SpringCloud 微服务介绍

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

1、微服务介绍

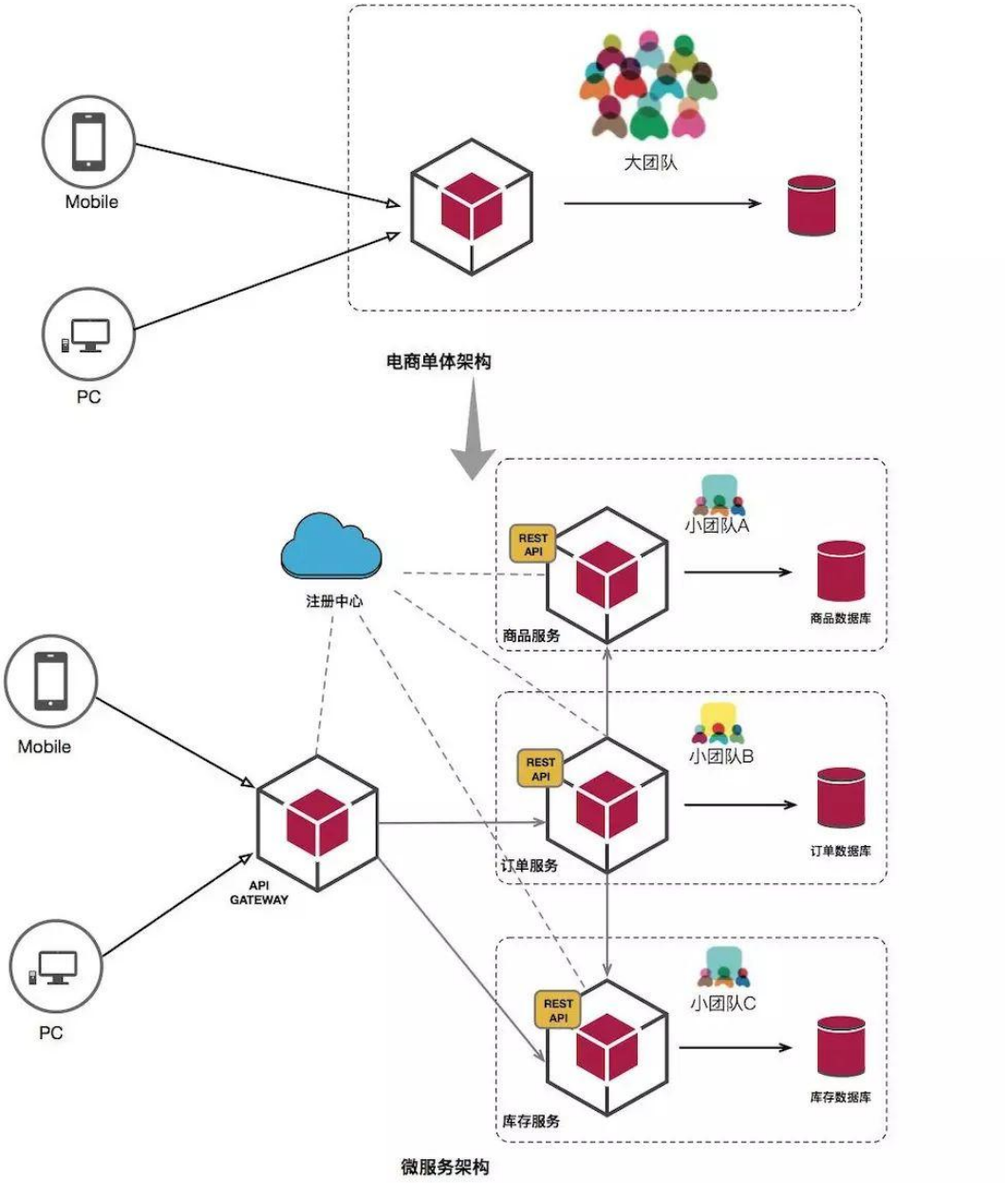
1.1、什么是微服务？

微服务是用于构建应用程序的架构风格，一个大的系统可由一个或者多个微服务组成，微服务架构可将应用拆分成多个核心功能，每个功能都被称为一项服务，可以单独构建和部署，这意味着各项服务在工作和出现故障的时候不会相互影响。

