目录

第一阶段：基础功能实现（内存存储） 2
01. 领域模型层 2
一、MyBatis Repository 的实现 27
3. 创建 MyBatis 配置文件 30
1．配置说明： 38
2．事务管理： 38
3．错误处理： 38
4．性能考虑： 38
5．使用建议： 38
二、Hibernate Repository 的实现 39
4. 使用示例 40
完整项目结构 40
1.依赖倒置 41
用户管理系统 45
用户管理系统业务需求 45
1. 概述 45
2. 功能需求 45
3. 非功能需求 46
第一阶段：实现基本功能 47
第二阶段：数据库存储与 Repository 模式 47
第三阶段：更换底层数据库访问逻辑 47
1. 创建 User 实体（model/User.java） 48
下一步：实验二的分层结构合并到此项目 55
用户管理系统接口需求文档 56
1. 项目概述 56
3.1 获取所有用户 56
3.2 创建用户 57
3.3 获取用户详情 57
3.4 更新用户 58
3.5 删除用户 58
4. 数据结构定义 58
一、实验项目需求：企业运营数据仪表盘 65
1．实时数据展示区 65
2．核心指标卡片（Ant Design Card） 65
3．图表展示（Ant Design Charts） 65
二、实现步骤（关键步骤与代码示例） 65
8. 测试验证 71
三、技术要点解析与教学知识点 71
1. 项目架构设计 71
2. 核心组件实现 72
3. 数据层关键技术 72
4. 交互与扩展能力 73
四、掌握的知识点 73
1．React 核心技术： 73
2．Ant Design 生态： 73
3．工程化实践： 73
4．数据处理能力： 73
5．UI/UX 原则： 73
五、页面组件加载活动图（Mermaid） 73
四、扩展学习建议 74
1．进阶方向： 74
2．优化方向： 74
一、项目搭建与配置 75
1．创建 React 项目： 75
二、第一阶段：Mock API 实现 75
三、第二阶段：连接真实 API 82

四、功能特点说明 85
1．用户界面： 85
2．Mock API 实现： 85
3．Ant Design 组件使用： 85
4．真实 API 集成： 85
五、项目结构 85
个人博客页面开发 - 完整实现 87
基本目标：锻炼学员基础知识的理解 120
1．首页设计 120
2．文章展示页 120
3．关于我页面 120
进阶目标：个人博客页面开发（Bootstrap + JavaScript） 120
在 Spring Boot 项目中，解决 CORS 的方法 121
目录建议： 121

第一阶段：基础功能实现（内存存储）

项目结构

```
user-management/
├── src/
│   └── main/
│       └── java/io/codescience
│           ├── model/
│           │   └── User.java
│           ├── service/
│           │   └── UserService.java
│           └── ui/
│               └── CliUI.java
└── pom.xml
```

核心代码

01. 领域模型层

```java
// User.java
package io.codescience.model;


public class User {
    private int id;
    private String name;
    private String gender;
    private int age;
    private String email;
    private String phone;

    public User(String name, String gender, int age, String email, String phone) {
        validateInput(name, "姓名");
        validateInput(gender, "性别");
        validateAge(age);
        this.name = name;
        this.gender = gender;
        this.age = age;
        this.email = email;
        this.phone = phone;
    }

    private void validateInput(String value, String field) {
        if (value == null || value.trim().isEmpty()) {
            throw new IllegalArgumentException(field + "不能为空");
        }
    }

    private void validateAge(int age) {
        if (age < 0) {
            throw new IllegalArgumentException("年龄不能为负数");
        }
    }
    // Getter/Setter methods...
}
```

02. 业务逻辑层


```java
// UserService.java
```

```java
package io.codescience.service;

import io.codescience.model.User;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class UserService {
    private Map<Integer, User> users = new HashMap<>();
    private int idCounter = 1;

    public User addUser(User user) {
        user.setId(idCounter++);
        users.put(user.getId(), user);
        return user;
    }

    public User deleteUser(int id) {
        return users.remove(id);
    }

    public User updateUser(int id, User newData) {
        User existing = users.get(id);
        if (existing != null) {
            existing.setName(newData.getName());
            existing.setGender(newData.getGender());
            existing.setAge(newData.getAge());
            existing.setEmail(newData.getEmail());
            existing.setPhone(newData.getPhone());
        }
        return existing;
    }

    public User getUser(int id) {
        return users.get(id);
    }

    public List<User> listUsers() {
        return new ArrayList<>(users.values());
    }
}
```


