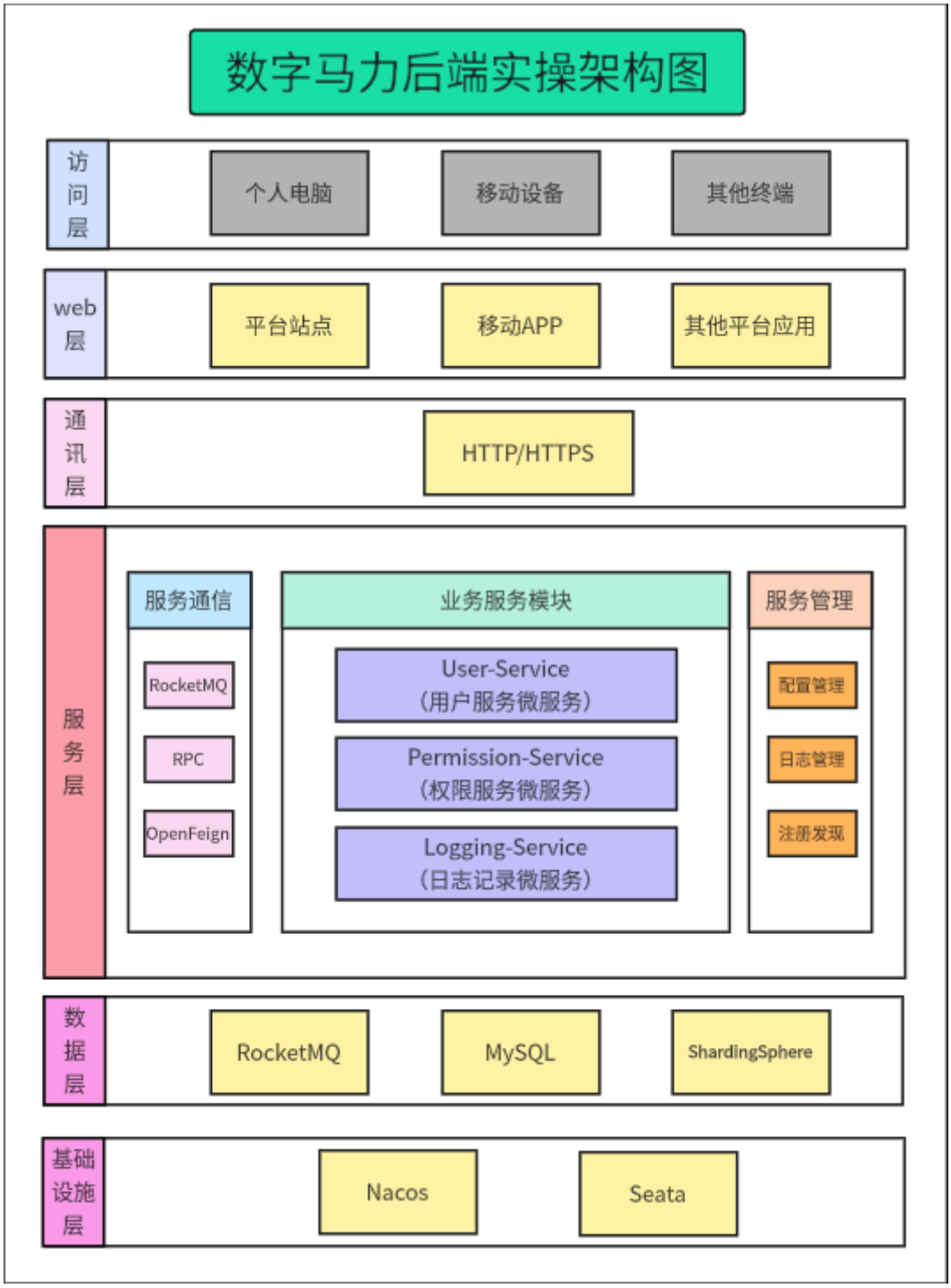
数字马力后端实操大题设计文档

1.基本概述：

本系统采用微服务架构，由三个独立的微服务组成，包括用户服务（user-service），权限服务（permission-service）和日志服务（logging-service）。用户服务和权限在nacos上进行注册之后可以相互通信，而为了进行异步并且持久化操作，日志服务可以通过RocketQM消息队列进行与用户服务的通信，从而使得在功能上实现高内聚，低耦合的特点。三个微服务相互独立且高度内聚。在**用户服务**中我完成了对用户的一些增删改查操作，其中包括用户的注册，登录，查询，修改密码等一系列操作。而在**权限服务**中我围绕者用户的一系列权限功能来写，其中包括在用户注册的时候通过用户id来进行角色的初始化（为user），根据用户的id来查询用户的权限（user，admin，super_admin），并且super_admin可以对用户权限发生改变。在**日志服务**中我通过RocketMQ消息队列将用户服务中的一些列操作进行日志记录，使得程序可以运行的更快并且使日志模块更加独立，方便日后对其他模块的日志管理。

