Momento项目文档

1. 需求分析

1.1 背景介绍

1.1.1 项目背景

Momento 是一个面向年轻人、内容创作者以及生活方式分享者的社交社区平台，用户可以通过平台发布笔记、点赞、收藏以及关注其他用户。随着短视频和图文分享平台的迅速发展，社交媒体平台的用户规模仍保持增长态势。根据艾瑞咨询的研究，用户使用社交产品主要迎合核心的七大需求：荷尔蒙需求、减少孤独感、自我表达、社交工具、炫耀、发泄吐槽和自我提升。 Momento 旨在提供一个兼具高性能、高可用性和个性化推荐的互动平台，以满足用户在社交和内容消费上的需求。

Momento 平台的核心价值在于：

1. **快速响应**：确保用户的操作在毫秒级响应，提升互动体验。
2. **数据一致性**：在分布式架构下，通过强一致性或最终一致性保障用户数据的完整性。
3. **高并发支持**：通过分布式缓存和消息队列设计，满足高峰期的百万级用户并发需求。

1.1.2 目标用户

目标用户包括以下人群：

1. **年轻用户群体**：注重生活方式分享的用户，例如旅行、时尚、健身爱好者。
2. **内容创作者**：希望通过平台获得曝光和粉丝关注的创作者。
3. **社交互动型用户**：关注社交互动和圈层文化的用户，喜欢通过点赞、收藏与关注建立连接。

用户需求分析：

- **内容创作需求**：简便的发布和编辑工具，支持多媒体（图片、视频）。
- **互动需求**：便捷的点赞、收藏、评论和关注功能，促进社交互动。
- **性能需求**：快速响应的界面和流畅的操作体验。
- **安全需求**：确保用户数据的隐私和账号安全。

1.2 需求概述

1.2.1 功能需求

1. **用户管理**：用户可通过账号密码或邮箱验证码注册与登录，支持个人信息的修改、查询以及密码更新功能，同时可以关注或取消关注其他用户，建立社交关系链。
2. **笔记管理**：用户能够发布、编辑、删除笔记，支持上传多媒体内容（如图片、视频），并通过关键词或标签筛选条件查询笔记。此外，用户可以设置笔记的公开或私密权限，并对重要内容进行置顶。
3. **互动功能**：支持点赞、收藏、评论功能，允许用户对笔记内容进行多维度互动。其中点赞和收藏需支持高并发写操作，评论功能支持一级评论和二级评论及其删除操作。
4. **内容展示**：用户主页展示功能包括已发布的笔记、粉丝列表和关注列表等内容。笔记详情页面需显示点赞数、评论数、收藏数等互动数据，并实时更新。
5. **计数服务**：平台需要提供用户维度（如粉丝数、关注数、点赞数）和笔记维度（如点赞数、评论数、收藏数）的实时统计功能，支持高并发场景下的精准计数。
6. **系统管理**：平台需实现分布式 ID 生成，以保证唯一性和高效性。同时，对象存储管理支持用户上传的图片和视频的安全存储，日志记录与异常处理功能需实时监控系统状态并对异常情况进行追踪和处理。

1.2.2 非功能需求

1. 用户规模和系统性能需求

- **性能要求**：平台需支持每秒至少 100,000 次高并发写操作，确保核心操作（如点赞、收藏）平均响应时间低于 1 秒，90%的请求需在100ms内完成。
- **用户规模**：
  - 目标初期用户量：100,000 日活跃用户。
  - 峰值并发请求：10,000 次/秒，主要集中在点赞、收藏、发布笔记等高频交互操作。
- **可用性**：系统需提供 99.9% 的在线时间，支持在服务故障时启用熔断和降级机制，确保核心功能可用。
- **数据一致性**：用户关键操作的数据（如点赞、收藏、发布）需在 1 秒内实现最终一致性，对重要事务采用分布式事务管理，保证操作的强一致性。

- **可维护性**：采用模块化设计，便于功能扩展和系统维护，同时要求所有功能模块的单元测试覆盖率不低于 80%，保障代码质量。

2. 扩展模式

- **水平扩展**：
  - 微服务架构下，所有服务通过增加实例数量支持水平扩展。
  - 使用负载均衡（Nginx 或 Spring Cloud Gateway）分发请求。
- **分库分表与读写分离**：
  - 按用户 ID 或笔记 ID 分库分表，分散数据库读写压力。
  - 对常见读请求采用主从数据库的读写分离策略。
- **缓存与异步优化**：
  - Redis 作为热点数据缓存（如点赞数、收藏数），减少数据库访问。
  - 通过消息队列（RocketMQ）异步处理写入，降低数据库压力。
- **异地多活部署**：
  - 支持跨区域用户的流量分发，结合 CDN 提升访问速度。

1.2.3 用户需求

1. **需求调研**

   通过用户访谈和问卷调查，发现用户对Momento平台的主要需求集中在内容创作、互动体验、性能要求、内容消费、安全性以及个性化推荐等方面。用户希望平台能够提供一个简洁高效的内容发布工具，支持文本、图片和视频的上传，同时允许对内容进行权限设置和分类管理，以满足不同场景的使用需求。在互动方面，用户强调点赞、收藏、评论和关注操作的实时性和便捷性，希望能够快速表达对内容的喜爱和与其他用户的社交联系。

   此外，性能是用户关注的重点，高并发场景下的平台稳定性和快速响应直接影响用户体验，尤其是在活动推广或高流量时期，平台需保障核心功能的流畅运行。在内容消费上，用户倾向于通过推荐算法快速找到与其兴趣相关的内容，同时希望筛选功能简单易用，例如通过关键词或标签快速定位目标内容。

   在安全方面，用户对个人数据隐私和账户保护有很高的要求，希望平台提供多因子认证和数据加密机制，尤其是在敏感操作（如密码修改）中实时通知用户，以确保数据安全和操作透明度。此外，用户对个性化推荐的期望也非常明确，他们希望平台根据历史行为提供精准的内容推荐，并通过个性化主页展示和动态更新，提升使用体验和内容发现效率。

2. **用户故事**

   注册与登录：作为一个新用户，我希望能通过邮箱或手机号快速注册和登录，以便立即体验平台的功能。

   发布笔记：作为一个内容创作者，我希望能够轻松发布包含图片和视频的笔记，并对笔记添加标签和设置权限，以更好地分享我的内容并保护隐私。

   点赞与收藏：作为一个普通用户，我希望能够快速点赞和收藏感兴趣的笔记，以便随时查看并支持创作者。

   关注用户：作为一个社交用户，我希望能关注喜欢的内容创作者，并能随时查看他们的最新动态和发布的笔记。

   互动数据展示：作为一个用户，我希望平台能实时显示笔记的点赞数、收藏数和评论数，以便了解内容的受欢迎程度并与之互动。

个性化推荐：作为一个用户，我希望平台能根据我的兴趣和浏览记录推荐相关内容，以便高效发现感兴趣的内容并优化使用体验。

1.3 功能详细描述

1.3.1 用户管理

1. 用户注册与登录

功能描述	**用户可以通过账号密码或手机验证码进行注册和登录。**
输入要求	注册：用户名、密码、邮箱、验证码登录：用户名/邮箱、密码或验证码
输出要求	成功注册/登录后返回用户身份认证信息（JWT token）
数据流程	用户提交注册/登录信息 → 系统验证信息 → 成功则返回认证信息
用户交互	注册和登录界面，输入表单验证，错误提示
错误处理	输入信息不完整或错误时提示用户重新输入

2. 用户信息管理

功能描述	**用户可以修改个人信息、查询用户信息、更新密码。**
输入要求	用户ID、修改的具体信息（如昵称、头像、密码）
输出要求	修改成功的确认信息或更新后的用户信息
数据流程	用户提交修改请求 → 系统验证权限 → 更新数据库 → 返回结果
用户交互	个人信息编辑页面，表单验证
错误处理	权限不足、输入信息不合法时提示错误

1.3.2 笔记管理

1. 笔记发布与编辑

功能描述	**用户可以创建新的笔记或编辑已有笔记。**
输入要求	笔记内容（文本、图片、视频）、标签、权限设置
输出要求	发布/编辑成功的确认信息及笔记ID
数据流程	用户提交笔记内容 → 系统保存到数据库和对象存储 → 返回结果
用户交互	笔记编辑界面，富文本编辑器，媒体上传功能
错误处理	上传失败、内容不合法时提示错误

2. 笔记浏览与查询

功能描述	**用户可以浏览和搜索笔记。**
输入要求	搜索关键词、过滤条件（如标签、作者）

功能描述	**用户可以浏览和搜索笔记。**
输出要求	符合条件的笔记列表
数据流程	用户提交查询请求 → 系统查询数据库和缓存 → 返回结果
用户交互	搜索框、过滤选项，分页显示
错误处理	无结果时提示无相关笔记

1.3.3 互动功能

1. 点赞功能

功能描述	**用户可以对笔记进行点赞操作，支持高并发写入。**
输入要求	用户ID、笔记ID
输出要求	点赞成功的确认信息，当前点赞数
数据流程	用户点击点赞 → 系统通过Redis Bloom过滤器判断是否已点赞 → 通过Redis ZSET记录点赞 → 异步落库
用户交互	点赞按钮的状态更新
错误处理	重复点赞或系统异常时提示错误

2. 收藏功能

功能描述	**用户可以对笔记进行收藏操作，支持高并发写入。**
输入要求	用户ID、笔记ID
输出要求	收藏成功的确认信息，当前收藏数
数据流程	用户点击收藏 → 系统通过Redis Bloom过滤器判断是否已收藏 → 通过Redis ZSET记录收藏 → 异步落库
用户交互	收藏按钮的状态更新
错误处理	重复收藏或系统异常时提示错误

3. 关注与取关功能

功能描述	**用户可以关注或取关其他用户，支持高并发写入。**
输入要求	用户ID、目标用户ID
输出要求	关注/取关成功的确认信息，当前关注数或粉丝数
数据流程	用户点击关注/取关 → 系统通过Redis判断是否已关注 → 更新关注列表 → 异步落库
用户交互	关注按钮的状态更新（关注/取关切换）
错误处理	重复关注、取关失败或系统异常时提示错误

1.3.4 计数服务

1. 统计用户互动数

功能描述	**统计用户的关注数、粉丝数、点赞数等，确保在高并发下的高性能。**
输入要求	用户行为数据（关注、取关、点赞等）
输出要求	实时统计数据
数据流程	用户行为触发计数请求 → 通过MQ异步处理 → 更新Redis缓存和数据库
用户交互	用户界面显示最新的统计数据
错误处理	计数失败时的重试机制

2. 笔记维度的计数

功能描述	**统计笔记相关的数据，如点赞数、评论数、收藏数等，确保高并发时的高性能。**
输入要求	笔记ID、用户行为数据（点赞、评论、收藏等）
输出要求	实时更新的笔记相关统计数据（如点赞数、评论数、收藏数等）
数据流程	用户对笔记进行点赞/评论/收藏操作 → 通过MQ异步处理 → 更新Redis缓存和数据库
用户交互	显示笔记页面时展示笔记的统计数据（如点赞数、评论数、收藏数等）
错误处理	计数失败时的重试机制，系统异常时提示用户操作失败

1.4 约束条件

1. **时间限制**：项目开发周期为3个月，需在规定时间内完成所有功能模块的开发和测试。
2. **DevOps限制**：项目成员人数有限，需要在合适的范围内选择合适的DevOps流程。
3. **技术约束**：必须使用指定的技术栈，如Spring Cloud Alibaba、MyBatis、Redis等；系统架构需采用微服务分布式架构，确保高可扩展性和高可用性。
4. **资源限制**：开发团队主要由Java后端开发人员组成，需要合理分配任务和资源。

1.5 用例分析

用例1：用户注册与登录

用例名称	**用户注册与登录**
参与者	新用户
前置条件	用户尚未注册或已注册但未登录
基本流程	1. 用户访问注册页面，填写注册信息（用户名、密码、手机号码）2. 系统验证信息有效性，发送手机验证码3. 用户输入验证码，完成注册4. 用户登录，系统返回JWT token
替代流程	- 用户忘记密码，选择密码重置流程
后置条件	用户成功注册并登录，获得JWT token
异常流程	1. 输入信息不完整或格式错误，系统提示“请检查输入信息”。2. 用户注册时已存在相同用户名或邮箱，系统提示“用户名或邮箱已被注册”。
扩展流程	- 用户注册时选择邮箱验证而非验证码，系统通过邮件发送验证码。

