

**《软件项目管理》课程个人作业**



**微服务架构的探讨与应用分析**

|  |  |
| --- | --- |
| 学 院： | 软件学院 |
| 专 业： | 软件工程 |
| 学生姓名： | 黄愫 |
| 学 号： | 2022141461212 |
| 指导老师： | 毌攀良 |
| 评阅意见： |  |
|  |  |
|  |  |
|  |  |

二零二四 年 十二 月 十九 日

目录

[一、 绪论 3](#_Toc32133)

[1.1 研究背景 3](#_Toc4037)

[二、 微服务架构 3](#_Toc19194)

[2.1 微服务架构概述 3](#_Toc23726)

[2.2 微服务架构应用场景 4](#_Toc21002)

[2.3 微服务架构的优点 5](#_Toc21821)

[2.4 微服务架构面临挑战 6](#_Toc14787)

[三、 微服务架构技术栈 7](#_Toc14849)

[3.1服务框架 7](#_Toc1865)

[3.2服务网关 8](#_Toc28442)

[3.3服务治理 8](#_Toc23453)

[3.3配置中心 8](#_Toc12636)

[3.4持续集成 9](#_Toc2233)

[四、 应用示例 9](#_Toc14729)

[4.1 微服务架构在分布式系统的应用 9](#_Toc83)

[4.1.1微服务架构在分布式系统中的应用场景及问题 9](#_Toc26226)

[4.1.2微服务架构在分布式系统中事务实践 10](#_Toc29199)

[4.2 微服务架构在twitter中应用 11](#_Toc3321)

[参考文献 14](#_Toc23527)

1. 绪论

## 研究背景

随着互联网用户的快速增长和技术的不断发展，互联网服务面临着巨大的挑战。虽然Java Web及其框架（如SSH和SSM）因其易用性受到中小型企业的青睐，但随着用户需求的增加，单体架构逐渐暴露出可维护性、扩展性和灵活性等方面的不足，尤其在快速变化的市场环境中，开发和部署的效率变得更加关键。

为了解决这些问题，各大互联网公司开始逐步转向微服务架构。这种架构风格通过将大型复杂的应用拆分为多个独立的微服务，使每个微服务专注于特定的任务，从而提高了系统的可扩展性和敏捷性。微服务的松耦合特性使得开发、维护和部署变得更加高效，特别是在云计算技术的支持下，微服务架构为复杂企业应用的快速实施和敏捷部署提供了极大的帮助。因此，微服务架构成为适应现代互联网发展的重要解决方案。

1. 微服务架构

## **2.1 微服务架构概述**

James Lewis and Martin Fowler对微服务是如下定义的：“The microservice architectural style is an approach to developing a single application as a suite of small services,each running in its own process and communicating with lightweight mechanisms,often an HTTP resource API.hese services are built around business capabilities and independently deployable by fully automated deployment machinery.There is a bare minimum of centralized management of these services,which may be written in different programming languages and use different data storage technologies.”[2]

微服务架构是一种软件架构风格，它将大型应用程序拆分为多个小型、独立的服务，每个服务围绕特定的业务功能进行构建。这些服务可以独立部署、扩展和升级，因此它们具有更高的灵活性和可维护性。微服务架构通常通过轻量级通讯协议（例如 HTTP/REST 或 gRPC）进行交互，并且每个微服务通常有自己独立的数据库，保持服务间的松耦合。总而言之，微服务架构思想即以专注于单一职责的小型功能模块为基础，通过API集成相互通信的方式，实现复杂的业务系统搭建[3]