2.系统架构图



3.微服务设计分块概述

3.1用户服务（user-service)

3.1.1功能简述:

用户服务负责处理用户的注册、登录、信息查询、更新以及密码重置等操作，同时支持用户信息的分页查询。

3.1.2架构设计:

- **控制器层（Controller）**：处理用户请求，接收并验证请求参数，调用服务层方法处理业务逻辑。
- **服务层（Service）**：实现具体的业务逻辑，如用户注册、登录、信息查询和更新等。
- **数据访问层（Mapper）**：负责与数据库进行交互，执行数据的增删改查操作。

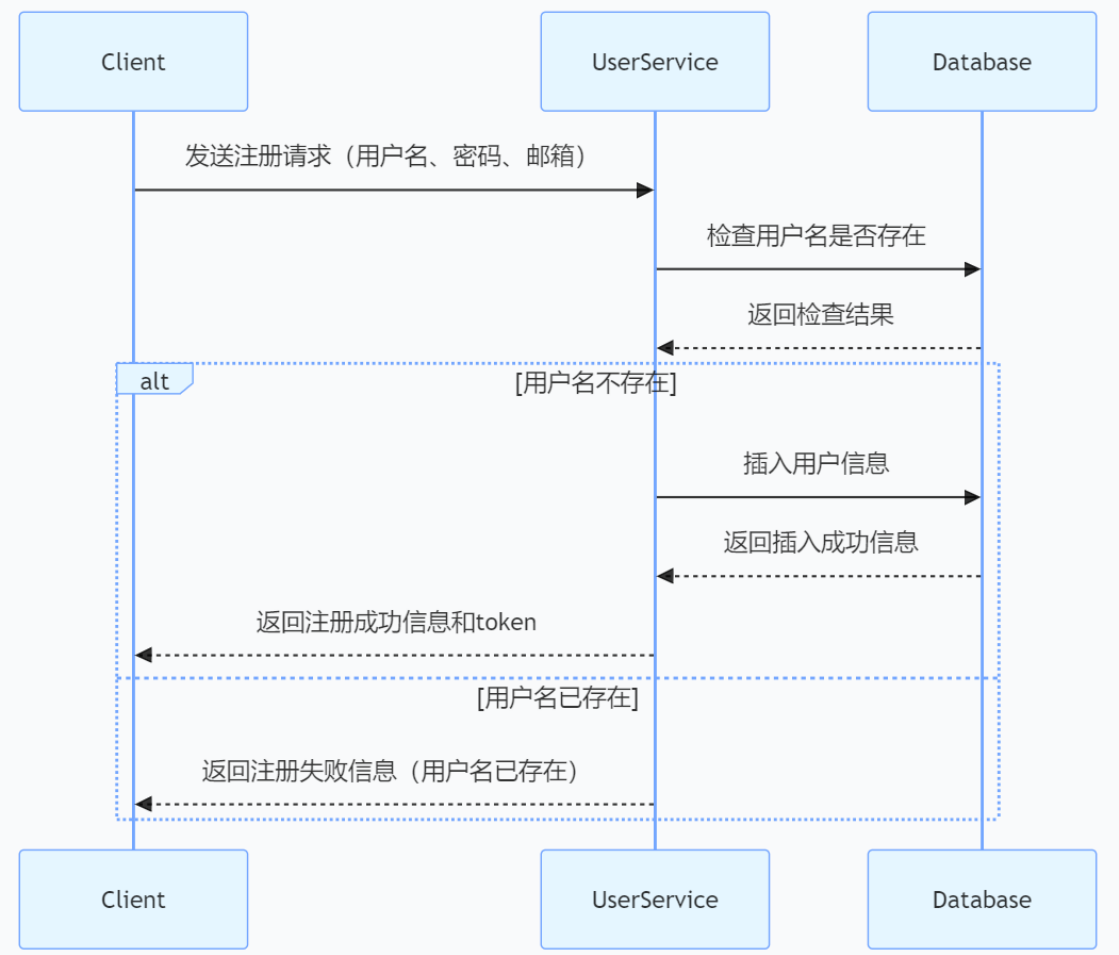
- **配置类（Config）**：配置 MyBatis-Plus 的拦截器，实现分页功能。