总结：微服务架构是把一个大的系统按照不同的业务单元分解成多个职责单一的小系统，并利用简单的方法使多个小系统相互协作，组合成一个大系统，各个小的系统是独立部署的，它们之间是松耦合的。

1.2、为什么要用微服务？

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



- 单体架构：

单体应用就是将应用程序的所有功能都打包成一个独立的单元，最终以一个 WAR 包或 JAR 包存在，没有外部的任何依赖，里面包含 Service、UI 等所有的逻辑。单体应用有以下优点：

1）便于开发：只需借助 IDE 的开发、调试功能即可完成。
2）易于测试：只需要通过单元测试或浏览器即可完成测试。
3）易于部署：打包成单一可执行 jar 包，执行 jar 包即可完成部署。

不幸的是，这种简单的单元有很大的局限性。应用程序随着业务需求的迭代，功能的追加扩展，最终成为一个庞然大物。变得更加复杂，逻辑耦合严重，难以理解，团队开发人员职责不清，部署困难，回归测试成本巨大，交付效率大大降

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

低，总结下来，单体应用有一下缺点：

1）复杂性高

代码难以理解。

在业务规模和团队规模发展的一定阶段，这些不足表现的更加明显，单体架构的不足首先表现在复杂性上，maven 模块增多，多个模块耦合在一起，代码结构混乱，使得团队成员没有一个人理解整个代码逻辑。

难以理解导致代码质量低，复杂性进一步增加。

难以理解导致代码复用度降低，因为你不知道哪些可以复用的。即便修改，影响范围也不好确定，这导致这样开发宁愿新建一个新方法和新的类，进一步导致重复代码越积越多。

代码难以被修改和重构

不理解代码当然也就写不出高内聚低耦合的代码，和代码质量持续下降。复杂性进一步增加随着复杂度的增加，耦合度越来越高，代码牵一发而动全身，代码已经很难修改和重构了。

团队职责不清晰

高度耦合的单体工程使得逻辑边界模糊不清，新的业务需求开发任务无法有效分配到人，团队人员职责不清晰，沟通成本增加。

2）交付效率低

构建和部署耗时长，难以定位问题，开发效率低。

代码量比较庞大，首先是编译耗时变长，开发调试将大部分时间花在重新编译上，代码量的增加又很难定位 bug，导致开发效率进一步降低，在代码合并过程中极易遇到代码冲突，又花上不少时间用在解决代码冲突上。这都是导致开发效率低下的因素。

代码复杂和变更影响难以理解，需要数天完成全量测试。

当我们开发完一个新的功能或者修复一个 bug，代码的变更影响是很难预估的，所以每次发布之前都要进去全量功能的回归测试。

全量部署耗时长、影响范围广、风险大，发布频次低。

正因为这种全量部署耗时长、影响范围广、风险大，导致我们将很多功能和修复聚集在一起进行开发完成，这导致了产品发布频次降低，新的功和更换的体验能不能及时呈现给用户，甚至被竞争对手赶超。

3）伸缩性(scalable)差

单体只能按整体横向扩展，无法分模块垂直扩展。

IO 密集型模块和 CPU 密集型模块无法独立升级和扩容。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

业务模块对资源的需求是不一样的，由于所有模块部署到一起，单体架构 IO 密集型模块和 CPU 密集型模块无法独立升级和扩容的，比如图片压缩，加解密这些都是 cpu 资源密集的应该升级 CPU，而 IO 密集型的模块比如日志收集服务 IO 操作比较多需要更大的内存，使用比如 SSD 性能更好的磁盘。

4）可靠性差

一个 bug 有可能引起整个应用的崩溃。

由于所有模块都是部署在一个实例中，一个 bug 会引起整个应用的崩溃，比如一个不重要的模块的内存泄露就将会导致所有应用实例一个个 crash 掉。

5）阻碍技术创新

受技术栈限制，团队成员使用同一框架和语言。

受技术栈限制，团队成员必须使用同一框架和语言，模块得不到拆分，不能使用新的语言和框架。

升级和变革技术框架变得困难

当有符合业务场景的新技术产生或者新版本时，升级和变革技术框架所带来的重构成本和风险变革很高。

尝试新语言变得困难

想尝试新的语言也变得很困难，因为开发成本的上升，重构和新需求迭代无法协调，所以最终只能是妥协继续使用原来的框架和语言。

那么如何解决单体的不足呢，通过迁移到微服务架构来解决，我们看一下什么是微服务。

- 微服务架构

我们通过上图来看下单体架构到微服务架构的对比。此图是一个简单电商单体到微服务架构的演进图，单体架构整个团队维护开发一个大工程及一个单库，到了微服务架构，用户请求经过 API Gateway 被路由到下游服务，服务之间以轻量级通信协议进行通信，服务通过注册中心发现彼此，每个服务都有专门的开发维护团队，每个服务对应独立的数据库，服务独立开发，独立部署和上线。接下来我们总结下微服务的优点。