用例2：发布笔记

用例名称	**发布笔记**
参与者	注册用户
前置条件	用户已登录
基本流程	1. 用户访问笔记发布页面，输入内容并上传媒体2. 系统保存笔记内容到数据库和对象存储3. 返回发布成功信息及笔记ID
替代流程	- 上传媒体失败，提示用户重新上传。 - 用户输入的内容为空，系统提示“内容不能为空”。
后置条件	系统成功保存笔记并分配笔记ID，笔记在系统中可见
异常流程	1. 媒体上传失败，用户收到“上传失败，请重新上传”的提示。2. 用户没有填写必要内容（如笔记标题、标签等），提示“请完整填写内容”。
扩展流程	- 用户选择保存草稿，系统将笔记保存为草稿状态，用户可随时编辑并发布。

用例3：点赞笔记

用例名称	**点赞笔记**
参与者	注册用户
前置条件	用户已登录，浏览笔记详情
基本流程	1. 用户点击点赞按钮2. 系统通过Redis Bloom过滤器判断用户是否已点赞3. 若未点赞，记录点赞信息并更新点赞数4. 返回点赞成功信息
替代流程	- 用户已点赞，提示取消点赞或不能重复点赞。
后置条件	笔记的点赞数增加，用户的点赞状态被记录
异常流程	1. 系统出现异常，无法记录点赞操作，提示“点赞失败，请稍后再试”。2. 网络不稳定，用户收到“点赞操作超时”提示。
扩展流程	- 用户取消点赞，系统记录取消操作并减少点赞数。

用例4：关注用户

用例名称	**关注用户**
参与者	注册用户
前置条件	用户已登录，浏览目标用户主页
基本流程	1. 用户点击关注按钮2. 系统通过Redis处理关注操作，发送顺序消息至MQ3. 更新用户关系数据，返回关注成功信息
替代流程	- 用户已关注，提示取消关注或不重复关注
后置条件	目标用户被添加到当前用户的关注列表中
异常流程	1. 网络错误，系统无法完成关注操作，提示“关注失败，请稍后再试”。2. 用户尝试关注已关注的用户，系统提示“您已关注此用户”。
扩展流程	- 用户选择取消关注，系统更新数据并返回取消关注成功信息。

用例5：收藏笔记

用例名称	**收藏笔记**
参与者	注册用户
前置条件	用户已登录，浏览笔记详情
基本流程	1. 用户点击收藏按钮2. 系统检查用户是否已收藏该笔记3. 若未收藏，记录收藏操作并返回成功信息4. 更新收藏数
替代流程	- 用户已收藏，提示取消收藏或不能重复收藏。
后置条件	笔记被加入用户的收藏列表，收藏数更新
异常流程	1. 收藏操作失败，提示“收藏失败，请稍后再试”。2. 用户网络不稳定，提示“收藏操作超时”。
扩展流程	- 用户取消收藏，系统更新数据并返回取消收藏成功信息。

用例6：查询笔记

用例名称	**查询笔记**
参与者	注册用户
前置条件	用户已登录
基本流程	1. 用户访问笔记查询页面，输入关键词或选择过滤条件（标签、作者等）2. 系统从数据库中查询符合条件的笔记3. 返回查询结果，并展示笔记列表
替代流程	- 无符合条件的笔记，系统提示“未找到相关笔记”。
后置条件	系统展示符合条件的笔记列表
异常流程	1. 网络错误，无法加载笔记列表，提示“加载失败，请稍后再试”。2. 查询参数错误，提示“查询条件无效”。
扩展流程	- 用户进行分页查询，系统返回当前页面的笔记列表。

用例7：修改个人信息

用例名称	**修改个人信息**
参与者	注册用户
前置条件	用户已登录
基本流程	1. 用户访问个人信息修改页面2. 输入新的信息（如昵称、头像、密码等）3. 系统验证并保存修改的信息4. 返回修改成功信息
替代流程	- 用户未填写必要信息（如新密码），系统提示“请填写必要信息”。 - 用户修改头像失败，提示“上传头像失败，请重试”。
后置条件	用户的个人信息被更新，用户页面显示最新的个人信息
异常流程	1. 系统验证失败，提示“信息格式错误，请重新输入”。2. 网络不稳定，保存信息失败，提示“保存失败，请稍后再试”。
扩展流程	- 用户选择取消修改，系统不保存任何变更并返回到个人信息页面。

用例8：重置密码

用例名称	**重置密码**
参与者	已注册用户
前置条件	用户忘记密码并请求重置密码
基本流程	1. 用户点击“忘记密码”链接，输入邮箱/手机号2. 系统发送验证码到用户邮箱或手机号3. 用户输入验证码，设置新密码4. 系统验证并保存新密码
替代流程	- 用户未收到验证码，系统提示“验证码发送失败，请重新请求”。 - 用户输入的验证码错误，系统提示“验证码错误，请重新输入”。
后置条件	用户密码被重置，用户能够使用新密码登录系统
异常流程	1. 用户输入错误的邮箱/手机号，提示“该邮箱/手机号未注册”。2. 系统无法发送验证码，提示“发送失败，请稍后再试”。
扩展流程	- 用户在忘记密码流程中选择通过邮箱重置密码，而非短信验证码。

1.6 优先级和版本控制

1.6.1 需求优先级排序

1. **高优先级**
    - 用户注册与登录
    - 笔记发布与编辑
    - 点赞与收藏功能
    - 关注与取关功能
    - 高并发处理与数据一致性
2. **中优先级**
    - 用户信息管理
    - 笔记浏览与查询
    - 粉丝列表与关注列表查询
3. **低优先级**
    - 计数服务的优化
    - 高级搜索与推荐功能

1.6.2 版本控制

- **初期版本（V1.0）**
    - 用户注册与登录
    - 笔记发布与编辑
    - 点赞与收藏功能
    - 关注与取关功能
    - 基础的高并发处理
- **后续版本（V1.1及以后）**
    - 用户信息管理
    - 笔记浏览与查询
    - 粉丝列表与关注列表查询
    - 计数服务优化
    - 高级搜索与推荐功能

1.7 需求验证

1.7.1 测试策略

并发测试

1. **测试工具**：使用 JMeter 模拟 10,000 QPS 的高并发场景。
2. **测试内容**：
    - 点赞、收藏的并发写操作。
    - 笔记浏览的高频读操作。
    - 消息队列的异步处理能力。

3. **测试结果**：
    - 峰值并发量达到 15,000 QPS，系统响应时间平均为 120ms。
    - Redis 缓存命中率达 92%，数据库压力显著降低。
    - 在高并发场景下，部分服务因锁冲突导致短暂超时。

扩容思路

1. **服务优化**：
    - 增加微服务实例数量，并通过负载均衡分发流量。
    - 优化热点服务代码（如点赞功能），减少锁冲突。
2. **数据库扩展**：
    - 针对高并发写入场景，采用 Redis 临时存储数据，批量同步至数据库。
    - 对数据库执行分区与索引优化，提升查询性能。
3. **动态资源调配**：
    - 使用 Kubernetes 自动扩展实例，确保服务弹性。

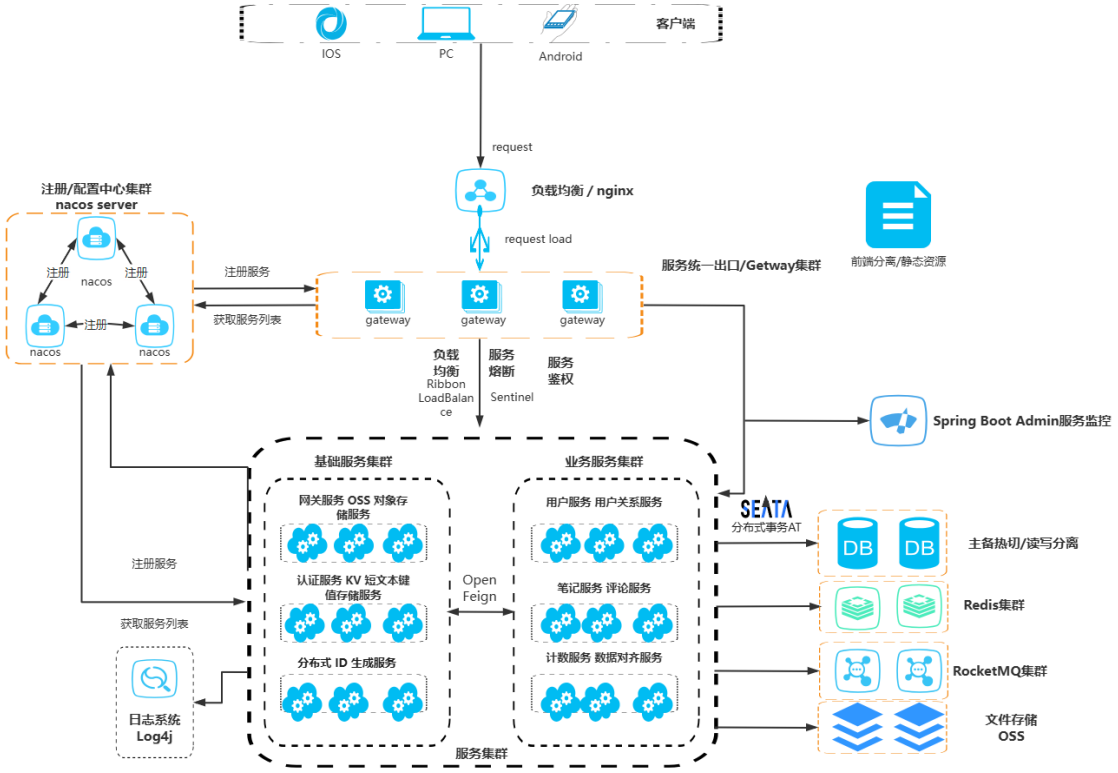
1.7.2 验收标准

- **功能完整性**：所有高优先级功能模块均按需求实现，无重大功能缺失。
- **性能指标达标**：系统能够支持预期的高并发用户访问，响应时间不超过2秒。
- **数据一致性**：在高并发场景下，数据保持一致，无数据丢失或错误。
- **安全性合格**：通过所有安全测试，未发现严重漏洞。
- **用户体验良好**：界面友好，操作流畅，用户反馈积极。

2. 系统设计

2.1 系统体系设计

整体系统采用**微服务架构**，通过分布式部署和异步通信机制，确保平台能应对大规模用户访问及高并发操作，以下是我们的**系统体系设计图：**



2.1.1 系统架构模式

Momento平台采用基于Spring Cloud Alibaba的微服务架构模式。微服务架构通过将系统拆分为多个独立的服务，每个服务负责单一的业务功能，从而提升了系统的可维护性和可扩展性。服务间通过轻量级协议（如RESTful API）进行通信，采用消息队列（如RocketMQ）进行异步处理，避免了同步调用对系统性能的影响。