3.1.3实体类设计：

- **User:**用户类，与数据库通过MyBatis-Plus进行连接访问，其中userId是用户的id，会通过学法算法生成分布式id，username是用户的名字，password是用户的密码，其中利用了md5加密使得在数据中无法直接的看出用户的密码，更加的安全，email是用户的邮箱，phone是用户的手机号，gmtCreate是用户的注册的时间，而salt是盐，对密码进行加盐的md5加密操作
- **UserInfo:**这个实体类主要是用于响应信息的反馈，当用户进行操作的时候，会反馈给这三条数据，token是用户的密令，在我构造token的时候，Claim中put了用户的id，这样用户的id信息会保存在请求头中，可以在其他的服务中判断用户当前状态（已经登录未超时，已经登录超时，未登录）
- **LogEvent:**与日志相关的服务所建立的日志类，其中userId是进行本次操作的用户的id，action是本次操作的行为，比如登录是LOGIN，注册是REGISTER等，ip是用户进行本次操作所在的ip，detail是用户进行操作的详细，比如A更改了自己的密码从1234abcd改为了2345sdfg。
- **PageResult:**用于分页操作的封装，其中有两个信息，分别为目标页码（pageNo）也一页的数据数量（pageSize）。
- **R:**响应结果实体类，分别包含三个数据，分别是状态码code，反馈消息message，反馈的数据data，并且我在这个类中定义了若干个静态方法，使得更方便进行调用。

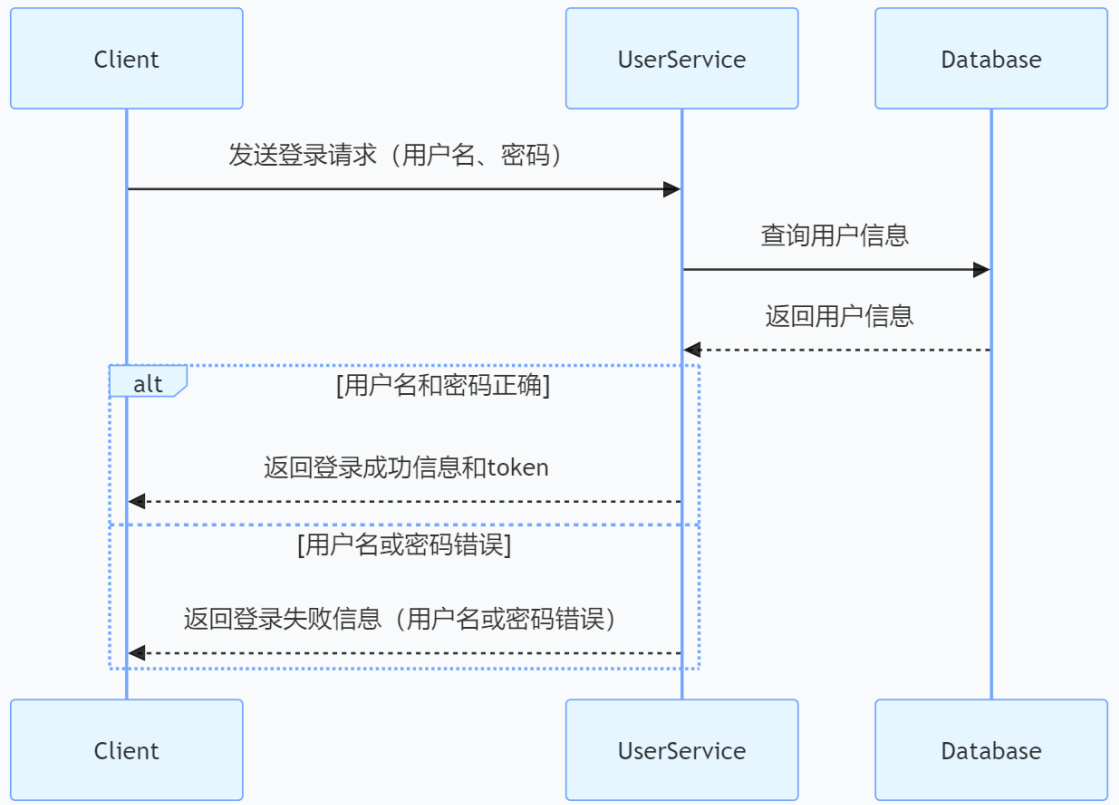
3.1.4接口设计：

- **register**：用户进行注册操作的接口。当用户进行注册操作的时候，首先会进行用户名的检测，如果在数据库中可以查出注册人的用户名，那么注册失败，返回的结果是用户名已经存在，如果在数据库中可以查出注册人的用户名，那么注册失败，返回的结果是用户名已经存在，加密之后写入数据库，并且调用权限服务绑定默认角色绑定成功之后发送日制到消息堆列完成注册操作。