微服务的优点：

1）灵活部署、独立扩展

传统的单体架构是以整个系统为单位进行部署，而微服务则是以每一个独立组件（例如订单服务，商品服务）为单位进行部署。

2）资源的有效隔离

每一个微服务拥有独立的数据源，假如微服务 A 想要读写微服务 B 的数据库，只能调用微服务 B 对外暴露的接口来完成。这样有效避免了服务之间争用数据库

和缓存资源所带来的问题。另外微服务各模块部署在 k8s 中，可以进行 CPU、内存等资源的限制和隔离。

3）高度可扩展性

随着某些服务模块的不断扩展，可以跨多个服务器和基础架构进行部署，充分满足业务需求。

4）易于部署

相对于传统的单体式应用，基于微服务的应用更加模块化且小巧，且易于部署。

5）服务组件化

在微服务架构中，需要我们对服务进行组件化分解，服务是一种进程外的组件，它通过 HTTP 等通信协议进行协作，而不是像传统组件那样镶入式的方式协同工作，每一个服务都独立开发、部署、可以有效避免一个服务的修改引起整个系统的重新部署。

6）去中心化治理

在整个微服务架构，通过采用轻量级的契约定义接口，使得我们对服务本身的具体技术平台不再那么敏感，这样整个微服务架构系统中的各个组件就能针对不同的业务特点选择不同的技术平台。

7）容错设计

在微服务架构中，快速检测出故障源并尽可能地自动恢复服务是必须被设计考虑的，通常我们都希望在每个服务中实现监控和日志记录。比如对服务状态、断路器状态、吞吐量、网络延迟等关键数据进行可视化展示。

8）技术栈不受限

在微服务架构中，可以结合项目业务及团队的特点，合理地选择技术栈。

9）局部修改容易部署

单体应用只要有修改，就得重新部署整个应用，微服务解决了这样的问题。

10）易于开发和维护

一个微服务只会关注一个特定的业务功能，所以它业务清晰，代码量较少。

1.3、什么样的项目适合使用微服务？

在复杂度比较低的项目中，单体架构就可以满足需求，而且部署效率也会比较高，在复杂度比较高的项目中，单体架构就不能满足了，需要进行微服务化。

微服务可以按照业务功能本身的独立性来划分，如果系统提供的业务是非常底层的，如：操作系统内核、存储系统、网络系统、数据库系统等，这类系统都

偏底层，功能和功能之间有着紧密的配合关系，如果强制拆分为较小的服务单元，会让集成工作量急剧上升，并且这种人为的切割无法带来业务上的真正的隔离，所以无法做到独立部署和运行，也就不适合做成微服务了。

那到底什么样的项目适合微服务呢？

1、业务并发量大，项目复杂，访问流量高，为了将来更好的扩展，随时对代码更新维护，可以使用微服务。

2、代码依赖程度高，想要解耦合，交给多个开发团队维护。

3、业务初期，服务器数量少，可以使用微服务，能有效节省资源。

4、从思想上： 对未来有清晰的认识，对技术更新要保持着一种自信，超前思维，知道这个东西在将来肯定会发展起来。

这就告诉了我们一个道理，在学习技术的时候，适合自己的才是最好的，比方说很多人说我们公司单体架构用的也挺好的啊，为什么还要用微服务，其实他们再用单体可能适合他们业务需求，但是我们公司可能业务规模大，项目复杂，我就想要用微服务，或者我们在未来上有更大的远见，那我也会选择用微服务，不要说看别人用，我也用，而是我用是符合我们实际需求的，一切脱离实际业务的微服务都是耍流氓。