概念： 把一个大型的单个应用程序和服务拆分为数个甚至数十个的支持微服务，它可扩展单个组件而不是整个的应用程序堆栈，从而满足服务等级协议。

定义： 围绕业务领域组件来创建应用，这些应用可独立地进行开发、管理和迭代。在分散的组件中使用云架构和平台式部署、管理和服务功能，使产品交付变得更加简单。

本质： 用一些功能比较明确、业务比较精练的服务去解决更大、更实际的问题。

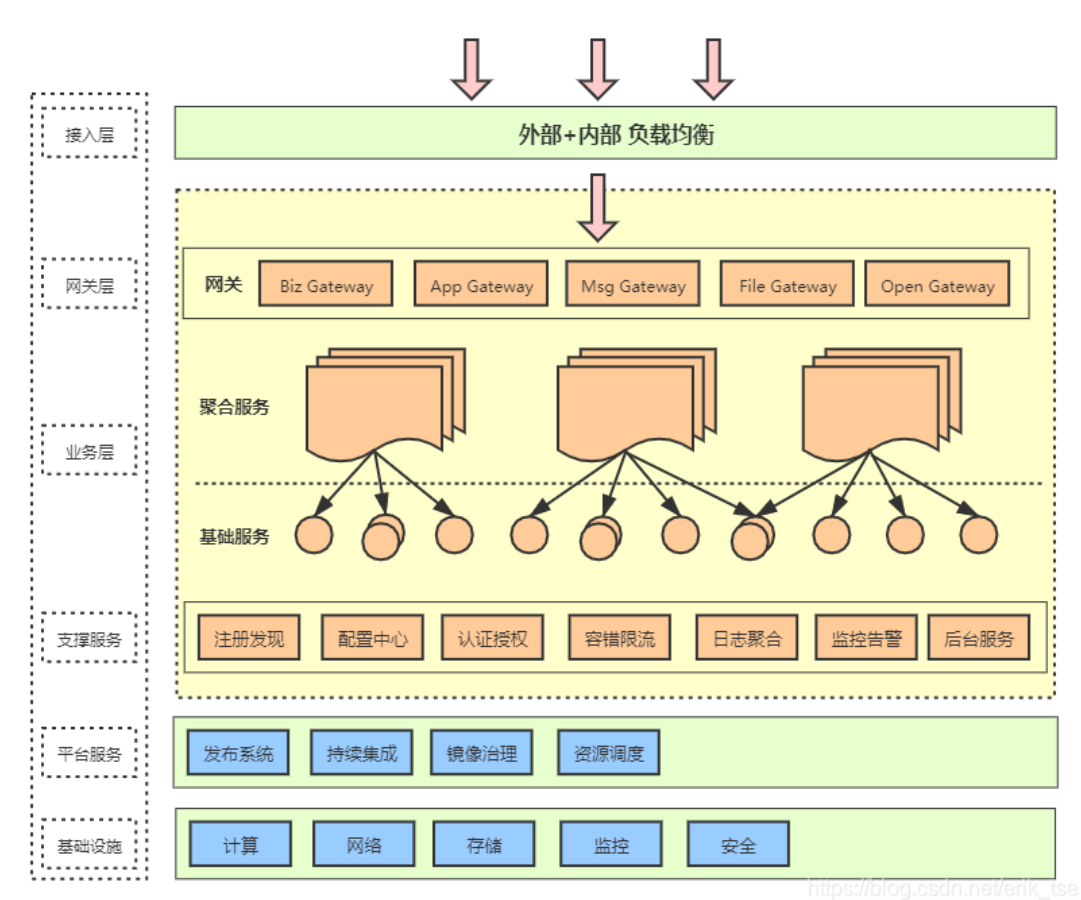


图2.1 微服务架构图

## **2.2 微服务架构应用场景**

以下是一些典型的微服务架构应用场景：

(1)大型复杂系统：当应用程序规模庞大且功能复杂时，微服务架构可以将系统拆分为多个小型服务，使得每个服务可以独立开发和部署。这种方式不仅能够提高系统的灵活性和可维护性，还能降低团队协作的复杂性。

(2)频繁迭代与快速交付：在快速变化的市场环境中，企业需要快速推出新功能和更新。微服务架构支持持续集成和持续交付（CI/CD），每个微服务可以独立开发、测试和部署，从而加速软件发布周期，满足业务需求的快速变化。

(3)多平台应用：在涉及多个平台（如网页、移动设备、IoT设备等）的应用中，微服务架构允许不同的团队专注于不同的平台实现。每个平台可以利用微服务提供的 API 来访问后端服务，促进跨平台的开发和维护。

(4)多技术栈兼容：微服务架构允许不同的微服务使用不同的编程语言和技术栈，这使得团队可以根据具体需求选择最佳的技术方案。例如，一个微服务可以使用Python进行数据处理，而另一个服务可以使用Java来实现业务逻辑。

(5)弹性与容错：微服务架构自然支持弹性设计，可以通过服务的独立性来优化系统的容错能力。在发生故障时，只有受到影响的微服务会受到影响，其他服务仍然可以正常运行，从而提高系统的可用性。

(6)云原生应用：微服务架构与云计算非常契合，适用于云原生应用开发。使用容器化和动态调度技术（如Docker和Kubernetes），微服务可以在云环境中快速部署、扩展和管理，以应对业务的高峰。

(7)电子商务平台：在大规模的电子商务平台中，微服务架构对不同功能模块的支持（如用户管理、产品管理、订单处理等）非常有利。不同的服务可以独立扩展，满足高并发的访问需求。

(8)数据驱动应用：在需要实时数据处理和分析的应用中，微服务可以分布式处理数据，优化性能。这在大数据和机器学习应用中尤为重要，可以通过构建特定的数据服务来简化数据流和处理逻辑。