- **login**：用户进行登录操作的接口。当用户进行登录操作的时候，首先会从请求头中获取token来判断用户是否是登录状态，如果是登录状态会报"错用户已经登录"。如果用户不是登录状态会检测用户输入用户名以及密码，并且分别在数据库中查询用户名与用户密码，如果二者不匹配或者不存

在返回401，当用户成功登录的时候会生成一个jwt令牌，在请求头中缓存，方便以后的检测操作。



- **getUser**：用户查询的接口。当用户进行查询操作的时候，首先也会判断用户是否是登录状态，如果用户不是在登录状态则无法进行下一步操作，之后会进行对目标用户的查询，如果该目标用户不存在会"返回用户不存在"，之后会分别判断查询角色和被查询角色的人物权限根据要求返回查询，结果以及是否越级查询。
- **updateUser**：用户更改信息的接口。当用户想更改信息的时候，同样也会判断是否在登录的状态下，如果未登录则无法进行下一步操作，之后会判断目标用户是否存在，如果不存在则退出查询，如果存在进行下一步对查询者和被查询者的权限比较，之后对被修改者的信息进行修改。
- **resetPassword**：用户重置密码的接口。在进行密码重置的操作的时候，同样会检测是否在登录状态下，如果用户在登录状态下会判断用户的权限，如果用户是普通用户只会重置他自己的密码，如果该用户是管理员则会重置非超级管理员所有人的密码，如果该用户是超级管理员那么就会重置所有人包括超级管理员的密码。
- **selectPage**：用户进行分页查询的接口。在进行分页查询的时候，同样会判断用户的登录状态，如果该用户已登录则会判断用户的权限，如果用户的权限是普通用户则只会查询到自己的信息，如果是管理员则会查询所有普通用户以及管理员的信息，如果是超级管理员则会查询所有人的信息。

接口名称	请求方法	请求路径	请求参数	响应参数	描述
用户注册	POST	/users/register	RegisterRequest	R	用户注册接口
用户登录	POST	/users/login	username, password	R	用户登录接口
根据 ID 获取用户信息	GET	/users/{userId}	userId	R	根据用户 ID 获取用户信息
更新用户信息	PUT	/users/{userId}	userId, User	R	更新用户信息
密码重置	POST	/users/resetPassword	无	R	密码重置接口
分页查询用户信息	POST	/users	PageResult	R<IPage>	分页查询用户信息

3.1.5工具类设计：

- **JwtUtil**：该工具类为生成JWT令牌以及检测用户登录状态的类。其中我设定了token的有效期为一天，加密的使用普通字符串密钥进行加密，最小的刷新间隔是三百秒，在我生成token的时候，我在claim中传递了id， 之后用户可以对id进行查询。在检测用户登录状态的时候存在着四个状态，第一个状态是有效，第二个状态是有效，但需要刷新第三个状态是过期，第四个状态是异常。当用户是有效的时候可以进行下一步操作，如果临期过期则需要刷新。
- **AuthorizeFilter**：该工具类会配合着JwtUtil使用，其中会检测用户是否登录以及返回用户id以及token 三个方法，其中获取token是从请求头中进行获取。

3.1.6消息队列生产者类设计：

- **LogProducer**：简单的注入了`RocketMQTemplate`进行消息队列的操作。

3.1.7权限服务RPC接口：

- **bindDefaultRole**：绑定用户ID并且初始化默认权限user。
- **getUserRoleCode**：获取用户的状态码（user，admin，super_admin）。
- **upgradeToAdmin**：（超管调用）升级用户为普通管理员
- **downgradeToUser**：（超管调用）降级用户为普通角色。

3.1.8分库分表逻辑：

在用户注册的时候我对用户表进行了水平分表的操作：

- 首先我在pom中配置了shardingsphere的相关依赖。
- 之后我在数据库`user_db`中创建了两个user表，分别为`users_1`与`users_2`
- 我在`application.yml`中配置了分表逻辑如下：

```
shardingsphere:
  datasource:
    names: ds0
    ds0:
      type: com.alibaba.druid.pool.DruidDataSource
      driver-class-name: com.mysql.cj.jdbc.Driver
```

```
        url: jdbc:mysql://localhost:3306/users_db?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf8&useSSL=false
        username: root
        password: 123456
    rules:
      sharding:
        tables:
          users:
            actual-data-nodes: ds0.users_$->{1..2}
            table-strategy:
              standard:
                sharding-column: user_id
                sharding-algorithm-name: users-inline
        sharding-algorithms:
          users-inline:
            type: INLINE
            props:
              algorithm-expression: users_$->{user_id % 2 + 1}
    props:
      sql-show: true
```