03. 命令行界面（CLI）

```java
// CliUI.java
```



```java
import io.codescience.model.User;
import io.codescience.service.UserService;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class CliUI {
    private final UserService service = new UserService();

    public CliUI() {
        initSampleData();
    }

    private void initSampleData() {
        // 添加示例用户数据
        User user1 = new User("张三", "男", 25, "zhangsan@example.com", "13800138001");
        User user2 = new User("李四", "女", 28, "lisi@example.com", "13800138002");
        User user3 = new User("王五", "男", 30, "wangwu@example.com", "13800138003");

        service.addUser(user1);
        service.addUser(user2);
        service.addUser(user3);

        System.out.println("示例数据初始化完成");
    }

    public void start() {
        System.out.println("用户管理系统启动（输入help查看命令帮助）");
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.print("\n> ");
            String input = scanner.nextLine().trim();
            if (input.isEmpty())
                continue;

            String[] args = input.split(" ");
```

```
            try {
                switch (args[0]) {
                    case "user":
                        handleUserCommand(args);
                        break;
                    case "exit":
                        System.out.println("系统退出");
                        return;
                    case "help":
                        printHelp();
                        break;
                    default:
                        System.out.println("未知命令");
                }
            } catch (Exception e) {
                System.out.println("错误: " + e.getMessage());
            }
        }
    }

    private void handleUserCommand(String[] args) {
        if (args.length < 2) {
            throw new IllegalArgumentException("命令不完整");
        }

        switch (args[1]) {
            case "add":
                addUserFlow(args);
                break;
            case "delete":
                deleteUserFlow(args);
                break;
            case "list":
                listUsersFlow(args);
                break;
            case "update":
                updateUserFlow(args);
                break;
            default:
                throw new IllegalArgumentException("未知命令: " + args[1]);
```



```
        }
    }

    private void addUserFlow(String[] args) {
        Map<String, String> params = parseParams(args);
        User user = new User(
                params.get("name"),
                params.get("gender"),
                Integer.parseInt(params.get("age")),
                params.get("email"),
                params.get("phone"));
        service.addUser(user);
        System.out.println("用户添加成功, ID: " + user.getId());
    }

    private Map<String, String> parseParams(String[] args) {
        Map<String, String> params = new HashMap<>();
        for (int i = 2; i < args.length; i++) {
            if (args[i].startsWith("--")) {
                String key = args[i].substring(2);
                params.put(key, args[++i]);
            }
        }
        return params;
    }

    private void printHelp() {
        System.out.println("可用命令: ");
        System.out.println("user list");
        System.out.println("user add --name <姓名> --gender <性别> --age <年龄> --email <邮箱> --phone <电话>");
        System.out.println("user delete --id <用户ID>");
        System.out.println("user update --id <用户ID> [--name <姓名>] [--gender <性别>] [--age <年龄>] [--email <邮箱>] [--phone <电话>]");
        System.out.println("exit");
    }

    private void deleteUserFlow(String[] args) {
        Map<String, String> params = parseParams(args);
        String userId = params.get("id");
```

```
118         if (userId == null) {
```

```
            throw new IllegalArgumentException("缺少用户ID参数");
        }
        try {
            int id = Integer.parseInt(userId);
            service.deleteUser(id);
            System.out.println("用户删除成功");
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("用户ID必须是数字");
        }
    }

    private void listUsersFlow(String[] args) {
        var users = service.listUsers();
        if (users.isEmpty()) {
            System.out.println("当前没有用户");
            return;
        }
        System.out.println("用户列表：");
        for (User user : users) {
            System.out.printf("ID: %d, 姓名: %s, 性别: %s, 年龄: %d, 邮箱: %s, 电话: %s%n",
                    user.getId(),
                    user.getName(),
                    user.getGender(),
                    user.getAge(),
                    user.getEmail(),
                    user.getPhone());
        }
    }

    private void updateUserFlow(String[] args) {
        Map<String, String> params = parseParams(args);
        String userId = params.get("id");
        if (userId == null) {
            throw new IllegalArgumentException("缺少用户ID参数");
        }

        try {
            int id = Integer.parseInt(userId);
            User user = service.getUser(id);
```



```
            if (user == null) {
```

```
                throw new IllegalArgumentException("用户不存在");
            }

            // 更新用户信息
            if (params.containsKey("name"))
                user.setName(params.get("name"));
            if (params.containsKey("gender"))
                user.setGender(params.get("gender"));
            if (params.containsKey("age"))
                user.setAge(Integer.parseInt(params.get("age")));
            if (params.containsKey("email"))
                user.setEmail(params.get("email"));
            if (params.containsKey("phone"))
                user.setPhone(params.get("phone"));

            service.updateUser(id, user);
            System.out.println("用户信息更新成功");
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("用户ID必须是数字");
    }   }

}
```