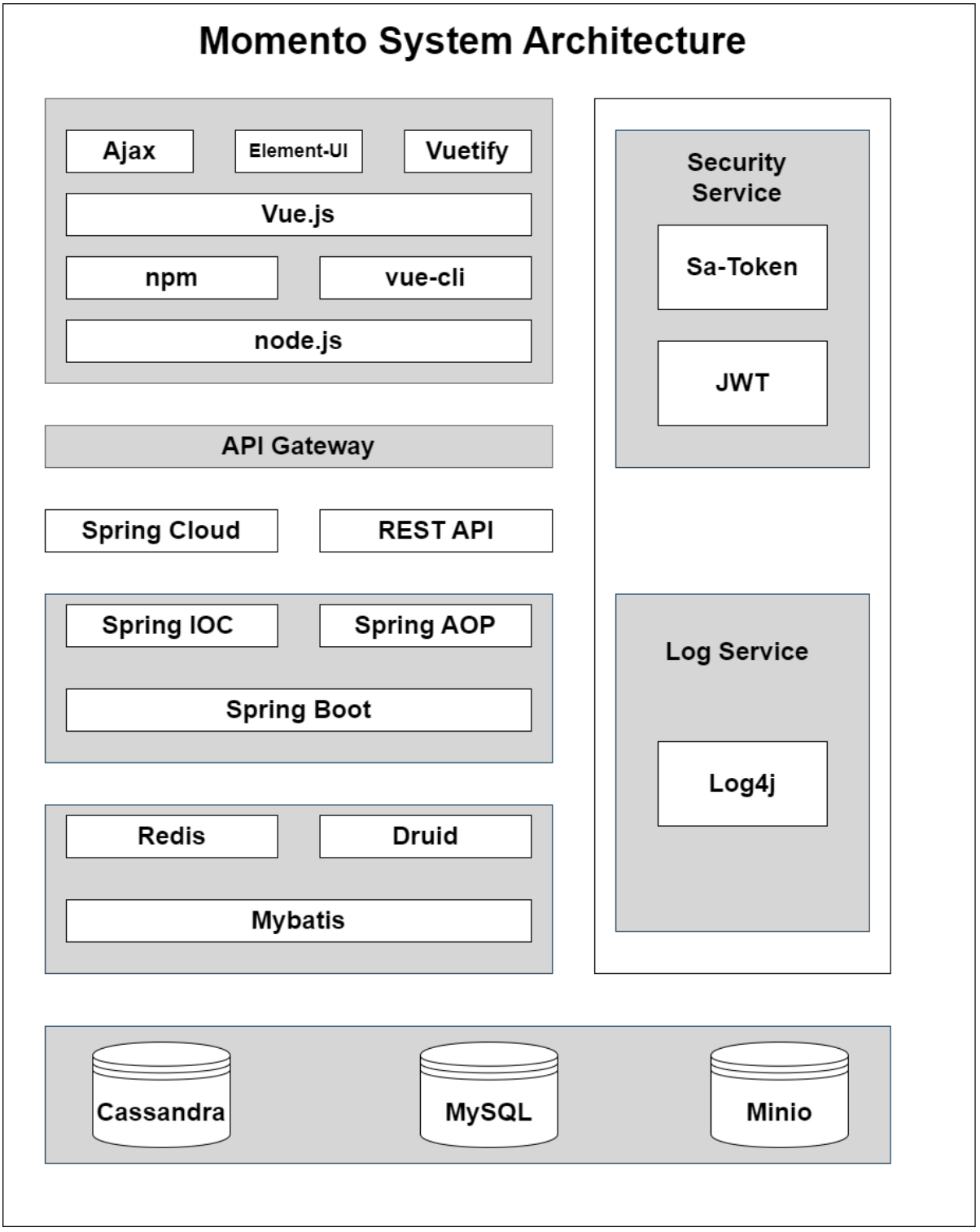
2.1.2 架构组成

1. **前端**：使用Vue.js开发Web应用，通过HTTP RESTful API与后端微服务进行数据交互。前端负责展示用户界面和交互逻辑。
2. **后端**：采用 Spring Boot开发，划分为多个微服务模块，每个模块专注于特定的业务功能，所有后端服务通过Spring Cloud进行集成与管理：
    - **网关服务**：处理所有外部请求，负责路由管理和全局鉴权。
    - **认证服务**：负责用户登录、注册与认证，基于 JWT 实现认证流程。
    - **用户服务**：管理用户信息，提供注册、登录、修改信息等功能。
    - **笔记服务**：处理笔记的创建、编辑、删除及权限管理。
    - **关系服务**：管理用户之间的社交互动，如关注与取关。
    - **计数服务**：负责统计用户的社交互动数据，如点赞、粉丝等。
    - **OSS 服务**：存储与管理用户上传的图片、视频等非结构化数据。
    - **消息队列**：确保异步操作顺序性，减少数据库压力。
    - 后端服务通过 **Nginx** 进行反向代理，负载均衡。
3. **数据库与缓存**：数据库采用 **MySQL** 存储用户信息、笔记信息、互动数据等。使用 **Redis** 缓存热点数据，如笔记列表、点赞数、收藏数等。数据库采用 **ShardingSphere** 进行分库分表，保证数据一致性和高性能。
4. **消息队列**：采用RocketMQ处理异步任务，确保用户操作数据的高效处理和系统解耦。

5. **对象存储**：使用MinIO对象存储系统保存用户上传的多媒体文件（如图片、视频等）。
6. **API 网关**：通过Spring Cloud Gateway实现统一的API网关，提供路由管理、身份验证和流量控制功能。

2.2 技术栈与环境

技术栈设计图：



2.2.1 技术栈

- **微服务框架**：基于 Spring Cloud Alibaba 实现微服务架构，支持服务治理、限流、熔断等功能。**Spring Boot**用于构建独立的、生产级别的Spring应用，支持快速开发、部署和运维。
- **API 网关**：Spring Cloud Gateway 作为统一入口，提供路由管理、权限认证、限流控制等功能。网关确保了后端服务的安全性和可扩展性。

- **消息队列**：用于异步消息处理，解耦服务之间的依赖，提供高吞吐量、高可靠性的消息传递。适用于处理点赞、评论、收藏等高并发操作。
- **数据持久化：**
  - **数据库**：MySQL 用于存储核心数据。Cassandra存储短文本数据。
  - **缓存层**：Redis 和 Caffeine 提供多级缓存，减轻数据库负载。
  - **对象存储**：MinIO 和阿里云 OSS 支持图片、视频等非结构化数据存储。
- **容器化与部署：**所有微服务均使用**Docker**容器化部署，确保环境的一致性，简化开发、测试和生产环境的配置。
- **持续集成/部署**：通过 CI/CD 工具链确保开发效率与快速迭代。
- **安全与认证**
  - **JWT**：用于实现无状态认证，保护API接口，确保用户数据的安全性。
  - **SaToken**：处理统一身份认证和授权，支持多种认证机制（如单点登录、多因子认证等）

2.3 核心功能模块设计

2.3.1 核心模块

1. 用户管理模块（User Service）

- 负责用户的注册、登录、修改个人信息、密码重置等功能。
- 关键功能：
  - 注册与登录：通过用户名、邮箱、验证码等方式进行认证，使用 JWT token 实现用户的身份认证。
  - 用户信息管理：用户可以修改个人信息，如昵称、头像等。
  - 密码重置：提供忘记密码功能，通过短信/邮箱验证码完成密码重置。

2. 笔记管理模块（Note Service）

- 负责笔记的发布、编辑、删除、查看等功能。
- 关键功能：
  - 笔记发布与编辑：用户可以发布新的笔记，上传图片、视频等媒体。
  - 笔记浏览与查询：用户可以浏览笔记列表，支持按照标签、关键词等条件进行筛选和查询。
  - 笔记删除：用户可以删除自己发布的笔记。

3. 互动模块（Interaction Service）

- 负责用户之间的互动，包括点赞、收藏、关注、取关等功能。
- 关键功能：
  - 点赞：用户可以对笔记进行点赞，点赞数将实时更新。
  - 收藏：用户可以将喜欢的笔记收藏，收藏数实时更新。
  - 关注与取关：用户可以关注其他用户，获取该用户的最新动态。

4. 计数服务模块（Statistics Service）

- 负责统计用户的互动数据（如点赞数、收藏数、关注数）以及笔记的互动数据（如点赞数、收藏数）。
- 关键功能：

- 统计用户互动数据：统计用户的关注数、粉丝数、点赞数等，支持高并发的计数操作。
- 统计笔记互动数据：统计笔记的点赞数、收藏数等。

2.4 数据库与存储设计

数据库设计
使用MySQL进行关系型数据存储，设计合理的表结构，包含用户表、笔记表、互动表等。对高并发访问的字段进行优化，如增加索引和缓存策略，确保查询效率。

存储设计

- **分布式存储**：使用MinIO或其他云存储服务，保存用户上传的多媒体文件（如图片、视频）。
- **缓存设计**：采用Redis缓存热点数据，减少数据库的压力，提高系统响应速度。

以下是用户表和笔记表的设计示例：

1. 用户表（user）

字段名	数据类型	描述
id	INT	用户ID，主键
username	VARCHAR(255)	用户名
password	VARCHAR(255)	密码
email	VARCHAR(255)	邮箱
phone	VARCHAR(20)	手机号码
avatar	VARCHAR(255)	头像链接
created_at	DATETIME	用户注册时间
updated_at	DATETIME	用户信息更新时间

2. 笔记表（note）

字段名	数据类型	描述
id	INT	笔记ID，主键
user_id	INT	用户ID（外键）
content	TEXT	笔记内容
media	VARCHAR(255)	媒体链接
tags	VARCHAR(255)	标签
created_at	DATETIME	笔记创建时间
updated_at	DATETIME	笔记更新时间

2.5 接口设计规范

2.5.1 统一响应格式

所有接口返回数据均采用统一响应格式，确保前端可以统一解析，简化错误处理和数据管理。

响应结构：

名称	类型	必选	约束	中文名	说明
`success`	boolean	true	none	响应状态	`true`表示成功，`false`表示失败
`message`	string	true	none	响应消息	错误时返回的错误消息
`errorCode`	string	true	none	错误码	错误时返回的异常码
`data`	object	true	none	响应数据	请求成功时返回具体的数据

返回示例：

```
{
  "success": true,
  "message": "",
  "errorCode": "",
  "data": {
    "userId": "12345",
    "userName": "John Doe"
  }
}
```

2.5.2 统一请求头

请求头需包含认证信息，所有需要身份验证的接口都应在请求头中传递 `Authorization` 字段。该字段的值应为登录成功后获取的 `JWT token`。

名称	位置	类型	必选	说明
`Authorization`	header	string	是	必须提供的 `JWT token`，用于身份验证

2.5.3 HTTP 状态码规范

接口应根据请求处理的结果返回相应的HTTP状态码，以帮助客户端判断请求是否成功。

常用状态码及其含义：

- **200 OK**：请求成功，返回请求的数据（适用于GET、PUT、DELETE等请求）
- **201 Created**：请求成功并创建资源（适用于POST请求）
- **204 No Content**：请求成功，但没有返回任何内容（适用于DELETE请求）
- **400 Bad Request**：请求无效或参数错误
- **401 Unauthorized**：未认证或认证失败
- **403 Forbidden**：认证通过，但没有权限访问该资源
- **404 Not Found**：请求的资源不存在

- **405 Method Not Allowed**：请求方法不允许，通常是请求方法与资源不匹配
- **409 Conflict**：请求与现有资源冲突，通常出现在资源已存在时
- **500 Internal Server Error**：服务器内部错误
- **502 Bad Gateway**：网关错误，通常由服务器的依赖服务出错引起
- **503 Service Unavailable**：服务不可用，通常是因为服务器正在维护或超载

2.5.4 接口命名规范

接口命名应简洁明了，采用RESTful风格，通常是名词复数形式，路径采用小写字母，多个单词使用连字符（`-`）分隔。

- **资源的命名**：使用名词的复数形式，例如：
  - `/users`：获取用户列表
  - `/users/{userId}`：获取指定用户信息
- **接口路径设计**：
  - 获取资源：`GET /resource`
  - 创建资源：`POST /resource`
  - 更新资源：`PUT /resource/{id}`
  - 删除资源：`DELETE /resource/{id}`
  - 搜索/过滤资源：`GET /resource?filter=value`
  - 批量操作：`POST /resource/batch`
  - 认证相关操作：`POST /auth/login`，`POST /auth/logout`

2.5.5 请求参数规范

- **路径参数**：路径中的动态部分表示资源的标识符，例如：`/users/{userId}`、`/posts/{postId}`。
- **查询参数**：用于过滤、排序、分页等，采用`key=value`的形式，例如：`/posts?page=2&size=20&sort=createdAt`。
- **请求体**：POST、PUT、PATCH等方法通过请求体传递数据。请求体应包含必要的字段，采用JSON格式。

示例：

```
{
  "userId": "12345",
  "userName": "John Doe"
}
```

2.5.5 错误处理与自定义错误码

所有接口都应在失败时返回清晰的错误信息。错误信息包含`message`字段，帮助开发者了解错误的原因。同时，为了方便前端的错误处理，可以设计统一的错误码系统。

错误码示例：

错误码	含义	解决办法
`1000`	参数缺失或不合法	检查请求参数是否缺失或格式错误
`1001`	未授权	用户未登录或token无效
`1002`	权限不足	当前用户无权限访问资源
`1003`	资源不存在	请求的资源未找到
`2000`	系统内部错误	请稍后再试或联系管理员

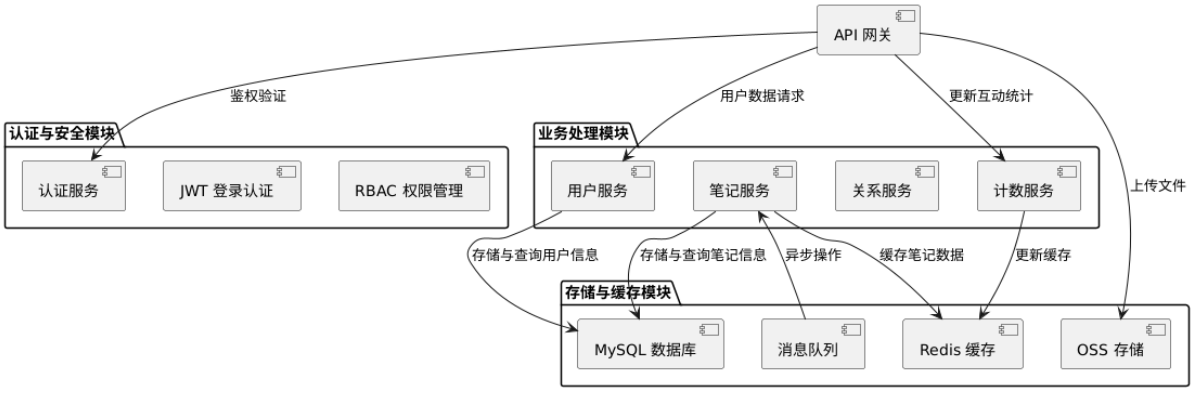
2.5.6 安全性与防护

- **输入验证**：所有输入参数都应进行严格的校验，防止SQL注入、XSS攻击等。
- **身份认证**：通过JWT token进行身份认证，保证只有经过认证的用户才能访问受保护的资源。