1.4、使用微服务需要考虑的问题

1、统一的配置管理中心

服务拆分以后，服务的数量非常多，如果所有的配置都以配置文件的方式放在应用本地的话，非常难以管理，可以想象当有几百上千个进程中有一个配置出现了问题，是很难将它找出来的，因而需要有统一的配置中心，来管理所有的配置，进行统一的配置下发。

在微服务中，配置往往分为几类，一类是几乎不变的配置，这种配置可以直接打在容器镜像里面，第二类是启动时就会确定的配置，这种配置往往通过环境变量，在容器启动的时候传进去，第三类就是统一的配置，需要通过配置中心进行下发，例如在大促的情况下，有些功能需要降级，哪些功能可以降级，哪些功能不能降级，都可以在配置文件中统一配置。

2、全链路监控

（1）系统和应用的监控

监控系统和服务的健康状态和性能瓶颈，当系统出现异常的时候，监控系统可以配合告警系统，及时地发现，通知，干预，从而保障系统的顺利运行。

（2）调用关系的监控

对代码调用关系进行监控

3、日志收集

业务层面、代码层面、系统层面

1.5、常见的微服务框架

- 第一代微服务框架
  SpringCloud
  SpringCloud 为开发者提供了快速构建分布式系统的通用模型的工具（包括配置管理、服务发现注册、熔断器、断路器、智能路由、微代理、控制总线、一次性令牌、全局锁、领导选举、分布式会话、集群状态、负载均衡、数据监控等）

- 第二代微服务框架
  dubbo
  dubbo 是一个阿里巴巴开源出来的一个分布式服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。

- 第三代微服务框架
  ServiceMesh（服务网格）
  istio 是开源的 ServiceMesh（服务网格），ServiceMesh 翻译成中文就是服务网格。

1.6、对不同的微服务框架进行对比分析

1.6.1、框架背景对比

1、SpringCloud
来源于 SpringSource ，具有 Spring 社区的强大背景支持，还有 Netflix 强大的后盾与技术输出。Netflix 作为一家成功实践微服务架构的互联网公司，在几年前就把几乎整个微服务框架开源贡献给了社区，这些框架开源的整套微服务架构套件是 SpringCloud 的核心。

Eureka：服务注册发现框架
Zuul：服务网关
Karyon：服务端框架
Ribbon：客户端框架
Hystrix：服务容错组件
Archaius：服务配置组件
Servo：Metrics 组件
Blitz4j：日志组件
Pinpoint：全链路监控组件

2、Dubbo
是一个分布式服务框架，是国内互联网公司开源做的比较不错的阿里开放的微服务化治理框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。其核心部分包含（官网）：

远程通讯：提供对多种基于长连接的 NIO 框架抽象封装，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。

集群容错：提供基于接口方法的透明远程过程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持。

自动发现：基于注册中心目录服务，使服务消费方能动态的查找服务提供方，使地址透明，使服务提供方可以平滑增加或减少机器。

Dubbo 也是采用全 Spring 配置方式，透明化接入应用，对应用没有任何 API 侵入，只需用 Spring 加载 Dubbo 的配置即可，Dubbo 基于 Spring 的 Schema 扩展进行加载。当然也支持官方不推荐的 API 调用方式。

3、Istio

作为用于微服务服务聚合层管理的新锐项目，是 Google、IBM、Lyft（海外共享出行公司、Uber 劲敌）首个共同联合开源的项目，提供了统一的连接，安全，管理和监控微服务的方案。

目前是针对 Kubernetes 环境的，社区宣称在未来几个月内会为虚拟机和 Cloud Foundry 等其他环境增加支持。Istio 将流量管理添加到微服务中，并为增值功能（如安全性，监控，路由，连接管理和策略）创造了基础。

HTTP、gRPC 和 TCP 网络流量的自动负载均衡。

提供了丰富的路由规则，实现细粒度的网络流量行为控制。

流量加密、服务间认证，以及强身份声明。

全范围（Fleet-wide）的策略执行。

深度遥测和报告。

1.6.2、开源社区活跃度对比

开源社区情况：现如今企业在采用云计算首选开源，而选择一个开源框架，社区的活跃度将作为重要参考选项。

查看下在 Github 上的更新时间

SpringCloud：GitHub 所有项目均更新于 1 小时内。

Dubbo：GitHub 核心项目最近更新于一个月乃至数月前。

istio：GitHub 所有项目均更新于 30 分钟内。

可见，项目在社区活跃度上，Istio > SpringCloud > Dubbo，结合稳定性来看，对于使用 Java 开发业务较多的企业，SpringCloud 是相对更优的选择，对于更多企业来说，与语言几乎无绑定的 Istio 也是可以好好期待一下其在社区的发展。