## **2.3 微服务架构的优点**

与微服务相对的另一个概念是传统的单体式应用程序，单体式应用内部包含了所有需要的服务。而且各个服务功能模块有很强的耦合性，也就是相互依赖彼此，很难拆分和扩容。而随着互联网技术的快速发展和应用需求的不断增长，传统的单体应用架构在应对复杂性和快速变化的业务环境时面临了诸多挑战。为了提高系统的灵活性、可维护性和容错能力，越来越多的企业开始转向微服务架构。 微服务架构通过将大型应用分解为多个功能单一且相互独立的微服务，能够有效降低系统的复杂性，实现各个服务的独立开发、测试和部署。这种架构不仅促进了团队的自治和技术选型的灵活性，还增强了系统的容错性和扩展能力。

微服务架构的几个核心优势包括复杂应用的解耦、独立性、技术选型的灵活性、容错机制以及松耦合特性如何为现代软件开发提供支持。

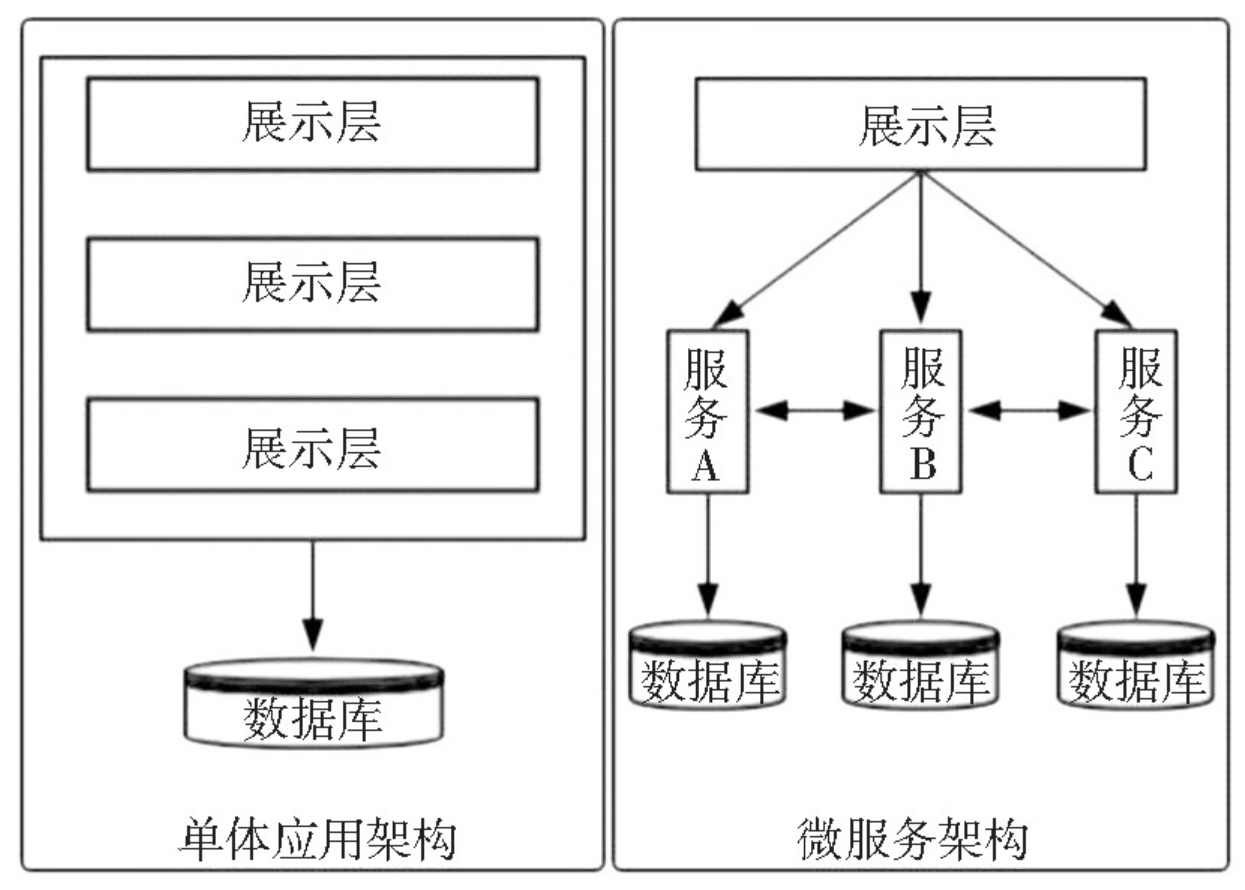


图3.1单体架构与微服务架构

