我们可以把一些通用的, 会被多个上层服务调用的共享业务, 提取成独立的基础服务, 组成一个个微小的服务. 这就是微服务.



简单来说, 微服务就是很小的服务. 小到一个服务只对应一个单一的功能, 只做一件事. 这个服务可以单独部署运行,

微服务之间可以采用REST和RPC协议进行通信.

REST和RPC 后面再讲, 此处把他理解为接口的约定.

从这个角度来看, 微服务架构是分布式架构的一种拓展, 这种架构模式下它拆分粒度更小, 服务更独立. 可以理解为: **微服务是一种经过良好架构设计的分布式架构方案.**

分布式架构&微服务架构

分布式: 服务拆分, 拆了就行.

微服务: 指非常微小的服务, 更细粒度的垂直拆分, 通常指不能再拆的服务

分布式架构侧重于压力的分散, 强调的是服务的分散化. 微服务侧重于能力的分散, 更强调服务的专业化和精细分工. 从实践的角度来看, 微服务架构通常是分布式服务架构, 反之则未必成立. 所以, **选择微服务通常意味着需要解决分布式架构的各种难题.**

1.4 微服务带来的挑战

随着产品的复杂性和流量的增加, 技术架构也在不断的发生变化. 不论是早期的单体架构, 还是现在广泛使用的微服务架构, 都是为了更好的服务产品, 解决问题.

微服务架构带来好处的同时, 也面临着一些挑战, 从单体服务转向微服务意味着管理更加复杂. 接下来我们从优势和挑战两个方面分析一下微服务架构.

优势

- 易开发和维护. 每个微服务负责的业务比较清晰, 体量小, 开发和维护成本降低.
- 容错性高. 一个服务发生故障, 可以使故障隔离在单个服务中, 不影响整体服务故障.
- 扩展性好. 每个服务都是独立运行的, 我们可以结合项目实际情况进行扩展, 按需伸缩.
- 技术选型灵活. 每个微服务都是单独的团队来运维, 可以根据业务特点和团队特点, 选择适合的技术栈.

挑战

虽然微服务具备很多的优势, 但由于服务数的增加, 服务治理也是我们面临的巨大挑战.

- 服务依赖. 随着服务的数量增多, 服务之间的关系也会变得更加复杂. 一个服务的更改, 需要考虑对其他服务的影响.
- 运维成本. 一个业务流程会涉及多个微服务共同完成, 有更多的服务需要编译, 部署, 运行, 甚至可能是不同的编程语言, 不同的运行环境, 当然也需要集群来处理故障转移等. 这对于运维人员而言, 挑战是巨大的.
- 开发和测试. 一个业务流程可能涉及多个微服务共同完成, 服务调用引入网络延迟, 不可靠的网络, 如何进行容错处理等问题. 这对开发和测试而言, 难度也会提升.
- 服务监控. 在一个单体结构中, 很容易实现服务的监控. 因为所有功能都在一个服务中, 微服务架构下, 不仅需要对整个链路进行监控, 还需要对每一个服务实现监控.
- 负载均衡. 微服务架构中的服务实例数量可能非常庞大, 因此需要有效的服务发现和负载均衡机制来管理请求流量和保证高可用性
- ...

比如企业员工管理

员工少的时候, 一个员工可能各方面的工作都要会, 但是员工之间的协同工作会少一点, 员工管理也会相对简单.

随着企业的发展, 事情越来越多, 一个员工的能力有限, 就需要招聘更多的人, 这时候就涉及到员工的管理了.

员工越多, 员工的管理就越复杂

服务也是类似.

选择微服务架构的话, 以上这些问题都需要我们解决, 我们是自己研发还是选择市场上比较成熟的技术拿来用呢?

全球的互联网公司都在积极尝试自己的微服务落地方案. 在Java领域, 最引人注目的就是Spring Cloud了

2. 微服务解决方案- Spring Cloud

2.1 什么是Spring Cloud

我们先看下官网的介绍:

Spring Boot

Spring Framework

Spring Data

Spring Cloud

Spring Cloud Azure

Spring Cloud Alibaba

Spring Cloud for Amazon Web Services

Spring Cloud Bus

Spring Cloud Circuit Breaker

Spring Cloud CLI

Spring Cloud - Cloud Foundry Service Broker

Spring Cloud Commons

Spring Cloud Config

Spring Cloud Consul

Spring Cloud Contract

Spring Cloud Function

Spring Cloud Gateway

Spring Cloud GCP

Spring Cloud Kubernetes

Spring Cloud

2023.0.0

OVERVIEWLEARNSAMPLES

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, short lived microservices and contract testing). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

Features

Spring Cloud focuses on providing good out of box experience for typical use cases and extensibility mechanism to cover others.

- Distributed/versioned configuration
- Service registration and discovery
- Routing
- Service-to-service calls
- Load balancing
- Circuit Breakers
- Distributed messaging
- Short lived microservices (tasks)
- Consumer-driven and producer-driven contract testing

Spring Cloud 提供了一些可以让开发人员快速构建分布式服务的工具, 比如配置管理, 服务发现, 熔断, 智能路由等. 他们可以在任何分布式环境中很好的工作.