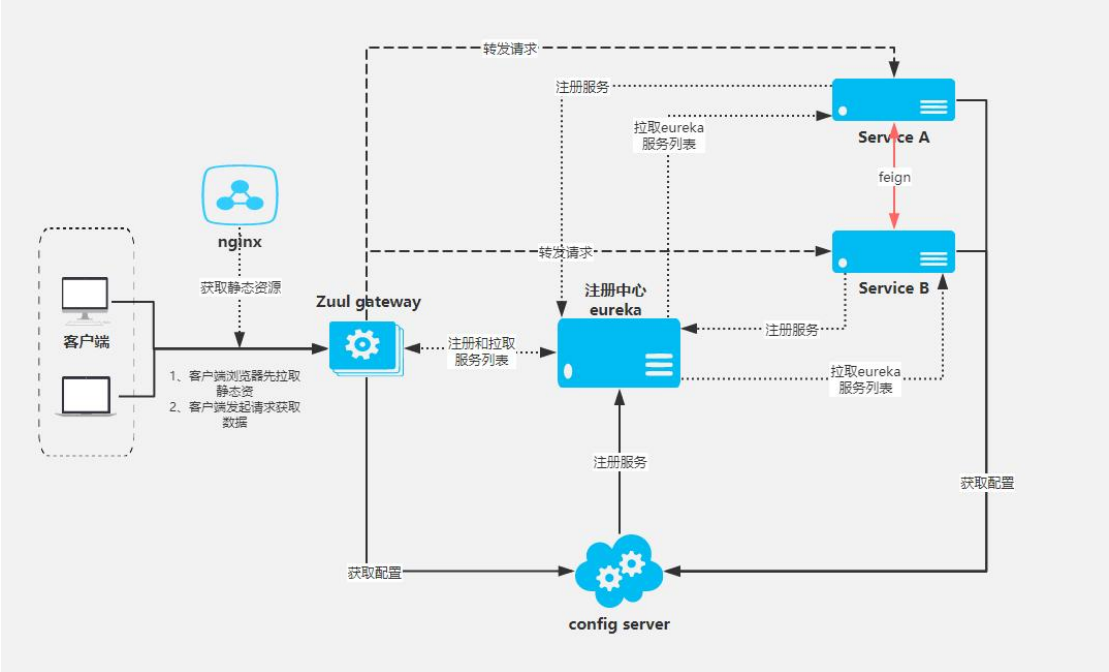
总结：结合项目背景、提供功能、社区更新活跃度，SpringCloud 是目前阶段发展最早的微服务框架方案，Istio 作为 Kubernetes 的优先支持来讲，也是一个值得关注的方案，而且发展潜力巨大，相信不久的将来 90%+的 k8s 用户都会使用 istio。目前对比来看，dubbo 则显得稍逊下来。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