## **2.4 微服务架构面临挑战**

微服务架构在规模较大的应用中具有明显优势，但其优势也是有代价的，微服务架构也会给人们带来新的问题和挑战。其中一个主要缺点是微服务架构分布式特点带来的复杂性，开发过程中，需要基于RPC或消息实现微服务之间的调用与通信，使服务发现与服务调用链跟踪变得困难。另一个挑战是微服务架构的分区数据库体系，不同服务拥有不同数据库。受限于CAP原理[21]约束以及NOSQL数据库的高扩展性，使人们不得不放弃传统数据库的强一致性，转而追求最终一致性，因此对开发人员提出了更高要求。微服务架构给系统测试也带来了很大挑战，微服务架构可能涉及多个服务，传统的单体Web应用只需测试单一API即可，然而对于微服务架构测试，需要启动其依赖的所有服务，该复杂性不可低估。在大规模应用部署中，在监控、管理分发及扩容等方面，微服务也存在着巨大挑战。

下面总结了微服务架构面临的主要挑战：

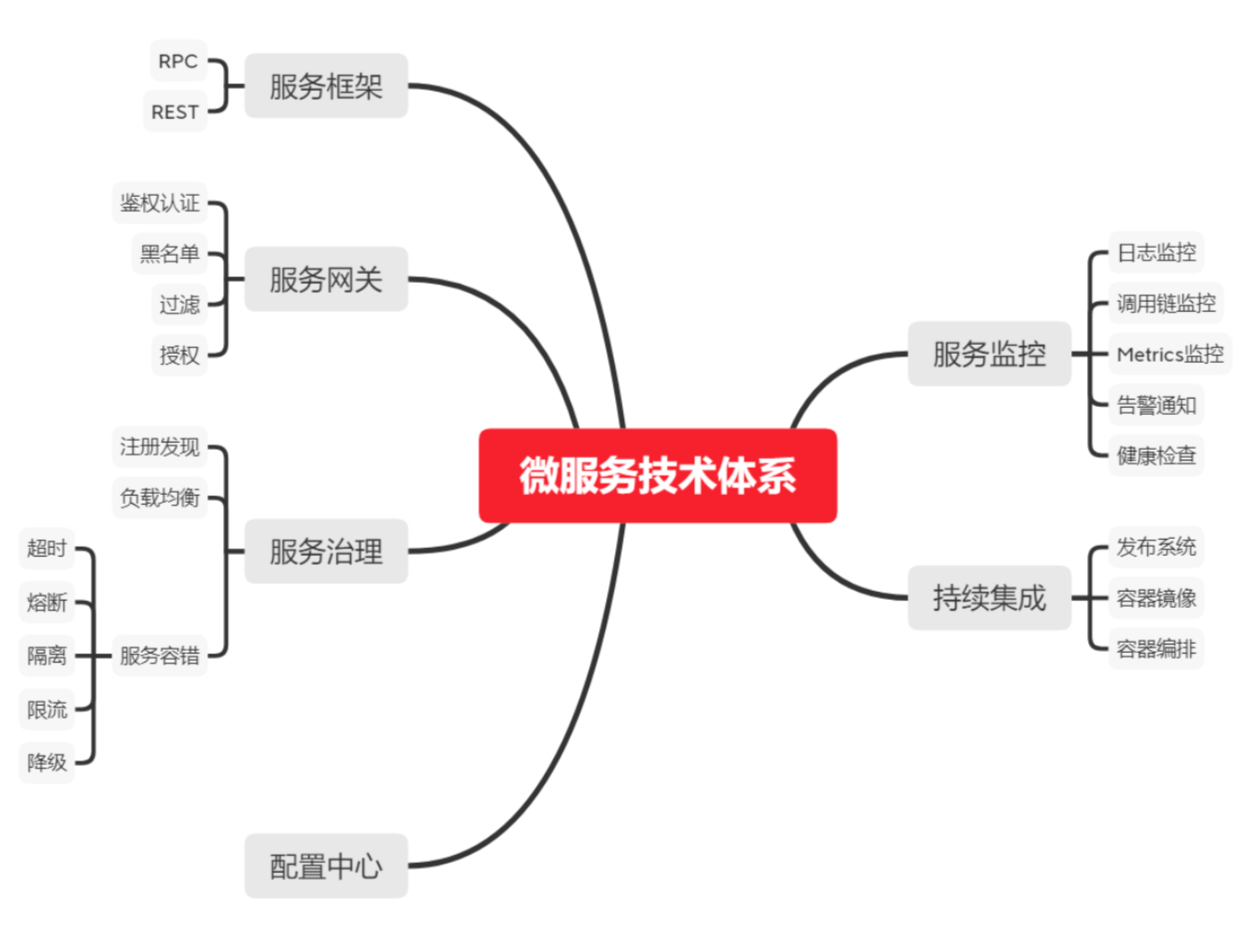
复杂性增加：微服务的分布式特性使得服务之间的通信和调用变得复杂。开发过程需要通过远程过程调用（RPC）或消息传递机制实现微服务之间的交互，这导致服务发现和调用链的跟踪变得困难。