- 其中datasource.names是我给数据库起的别名
- ds0是我配置的数据库的相关源
- rules是分库分表的规则，这里我要求按照user_id进行分表，如果user_id%2==0则会分到1表，反之会分到2表。

3.1.9nacos配置

这里我仅仅对该微服务进行了注册操作，其`application.yml`配置如下：

```
  application:
    name: userservice
  cloud:
    nacos:
      server-addr: localhost:8848
```

3.1.10rocketMQ配置

其配置文件如下：

```
rocketmq:
  name-server: 127.0.0.1:9876
  producer:
    group: producer-group
    send-message-timeout: 3000
    retry-times-when-send-failed: 2
    max-message-size: 4194304
```

- name-server是命名服务器的地址，这里由于我是本地部署于是地址为这个
- producer.group是生产者组的名称，这里我就简单的命名为producer-group
- producer.send-message-timeout是消息发送超时的配置，如果在3s中为未发送到broker则发送失败
- producer.retry-times-when-send-failed是尝试发送次数，我这里写了两次
- producer.max-message-size是消息最大可以为多少，这里我写了最多可以发送4MB。

在我的配置中体现了消息可靠性的保障策略，因为如果3s没用发送到并且两次都失败了那么就说明消息可能有问题，并且我消息发送的都是一些简短的日志，如果出现大于4MB的情况那么肯定是存在问题的！

3.1.11分布式事务配置

这里我使用了XA而非AT，因为在我的项目中AT与某些依赖存在冲突，于是我使用了XA分布式事务而非AT，其配置如下：

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>5.2.1</version>
    <exclusions>
        <exclusion>
            <artifactId>transactions-jdbc</artifactId>
            <groupId>com.atomikos</groupId>
        </exclusion>
        <exclusion>
            <artifactId>transactions-jta</artifactId>
            <groupId>com.atomikos</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

3.2权限服务（permission-service）

3.2.1功能简述：

权限服务负责处理用户的角色分配和权限验证，为其他服务提供角色查询和权限验证的接口。

3.2.2架构设计:

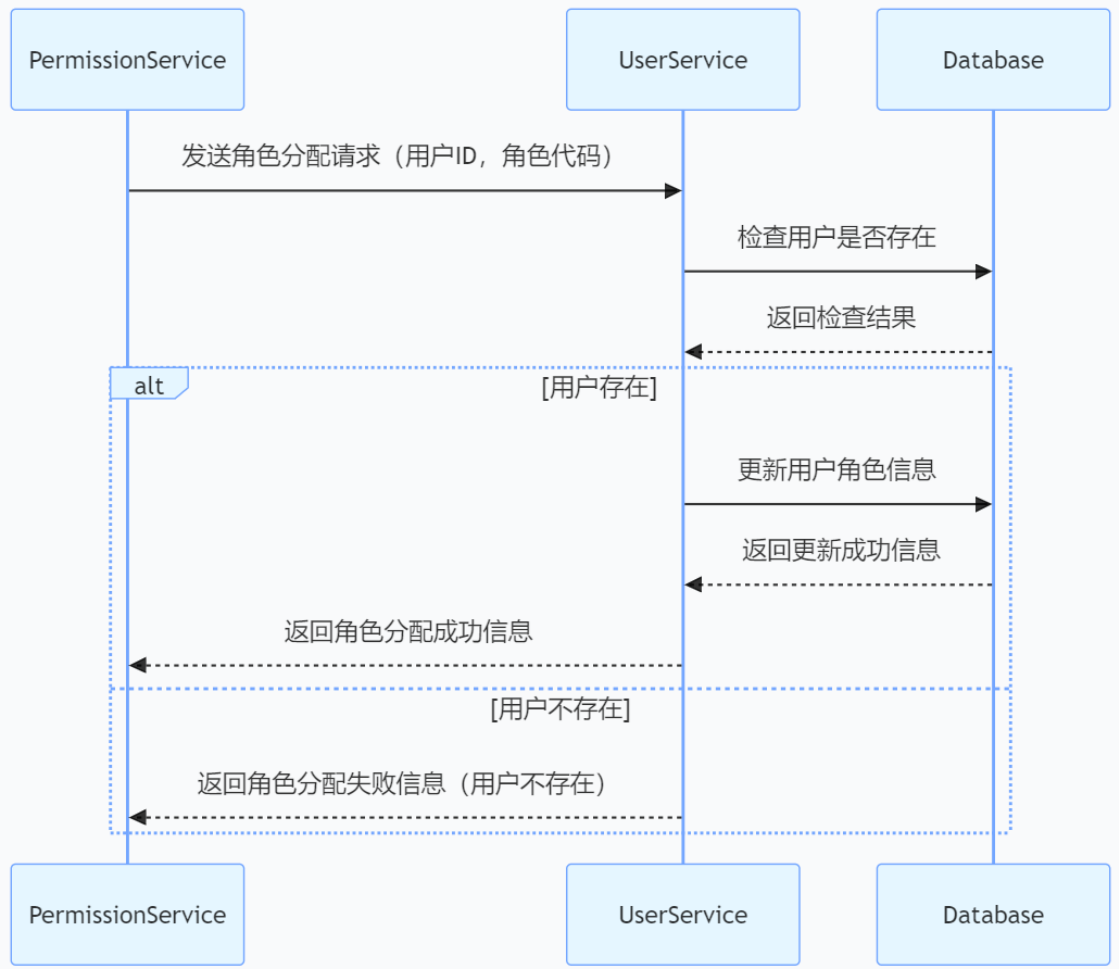
- **控制器层（Controller）**：处理权限相关的请求，如角色查询、权限验证等。
- **服务层（Service）**：实现具体的权限业务逻辑，如角色分配、权限验证等。
- **数据访问层（Mapper）**：负责与数据库中的角色表和用户角色关联表进行交互。