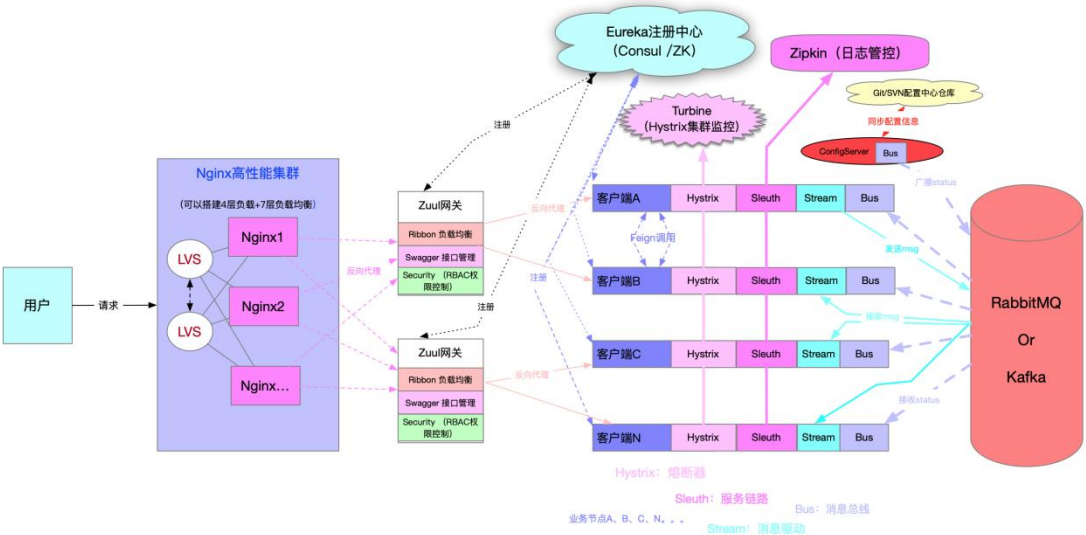
2、SpringCloud 概述

SpringCloud 架构图

图一：



图二：



2.1、SpringCloud 是什么？

官网： https://spring.io/projects/spring-cloud/

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

官方解释：

SpringCloud 是一系列框架的有序集合。它利用 SpringBoot 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 SpringBoot 的开发风格做到一键启动和部署。SpringCloud 并没有重复制造轮子，它只是将各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 SpringBoot 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。

2.2、SpringCloud 和 SpringBoot 什么关系？

SpringBoot 专注于快速方便的开发单个个体微服务。

SpringCloud 是关注全局的微服务协调整理治理框架，它将 SpringBoot 开发的一个个单体微服务整合并管理起来，SpringBoot 可以离开 SpringCloud 独立开发项目，但是 SpringCloud 离不开 SpringBoot，属于依赖关系。

2.3、SpringCloud 优缺点

优点：

1）SpringCloud 来源于 Spring，质量、稳定性、持续性都可以得到保证。

SpirngCloud 以 SpringBoot 为基础开发框架，可以给开发者大量的微服务开发经验，例如，只要极少量的标签，你就可以创建一个配置服务器，再加一些标签，你就可以得到一个客户端库来配置你的服务，更加便于业务落地。

2）SpringCloud 是 Java 领域最适合做微服务的框架，对 Java 开发者来说就很容易开发。

3）耦合度低，不影响其他模块

4）多个开发团队可以并行开发项目，提高开发效率

5）直接写自己的代码即可，然后暴露接口，通过组件进行服务通信。

缺点：

1）只能针对 Java 开发

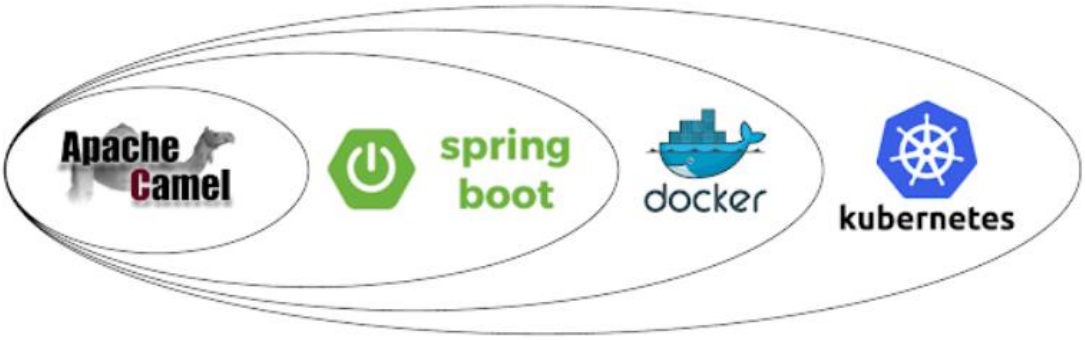
2）部署麻烦、组件多

3）每个微服务都可以用一个数据库，导致数据管理复杂

4）一套完整的微服务包括自动化部署、调度、资源管理、进程隔离、自愈、构建流水线等功能，单靠 SpringCloud 是无法实现的，所以 SpringCloud+k8s 才是最好的方案。

2.4、为何要将 SpringCloud 项目部署到 k8s 平台？

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



SpringCloud 只能用在 SpringBoot 的 java 环境中，而 kubernetes 可以适用于任何开发语言，只要能被放进 docker 的应用，都可以在 kubernetes 上运行，而且更轻量，更简单。