04. 程序入口

```
package io.codescience;

im ort io.codescience.ui.CliUI;

pu lic class Application {
    public static void main(String[] args) {
        CliUI cli = new CliUI();
        cli.start();
    }
}
```

运行示例

```
> user add --name "张三" --gender Male --age 30 --email z@example.com --phone 13800138000
用户添加成功，ID: 1

> user list
1 | 张三 | Male | 30 | z@example.com | 13800138000
```

第二阶段：数据库存储（Repository模式）

新增结构

```
├── repository/
│   ├── UserRepository.java
│   └── UserRepositoryImpl.java
└── config/
    └── DatabaseConfig.java
```

核心代码

01. 数据库配置

```java
package io.codescience.config;

public class DatabaseConfig {
    public static final String URL = "jdbc:mysql://localhost:3306/user_management?useSSL=false&serverTimezone=UTC";
    public static final String USER = "root";
    public static final String PASSWORD = "Craftsman@2025";

    static {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException("MySQL JDBC Driver not found", e);
        }
    }
}

```

02. 数据访问层

添加POM引用

```xml
<dependencies>
    <!-- MySQL JDBC Driver -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>
</dependencies>
```

UserRepository.java


```java
// UserRepository.java
package io.codescience.repository;

import io.codescience.model.User;
import java.util.List;
import java.util.Optional;

public interface UserRepository {
    User save(User user);

    Optional<User> findById(int id);

    List<User> findAll();

    void deleteById(int id);

    void update(User user);
}
```

