



《软件项目管理》个人作业

构建高效分布式系统的现代选择

——微服务架构学习心得



专 业： _____ 软件工程 _____

学生姓名： _____ 张可心 _____

学 号： _____ 2022141461181 _____

指导教师： _____ 毋攀良 _____

评阅意见： _____

提交时间：2024 年 12 月 28 日

目录

1. 微服务架构简介	2
2. 适用场景	2
3. 核心组件	2
4. 优缺点	3
4.1 架构优点	3
4.2 架构缺点	4
5. 技术栈	5
6. 使用微服务架构的系统和第三方软件、库	5
7. 思考与收获	6

1. 微服务架构简介

随着信息技术的飞速发展，软件系统的规模和复杂度不断增加，传统的单体架构逐渐难以满足业务需求。微服务架构作为一种新兴的软件架构模式，以其模块化强、灵活扩展、技术多样性、容错性高以及便于持续交付等诸多优势，受到了广泛关注。

微服务架构是一种将应用程序分解为多个小型独立服务的架构风格。每个服务都是围绕单一业务功能构建的，可以独立部署、运行并通过轻量级通信机制（通常是 HTTP/REST 或消息队列）与其他服务进行交互。微服务架构强调服务的高度自治性和模块化，这种方法可以有效解决单体架构在扩展性、开发效率和部署灵活性方面的瓶颈。

2. 适用场景

1. 大型复杂系统

需要管理多个模块，并且每个模块功能独立。

2. 高频更新和迭代

需要快速上线新功能或修复漏洞的系统。

3. 高可用和高扩展性要求

系统需要支持弹性伸缩，确保在高峰期也能正常运行。

4. 分布式团队开发

团队可以并行开发不同的服务模块。

5. 多技术栈需求

希望为每个服务选择最适合的编程语言和技术。

典型应用包括电子商务平台、社交媒体应用、大型企业级系统等。

3. 核心组件

1. 服务发现

在复杂的微服务集群中，服务实例动态增减，服务发现组件至关重要。像 Netflix 的 Eureka，服务启动时向其注册自身信息，客户端调用时先从 Eureka 查询目标服务实例地址，确保请求准确路由，保障服务间可靠通信。

2. API 网关

作为系统对外统一入口，负责请求路由、鉴权、限流等。例如 Kong API 网关，外部请求先抵达网关，经身份验证后按预设规则转发至相应微服务，还能对恶意流量进行限制，保护后端服务安全稳定运行。

3. 配置中心

集中管理各微服务配置信息，如 Spring Cloud Config。当配置需要变更时，在配置中心修改后可实时推送给相关服务，避免逐个服务手动更新配置，确保配置一致性，尤其在多环境部署场景下优势显著。

4. 分布式跟踪

面对分布式系统中复杂的服务调用链路，如 Zipkin 可实现分布式跟踪。它为每个请求生成唯一 ID，记录服务间调用顺序、耗时，便于排查性能瓶颈、故障根源，助力系统优化。

5. 消息队列

用于异步通信和解耦服务，如 RabbitMQ。在订单处理微服务与库存微服务间，订单创建消息放入队列，库存微服务异步消费处理，缓解即时响应压力，提升系统吞吐量与稳定性。

4. 优缺点

4.1 架构优点

1. 模块化强

每个微服务独立运行，拥有自己的代码库、数据存储和运行时环境，这使得服务的维护和更新变得极为便捷。开发团队可以专注于单个服务的优化，而不会对整个系统造成大面积影响，例如电商系统中订单管理微服务的升级，不会干扰商品展示或用户认证等其他服务。

2. 灵活扩展

依据业务负载变化，能够单独对特定服务进行扩展。如社交平台在热门话题爆发时，仅需对帖子浏览、评论相关微服务进行资源扩充，提升服务器实例数量或优化算法，而不必对整个平台全面扩容，极大提升了资源利用率。