2.6 系统组件图与流程图

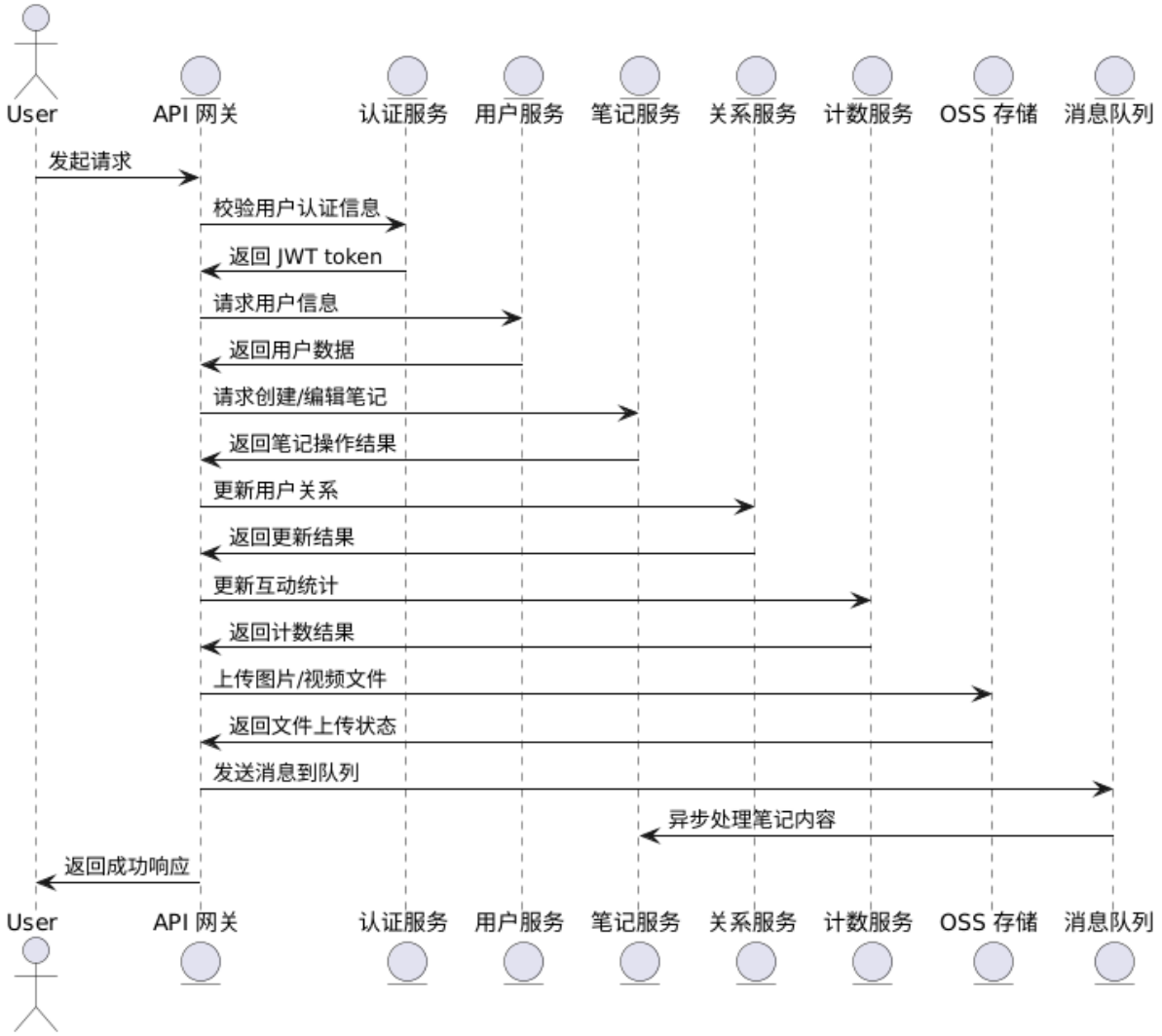
2.6.1 系统组件图

为了更加清晰地展示系统的整体结构，我们引入了**系统组件图**。该图展示了系统中各个主要模块及其之间的关系，帮助理解每个服务如何协同工作，满足系统的高可用性与可扩展性需求。例如，用户服务、笔记服务、关系服务等通过 API 网关对外提供服务，同时使用消息队列进行异步任务处理，确保系统在高并发环境下的流畅运行。



2.6.2 系统流程图

系统流程图帮助我们理解各个模块之间的交互流程。举个例子，用户发起请求后，首先通过网关进行路由和认证，认证通过后，相关模块（如用户服务、笔记服务等）会处理具体操作并返回响应。该流程图清晰地展示了每个请求的生命周期，以及涉及到的多个微服务和缓存、队列等中间件的作用，帮助我们在设计时更好地规划模块间的通信与数据流动。



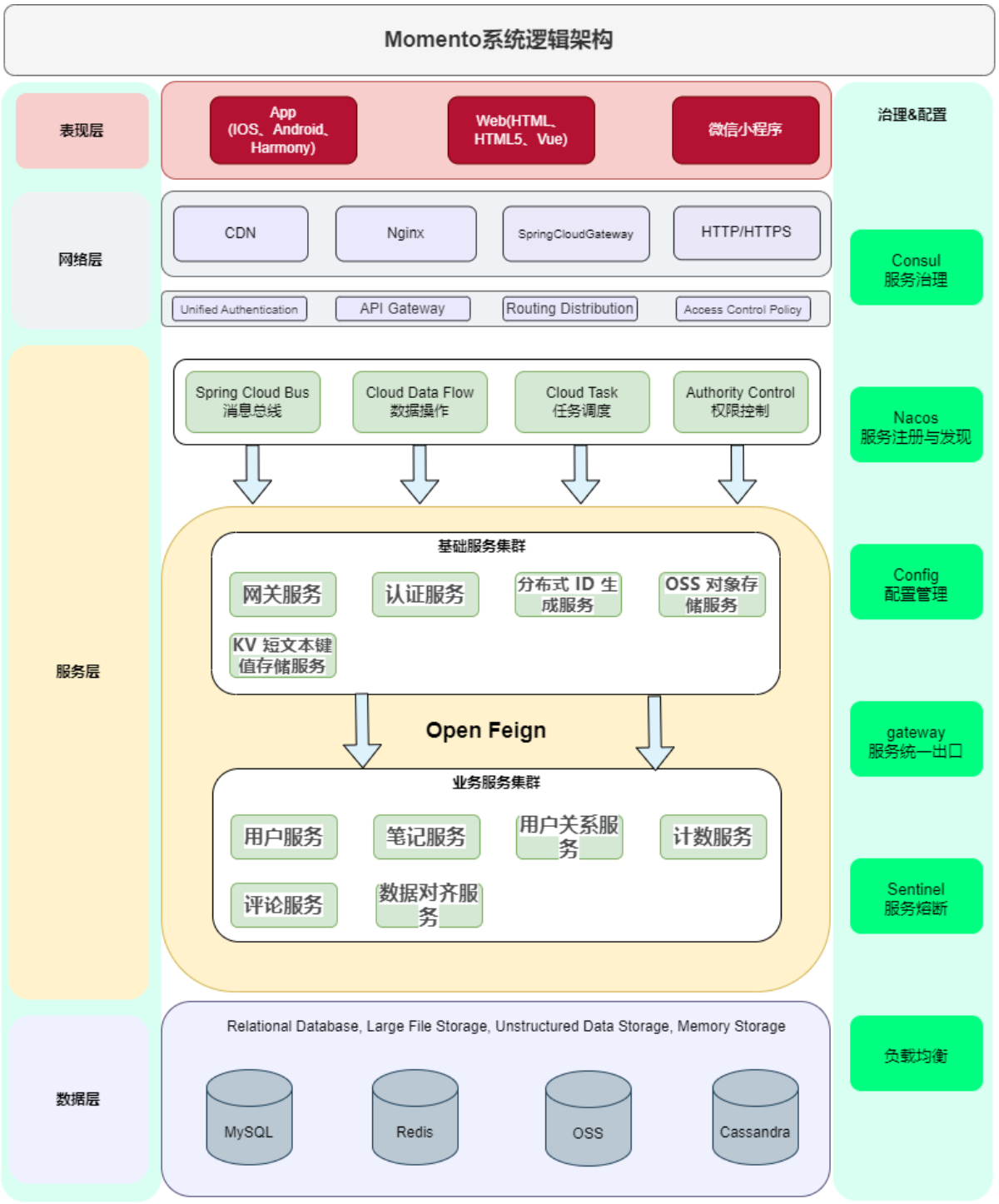
2.7 性能与可靠性考量

- **水平扩展**：所有微服务都可以根据实际负载情况进行横向扩展，从而适应不断增长的用户基数。
- **容错与恢复**：通过分布式事务、消息队列等方式保证即使在部分节点故障的情况下也能维持正常的服务。
- **高效的缓存策略**：对于频繁访问的数据采用了多级缓存机制，既加快了响应速度又降低了主数据库的压力。
- **安全性保障**：从用户身份验证到敏感数据保护都做了充分的设计，确保用户信息安全。
- **持续集成与部署**：借助 CI/CD 工具链实现快速迭代开发，保证新功能及时上线的同时不影响现有用户体验。

3. 架构描述

3.1 整体架构设计

Momento 平台的架构采用**微服务架构**，结合分布式存储和消息队列等技术，确保系统具有高可用性、可扩展性和高性能。整体架构由四个层次组成：客户端、网关层、服务层和数据层，系统逻辑架构图如下：



客户端层包括Web端和移动端（iOS与Android）。Web端使用Vue.js开发，移动端使用Flutter开发，确保在不同设备上提供一致的用户体验。客户端主要负责用户界面展示和交互逻辑的实现，例如笔记浏览、发布、点赞、收藏等操作。所有用户请求通过统一接口与后端服务交互。

网关层通过Spring Cloud Gateway实现，作为系统的统一入口。它负责路由转发、身份认证、流量控制以及负载均衡。网关不仅屏蔽了服务间的复杂交互逻辑，还提供了流量隔离和熔断保护功能，从而提高系统的安全性和稳定性。在网关之前，Nginx作为第一层负载均衡器，对用户请求进行分发，进一步提升了系统的并发处理能力。

服务层包括多个独立的微服务模块，例如用户管理服务、笔记管理服务、互动服务、计数服务等。这些服务按照业务功能进行拆分，采用Spring Cloud Alibaba构建，彼此之间通过HTTP RESTful API或消息队列（RocketMQ）进行通信。服务层还支持动态扩容，确保在高并发场景下保持稳定运行。此外，为了解决跨服务数据一致性的问题，服务层采用分布式事务框架Seata，保障分布式操作的可靠性。