数据一致性问题：在微服务架构中，各个微服务通常拥有独立的数据库，导致数据分布的分散性。这使得跨多个数据库保持事务一致性成为一大挑战。受CAP原理和NoSQL数据库高扩展性的限制，开发者往往需放弃传统数据库的强一致性，转而追求最终一致性，这在某种程度上造成了妥协。

安全风险：微服务架构的数量众多且相互之间频繁暴露API接口，这扩大了攻击面并增加了系统受到攻击的风险。此外，微服务之间的信任关系也导致了“单点脆弱”问题，一旦某个服务被攻破，攻击可能迅速传播，影响整个系统的安全。

监控与调试挑战：微服务架构中服务实例的数量庞大，任何一个服务的响应慢或出现故障都可能导致系统延迟甚至瘫痪。系统的性能监控和故障分析因复杂的依赖关系而变得困难。调试方面，由于微服务依赖于多个服务，测试需要启动所有相关服务，这无疑增加了测试的复杂性和成本。

1. 微服务架构技术栈

重点说明下相关的微服务核心关键技术栈，本文讨论的基于JAVA开发。  


## **3.1服务框架**

用JAVA技术开发微服务，比较主流的选择有：Spring Cloud 和 Dubbo。

Spring Cloud是在Spring基础上构建的，它后面有2大公司支撑，Pivotal和Netflix的技术支持。它的核心就是Netflix贡献的源码，目前可以认为是构建 Java 微服务的一个社区标准。

Dubbo 是阿里多年构建生产级分布式微服务的技术结晶，服务治理能力非常丰富，在国内技术社区具有很大影响力，Dubbo 本质上是一套基于 Java 的 RPC 框架。

此外还有：

当当的Dubbox，Dubbox在Dubbo的基础上拓展了RESTful的接口暴露能力。

新浪微博的Motan，功能和Dubbo类似，可以认为是一个轻量裁剪版的Dubbo。

谷歌的gPRC，是谷歌近年新推出的RPC框架，基于protobuf的强契约编程模型，能自动生成各种语言客户端，且保证互操作。

## **3.2服务网关**

如果采用Spring Cloud体系，则可以选择Zuul网关，Zuul在Netflix经过大规模生产验证，支持灵活的动态过滤器脚本机制。

此外还有：

Spring Cloud Gateway，Spring Cloud新推出的网关框架；

基于 Nginx/OpenResty 的 API 网关 Kong，因为采用 Nginx 内核，Kong 的异步性能较强，另外基于 lua 的插件机制比较灵活，社区插件也比较丰富。

## **3.3服务治理**

服务注册和发现：Netflix Eureka，支持跨数据中心，除此之外，还有Consul（天然支持跨数据中心，还支持 KV 模型存储和灵活健康检查能力），Zookeeper等

断路器：Hystrix，Netflix 的Hystrix把熔断、隔离、限流和降级等能力封装成组件，任何依赖调用（数据库，服务，缓存）都可以封装在Hystrix Command之内，封装后自动具备容错能力。除此之外，Netflix的Hystri停更后，SpringCloud家族推荐Resilience4j，另外阿里也推出了Sentinel。

调用端负载均衡：Ribbon

REST客户端：Feign

分布式事务：阿里开源的Seata，个人开源 TX-LCN

消息队列：Spring Cloud的Spring Cloud Bus和Spring Cloud Stream，阿里开源的RocketMQ，Linkedin开源的Kafka。