3. 技术多样性

不同微服务可基于最合适的编程语言或技术框架构建。对于计算密集型的图像处理微服务，采用 C++ 等高效语言；而注重快速开发迭代的用户界面微服务，选用 JavaScript 框架，充分发挥各类技术优势。

4. 容错性高

单个微服务出现故障，如推荐系统微服务因算法异常崩溃，由于服务间松散耦合，只会影响该功能模块，不会致使整个软件系统瘫痪，其他如登录、交易流程等服务仍正常运转。

5. 便于持续交付

支持小步快跑的敏捷开发模式，开发团队能频繁修改、测试并快速部署单个微服务。如每日多次将新特性推送到线上环境，及时响应业务需求变化，提升开发效率。

4.2 架构缺点

1. 分布式事务管理

跨多个微服务的业务操作，如电商下单涉及库存扣减、订单生成、支付通知，确保事务一致性复杂。传统数据库事务难以适用，需采用 Saga 模式、TCC 等分布式事务解决方案，但增加开发与运维难度。

2. 服务间通信开销

大量微服务间频繁网络调用，相较于单体应用内部函数调用，通信延迟、带宽消耗增加，影响系统性能。优化服务布局、采用高效序列化协议、引入缓存机制可缓解但无法根除。

3. 运维复杂性

众多微服务实例的部署、监控、故障排查任务繁重，要求运维团

队具备多领域知识技能。容器编排工具虽减轻部分负担，但面对复杂故障场景，定位问题根源仍耗时费力。

4. 数据一致性

各微服务有独立数据存储，数据实时同步更新困难。如社交平台用户信息修改，涉及多个关联微服务数据更新，易出现数据不一致，需设计合理的数据同步策略与补偿机制。

5. 技术栈

开发语言和框架：Spring Boot (Java)、Flask (Python)、Node.js (JavaScript/TypeScript)、Go 等。

API 网关：Kong、Zuul、Traefik 等，用于统一管理服务的入口。

服务通信：REST、gRPC、GraphQL。

服务注册与发现：Eureka、Consul、etcd。

消息队列：Kafka、RabbitMQ、ActiveMQ。

容器化和编排：Docker、Kubernetes。

监控与日志：Prometheus、Grafana、ELK Stack。

CI/CD 工具：Jenkins、GitLab CI/CD、ArgoCD。

6. 使用微服务架构的系统和第三方软件、库

1. Netflix：微服务架构的先驱，提供了一套成熟的微服务工具，如 Eureka（服务发现）、Hystrix（断路器）。Netflix 的实践证明了微服务架构在高并发场景下的可靠性和扩展性。

2. Amazon：其电商平台采用微服务架构，将订单处理、支付、推荐系统等功能拆分为多个独立服务。通过微服务架构，Amazon 可以实现快速迭代并高效处理全球范围的订单。

3. Uber：采用微服务来管理其复杂的司机-乘客匹配、地图服务等。微服务帮助 Uber 实现了全球化扩展，同时降低了系统单点故障

的风险。

4. Spring Cloud: 为构建微服务提供了完整的解决方案。Spring Cloud 降低了微服务的入门难度，是企业级项目的常用框架。

7. 思考与收获

微服务架构的出现无疑为软件行业带来了巨大变革，但在拥抱这一架构的过程中，团队文化与组织架构的适应性调整至关重要。微服务强调独立自主的团队负责单个或多个微服务开发运维，这需要打破传统的集中式开发模式，建立更加扁平化、敏捷的团队协作机制，赋予团队更多决策权。此外，从技术选型到架构落地，是一个长期迭代优化的过程，不能一蹴而就。企业需结合自身业务规模、技术实力、发展阶段谨慎决策，既要充分发挥微服务架构的优势，又要清醒认识并妥善应对各种挑战，如此才能在数字化转型浪潮中借助微服务架构驶向成功彼岸。

微服务架构不是万能的，但在适合的场景下，它可以显著提高开发效率和系统稳定性，为企业带来巨大价值。同时，微服务的实施也要求团队不断优化技术能力，以应对分布式系统带来的复杂挑战。