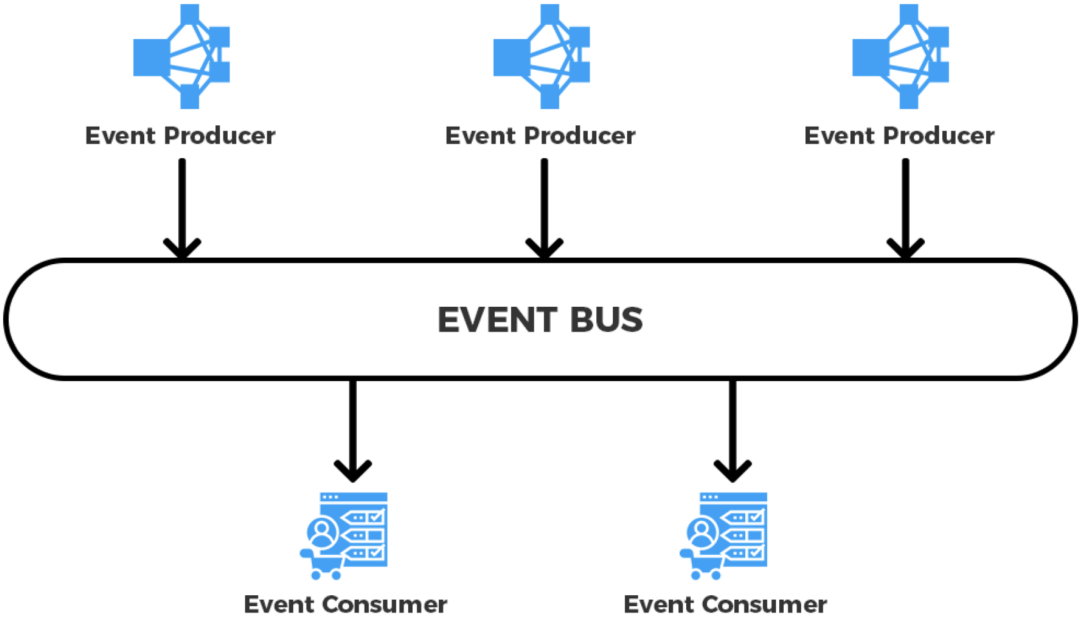
数据层承担了系统的数据存储和处理任务。结构化数据存储在MySQL中，使用分库分表策略以分散读写压力。Redis作为缓存层，存储热点数据（如用户信息、笔记数据），减少数据库的访问频率，提升响应速度。对象存储采用MinIO保存用户上传的图片和视频文件，支持高效的文件分块和快速访问。同时，系统通过log4j进行日志管理与可视化分析，为问题排查和性能优化提供支持。

3.2 架构风格

3.2.1 微服务架构

Momento 平台采用微服务架构，通过 Spring Cloud Alibaba 构建系统的服务框架。各业务功能（如用户管理、笔记管理、互动处理、计数服务）被拆分为独立的微服务，服务间通过 RESTful API 或 gRPC 进行通信。这种架构设计确保了服务的独立性，每个服务可以独立部署、升级和扩展，降低了系统耦合性。此外，微服务架构便于快速定位问题，提升了系统的可维护性和容错能力。

3.2.2 事件驱动架构



Momento 平台采用事件驱动架构处理高并发的用户操作。通过引入 RocketMQ 消息队列，系统实现了异步任务处理和服务解耦。例如，当用户点赞笔记时，点赞事件会被推送至消息队列，计数服务异步消费该事件并更新点赞数。这种架构不仅提高了系统的吞吐量，还确保了任务处理的可靠性和稳定性，避免了服务间的直接调用对性能的影响。

3.2.3 分层架构

系统采用分层架构，将系统划分为**表示层、网关层、业务逻辑层和数据访问层**。表示层通过基于 Flutter 的前端与用户交互，Nginx 实现反向代理和负载均衡网关层使用 Spring Cloud Gateway处理认证、流量控制和动态路由业务逻辑层以独立微服务形式运行，处理核心功能数据访问层整合 MySQL、Redis、Cassandra 和 MinIO，优化不同数据类型的存储与访问效率。该设计实现了关注点分离，各层独立运行，便于开发和优化，同时通过缓存和 API Gateway 等机制增强系统灵活性和性能。

3.2.4 分布式架构

Momento 平台通过分布式架构支持大规模用户访问需求。系统利用 Nacos 实现服务注册与发现，确保服务动态扩展和负载均衡。分布式事务框架 Seata 保障跨服务操作的数据一致性，尤其在多服务协作场景中，能够有效避免数据不一致问题。此外，数据库采用分库分表策略，结合 Redis 缓存热点数据，显著提升了高并发场景下的读写性能。

3.3 技术栈应用

3.3.1 前端

- **技术栈**：**Vue**
- **功能**：提供用户注册、登录、发布笔记、点赞、收藏、关注等功能的用户界面。通过RESTful API与后端服务交互，展示动态内容，确保流畅的用户体验。
- **职责**：负责用户界面的展示和用户交互，通过调用后端API获取和提交数据。

3.3.2 API Gateway

- **技术栈**：**Spring Cloud Gateway**
- 功能：
  - **统一入口**：所有客户端请求通过API Gateway进行路由和处理。
  - **路由管理**：根据请求路径将请求转发到相应的微服务。
  - **负载均衡**：实现对后端服务的负载均衡，提升系统的并发处理能力。
  - **鉴权**：集成**SaToken**进行统一的身份验证和授权，确保只有合法用户才能访问受保护的资源。
  - **限流与熔断**：防止恶意请求和高并发请求对后端服务造成冲击，确保系统稳定运行。
  - **监控与日志**：记录所有请求的日志，便于后续的监控和故障排查。

3.3.3 后端服务

Authentication Service（认证服务）

- **技术栈**：**Spring Boot**, **Spring Security**, **SaToken**
- 功能：
  - 用户注册与登录（支持密码登录和验证码登录）
  - 用户认证（生成和校验JWT Token）
  - 密码重置
- **存储**：用户认证信息存储在**MySQL**中，登录状态缓存于**Redis**中。

User Service（用户服务）

- **技术栈**：**Spring Boot**, **MyBatis**, **Redis**
- 功能：
  - 用户信息管理（查询、修改个人信息、更新密码）
  - 用户关系管理（关注与取关）
- **存储**：用户信息存储在**MySQL**中，用户关系和部分高频访问数据缓存于**Redis**中。

Note Service（笔记服务）

- **技术栈**：**Spring Boot**, **MyBatis**, **Redis**, **Cassandra**
- 功能：
  - 笔记发布、编辑、删除
  - 笔记浏览与查询（支持标签、关键词过滤）
  - 笔记置顶与权限设置

- **存储**：笔记索引存储在**MySQL**中，笔记内容存储在**Cassandra**中，媒体文件存储在**Minio**对象存储中，热点数据缓存于**Redis**中。

Interaction Service（互动服务）

- **技术栈**：**Spring Boot**, **Redis**, **RocketMQ**
- 功能：
  - 点赞、收藏操作的处理（高并发写入）
  - 关注与取关操作的处理
  - 用户互动数据的缓存与异步落库
- **存储**：互动数据主要存储在**Redis**中，通过**RocketMQ**实现异步落库至**MySQL**。

Count Service（计数服务）

- **技术栈**：**Spring Boot**, **Redis**, **MySQL**
- 功能：
  - 统计用户的关注数、粉丝数、点赞数、收藏数等
  - 统计笔记的点赞数、评论数、收藏数等
  - 提供实时统计数据接口
- **存储**：统计数据存储在**Redis**和**MySQL**中，确保高并发下的高性能。

3.3.4 数据存储

- **MySQL**：
  - 用于存储结构化数据，包括用户信息、笔记信息、互动数据等。
  - 采用分库分表策略（如基于用户ID分片）以支持高并发和海量数据存储。
- **Redis**：
  - 作为缓存层，缓存热点数据（如用户信息、笔记列表、点赞数、收藏数等），减少数据库访问压力。
  - 使用Redis的高性能数据结构（如Hash、Set、ZSet）优化数据存取效率。
- **Cassandra**：
  - 用于存储高写入吞吐量的数据，如用户操作日志、评论日志等。
  - 提供高可用性和无单点故障的数据存储方案。

3.3.5 对象存储

- **Minio**：
  - 用于存储笔记中的媒体文件（图片、视频等），提供高可用、可扩展的对象存储服务。
  - 支持多种存储策略，通过工厂模式和策略模式实现存储扩展性（如支持阿里云OSS、Minio等不同类型）。

3.3.6 消息队列

- **RocketMQ**：
  - 用于处理异步任务和解耦系统模块，如用户互动操作（点赞、收藏、关注等）的异步处理。
  - 支持高吞吐量和高可靠性的消息传递，确保消息的顺序性和幂等性。

3.3.7 缓存设计

- **Redis**：
  - 缓存热点数据，提高数据读取速度，减少数据库压力。
  - 使用二级缓存策略（本地缓存 + Redis）提高数据访问效率。
  - 实现缓存穿透、缓存击穿和缓存雪崩的防护机制，如使用Bloom过滤器、互斥锁等技术。

3.3.8 安全性与高可用性设计

- **安全性**：
  - **认证与授权**：通过**SaToken**和**JWT**实现用户身份认证和权限管理，确保只有合法用户才能访问受保护的资源。
- **高可用性**：
  - **负载均衡**：通过**API Gateway**和**Nginx**实现请求的均衡分配，提升系统的并发处理能力。
  - **分布式协调**：使用**Nacos**和**Zookeeper**进行服务发现、配置管理和分布式锁，保证系统的一致性和协调性。

3.3.9 监控与日志

- **日志管理**：ELK Stack（Elasticsearch, Logstash, Kibana）：
  - **Elasticsearch**：存储和索引日志数据，支持快速搜索和分析。
  - **Logstash**：收集、处理和转发日志数据。
  - **Kibana**：可视化日志数据，提供实时监控和分析界面。
- **系统监控**：Prometheus + Grafana：
  - **Prometheus**：收集和存储系统的性能指标，如CPU使用率、内存使用情况、网络流量等。
  - **Grafana**：基于Prometheus的数据源，提供实时监控仪表盘，帮助运维团队实时了解系统健康状况。

3.3.10. 容器化与部署

- **Docker**：
  - 所有微服务均容器化部署，确保环境一致性和部署的便捷性。
  - 使用**Docker Compose**进行容器编排和管理，实现服务的自动化部署、扩展和管理。
- **CI/CD**：
  - 集成持续集成/持续部署工具（GitLab CI），实现代码的自动化测试、构建和部署，提高开发效率和代码质量。

3.4 模块设计

3.4.1 用户管理模块（User Service）

- 职责：
  - 处理用户注册、登录、信息查询与修改、密码更新等功能。
  - 管理用户的关注与取关关系，维护用户的社交关系链。
- 关键功能：
  - **注册与登录**：支持密码登录和验证码登录，通过**JWT Token**实现无状态认证。

    - **信息管理**：用户可以修改昵称、头像等个人信息，更新密码。
    - **关注与取关**：用户可以关注其他用户，系统维护用户的关注列表和粉丝列表。
- 数据模型：
    - **用户表（user）**：存储用户基本信息。
    - **用户计数表（user_count）**：记录用户的关注数、粉丝数、点赞数、收藏数等统计数据。
    - **关注表（follow）**：记录用户之间的关注关系。

3.4.2 认证模块（Authentication Service）

- 职责：
    - 负责用户的身份认证和授权，生成和校验**JWT Token**。
- 关键功能：
    - **登录认证**：验证用户的登录信息，生成JWT Token。
    - **登出**：处理用户登出请求，失效JWT Token。
    - **密码重置**：通过验证码验证后，允许用户重置密码。
- 数据模型：
    - **用户表（user）**：存储用户的认证信息。

3.4.3 笔记管理模块（Note Service）

- 职责：
    - 处理笔记的创建、编辑、删除、查询等操作，管理笔记的内容和媒体文件。
- 关键功能：
    - **发布与编辑**：用户可以发布新的笔记或编辑已有笔记，支持文本、图片、视频等多媒体内容。
    - **浏览与查询**：用户可以浏览笔记列表，支持按标签、关键词、作者等条件过滤和查询。
    - **置顶与权限**：用户可以将笔记置顶，设置笔记的可见性权限（公开、私密等）。
- 数据模型：
    - **笔记表（note）**：存储笔记的基本信息和内容。
    - **笔记计数表（note_count）**：记录笔记的点赞数、收藏数、评论数、浏览数等统计数据。