## **3.3配置中心**

Spring Cloud 自带 Spring Cloud Config

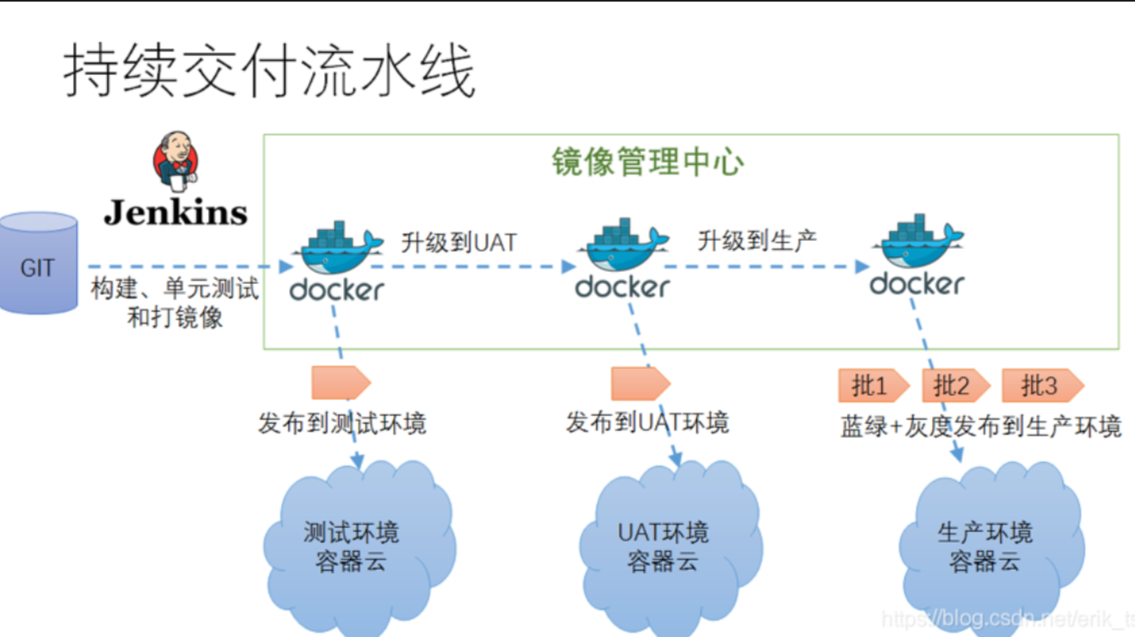
携程开源的Apollo，在携程经过生产级验证，具备高可用，配置实时生效（推拉结合），配置审计和版本化，多环境多集群支持等生产级特性，建议中大规模需要对配置集中进行治理的企业采用。

阿里开源的Nacos，Nacos集注册发现和配置中心于一体。

## **3.4持续集成**

持续交付流水线（CD Pipeline）是微服务发布重要环节，包括基于Docker的镜像治理、基于Kubernetes的集群资源调度，以及资源治理和发布平台等。

下图为可供参考的流水线模型：



1. 应用示例

## 4.1 微服务架构在分布式系统的应用

4.1.1微服务架构在分布式系统中的应用场景及问题

微服务架构在分布式系统中的权限认证应用旨在解决现有技术中的问题。该架构的广泛应用依赖于不同分布式服务的协同工作。在分布式环境中，模块间存在复杂的依赖关系，整体的高耦合度是系统平台的一大特征。不同模块在系统平台中发挥着各自的功能：

结构管理模块：提供组织机构的编辑、注销和添加功能。

数据管理模块：包括系统登录、首页界面、个人信息及功能菜单等子模块。

服务模块：涉及用户访问认证、数据权限更新、用户角色识别和令牌续期等服务。

数据访问模块：包括资源访问拦截器、页面资源访问权限识别和数据资源访问权限识别等功能。

以上四个模块在分布式系统中协同工作，独立执行各自的运算功能，并整合到整体系统中，以提升用户体验和系统效率。

4.1.2微服务架构在分布式系统中事务实践

微服务架构下，不同分布式系统间存在大量的基于http或rpc消息的网络通信，接口的重复调用以及消息的重复可能会经常发生。在具体网络事务实践中，网络波动导致服务提供方的接口被调用了两次，当接口被重复调用时，消息被系统组织重复传送。这种实践错误使系统接口的调用程序发生紊乱，从而导致系统bug呈现出来，甚至触发未关闭的重试机制。微服务架构在实践过程中表现出幂等性，即包含相同业务ID参数的请求将被拒绝。后台有通讯，前台就需要依据数据管理通讯过程，一个服务下线/更新/升级，前台就要重新部署，当前台是移动应用的时候，通常业务变化的节奏更快。另外，通讯服务的接口调用也在微服务架构的幂等性原则下发生，用户登录信息和权限管理是在系统内部一个名称为(OAuth)的地方统一维护管理。它利用幂等性原则提升远程接口服务的易用保障微服务架构集群的服务治理和故障处理能力。[5]

4.1.3微服务架构在分布式系统的应用优势

独立性和隔离性：微服务架构将应用拆分为多个独立的服务实例，每个服务负责特定的功能。这种独立性使得每个服务可以独立配置、开发和部署，降低了系统各部分之间的耦合。这样，某个服务出现故障时，其他服务仍然能够正常运行，提高了系统的稳定性和容错性。

弹性和可扩展性：微服务架构支持按需扩展。在面对高负载时，可以根据具体服务的需求独立扩展特定的微服务，而不是整个系统。这种灵活的资源分配和处理能力使得系统能够更好地应对流量波动和用户数量的变化。

