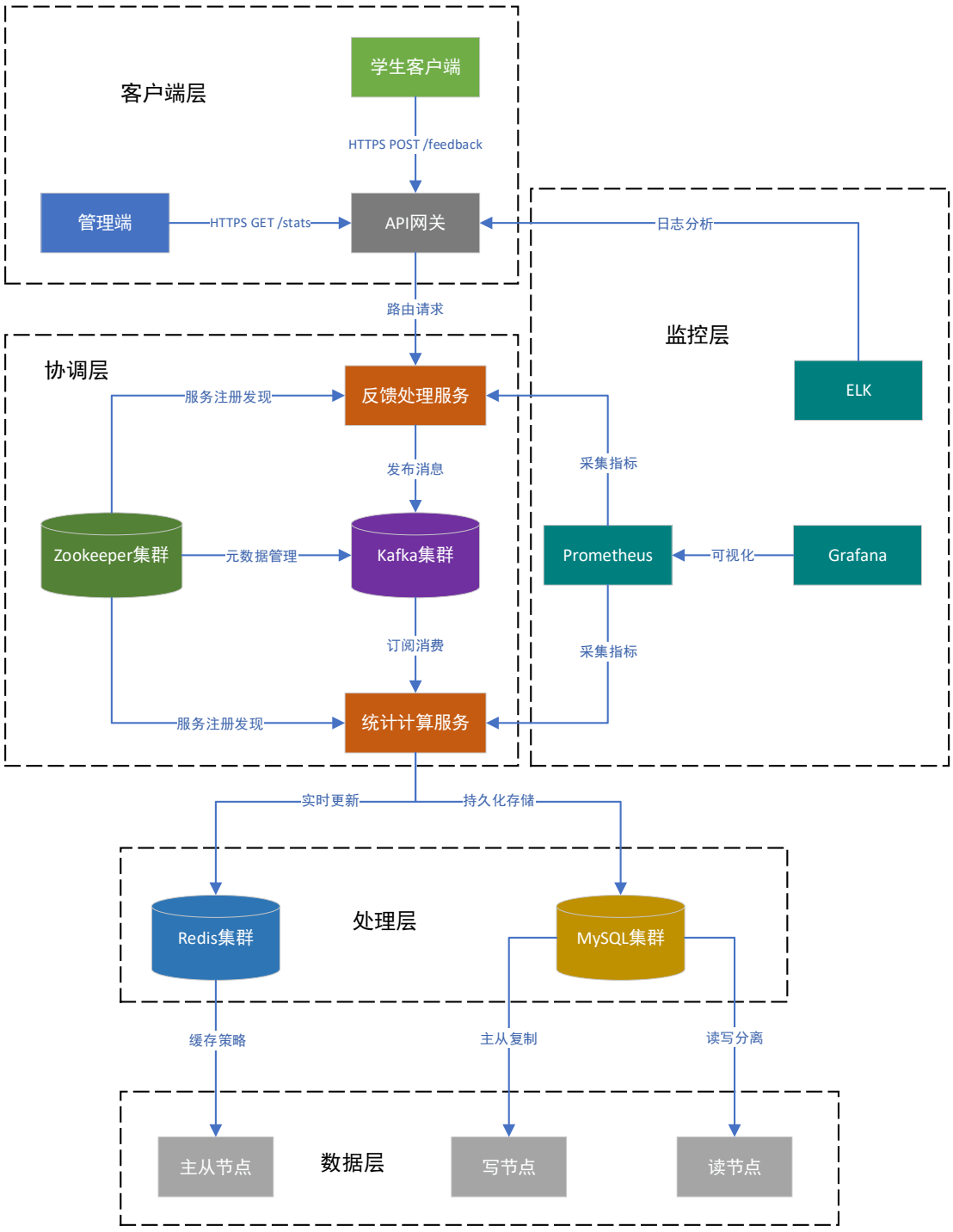
课程学习反馈系统

一、架构设计



二、组件作用

1、消息队列（Kafka）

异步削峰：消息队列作为系统的缓冲层，能够接收高并发的反馈请求，避免服务过载。当学生提交反馈时，消息队列会暂时存储这些请求，等待后续处理。

数据持久化：消息队列会将消息持久化到磁盘，确保即使系统发生故障，数据也不会丢失。Kafka 默认保留 7 天的消息，便于故障恢复和数据分析。

顺序保证：Kafka 通过分区机制保证同一课程的评价消息按顺序处理，避免数据错乱。

2、Zookeeper 集群

Kafka 元数据管理：Zookeeper 负责管理 Kafka 集群的元数据，包括 Broker 列表、Topic 配置、消费者偏移量等，确保 Kafka 的高可用性和一致性。