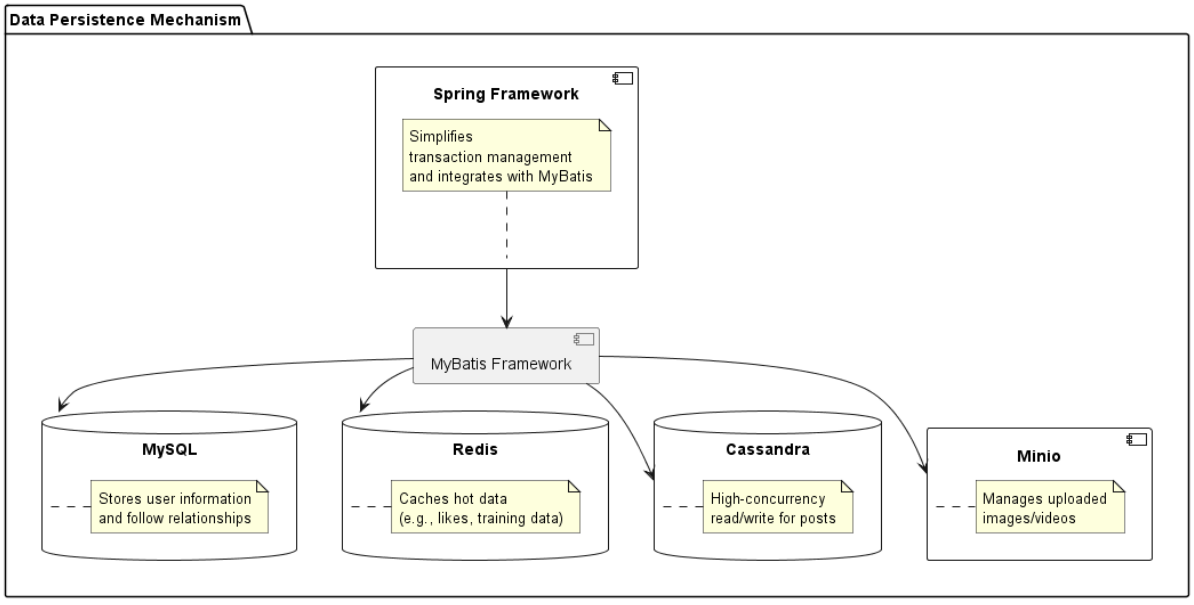
3.4.4 互动模块（Interaction Service）

- 职责：
    - 处理用户的互动操作，如点赞、收藏、关注与取关等，维护互动数据的一致性和高并发处理能力。
- 关键功能：
    - **点赞**：用户可以对笔记进行点赞，系统实时更新点赞数。
    - **收藏**：用户可以将喜欢的笔记收藏，系统实时更新收藏数。
    - **关注与取关**：用户可以关注或取关其他用户，系统维护关注关系和粉丝关系。
- 数据模型：
    - **点赞表（like）**：记录用户对笔记的点赞行为。
    - **收藏表（favorite）**：记录用户对笔记的收藏行为。

3.4.5 统计模块（Statistics Service）

- 职责：
  - 负责统计和提供用户和笔记的互动数据，确保数据的实时性和准确性。
- 关键功能：
  - **用户互动统计**：统计用户的关注数、粉丝数、点赞数、收藏数等。
  - **笔记互动统计**：统计笔记的点赞数、收藏数、评论数、浏览数等。
  - **数据接口**：提供API接口供前端查询统计数据。
- 数据模型：
  - **用户计数表（user_count）**：记录用户的互动统计数据。
  - **笔记计数表（note_count）**：记录笔记的互动统计数据。

3.5 数据库与存储设计

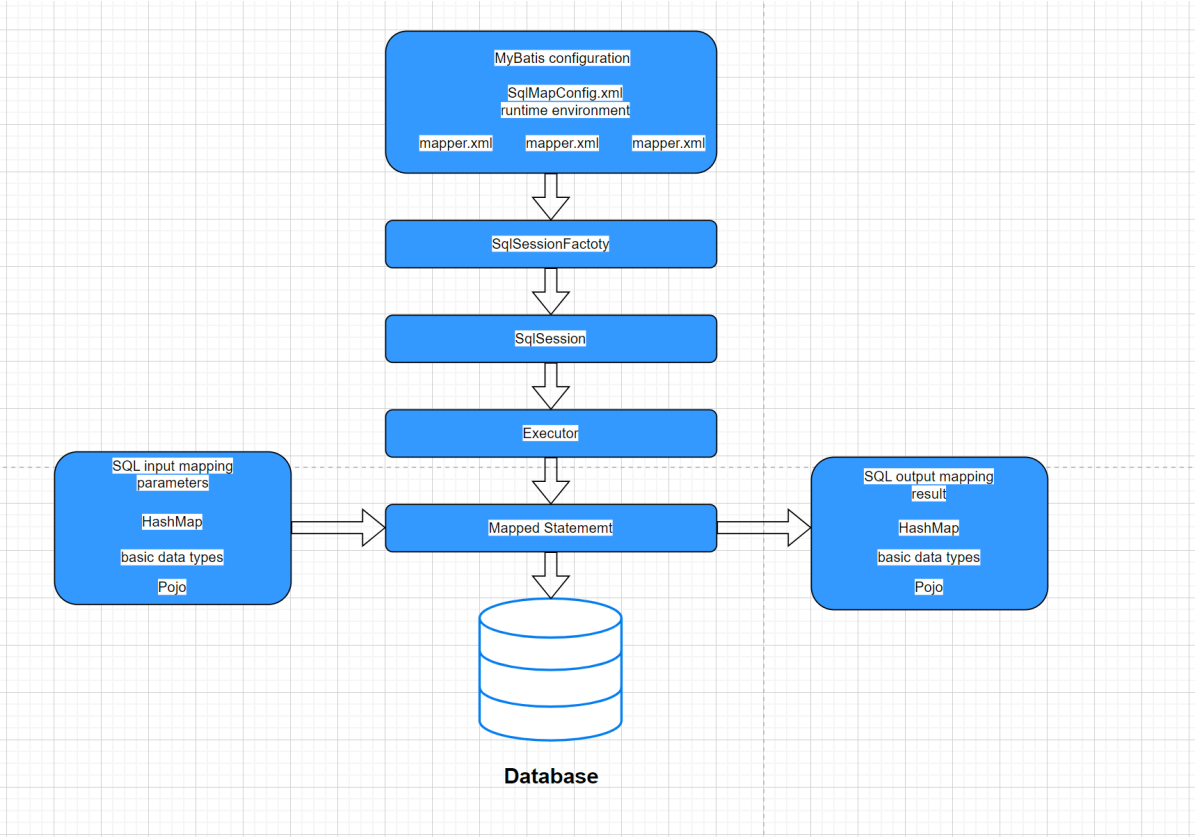


3.5.1 存储技术与应用场景

- **MySQL**
  - **存储数据**：关系型数据（例如：用户信息、用户关系）。
  - **特点**：支持复杂的事务操作以确保数据一致性；与MyBatis结合使用，具备灵活的SQL定制和映射能力。
  - **应用场景**：用户注册与登录；用户训练数据管理；用户关系管理。
- **Cassandra**
  - **存储数据**：帖子内容、评论及其他短文本数据。
  - **特点**：分布式存储，支持高并发的写入和查询操作。
  - **应用场景**：短文本数据的存储和快速查询。
- **Redis**
  - **存储数据**：高频访问的数据（例如：点赞数、帖子详情、用户信息）。
  - **特点**：超快的读写速度，适用于缓存热点数据。
  - **应用场景**：点赞和评论计数；热门帖子快速访问。
- **Minio**

- **存储数据**：用户上传的图片、视频及其他非结构化数据。
- **特点**：分布式对象存储，支持高并发的大文件存储。
- **应用场景**：帖子附件的存储与管理；用户个人资料信息的管理。

3.5.2 基于 MyBatis 的实现机制



a. 事务管理

功能

- 使用Spring的事务管理机制（`@Transactional`注解），确保多表操作或复杂业务逻辑的事务一致性，避免数据不一致问题。
- 支持嵌套事务和回滚机制，确保数据库操作的原子性。

技术特点

- **事务传播机制**：在多个服务调用中保持事务一致性，例如当用户服务调用笔记服务并共享同一事务上下文时。
- **事务回滚**：在捕获运行时异常时自动回滚事务，确保数据不会部分提交。

---

b. 缓存集成机制

功能：MyBatis支持一级（会话级）和二级（全局级）缓存，通过与分布式缓存（如Redis）集成进一步优化。

集成方式

1. **一级缓存**：
   - 默认启用，在同一数据库会话内重用查询结果，减少冗余查询。
   - 适合在同一服务方法多次调用时缓存数据。
2. **二级缓存与Redis集成**：

- 将MyBatis的二级缓存与Redis结合，实现在多个会话间的全局数据缓存：

```
<cache type="org.mybatis.caches.redis.RedisCache"/>
```

- 适用于缓存热点数据，如热门帖子和用户信息。

技术特点

- **缓存与数据一致性**：利用事务监听器将缓存更新与数据库变更同步。
- **适配场景**：非常适合高频访问的场景，平台需要快速响应时，如浏览热门帖子。

---

c. 错误处理和日志管理

功能：集成MyBatis的错误处理机制和日志功能，增强系统的调试和监控能力。

应用场景

1. **错误处理**：
   - 在系统中，如果查询失败，全球异常处理器可以捕获并记录错误：

```
@ExceptionHandler(DataAccessException.class)
public ResponseEntity<String> handleDatabaseError(DataAccessException ex) {
    logger.error("Database error: {}", ex.getMessage());
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Database error");
}
```

2. **日志管理**：
   - 使用MyBatis内置的SQL日志功能记录执行的查询及其执行时间：

```
log4j.logger.org.mybatis=DEBUG
```

技术特点

- **错误透明性**：将底层数据库错误封装为业务异常，简化系统层面的处理。
- **日志优化**：通过SQL日志分析慢查询，提升系统性能。

d. Redis 缓存集成

1. **二级缓存机制（Redis + Caffeine）**
   - 功能：
     - Redis缓存频繁访问的数据，减少数据库查询。
     - Caffeine本地缓存进一步加速数据访问。
   - 缓存策略：
     - 热数据的定期刷新。
     - 使用Bloom过滤器防止缓存穿透，并设置随机过期时间以避免缓存雪崩。
2. **Redis 示例**：
   - 实时更新点赞数：

```java
public void likePost(int postId) {
    redisTemplate.opsForValue().increment("post:likes:" + postId);
    // 异步同步到数据库
    rocketMQTemplate.convertAndSend("likeSync", postId);
}
```

3.5.3 历史数据管理

管理需求

1. **存储增长问题**：随着用户和笔记数量增长，历史数据对存储和查询性能构成挑战。
2. **数据访问频率**：活跃用户的近期数据访问频率高于历史数据。

设计方案

1. **冷热数据分离**：
   - 热数据：存储近 3 个月的用户和笔记数据，支持实时查询。
   - 冷数据：超过 3 个月的数据定期迁移至归档数据库（Cassandra）。
2. **归档策略**：
   - 每月定期将历史数据分区归档到对象存储（MinIO）。
   - 提供离线查询接口，仅在需要时加载归档数据。
3. **清理与压缩**：
   - 定期清理无用数据（如已删除笔记的相关记录）。
   - 对冷数据进行压缩存储，减少空间占用。
4. **查询优化**：
   - 查询接口根据数据存储位置（热数据、冷数据）动态选择数据源。
   - 热数据使用索引提升查询效率，冷数据查询通过分布式计算引擎（Spark）处理。

实施步骤

1. 创建数据分区策略，根据时间戳字段对数据分区。
2. 编写定期归档任务，将冷数据迁移至归档存储。
3. 优化查询逻辑，支持跨数据源查询。
4. 定期清理无效数据并压缩冷数据存储。

3.6 安全与权限管理

3.6.1 认证与授权机制

1. **Sa-Token**

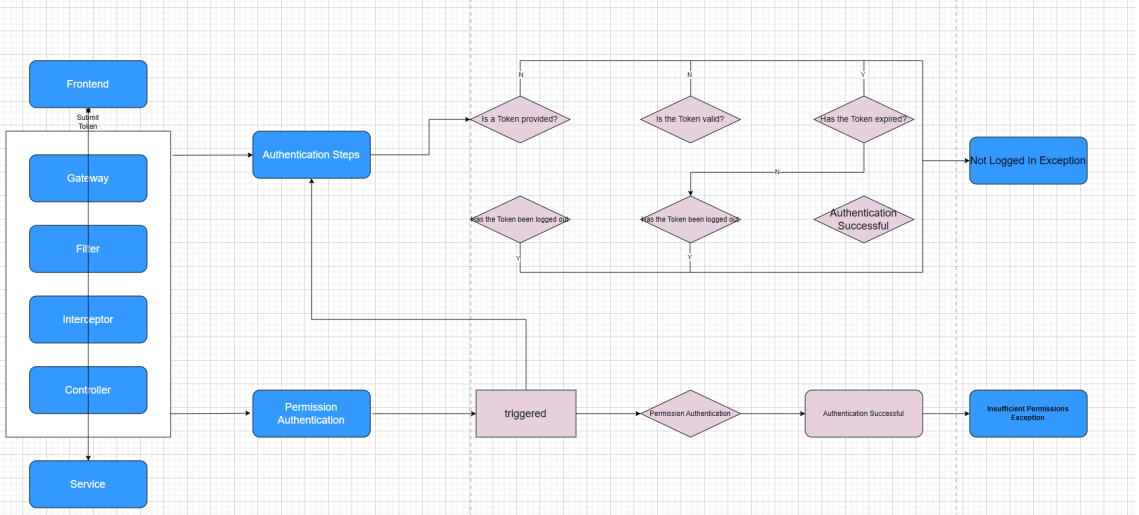
   设计理念：

   - 用户成功登录后，认证服务通过Sa-Token生成一个`Token`（可选为JWT格式），其中包含用户ID、角色信息和过期时间等数据。
   - 对于每个请求，前端将`Token`附加在HTTP头中，后端使用Sa-Token工具解析并验证用户身份。
   - Sa-Token支持会话管理和动态权限更新，非常适合系统的分布式架构。