UserRepositoryImpl

```java
package io.codescience.repository;
```



```java

import io.codescience.config.DatabaseConfig;
import io.codescience.model.User;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class UserRepositoryImpl implements UserRepository {

    @Override
    public User save(User user) {
        String sql = "INSERT INTO users(name, gender, age, email, phone) VALUES (?,?,?,?,?)";

        try (Connection conn = DriverManager.getConnection(
                DatabaseConfig.URL,
                DatabaseConfig.USER,
                DatabaseConfig.PASSWORD);
                PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, user.getName());
            stmt.setString(2, user.getGender());
            stmt.setInt(3, user.getAge());
            stmt.setString(4, user.getEmail());
            stmt.setString(5, user.getPhone());

            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    user.setId(rs.getInt(1));
                }
            }
            return user;
        } catch (SQLException e) {
            throw new RuntimeException("保存用户失败", e);
        }
```

```
    }

    @Override
    public Optional<User> findById(int id) {
        String sql = "SELECT * FROM users WHERE id = ?";

        try (Connection conn = DriverManager.getConnection(
                DatabaseConfig.URL,
                DatabaseConfig.USER,
                DatabaseConfig.PASSWORD);
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, id);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return Optional.of(mapResultSetToUser(rs));
                }
            }
            return Optional.empty();
        } catch (SQLException e) {
            throw new RuntimeException("查询用户失败", e);
        }
    }

    @Override
    public List<User> findAll() {
        String sql = "SELECT * FROM users";
        List<User> users = new ArrayList<>();

        try (Connection conn = DriverManager.getConnection(
                DatabaseConfig.URL,
                DatabaseConfig.USER,
                DatabaseConfig.PASSWORD);
                PreparedStatement stmt = conn.prepareStatement(sql);
                ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                users.add(mapResultSetToUser(rs));
            }
```



```
            return users;
        } catch (SQLException e) {
            throw new RuntimeException("查询用户列表失败", e);
        }
    }

    @Override
    public void deleteById(int id) {
        String sql = "DELETE FROM users WHERE id = ?";

        try (Connection conn = DriverManager.getConnection(
                DatabaseConfig.URL,
                DatabaseConfig.USER,
                DatabaseConfig.PASSWORD);
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, id);
            stmt.executeUpdate();
        } catch (SQLException e) {
            throw new RuntimeException("删除用户失败", e);
        }
    }

    @Override
    public void update(User user) {
        String sql = "UPDATE users SET name = ?, gender = ?, age = ?, email = ?, phone = ? WHERE id = ?";

        try (Connection conn = DriverManager.getConnection(
                DatabaseConfig.URL,
                DatabaseConfig.USER,
                DatabaseConfig.PASSWORD);
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, user.getName());
            stmt.setString(2, user.getGender());
            stmt.setInt(3, user.getAge());
            stmt.setString(4, user.getEmail());
            stmt.setString(5, user.getPhone());
            stmt.setInt(6, user.getId());
```

```
118
119             int affectedRows = stmt.executeUpdate();
120             if (affectedRows == 0) {
121                 throw new IllegalArgumentException("用户不存在");
122             }
123         } catch (SQLException e) {
124             throw new RuntimeException("更新用户失败", e);
125         }
126     }
127
128     private User mapResultSetToUser(ResultSet rs) throws SQLException {
129         User user = new User();
130         user.setId(rs.getInt("id"));
131         user.setName(rs.getString("name"));
132         user.setGender(rs.getString("gender"));
133         user.setAge(rs.getInt("age"));
134         user.setEmail(rs.getString("email"));
135         user.setPhone(rs.getString("phone"));
136         return user;
137     }
138 }
```

服务层改造

```
// UserService.java
```



```
package io.codescience.service;

import io.codescience.model.User;
import io.codescience.repository.UserRepository;
import io.codescience.repository.UserRepositoryImpl;

import java.util.List;

public class UserService {
    private final UserRepository repository;

    public UserService() {
        this.repository = new UserRepositoryImpl();
    }

    public User addUser(User user) {
        return repository.save(user);
    }

    public User getUser(int id) {
        return repository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("用户不存在"));
    }

    public List<User> listUsers() {
        return repository.findAll();
    }

    public void deleteUser(int id) {
        if (!repository.findById(id).isPresent()) {
            throw new IllegalArgumentException("用户不存在");
        }
        repository.deleteById(id);
    }

    public void updateUser(int id, User user) {
        if (!repository.findById(id).isPresent()) {
            throw new IllegalArgumentException("用户不存在");
```

```
        }
        user.setId(id);
        repository.update(user);
    }
}
```

数据库初始化脚本

#04. Docker 数据管理（Volumes） （可选）

```
CREATE DATABASE IF NOT EXISTS user_management;
USE user_management;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    gender ENUM('Male','Female','Other') NOT NULL,
    age INT,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15)
);
```

第三阶段：换底层数据库访问逻辑

一、MyBatis Repository的实现


```
<dependencies>
    <!-- MySQL JDBC Driver -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>

    <!-- MyBatis -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.13</version>
    </dependency>
</dependencies>

```

2. 创建数据库配置

创建 `DatabaseConfig.java` ：

```
package io.codescience.config;

public class DatabaseConfig {
    public static final String URL = "jdbc:mysql://localhost:3306/user_management?useSSL=false&serverTimezone=UTC";
    public static final String USER = "root";
    public static final String PASSWORD = "your_password";

    static {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException("MySQL JDBC Driver not found", e);
        }
    }
}
```

```
16
```