服务注册发现：Zookeeper 还用于微服务的注册与发现，反馈处理服务和统计计算服务会将自己的地址注册到 Zookeeper，API 网关通过 Zookeeper 动态获取服务地址。

分布式锁：Zookeeper 提供分布式锁功能，确保在并发场景下统计计算的原子性。

3、Redis 集群

实时统计存储：Redis 用于存储课程的实时统计数据，如总评分、评价次数和平均分。这些数据通过原子操作（如 HINCRBY）实时更新，确保统计结果的准确性。

缓存策略：Redis 作为缓存层，能够显著减少对数据库的查询压力。热点课程数据会被主动预热到 Redis 中，并通过动态 TTL（如 24 小时过期）管理缓存的生命周期。

4、MySQL 数据库

持久化存储：MySQL 用于长期存储学生的反馈数据，包括评分、评价内容和时间戳等。这些数据可以用于历史查询、审计和离线分析。

高可用性：通过主从复制和读写分离，MySQL 能够提供高可用的数据服务。写操作由主库处理，读操作由从库处理，确保系统的高性能和容错能力。

三、流程说明

1、学生提交反馈

学生通过 Web/App 客户端提交反馈（评分和评价内容），请求通过 HTTPS 发送到 API 网关。

2、API 网关处理

API 网关对请求进行身份验证（JWT）、限流（令牌桶算法）和路由分发，将请求转发给反馈处理服务。

3、反馈处理服务

反馈处理服务对数据进行校验（如评分范围、敏感词过滤），并将合法的反馈数据封装为 JSON 格式的消息，发送到 Kafka 的 feedback-events Topic 中。

Kafka 返回 ACK 确认消息已接收，反馈处理服务向 API 网关返回 202 Accepted，表示请求已成功接收。

4、统计计算服务

统计计算服务从 Kafka 消费消息，解析出课程 ID 和评分，然后通过 Redis 的原子操作（HINCRBY）更新课程的总评分和评价次数。

统计计算服务还会将原始反馈数据批量写入 MySQL 数据库（每 100 条数据批量插入一次），确保数据的持久化。

5、管理端查询统计

当管理端需要查看某课程的统计信息时，API 网关将请求路由到查询服务。

查询服务首先尝试从 Redis 中获取课程的统计信息（如总评分、评价次数）。如果缓存命中，直接返回结果；如果缓存未命中，则从 MySQL 中查询并更新 Redis 缓存。

四、异常处理

1、消息队列故障（Kafka 宕机）

本地磁盘队列：反馈处理服务启用本地磁盘队列，暂存无法发送到 Kafka 的消息。

故障转移：Zookeeper 触发 Kafka Broker 的故障转移，确保集群的高可用性。

恢复后处理：Kafka 恢复后，优先消费积压的消息，确保数据不丢失。

2、Redis 缓存失效

降级查询：查询服务直接访问 MySQL 数据库，获取统计信息，牺牲一定的性能保证系统的可用性。

临时单节点 Redis：启动临时单节点 Redis 接管流量，确保缓存服务不中断。

数据重建：通过 MySQL 的全量+增量同步机制，重建 Redis 缓存。

3、数据库连接异常（MySQL 主库宕机）

主从切换：通过 MHA（Master High Availability）工具自动将从库提升为新主库，确保写操作不中断。

暂停批量写入：统计计算服务暂停批量写入操作，避免数据不一致。

数据同步：主库修复后，通过增量同步机制将缺失的数据同步到主库。

4、Zookeeper 失联

静态配置：服务切换到静态配置模式，使用本地配置文件获取 Kafka 和服务的地址。

本地元数据缓存：Kafka 启用本地元数据缓存，确保在 Zookeeper 失联期间仍能正常运行。

限制新节点加入：在 Zookeeper 恢复之前，禁止新节点加入集群，避免配置不一致。

五、性能优化

1、Kafka 调优

提高吞吐量：通过调整 num.io.threads 和 log.flush.interval.messages 等参数，提升 Kafka 的处理能力。

分区策略：按 course_id 哈希分区，确保同一课程的消息顺序性。

2、Redis 热点处理

使用 Lua 脚本保证原子操作，避免并发问题。

热点数据主动预热，确保高并发场景下的缓存命中率。

3、数据库优化

添加覆盖索引：如 CREATE INDEX idx_course_time ON feedback (course_id, created_at)，提升查询性能。

批量插入：通过批量插入（如每 100 条数据插入一次）减少数据库的写入压力。