   优点：

- 内置会话管理，支持多种场景（例如：单点登录、多设备登录）。
- 动态权限更新允许实时调整权限，无需重新登录。
- 轻量级设计，相比传统认证框架具有性能优势，简化了集成和扩展。

验证流程：



- **前端登录**：
  - 用户提交用户名和密码到认证服务。
  - 认证服务调用Sa-Token的`StpUtil.login()`方法生成`Token`：

```
StpUtil.login(userId);
String token = StpUtil.getTokenValue();
```

  - 后端将生成的`Token`返回给前端，前端将其存储在Cookie或Local Storage中。
- **前端发送带有Token的请求**：
  - 每个API请求，前端将`Token`包含在发送给后端的HTTP头中：

```
Authorization: Bearer <token>
```

- **网关解析Token**：
  - API网关使用Sa-Token验证`Token`：

```
StpUtil.checkLogin();
```

  - 验证成功后，提取用户信息（例如用户ID、角色等），并将其附加到请求中。
- **权限验证**：
  - 微服务使用Sa-Token的注解验证权限：

```
@SaCheckPermission("note:edit")
public void editNote(Note note) {
    noteMapper.update(note);
}
```

- **请求处理与响应**：
  - 目标微服务处理用户请求并将结果返回给API网关，网关将响应转发到前端。

2. **RBAC权限模型**
    - **实现机制**：
        - 基于角色的访问控制：根据角色（如普通用户、管理员）控制资源访问。
        - 权限在服务层通过Sa-Token的注解进行强制执行。

```java
@SaCheckPermission("note:edit")
public void editNote(Note note) {
    noteMapper.update(note);
}
```

3. **防刷机制**
    - 使用Sentinel配置API调用的限流，防止恶意刷接口：

```java
@SentinelResource(value = "likeApi", blockHandler = "handleBlock")
public String likePost(int postId) {
    return "Liked!";
}
public String handleBlock(int postId, BlockException ex) {
    return "Too many requests!";
}
```

3.6.2 API网关实现

1. 请求拦截与路由

- 所有请求都通过网关，网关统一处理无效请求（如缺少JWT或无效Token）。
- 解析请求后，合法请求被路由到对应的微服务。

2. 用户身份认证

- **实现机制**：
    - 将Sa-Token与内置的JWT支持集成，进行身份认证。
    - 在网关中通过中间件解析和验证JWT Token，配置认证规则：

```java
@Bean
public GatewayFilterFactory<JwtFilter> jwtFilter() {
    return new JwtFilter();
}
```

3. 限流与熔断保护

1. **动态限流：**
    - 为特定接口配置限流规则，例如设置“点赞”API每秒请求的最大数量：

```java
@SentinelResource(value = "likeApi", blockHandler = "handleBlock")
public String likePost() {
    return "Liked successfully!";
}

public String handleBlock(BlockException ex) {
    return "Too many requests, try again later!";
}
```

- **适配场景**：在社区服务中，对高频率API（如“点赞”和“评论”）进行动态流量分配，以保护后端服务。

2. **熔断机制**：
   - 配置熔断规则，在服务响应时间过长或错误率过高时触发：

```java
@SentinelResource(value = "noteApi", fallback = "fallbackResponse")
public String getNote() {
    // 调用后端服务
}

public String fallbackResponse(Throwable ex) {
    return "Service temporarily unavailable, please try again later.";
}
```

   - **适配场景**：在训练服务中，如果出现延迟过长的情况，熔断机制可以返回默认响应，避免影响用户体验。

3.6.3 系统特性与优势

1. **高性能与并发支持**：
   - 网关集中认证减少了单个服务的性能开销。
   - 限流与熔断机制在高并发场景下防止服务过载。
2. **安全性**：
   - JWT包含签名验证，确保用户身份的真实性。
   - RBAC权限模型提供了细粒度的资源访问控制。
3. **灵活性与可扩展性**：
   - 支持OAuth2.0第三方登录（如微信登录），提升用户体验。
   - API网关允许动态扩展认证和限流规则，以适应不断变化的业务需求。

3.7 系统可扩展性与高可用性设计

3.7.1 服务拆分与独立部署

- **单一职责**：每个微服务模块承担单一的业务职责，降低模块之间的耦合度，便于独立开发、测试和部署。
- **独立部署**：每个微服务可以独立部署和扩展，确保系统的灵活性和可维护性。
- **水平扩展**：通过增加微服务实例的数量，实现系统的水平扩展，满足不断增长的用户和请求量。

3.7.2 数据库分库分表

- **分库分表策略**：根据业务需求和数据访问模式，采用基于用户ID、笔记ID等字段进行分库分表，提升数据库的并发处理能力和查询性能。
- **数据一致性**：通过分布式事务管理和最终一致性机制，确保分库分表后的数据一致性和完整性。

3.7.3 缓存优化

- **热点数据缓存**：将高频访问的数据（如用户信息、笔记列表、点赞数、收藏数等）缓存于**Redis**中，减少数据库的查询压力，提升数据读取速度。
- **二级缓存策略**：结合本地缓存（如Caffeine）和Redis缓存，实现数据的快速访问和高效管理。
- **缓存失效策略**：设置合理的缓存过期时间和更新机制，确保数据的实时性和准确性。

3.7.4 负载均衡与高可用性

- **API Gateway负载均衡**：通过**Spring Cloud Gateway**实现对后端微服务的负载均衡，均匀分配请求，防止单一服务实例过载。
- **服务冗余**：每个微服务部署多个实例，确保单个实例故障不会影响整体服务的可用性。

3.7.5 消息队列与异步处理

- **解耦服务模块**：通过**RocketMQ**实现服务之间的异步通信，降低服务之间的耦合度，提升系统的灵活性。
- **高吞吐量与可靠性**：选择**RocketMQ**作为消息队列，支持高吞吐量和高可靠性的消息传递，确保关键业务消息的可靠传输和处理。
- **消息幂等性**：设计消息处理机制，确保消息的幂等性，避免重复处理导致的数据不一致。

3.7.6 监控与日志

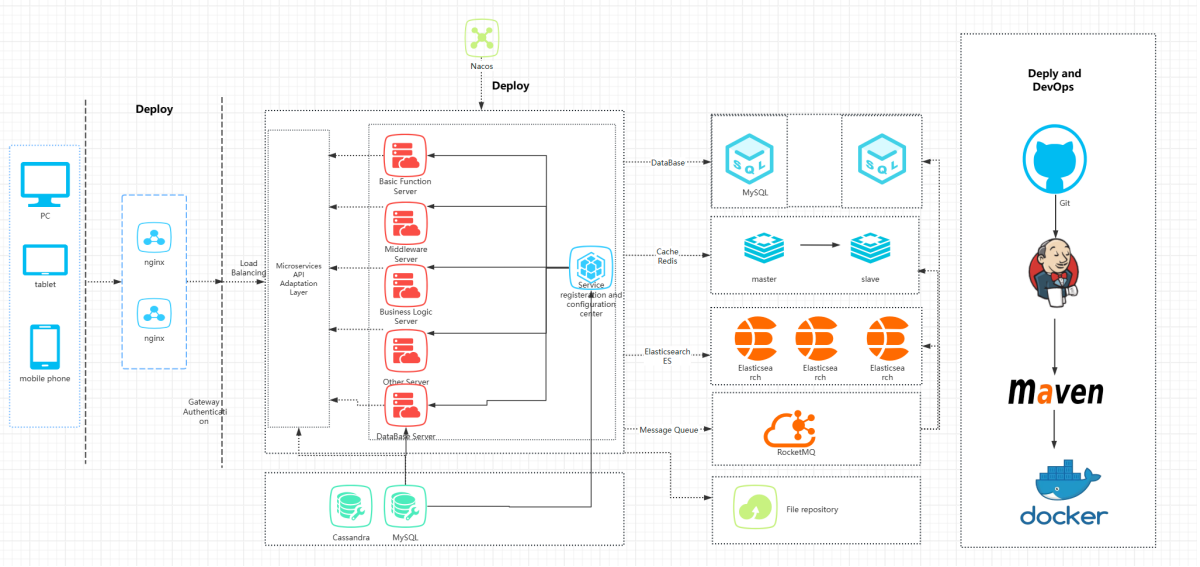
- **实时监控**：通过**Prometheus**和**Grafana**实时监控系统的性能指标和服务健康状况，及时发现和响应异常情况。
- **集中式日志管理**：采用**ELK Stack**进行日志的集中收集、存储和分析，便于故障排查和性能优化。

3.7.7 容器化与自动化部署

- **容器化部署**：使用**Docker**将各个微服务打包成容器镜像，确保开发、测试和生产环境的一致性。
- **自动化部署**：集成持续集成/持续部署（CI/CD）工具，实现代码的自动化测试、构建和部署，提升开发效率和代码质量。

4.Deploy与DevOps设计

4.1 部署架构图



4.2 Deploy

在部署方面，系统采用**Nginx**作为主要的负载均衡器，通过轮询、最少连接和IP哈希等策略，将客户端请求高效分配到后端多台服务器，确保系统的可用性和扩展性。为了提高负载均衡器的高可用性，部署了多个Nginx实例，并结合**Keepalived**实现主备切换，避免单点故障。

系统架构采用**微服务架构**，包含基础功能服务器、中间件服务器和业务逻辑服务器等多个独立服务。所有微服务通过**Nacos**进行注册和管理，支持动态服务发现和健康检查，确保只有健康的服务实例处理请求。同时，Nacos集中管理各微服务的配置，支持动态刷新和灰度发布，减少配置变更带来的停机时间。外部请求统一通过**API网关**（如Zuul或Spring Cloud Gateway）进行路由、鉴权和限流，内部服务间通信则采用HTTP/REST或gRPC协议，确保高效低延迟。

数据库与缓存方面，主要使用**MySQL**和**Cassandra**。MySQL作为关系型数据库，采用主从架构实现读写分离，支持复杂事务和查询操作；Cassandra作为分布式NoSQL数据库，适用于大规模数据和高并发读写，提供高可用性和横向扩展能力。缓存系统方面，使用**Redis**进行热点数据缓存，采用主从复制架构实现读写分离，并通过合理的缓存策略（如LRU、TTL）提高缓存命中率，减少数据库访问。

搜索与消息队列部分，采用**Elasticsearch**作为分布式搜索引擎，支持高效的全文搜索和实时数据分析，通过日志收集和数据同步机制将业务数据实时同步至Elasticsearch，确保搜索的实时性和准确性。消息队列选用**RocketMQ**，适用于订单处理、通知发送和日志收集等异步处理场景，配置了消息持久化和重试机制，确保消息的可靠传输和处理。

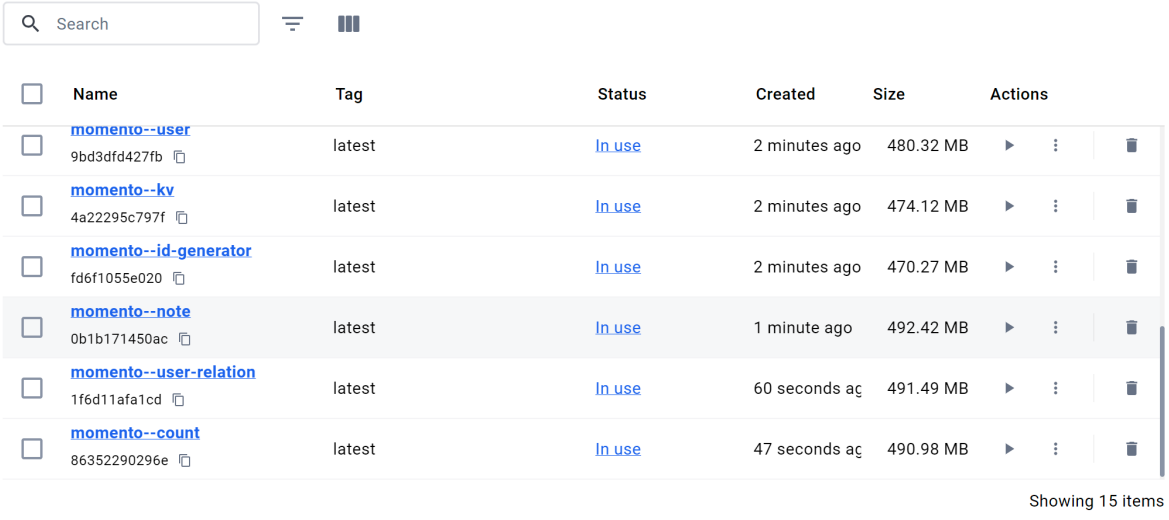
文件存储方面，采用**分布式文件系统**（如HDFS或Ceph）或**对象存储服务**（如MinIO、阿里云OSS、AWS S3）进行文件数据的存储，确保文件的安全性和高可用性。定期备份文件存储库，并制定数据恢复策略，确保在数据丢失或损坏时能够快速恢复。