技术多样性：在微服务架构下，不同的微服务可以使用不同的技术栈和编程语言，根据各自的业务需求和团队特长选择最合适的工具。这种去中心化的技术选型使得系统在不断变化的技术环境中保持竞争力。

灵活的部署方式：微服务架构允许通过容器化技术如Docker和编排工具如Kubernetes进行部署。这样的策略不仅简化了服务的配置与管理，还能实现持续集成和持续交付（CI/CD），提供快速部署和版本控制的能力，提高了开发效率。

增强的监控与维护能力：微服务架构可以利用分布式监控工具（如Prometheus、Grafana等）进行各个服务的性能和健康状态监控。这种监控能力使得开发和运维团队能够更及时地识别和解决问题，提升了系统的可靠性。

简化的开发流程：通过将应用拆分为多个微服务，团队可以并行开发不同的服务，每个团队专注于自己的服务功能。这样可以缩短开发周期，提高交付速度，适应快速变化的市场需求。

服务发现与负载均衡：在微服务架构中，采用服务注册与发现机制（如Eureka、Consul等）可以动态地管理服务实例。负载均衡器可以根据当前的流量和服务器负载情况智能地将请求路由到特定的服务实例，优化系统性能和资源利用。

易于集成与交互：微服务之间使用标准化的协议（如HTTP/REST、gRPC等）进行通信，使得不同服务之间的集成变得更加简单。这样的架构可以方便地整合不同的外部服务和API，提升了系统的灵活性。

