

用户管理系统

用户管理系统业务需求

1. 概述

本系统旨在通过命令行界面（CLI）实现一个简单的用户管理系统。用户可以通过命令行进行增、删、改、查操作，并且系统将用户信息持久化到数据库中。系统将采用分层架构设计，确保代码的可维护性和扩展性。

2. 功能需求

2.1 用户管理功能

- 增加用户：用户可以通过命令行输入用户信息（姓名、性别、年龄、电子邮件、电话号码），系统将生成一个唯一的用户ID，并将用户信息存储到数据库中。
- 删除用户：用户可以通过输入用户ID删除指定的用户信息。
- 修改用户：用户可以通过输入用户ID修改指定用户的姓名、性别、年龄、电子邮件、电话号码等信息。
- 查询用户：用户可以通过输入用户ID查询指定用户的详细信息，或者列出所有用户的信息。

2.2 命令行界面（CLI）

- 操作菜单：系统启动后，展示一个操作菜单，用户可以选择进行增、删、改、查操作。
- 命令格式：系统支持类似 git 或 Docker 的命令格式，用户可以通过输入命令和参数来执行相应的操作。
 - 示例命令：
 - `user add --name "John Doe" --gender Male --age 30 --email john@example.com --phone 1234567890`
 - `user delete --id 1`
 - `user update --id 1 --name "Jane Doe" --email jane@example.com`
 - `user get --id 1`
 - `user list`

2.3 数据库存储

- 数据库：使用 MySQL 作为关系型数据库。
- 表结构：在 `user_management` 数据库中创建 `users` 表，表结构如下：

```
1 CREATE TABLE users (  
2     id INT AUTO_INCREMENT PRIMARY KEY,  
3     name VARCHAR(100) NOT NULL,  
4     gender ENUM('Male', 'Female', 'Other') NOT NULL,  
5     age INT,  
6     email VARCHAR(100) UNIQUE,  
7     phone VARCHAR(15)  
8 );
```

3. 非功能需求

3.1 性能

- 系统应能够快速响应用户的操作请求，尤其是在查询和列出用户信息时，应尽量减少数据库查询时间。

3.2 安全性

- 系统应对用户输入进行验证，防止 SQL 注入等安全问题。
- 数据库连接信息应妥善保管，避免泄露。

3.3 可维护性

- 代码应具有良好的结构和注释，便于后续维护和扩展。
- 系统应采用模块化设计，确保各层之间的职责清晰。

3.4 可扩展性

- 系统应设计为易于扩展，未来可以方便地添加新的功能模块或修改现有功能。

4. 分层设计与架构

1. 分层架构：

- 将系统设计为三层架构，包括：
 - 表示层 (UI层)：负责与用户交互，通过命令行界面显示操作菜单，并接收用户输入。
 - 业务逻辑层 (Service层)：负责处理业务逻辑，如验证输入、执行增、删、改、查操作。
 - 数据访问层 (DAO层)：负责与数据库交互，提供增、删、改、查的数据库操作方法。

2. Repository与分层结合：

- 在 Service 层调用 Repository 类进行数据存储操作，将数据访问逻辑与业务逻辑分开，提高系统的可维护性和扩展性。

3. 代码组织：

- com.example.model：定义用户模型类 (User)，包括用户属性。
- com.example.dao：实现数据库操作类 (UserRepository)，使用 JDBC 进行数据存取。
- com.example.service：实现业务逻辑层，调用 Repository 层进行增、删、改、查操作。
- com.example.ui：提供命令行交互界面，展示操作菜单，接收用户输入。

5. 测试需求 (可选-根据能力实现)

- 单元测试：对 UserRepository 和 UserService 进行单元测试，确保每个方法的功能正确。
- 集成测试：测试整个系统的功能，确保从命令行输入到数据库操作的整个流程正确无误。
- 性能测试：测试系统在高并发情况下的性能表现，确保系统能够处理大量用户请求。

6. 未来扩展

- 用户权限管理：未来可以扩展系统，增加用户权限管理功能，不同权限的用户可以执行不同的操作。
 - 日志记录：增加日志记录功能，记录用户的操作历史，便于审计和排查问题。
 - 多数据库支持：未来可以扩展系统，支持多种数据库 (如 PostgreSQL、Oracle 等)。
-

第一阶段：实现基本功能

1. 功能要求：

- 实现用户的增、删、改、查操作。
- 每个用户应包含以下信息：
 - 用户ID（唯一标识符）
 - 姓名
 - 性别
 - 年龄
 - 电子邮件
 - 电话号码

2. 命令行UI：

- 在命令行界面展示一个操作菜单，用户可选择进行增、删、改、查操作。
 - 提供用户输入信息并执行相应的操作。
-

第二阶段：数据库存储与Repository模式

1. 数据库要求：

- 使用 MySQL 或其他关系型数据库。
- 创建一个 user_management 数据库，并在其中创建一个 users 表，字段与用户信息相匹配。

2. Repository模式：

- 实现一个 Repository 类，负责与数据库交互，提供增、删、改、查等操作。
- 通过 JDBC 连接数据库，并确保数据存储与读取正确。

3. 功能增强：

- 将用户信息持久化到数据库中，确保增、删、改、查操作都能反映在数据库中。
-

第三阶段：更换底层数据库访问逻辑

目前使用JDBC，请尝试使用MyBatis和 Hibernate 实现Repository（此部分没有提供示例 - 扩展功能）。