简单来说, **Spring Cloud 就是分布式微服务架构的一站式解决方案**, 是微服务架构落地的多种技术的集合.

比如:

- Distributed/versioned configuration 分布式版本配置
- Service registration and discovery 服务注册和发现
- Routing 路由
- Service-to-service calls 服务调用
- Load balancing 负载均衡
- Circuit Breakers 断路器
- Distributed messaging 分布式消息
-



Spring Cloud 并不是Spring 团队研发的框架, 它只是把一些比较优秀的解决微服务架构中常见问题的开源框架基于SpringCloud规范进行了整合, 并基于SpringBoot的风格,对这些组件进行封装, 屏蔽掉了复杂的配置和实现原理. 为开发者提供了开箱即用的微服务开发体验.

这些开源技术的框架是由各个公司来维护的. Spring Cloud 就是这些微服务的大管家.

2.2 Spring Cloud版本

Spring Cloud 是一个由很多子项目组成的庞大项目, 这些子项目由各个公司来维护的, 所以发布阶段也是不同的.

为了管理主项目和子项目的依赖关系, 以及为了避免和子项目版本的冲突, 主项目版本命名并没有采用和子项目数字版本化的形式, 而是采用了英文名称.

这个英文版本名称也比较有趣, Spring Cloud 采用了英国伦敦地铁站的名称来命名, 并由地铁站名称字母A-Z依次类推的形式来发布迭代版本.

- Angel
- Brixton
- Camden
- Dalston
- Edgware
- Finchley
- Greenwich
- Hoxton

但英文版本号太复杂了, 从 Hoxton 版本之后, Spring Cloud的版本就变成了2020.0.0 这样的日期版本号了

- 2020.0.x aka Ilford
- 2021.0.x aka Jubilee
- 2022.0.x aka Kilburn
- 2023.0.x aka Leyton

Spring Cloud和SpringBoot的关系

Spring Cloud中的所有子项目都依赖SpringBoot, 所以SpringBoot 和Spring Cloud的版本之间也存在一定的对应关系

Release Train	Spring Boot Generation
2023.0.x aka Leyton	3.2.x
2022.0.x aka Kilburn	3.0.x, 3.1.x (Starting with 2022.0.3)
2021.0.x aka Jubilee	2.6.x, 2.7.x (Starting with 2021.0.3)
2020.0.x aka Ilford	2.4.x, 2.5.x (Starting with 2020.0.3)
Hoxton	2.2.x, 2.3.x (Starting with SR5)
Greenwich	2.1.x
Finchley	2.0.x
Edgware	1.5.x
Dalston	1.5.x

比如SpringBoot 3.2.X对应的SpringCloud版本是2023.0.X

如果我们有一个SpringBoot项目, 我们希望在这个项目中添加SpringCloud的一些组件, 需要根据当前项目的SpringBoot版本, 选择SpringCloud的版本(当然, 新项目不存在这个问题)

2.3 Spring Cloud实现方案

在Spring Cloud的规范下, 有很多实现, 其中最为出名的是

- Spring Cloud Netflix
- Spring Cloud Alibaba


Spring Cloud Netflix

Spring Cloud Netflix是 Netflix OSS(Netflix Open Source Software)在Spring Cloud规范下的实现. 包含的组件及其主要功能大致如下:

- Eureka: 服务注册和发现

- Zuul: 服务网关
- Ribbon: 负载均衡
- Feign: 服务调用组件
- Hystrix: 断路器, 提供服务熔断和限流
- Hystrix Dashboard: 监控面板
- ...

在很长的一段时间里, Spring Cloud 一度被泛指 Spring Cloud Netflix. Spring Cloud 一直以来把 `Netflix OSS` 套件作为其官方默认的一站式解决方案. 然而, Netflix公司在2018年前后宣布其核心组件Hystrix、Ribbon、Zuul等均进入维护状态, Spring Cloud 也被迫宣布删除这些维护模块.

 spring-cloud-netflix 并没有从Spring Cloud的依赖中完全删除, 只是从2020.0版本起, 他只管理Eureka.

Spring Cloud Netflix 在很多公司都有大规模使用, 一旦停止更新, 短期看影响不大, 但长期显然是不合适的, Spring Cloud官方也提供了一些替换建议.

Netflix	推荐替代品	说明
Hystrix	Resilience4j	Hystrix也推荐大家使用Resilience4j代替自己
Hystrix Dashboard / Turbine	Micrometer + Monitoring System	说白了, 监控这件事交给更专业的组件去做
Ribbon	Spring Cloud Loadbalancer	忍不住了, Spring终究亲自出手
Zuul 1	Spring Cloud Gateway	忍不住了, Spring终究亲自出手
Archaius 1	Spring Boot外部化配置 + Spring Cloud配置	比Netflix实现的更好、更强大