每个微服务可以部署多个，没有多少依赖，并且有负载均衡能力，比如一个服务部署一个副本或 5 个副本，通过 k8s 可以更好的去扩展我们的应用。

Spring 提供应用的打包，Docker 和 Kubernetes 提供部署和调度。Spring 通过 Hystrix 线程池提供应用内的隔离，而 Kubernetes 通过资源，进程和命名空间来提供隔离。Spring 为每个微服务提供健康终端，而 Kubernetes 执行健康检查，且把流量导到健康服务。Spring 外部化配置并更新它们，而 Kubernetes 分发配置到每个微服务。

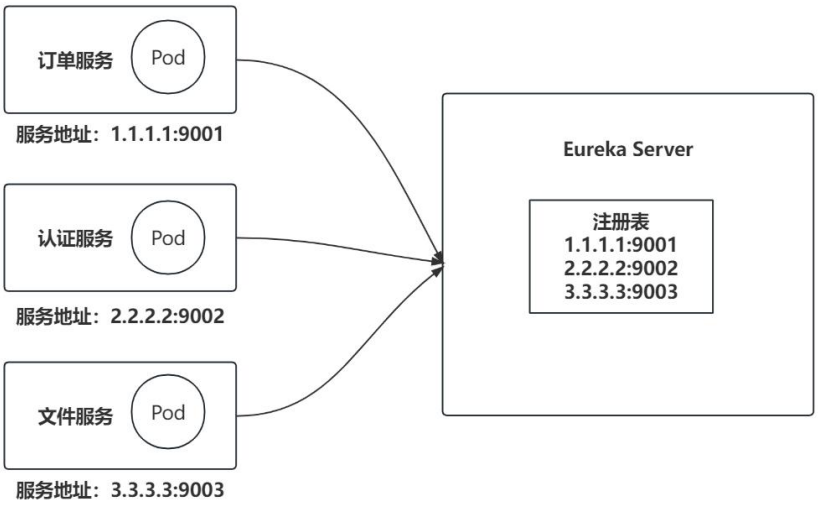
SpringCloud 很多功能都跟 kubernetes 重合，比如服务发现，负载均衡，配置管理，所以如果把 SpringCloud 部署到 k8s，那么很多功能可以直接使用 k8s 原生的，减少复杂度。

SpringCloud 容易上手，是对开发者比较友好的平台。Kubernetes 是可以实现 DevOps 流程的，SpringCloud 和 kubernetes 各有优点，只有结合起来，才能发挥更大的作用，达到最佳的效果。

3、SpringCloud 组件介绍

3.1、服务发现与服务注册组件 Eureka

在分布式微服务架构中，服务众多，如何确定哪台服务器对应哪个服务，Eureka 就是这样一个组件，用于服务注册与发现。Eurake 分为两大组成部分，Eureka 客户端负责将信息注册到 Eureka 服务端内，Eureka 服务端内有注册表，相当于注册中心，连接各个服务，如下图所示：