3. 创建 MyBatis 配置文件

创建 `src/main/resources/mybatis-config.xml` ：

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql://localhost:3306/user_management?
useSSL=false&amp;serverTimezone=UTC"/>
                <property name="username" value="root"/>
                <property name="password" value="your_password"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="mapper/UserMapper.xml"/>
    </mappers>
</configuration>

```

4. 创建 Mapper 接口

创建 `UserMapper.java` ：

```java
package io.codescience.repository;

import io.codescience.model.User;
import java.util.List;
import java.util.Optional;

public interface UserMapper {
    void insert(User user);
    Optional<User> findById(int id);
    List<User> findAll();
    void deleteById(int id);
    void update(User user);
}

```

5. 创建 Mapper XML

创建 `src/main/resources/mapper/UserMapper.xml` ：


```xml
<?xml version="1.0" encoding="UTF-8" ?>
```

```xml
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="io.codescience.repository.UserMapper">
    <resultMap id="userMap" type="io.codescience.model.User">
        <id property="id" column="id"/>
        <result property="name" column="name"/>
        <result property="gender" column="gender"/>
        <result property="age" column="age"/>
        <result property="email" column="email"/>
        <result property="phone" column="phone"/>
    </resultMap>

    <insert id="insert" parameterType="io.codescience.model.User"
useGeneratedKeys="true" keyProperty="id">
        INSERT INTO users (name, gender, age, email, phone)
        VALUES (#{name}, #{gender}, #{age}, #{email}, #{phone})
    </insert>

    <select id="findById" resultMap="userMap">
        SELECT * FROM users WHERE id = #{id}
    </select>

    <select id="findAll" resultMap="userMap">
        SELECT * FROM users
    </select>

    <delete id="deleteById">
        DELETE FROM users WHERE id = #{id}
    </delete>

    <update id="update" parameterType="io.codescience.model.User">
        UPDATE users
        SET name = #{name},
            gender = #{gender},
            age = #{age},
            email = #{email},
            phone = #{phone}
        WHERE id = #{id}
    </update>
</mapper>
```


6. 实现 Repository

创建 `MyBatisUserRepository.java` ：

```
package io.codescience.repository;
```



```

import io.codescience.model.User;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import java.util.Optional;

public class MyBatisUserRepository implements UserRepository {
    private final SqlSessionFactory sqlSessionFactory;

    public MyBatisUserRepository() {
        try {
            String resource = "mybatis-config.xml";
            InputStream inputStream = Resources.getResourceAsStream(resource);
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            throw new RuntimeException("初始化 MyBatis 失败", e);
        }
    }

    @Override
    public User save(User user) {
        try (SqlSession session = sqlSessionFactory.openSession()) {
            UserMapper mapper = session.getMapper(UserMapper.class);
            mapper.insert(user);
            session.commit();
            return mapper.findById(user.getId())
                    .orElseThrow(() -> new RuntimeException("保存用户后无法获取用户信
息"));
        }
    }

    @Override
    public Optional<User> findById(int id) {
```

```
        try (SqlSession session = sqlSessionFactory.openSession()) {
            UserMapper mapper = session.getMapper(UserMapper.class);
            return mapper.findById(id);
        }
    }

    @Override
    public List<User> findAll() {
        try (SqlSession session = sqlSessionFactory.openSession()) {
            UserMapper mapper = session.getMapper(UserMapper.class);
            return mapper.findAll();
        }
    }

    @Override
    public void deleteById(int id) {
        try (SqlSession session = sqlSessionFactory.openSession()) {
            UserMapper mapper = session.getMapper(UserMapper.class);
            mapper.deleteById(id);
            session.commit();
        }
    }

    @Override
    public void update(User user) {
        try (SqlSession session = sqlSessionFactory.openSession()) {
            UserMapper mapper = session.getMapper(UserMapper.class);
            mapper.update(user);
            session.commit();
        }
    }
}
```

7. 修改 Service 层

修改 `UserService.java` ：