Docker部署各个微服务

dockerfile与docker-compose编写示例如下：

运行docker-compose进行部署：

在docker中可以查看已经搭建好的微服务：



4.3 DevOps部分

在DevOps方面，系统使用**Git**进行代码管理，采用**GitFlow**分支模型，定义主分支、开发分支、功能分支、发布分支和热修复分支，确保代码的有序开发和发布。通过**Pull Request**机制进行代码审核和合并，使用**GitHub**、**GitLab**或**Gitee**等平台托管代码仓库，支持权限管理和持续集成功能，提升团队协作效率。

持续集成与持续部署（CI/CD）流程由 Jenkins负责自动化构建、测试和部署。Jenkins流水线自动从Git仓库拉取最新代码，使用**Maven**进行项目构建，执行单元测试和集成测试，并进行代码质量检查。构建产物打包成**Docker**镜像后上传到Docker Registry，通过**Docker Compose**或**Kubernetes**等容器编排工具将镜像部署到测试或生产环境，实现自动化部署。同时，集成**Prometheus**和**Grafana**等监控工具，实时监控构建和部署过程，及时反馈构建状态和异常情况，快速响应问题。

在**构建与容器化**方面，使用Maven管理项目的依赖和构建生命周期，优化构建过程以提升速度和稳定性。应用程序及其依赖通过Docker进行容器化，编写高效的Dockerfile优化镜像构建和启动速度，使用**Docker Registry**或其他镜像仓库（如Harbor、AWS ECR）集中管理镜像，确保镜像的安全性和可管理性。采用**Kubernetes**作为主要的容器编排平台，实现容器的自动部署、扩展和管理，配置**Horizontal Pod Autoscaler**根据负载自动调整容器实例，集成**Prometheus**和**ELK Stack**进行监控和日志管理，确保系统的弹性和可维护性。

5. 挑战与未决问题

5.1 系统挑战

1. 分布式ID生成的性能与可靠性

- **性能与吞吐量**：随着系统请求量的增加，ID生成服务需要能够处理大量并发请求，而不会造成瓶颈或延迟。
- **ID唯一性**：在分布式环境中，如何保证生成的ID是唯一的，避免因网络延迟或节点失效导致重复ID的生成。
- **容错处理**：在ID生成服务发生故障或宕机时，系统如何保证依然能够正常工作，避免影响业务流程，尤其是在ID生成服务不可用的情况下。
- **时钟回拨问题**：分布式ID生成通常依赖系统时间戳，时钟回拨可能会导致生成重复或无序的ID，如何处理系统时间不一致的情况，以确保生成的ID按时间递增且无重复。

2. 数据一致性问题

- **Redis与MySQL数据一致性**：由于使用了Redis作为缓存，如何确保缓存与数据库中的数据保持一致，特别是在更新缓存和数据库时的同步问题。高并发情况下，如何防止缓存不一致带来的脏数据问题。
- **分布式事务处理**：在多个微服务之间进行数据操作时，如何保证分布式事务的原子性和一致性，避免在分布式环境中出现事务回滚失败或数据不一致的情况。
- **高并发下的数据竞争**：在多个请求并发处理时，如何有效避免数据竞争，特别是在涉及到复杂业务逻辑时，如何确保操作的顺序性和数据的准确性。

3. 微服务通信的可靠性

- **服务间调用的超时处理**：在分布式系统中，服务之间的调用可能会因为网络延迟、负载过高等原因发生超时，如何有效处理服务超时，避免对业务流程造成影响。
- **服务降级与熔断策略**：当某些服务不可用或出现异常时，如何快速做出响应，采取服务降级或熔断策略，确保系统整体可用性。
- **网络分区容错处理**：在网络出现分区时，如何保证服务之间的通信不中断，确保系统能够在一定程度上容忍网络分区，并继续保持部分功能的正常运行。
- **服务注册与发现的可靠性**：使用Spring Cloud进行服务注册与发现时，如何确保服务注册中心的可靠性，防止由于服务实例丢失或无法被发现导致服务无法正常调用。

4. 消息队列的可靠性

- **消息丢失处理**：如何确保消息在传递过程中不丢失，特别是在网络不稳定或服务宕机时，确保消息能够可靠地到达目标服务。
- **消息重复消费**：在分布式环境中，如何避免同一消息被多个消费者重复消费，导致数据冗余或业务逻辑出错。
- **消息顺序性保障**：对于一些需要保证严格顺序的业务场景，如何确保消息在队列中的顺序不被打乱，避免因并发处理导致的业务异常。
- **大流量下的性能保障**：在高并发、高流量的场景下，如何保证消息队列的吞吐量和延迟，避免消息积压或处理瓶颈，确保系统的高效性。

5. 系统待优化的问题

- **缓存策略优化**：如何优化缓存的使用，包括缓存预热、缓存穿透、缓存击穿、缓存雪崩等问题的防护，以及如何制定合理的缓存更新策略，确保缓存的高效性和一致性。
- **性能优化**：包括SQL查询优化、服务间调用的性能提升、大数据量下分页查询的性能优化等。如何减少数据库的查询压力，提升系统响应速度，避免高并发环境下的性能瓶颈。
- **监控与告警体系**：需要建立全面的服务健康检查机制、性能监控、业务异常监控以及资源使用监控，确保能够及时发现系统故障和性能瓶颈，做出预警和处理。
- **安全性增强**：加强系统的安全性，包括接口防刷、数据脱敏、权限细粒度控制、敏感操作审计等，防止恶意攻击或非法操作，保护用户数据安全。

5.2 未决问题：内容检索与搜索服务

5.2.1 服务概述

内容检索与搜索服务专注于融合传统信息检索与人工智能语义分析技术。该模块旨在实现高效、精准的笔记搜索与推荐功能。通过引入深度学习模型，突破了传统关键词匹配的局限，为用户提供基于语义理解的智能化搜索体验

5.2.2 AI 搜索的技术方案与资源分析

1. 功能描述

- **语义搜索**：支持用户自然语言查询，提升检索精准度和用户体验。
- **智能问答**：通过上下文理解生成与用户意图高度匹配的结果。
- **内容推荐**：结合语义匹配与用户行为数据挖掘，推送个性化笔记内容。

2. 技术实现方案

- **语义向量生成与存储**
  - 使用预训练 NLP 模型（如 BERT 或 GPT），将笔记内容、标题等转化为高维语义向量。
  - 采用 Milvus 或 FAISS 向量存储引擎：
    - **Milvus**：提供分布式存储支持，适用于大规模数据场景。
    - **FAISS**：以高效的向量相似度计算和 GPU 加速能力著称，显著提升检索速度。
- **查询向量化与匹配**

  用户查询通过 NLP 模型生成语义向量，与存储的向量进行相似度计算（如余弦相似度），返回语义最相关的笔记内容。
- **多模型协作优化**

  结合关键词搜索与语义检索结果，采用加权融合策略提高排序精准度。对预训练模型进行领域微调，增强其对专业术语与上下文的理解能力。

3. 资源需求与代价分析

- **计算成本**
  - 语义向量生成与实时匹配需要 GPU 支持，特别是在大规模数据场景下需部署高性能计算资源。
  - 通过离线批量生成语义向量与增量更新策略降低实时推理需求：
    - **离线生成**：预处理静态数据，减少在线计算负担。
    - **增量更新**：快速处理新增或修改数据并更新索引，实现动态响应。
- **存储成本**
  - 高维语义向量的存储需求显著高于传统关键词索引。
  - 采用分片与层次化存储策略，以平衡存储成本与检索性能。
- **开发周期**
  - 集成 NLP 模型与向量检索引擎预计需 4-6 周。
  - 模型微调与优化需根据业务需求增加 2-3 周。

4. 性能优化方向

- **实时性保障**：利用 Redis 等缓存存储高频查询结果，减少延迟。采用粗筛-精筛两阶段检索策略，先过滤低相关性结果，再精准匹配高相关性内容。
- **扩展性提升**：水平扩展向量检索节点，支持高并发访问。优化检索算法，实现动态负载均衡。
- **模型迭代与更新**：结合用户反馈进行有监督微调，提升模型对多样化查询的适应能力。

5.2.3 数据同步与一致性管理

使用 Canal 捕获 MySQL 数据库的增量更新，实时同步至 Elasticsearch 和向量存储引擎。保障新增、修改、删除操作在检索服务中的一致性。

实现方案

1. **增量同步**：解析 binlog 日志，实现对数据库变更的轻量化监听。
2. **更新策略**：批量更新低频数据，实时同步高频数据，平衡系统负载与实时性。

5.2.4 挑战与解决方案

1. 高并发与性能瓶颈

- **挑战**：GPU 推理响应速度可能成为大规模用户查询场景下的瓶颈。
- **解决方案**：
  - 引入多级缓存：
    1. 高频查询存储于内存缓存（如 Redis）中，确保低延迟响应。
    2. 热点但变化不频繁的查询存储于磁盘缓存，降低内存占用。
    3. 通过缓存过期策略与访问频率监控动态调整缓存内容。
  - 模型蒸馏与轻量化优化，减少推理计算成本。

2. 系统复杂性与维护成本

- **挑战**：多存储层（Elasticsearch 与向量存储）与模型服务的协同管理增加系统复杂度。
- **解决方案**：
  - 构建统一的监控与日志分析系统，实时监控关键模块。
  - 引入自动化部署与版本控制工具，提升维护效率。

3. 隐私与安全保障

- **挑战**：用户搜索历史和行为数据涉及敏感信息。
- **解决方案**：
  - 应用差分隐私技术保护用户行为数据。
  - 通过加密传输与分级访问控制机制，确保数据传输与存储安全。

6.团队反思

成员	贡献	反思
2253551 李沅衡	- 负责系统的设计和项目代码框架的搭建，确保成员之间的职责明确。- 负责**基础服务**和**计数服务** 的设计与实现，技术服务包括高并发环境下的计数策略、数据对齐服务和最终一致性保障。- 设计并实现了数据对齐服务，定期校正 Redis 和数据库中计数数据的差异。	- 在处理高并发计数时，虽然采用了 Redis 缓存和异步写入策略，但在极高并发下，如何进一步优化高频计数的更新逻辑仍然是一个挑战。 - 数据对齐服务的效率和性能优化仍需改进。
2250763 李俊旻	- 负责 用户相关的业务服务需求分析与设计，包括用户服务、用户关系服务等核心模块的需求分析和整体设计。- 确定了用户注册、登录、信息查询和更新等功能的业务流程	- 在需求分析和设计阶段，虽然已经明确了业务逻辑和接口规范，但在细化实现过程中，如何处理用户关系和权限控制的复杂性仍然需要进一步探讨。- 面对高并发时，如何保证用户数据的一致性和及时性，尤其是在用户行为（如关注、取关）频繁变化的情况下，仍需在性能和数据一致性之间找到平衡
2254272 赵子毅	-负责笔记相关的业务服务需求 - 负责 **DevOps** 配置与管理，包括容器化部署、CI/CD 流水线优化和基础设施自动化管理。 - 配置负载均衡与自动化扩展，优化系统在流量高峰期的稳定性。	- 容器化部署和 CI/CD 流水线虽然已实现，但在高并发场景下，如何进一步提升系统的弹性和高可用性是亟待解决的问题。 - 监控和日志系统的细粒度管理仍需优化，提升自动化运维能力。

团队总结

协作与沟通：
团队成员之间的沟通顺畅，分工明确，工作协调得当，确保了项目按时推进。大家的技能互补，有效推动了项目的多个模块并行开发。

技术挑战与解决方案：
项目中的主要技术挑战包括 **计数服务** 的高并发处理、**增量同步** 和 **搜索服务** 的查询优化等。通过 Redis 缓存、消息队列、Elasticsearch 的增量同步机制，我们初步解决了部分问题，但仍需进一步优化和调整。

未来展望：
下一步，团队将集中精力设计 **内容搜索与检索服务**，解决查询性能问题，同时进一步提升 **计数服务** 的处理能力，确保系统在高并发和大数据量场景下的稳定性。同时，我们也将加强 **用户服务模块** 的开发，特别是处理复杂用户关系和高并发访问的方案设计。