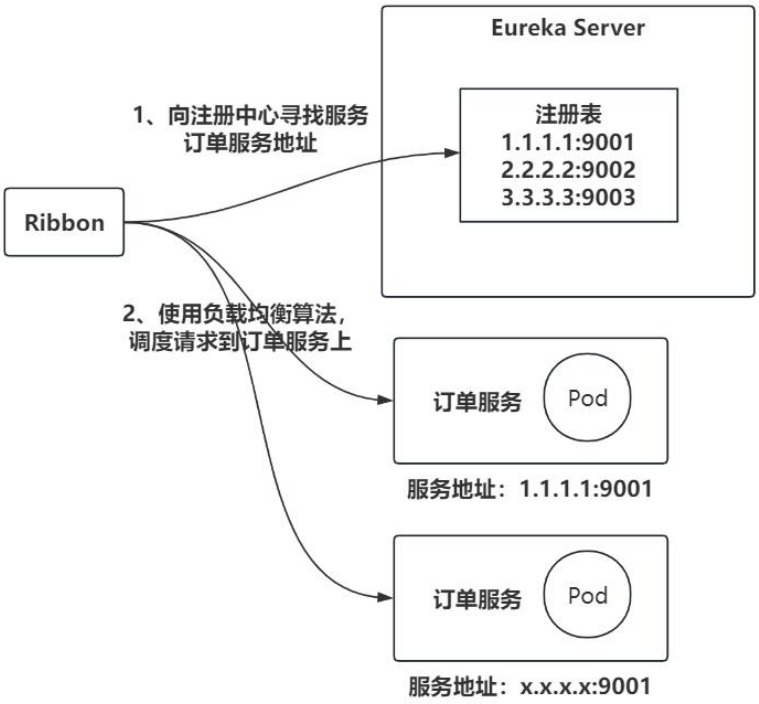


Eureka 社区已经停止维护了，目前微服务项目使用最多的服务发现与注册组件是 Nacos 组件。

Nacos 可以用于服务发现与服务注册。相比 Eureka，Nacos 还可以用于做配置中心。

3.2、负载均衡组件 Ribbon

在分布式微服务架构中，为了保证高可用性，通常采用集群架构。一个服务通常布置在多个服务器节点上，通过 Ribbon 可以起到负载均衡的效果，采用轮询算法，选择最终的服务器节点。下图，表示它的使用：



3.3、熔断器 Hystrix

Hystrix 是一个容错管理组件，为服务中出现的延迟和故障提供强大的容错能力。

- 服务降级

在微服务架构中，时常会出现高并发的情况，假如我们的订单服务只能抗住 10 万用户同时访问，那么当有 11 万用户同时访问时，肯定会出现有用户下单时服务报错。

那么这个时候，服务降级的作用就体现了。服务出现故障时，给故障服务降级到事先准备好的故障处理结果，将此结果返回给服务消费者，如：“当前请求人数多，请稍后重试”。

- 服务熔断

假设有 A、B、C 三个服务，服务 A 调用服务 B 和 C，链路关系如下：

假设服务 C 因为请求量大，扛不住请求，变得不可用，这样就是积累大量的请求，服务 B 的请求也会阻塞，会逐渐耗尽线程资源，使得服务 B 变得不可用，那么服务 A 在调用服务 B 就会出现问题，导致服务 A 也不可用，那么整条链路的服务调用都失败了，我们称之为雪崩。

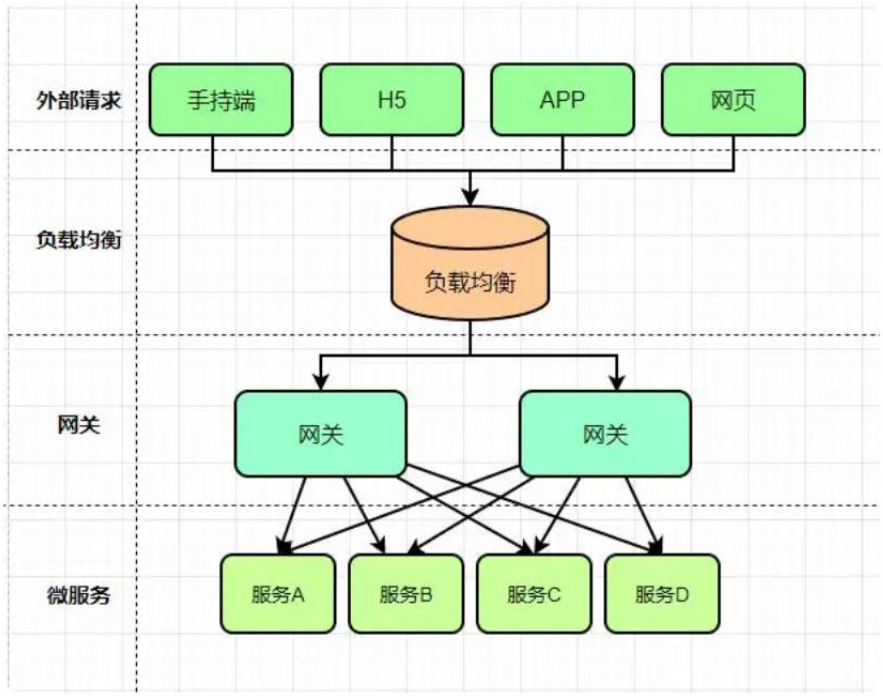
熔断机制是应对服务雪崩的一种链路保护机制，当服务出现故障时，服务会进行降级，熔断该服务节点，迅速返回错误响应信息。当检测到服务访问正常时，恢复其链路节点。

3.4、服务网关 Gateway

Gateway 是一个基于 Spring 5.0，Spring Boot 2.0 和 Project Reactor 等技术开发的网关框架，它使用 Filter 链的方式提供了网关的基本功能，例如安全、监控/指标和限流等。

网关在微服务中的位置:

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。