```
public class UserService {
    private final UserRepository repository;

    public UserService() {
        this.repository = new MyBatisUserRepository();
    }
    // ... 其他方法保持不变
}
```

重要说明：

1. **配置说明**：
   - 确保数据库连接信息正确（URL、用户名、密码）
   - 数据库和表需要提前创建
   - MyBatis 配置文件必须放在 resources 目录下
2. **事务管理**：
   - 每个方法都使用独立的 SqlSession
   - 写操作（insert、update、delete）后需要调用 commit()
   - 使用 try-with-resources 自动关闭 SqlSession
3. **错误处理**：
   - 所有数据库操作都包含在 try-catch 块中
   - 适当的错误信息转换
   - 事务回滚处理
4. **性能考虑**：
   - SqlSessionFactory 是单例的
   - 使用连接池（POOLED）
   - 及时关闭资源
5. **使用建议**：
   - 在开发环境中使用
   - 生产环境建议使用连接池（如 HikariCP）
   - 考虑添加日志记录
   - 可以添加缓存机制

测试步骤：

1. 确保 MySQL 服务运行
2. 创建数据库和表

3. 运行程序
4. 测试基本操作：

```
user add --name Jessica --gender Female --age 27 --email 1111 --phone 1111111
user list
user update --id 1 --name Jessica2
user delete --id 1

```

这个实现提供了：

- 清晰的代码结构
- 良好的错误处理
- 事务管理
- 资源管理
- 可扩展性

二、Hibernate Repository的实现

1. 环境准备

2. 实现步骤

3. 关键点说明

3.1 Session 管理

- 使用 try-with-resources 自动管理 Session 的生命周期
- 确保 Session 在使用后正确关闭
- 避免 Session 泄漏

3.2 事务管理

- 每个写操作（save、delete、update）都需要事务
- 使用 try-catch 处理事务异常
- 发生异常时回滚事务

3.3 异常处理

- 初始化异常：包装为 RuntimeException
- 数据库操作异常：在事务中处理
- 查询异常：返回 Optional 或空列表

3.4 性能优化

- 使用 Session 的 get 方法进行主键查询

- 使用 HQL 进行复杂查询
- 合理使用事务范围

4. 使用示例

4.1 创建用户

```
User user = new User();
user.setName("张三");
user.setEmail("zhangsan@example.com");
userRepository.save(user);

```

4.2 查询用户

```
Optional<User> user = userRepository.findById(1);
user.ifPresent(u -> System.out.println(u.getName()));

```

4.3 更新用户

```
User user = userRepository.findById(1).orElseThrow();
user.setName("李四");
userRepository.update(user);

```

4.4 删除用户

```
userRepository.deleteById(1);
```

分层架构设计（仅仅是介绍）

完整项目结构

```
src/
└── main/
    ├── java/
    │   ├── model/User.java
    │   ├── repository/UserRepository.java
    │   ├── service/UserService.java
    │   ├── ui/CliUI.java
    │   └── exception/
    │       ├── DataAccessException.java
    │       └── ValidationException.java
    └── resources/
        └── db/
            └── migration/
                └── V1__Create_users_table.sql
```

分层调用流程

```
sequenceDiagram
    participant UI as CLI界面
    participant Service as UserService
    participant Repository as UserRepository
    participant DB as MySQL

    UI->>Service: 执行user add命令
    Service->>Repository: save(user)
    Repository->>DB: INSERT INTO users...
    DB-->>Repository: 返回生成的ID
    Repository-->>Service: 返回User对象
    Service-->>UI: 显示操作结果
```

关键设计原则

1. **依赖倒置**


```java
// Service层通过接口依赖Repository
public interface UserRepository {
    User save(User user);
    User findById(int id);
    // 其他方法...
}

// 实现类
public class JdbcUserRepository implements UserRepository {
    // JDBC具体实现...
}
```

1. **异常处理分层**

```java
// 自定义异常类
public class DataAccessException extends RuntimeException {
    public DataAccessException(String message, Throwable cause) {
        super(message, cause);
    }
}