3.2.3实体类设计：

- **Role:**与数据库`permisson_db`中的`roles`表进行关联。其中roleId是权限id，为1，2，3。roleCode是用户的状态码分别为user，admin，super_admin。1对应的是super_admin，2对应的是user，3对应的是admin。
- **UserRole:**与数据库`permisson_db`中的`user_roles`表进行关联，其中id是存储的编号，通过分布式雪花算法进行分配，userId是用户的id，而rold是权限对应的id。

3.2.4接口设计：

- **bindDefaultRole**：初始化用户权限的接口。在这里我直接创建了一个UserRole对象，之后把roleid设置成2插入到了表中。



- **getUserRoleCode**：获取用户权限的接口。首先我会判断在数据库中是否存在对应的Id，如果不存在则未查询到用户信息，反之查出他的Id之后可以获取他的权限。
- **upgradeToAdmin**：升级用户的接口。在这个方法中，我也先获取他的用户Id 如果无法在数据库中查到则该用户不存在，如果该用户的状态码是3则说明该用户已经是管理员，反之我可以对这个进行更新操作。
- **downgradeToUser**：降级管理员的接口。同上。
- **getAllUser**：获取所有userid的接口。这个方法可以返回所有权限为user的id，可以在进行分页查询的时候快速获取信息（其实可以直接进行一个缓存，但是感觉有点麻烦没有必要所以没写）。

以上的接口全部是RPC接口，因为我的permission已经在nacos上成功注册，因此可以使用这些接口。

接口名称	请求方法	请求路径	请求参数	响应参数	描述
获取用户角色代码	GET	/permissions/userRoleCode/{userId}	userId	String	根据用户 ID 获取用户的角色代码
绑定默认角色	POST	/permissions/bindDefaultRole/{userId}	userId	无	为用户绑定默认角色
降级用户为普通角色	Put	/permissions/downgradeToUser/{userId}	userId	无	降级用户为普通角色
升级用户为管理员	Put	/permissions/upgradeToAdmin/{userId}	userId	无	升级用户为管理员
获取所有用户 ID	GET	/permissions/allUserIds	无	List	获取系统中所有用户的 ID

3.2.5nacos配置：

```yaml
application:
  name: permissionservice
cloud:
  nacos:
    server-addr: localhost:8848
```