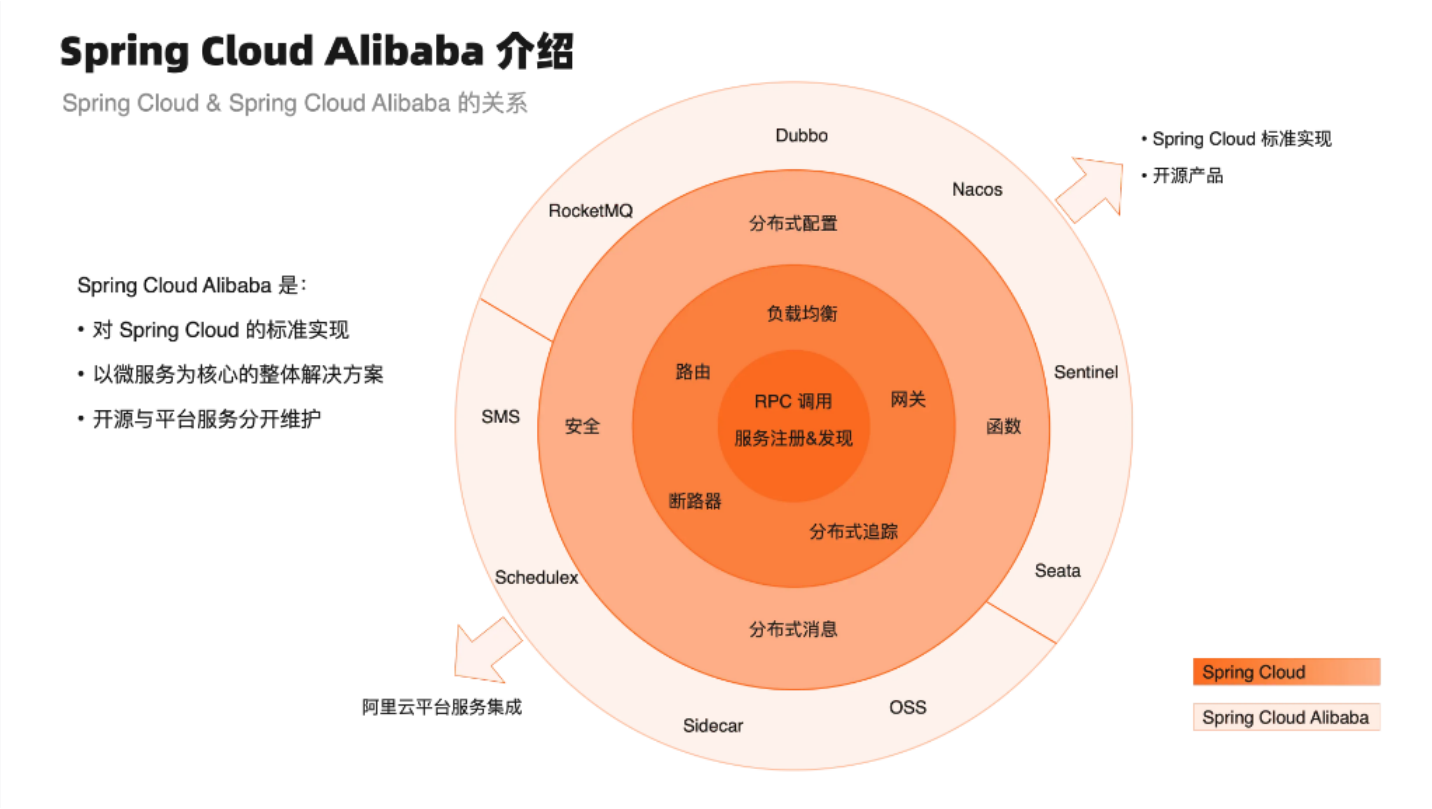
Spring Cloud Alibaba

Spring Cloud Alibaba 是阿里巴巴集团下的开源组件和云产品在Spring Cloud规范下的实现.

虽然Spring Cloud Alibaba目前并不是Spring Cloud官方推荐的默认方案, 但是Spring Cloud Alibaba是阿里中间件团队主导的一个新生项目, 正处于高速迭代中. 甚至在Alibaba的开源组件还没有织入SpringCloud生态之前, 就已经在各大公司广泛使用了.

官方网站: [Spring Cloud Alibaba 是什么 | Spring Cloud Alibaba](#)

如果说Spring Cloud Netflix 是 Spring Cloud 的第一代实现, 那么Spring Cloud Alibaba 也可以看做是 Spring Cloud 的第二代实现, 主要由 Nacos、Sentinel、Seata 等组成.



Spring Cloud Alibaba 吸收了 Spring Cloud Netflix 微服务框架的核心架构思想, 并进行了高性能改进. 自 Spring Cloud Netflix 进入停更维护后, Spring Cloud Alibaba 逐渐代替它成为主流的微服务框架.

Spring Cloud 实现对比

	SpringCloud官方	Spring Cloud Netflix	Spring Cloud Alibaba
服务注册/发现	Eureka	Eureka	Nacos
服务调用	OpenFeign	Feign	Dubbo
配置中心	SpringCloudConfig	Archaius	Nacos
服务网关	SpringCloudGateway	Zuul	SpringCloudGateway
负载均衡	SpringCloud LoadBalance	Ribbon	Dubbo