// Service层抛出业务异常
public class ValidationException extends RuntimeException {
    public ValidationException(String message) {
        super(message);
    }
}
```

1. **输入验证增强**

```java
public User addUser(User user) {
    validateEmailFormat(user.getEmail());
    validatePhoneFormat(user.getPhone());
    return repository.save(user);
}

private void validateEmailFormat(String email) {
    if (!email.matches("^[A-Za-z0-9+_.-]+@(.+)$"))
        { throw new ValidationException("邮箱格式不正确");
    }
}
```

扩展功能建议

1. 日志记录

```java
// 添加日志依赖
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.36</version>
</dependency>

// 在关键位置添加日志
private static final Logger logger = LoggerFactory.getLogger(UserService.class);

public User addUser(User user) {
    logger.info("添加用户: {}", user.getName());
    // ...
}
```

1. 分页查询优化


```java
public List<User> listUsers(int page, int pageSize) {
    String sql = "SELECT * FROM users LIMIT ? OFFSET ?";
    int offset = (page - 1) * pageSize;
    // 执行分页查询...
}
```

1. 连接池配置

```java
// 使用HikariCP连接池
HikariConfig config = new HikariConfig();
config.setJdbcUrl(DatabaseConfig.URL);
config.setUsername(DatabaseConfig.USER);
config.setPassword(DatabaseConfig.PASSWORD);
HikariDataSource ds = new HikariDataSource(config);
```

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：

```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    gender ENUM('Male', 'Female', 'Other') NOT NULL,
    age INT,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15)
);
```

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）。

✓ 技术准备

- Java 17+
- Spring Boot 3.x
- IDE：IntelliJ IDEA
- 使用 Spring Web 模块即可（不需 JPA、数据库等）

项目结构建议

1. 创建 User 实体（model/User.java）

```
1
```

```java
public class User {
    private Integer id;
    private String name;
    private String gender;
    private Integer age;
    private String email;
    private String phone;

    // 构造方法
    public User() {}

    public User(Integer id, String name, String gender, Integer age, String email, String phone) {
        this.id = id;
        this.name = name;
        this.gender = gender;
        this.age = age;
        this.email = email;
        this.phone = phone;
    }

    // Getter & Setter
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getGender() { return gender; }
    public void setGender(String gender) { this.gender = gender; }

    public Integer getAge() { return age; }
    public void setAge(Integer age) { this.age = age; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }
}
```

```
41
```

---

2. 创建 Controller（controller/UserController.java）


```
1
```

```java
import io.codescience.model.User;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.*;

@RestController
@RequestMapping("/api/users")
public class UserController {

    // 模拟内存数据存储
    private final Map<Integer, User> userMap = new HashMap<>();
    private int currentId = 1;

    // 获取所有用户
    @GetMapping
    public ResponseEntity<List<User>> getAllUsers() {
        return ResponseEntity.ok(new ArrayList<>(userMap.values()));
    }

    // 创建用户
    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user) {
        user.setId(currentId++);
        userMap.put(user.getId(), user);
        return ResponseEntity.ok(user);
    }

    // 获取指定 ID 用户
    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Integer id) {
        User user = userMap.get(id);
        if (user != null) {
            return ResponseEntity.ok(user);
        }
        return ResponseEntity.notFound().build();
    }

```



```java
    // 更新用户
    @PutMapping("/{id}")
    public ResponseEntity<?> updateUser(@PathVariable Integer id, @RequestBody User updatedUser) {
        User existing = userMap.get(id);
        if (existing == null) {
            return ResponseEntity.notFound().build();
        }
        updatedUser.setId(id);
        userMap.put(id, updatedUser);
        return ResponseEntity.ok().build();
    }

    // 删除用户
    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteUser(@PathVariable Integer id) {
        if (userMap.remove(id) != null) {
            return ResponseEntity.ok().build();
        }
        return ResponseEntity.notFound().build();
    }
}

```

---

3. 主程序入口（UserManagementApplication.java）

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UserManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserManagementApplication.class, args);
    }
}