3.3日志服务（loggin-service）

3.3.1功能简述：

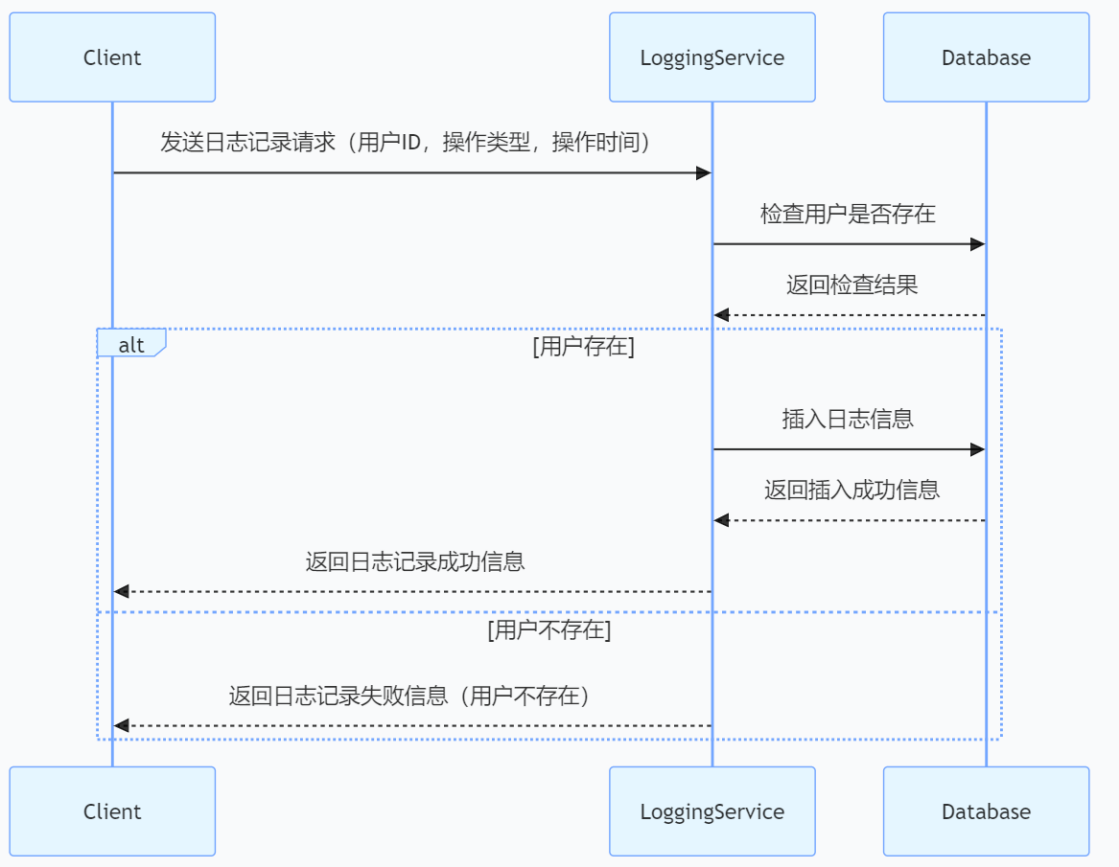
日志服务负责接收和处理系统中的操作日志，将日志信息存储到数据库或消息队列中。

3.3.2 架构设计：

- **控制器层（Controller）**：处理日志相关的请求，如日志记录、查询等。
- **服务层（Service）**：实现具体的日志业务逻辑，如日志记录、查询等。
- **数据访问层（Mapper）**：负责与数据库中的日志表进行交互。