## 4.2 微服务架构在twitter中应用

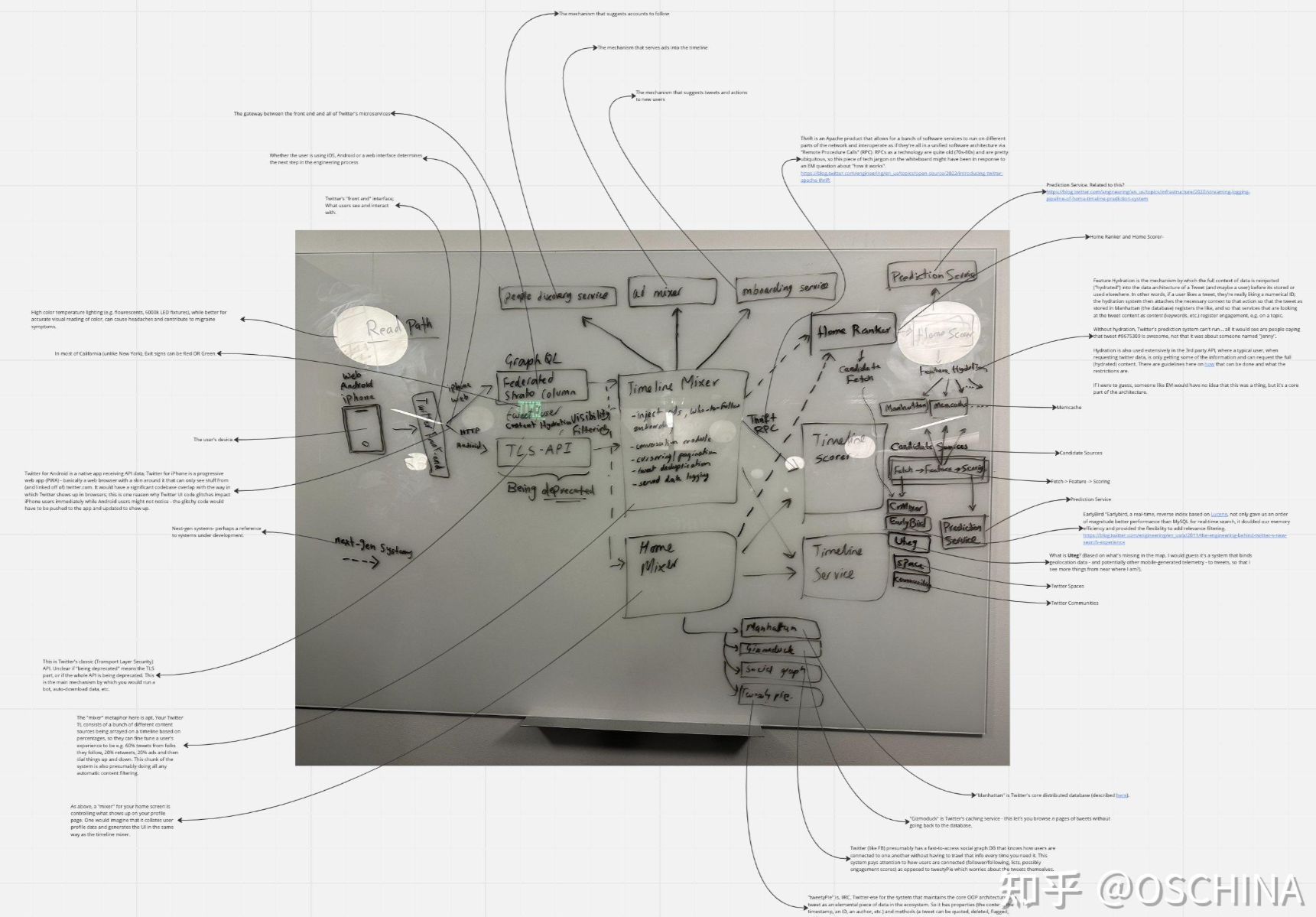


图4.1 马斯克晒出twitter架构图

Twitter的软件架构主要采用微服务架构，下面是一些关键组件:

1.用户服务(User service):管理用户账户、认证与授权。

2.推文服务(Tweet Service):处理推文的添加、删除和获取。

3.时间线服务(Timeline Service):生成用户的时间线，整合来自多个推文源的数据。4.消息队列(Message Queue):用于异步处理推文、通知和消息传递，以提高系统的响应速度5.缓存层(Cache Layer):使用Redis等高速缓存，减少数据库的读取压力。6.数据库(Database):持久化存储用户信息和推文历史。

采用微服务架构为其系统带来了诸多优势，主要包括以下几个方面：

1. 灵活性与可扩展性

微服务架构允许 Twitter 将其庞大的平台分解为多个独立的服务，每个服务专注于特定的功能（如推文发布、通知管理、消息处理等）。这种灵活性使得 Twitter 能够根据实时流量和用户需求迅速扩展特定的服务，而无需重新构建整个系统。

2. 快速迭代与部署

通过微服务架构，Twitter 的开发团队可以独立开发、测试和部署各个微服务。这种独立性使得新功能的推出和现有功能的优化更加高效，团队可以更快地响应市场变化，从而提高用户体验。

3. 故障隔离与系统可靠性

微服务架构的设计使得单个服务的故障不会直接影响到整个系统的运行。Twitter 能够确保即使某一微服务出现问题，其他服务依然能够维持正常运作，从而提升整个平台的可靠性和用户的连续体验。

4. 技术多样性

在微服务架构中，Twitter 可以根据每个服务的需求选择适合的技术栈。这种技术多样性不仅支持开发团队使用最佳工具还可以便于引入新技术以优化服务性能，如在特定服务中使用不同的编程语言或数据库。

5. 增强的团队协作

将平台划分为微服务后，各个团队可以独立负责不同的服务，减少了团队之间的依赖和沟通成本。不同团队能够专注于各自的功能开发，促进了协调合作，提高了开发效率。

6. 改善性能和响应时间

微服务架构让 Twitter 能够为不同的功能进行特别的性能优化。比如，可以针对高频使用的功能（如推文显示）进行专门优化，与其他较少使用的功能分开，提高了系统的整体性能和用户的实时响应时间。

7. 更好的监控与维护

微服务架构配合相关的监控工具，使得 Twitter 可以对每个服务的性能进行单独监控，及时发现和解决问题。团队可以更轻松地定位故障，并执行针对性的维护，提升了系统的整体健康水平。

8. 服务发现与负载均衡

微服务架构能够集成服务发现和负载均衡机制，这使得 Twitter 能够根据实时流量自动调整服务实例，并在不同的服务器之间分配请求，优化资源利用，提高应用的稳定性。

然而微服务架构也会带来一些问题，在高并发情况下，推文的发布可能导致:

服务过载:用户同时发推，服务器无法处理，

数据一致性:推文可能在短时间内被多个服务访问和修改。延迟:消息从用户发送到显示在时间线上的延迟。

# **参考文献**

1. 张墨涵, 王雪英, 沈学东, 吴茂, 永康昊. 微服务架构技术与挑战[J]. 网络安全技术与应用, 2023(02).
2. Fowler M,Lewis J.Microservices--a definition of this new architectural term[EB/OL].(2014-03-25).

https://martinfowler.com/articles/microservices.html.

1. 汪友杰, 王平, 李咀庆, 王睿昕, 张云鹏. 浅谈微服务架构[C]// 2020年中国通信学会能源互联网学术报告会论文集. 中国四川成都: 中国通信学会, 2020-12-06. DOI: 10.26914/c.cnkihy.2020.065772.
2. 于晓虹. 微服务架构在分布式系统的设计和应用[J]. 电子技术与软件工程, 2021(06).
3. 张炼. 基于微服务架构的企业级软件开发实践探讨[J]. 中国信息界, 2024(02).