3.3.3实体类设计：

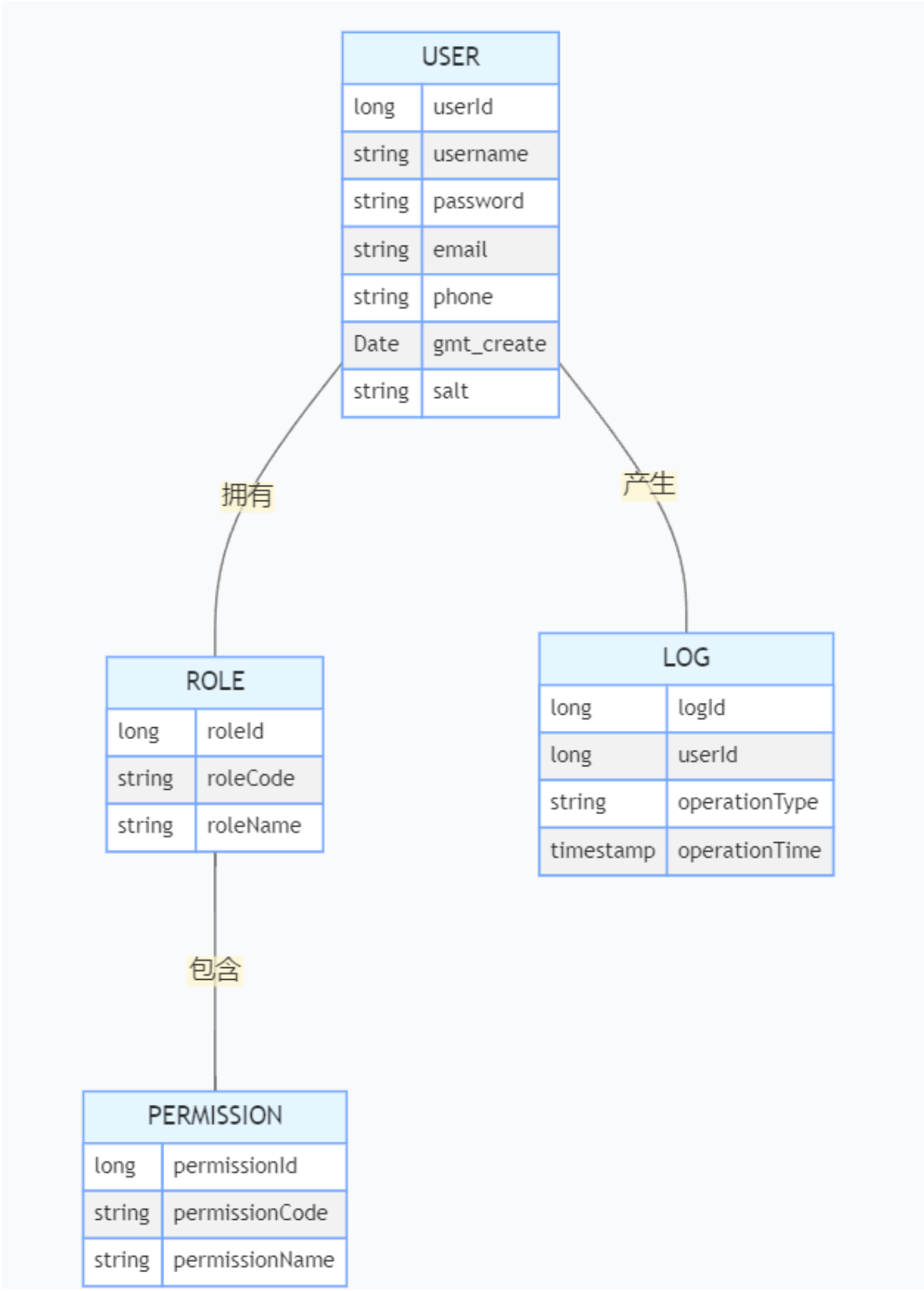
- **LogEvent:**该实体类与数据库`logs_db`中`operation_logs`进行绑定，的与日志相关的服务所建立的日志类，其中userId是进行本次操作的用户的id，action是本次操作的行为，比如登录是LOGIN，注册是REGISTER等，ip是用户进行本次操作所在的ip，detail是用户进行操作的详细，比如A更改了自己的密码从1234abcd改为了2345sdfg。



3.3.4消息队列消费者类设计：

- **LogConsumer**：启动监听服务，当Broker中有消息的时候该类会自动获取这个消息，写入数据库并且打印到控制台。

4.数据库设计



5.基础环境配置

组件	版本	部署方式	启动步骤	访问地址	描述
MySQL	8.0.36	通过 Navicat Premium 启动	点击即可		根据用户ID获取用户的角色代码
Nacos	2.2.0	本机部署	打开文件夹下的bin目录的cmd输入startup.cmd -m standalone 即可	localhost:8848	进行微服务的注册以及配置
RocketMQ	5.1.2	本机部署	打开文件夹下的bin目录的cmd依次输入start mqnamesrv.cmd（启动命名服务器集群）start mqbroker.cmd -n 127.0.0.1:9876 autoCreateTopicEnable=true（启动消息服务集群）	localhost:9876	进行日志的异步存储

5.服务间通信：

本系统采用 Spring Cloud OpenFeign 实现服务间的通信。例如，用户服务（user-service）通过 Feign 客户端调用权限服务（permission-service）的接口，获取用户的角色代码和权限信息。

6.安全设计：

- 用户登录采用 JWT（JSON Web Token）进行身份验证，保证用户信息的安全性。
- 对敏感信息（如用户密码）进行加密存储，使用 MD5 加盐的方式对密码进行加密。
- 对服务间的通信进行加密，使用 HTTPS 协议保证数据传输的安全性。

7.作者信息：

- 姓名：邢凯宁
- 学校：华北水利水电大学
- 学院：信息工程学院
- 专业：软件工程
- 联系电话（微信同）：13683837515

- QQ：769472984
- 项目地址：kaining667/NewService: 数字马力实操 (github.com)