```

---

4. application.properties（任选其一）

```
server:
  port: 8080
```

测试建议

下一步：实验二的分层结构合并到此项目

请自行实现该部分

用户管理系统接口需求文档

1. 项目概述

本项目为“用户管理系统”，提供一套 RESTful 风格的接口，用于实现用户的基本增删查改功能。接口返回数据统一使用 JSON 格式，遵循 OpenAPI 3.0 规范。

---

2. 功能模块概述

功能模块	描述
获取用户列表	查询所有用户数据
获取用户详情	根据 ID 查询单个用户
创建用户	添加新用户信息
更新用户	修改已有用户信息
删除用户	根据 ID 删除用户

---

3. 接口定义

3.1 获取所有用户

- **方法**：GET
- **路径**：`/api/users`
- **说明**：获取用户列表
- **响应**：
  - 200 OK：返回用户数组

✅ **示例响应：**

```
[
  {
    "id": 1,
    "name": "张三",
    "gender": "男",
    "age": 30,
    "email": "zhangsan@example.com",
    "phone": "13800000000"
  }
]
```

3.2 创建用户

- **方法**：POST
- **路径**：`/api/users`
- **说明**：提交用户信息进行创建
- **请求体**（必须）：

字段	类型	是否必须	描述
name	string	是	用户姓名
gender	string	是	用户性别
age	integer	是	用户年龄
email	string	是	邮箱（格式）
phone	string	是	电话

- **响应**：
  - 200 OK：返回创建的用户对象
  - 400 Bad Request：参数验证失败

3.3 获取用户详情

- **方法**：GET
- **路径**：`/api/users/{id}`
- **路径参数**：

参数名	类型	是否必须	描述
id	integer	是	用户 ID

- **响应**：
  - 200 OK：返回对应的用户信息
  - 404 Not Found：用户不存在

3.4 更新用户

- **方法**：PUT
- **路径**：`/api/users/{id}`
- **路径参数**：

参数名	类型	是否必须	描述
id	integer	是	用户 ID

- **请求体**：同创建用户
- **响应**：
  - 200 OK：用户更新成功
  - 400 Bad Request：参数验证失败
  - 404 Not Found：用户不存在

3.5 删除用户

- **方法**：DELETE
- **路径**：`/api/users/{id}`
- **路径参数**：

参数名	类型	是否必须	描述
id	integer	是	用户 ID

- **响应**：
  - 200 OK：用户删除成功
  - 404 Not Found：用户不存在

---

4. 数据结构定义

User 对象结构

字段	类型	是否必须	描述	约束条件
id	integer	否	用户ID	系统生成
name	string	是	用户姓名	最长 50 字符
gender	string	是	性别	最长 10 字符
age	integer	是	年龄	≥ 0
email	string	是	邮箱	email 格式
phone	string	是	电话	最长 20 字符

---

5. 接口错误码定义（建议扩展）

HTTP 状态码	含义	说明
200	OK	请求成功
400	Bad Request	请求参数格式有误
404	Not Found	请求的资源不存在
500	Internal Error	服务内部错误（保留扩展）

---

6. Open API 源代码


```
openapi: 3.0.0
```

```yaml
info:
  title: 用户管理系统 API
  version: 1.0.0
  description: 用户管理系统的RESTful API接口文档

servers:
  - url: http://localhost:8080
    description: 本地开发服务器

paths:
  /api/users:
    get:
      summary: 获取所有用户
      description: 返回所有用户的列表
      responses:
        '200':
          description: 成功获取用户列表
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User'
    post:
      summary: 创建用户
      description: 创建一个新的用户
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      responses:
        '200':
          description: 用户创建成功
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
        '400':
          description: 请求参数验证失败

  /api/users/{id}:
    get:
      summary: 获取用户
      description: 根据ID获取用户信息
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: 成功获取用户信息
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
        '404':
          description: 用户不存在
    put:
      summary: 更新用户
      description: 更新指定ID的用户信息
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      responses:
        '200':
```

```
          description: 用户更新成功
        '400':
          description: 请求参数验证失败
        '404':
          description: 用户不存在
    delete:
      summary: 删除用户
      description: 删除指定ID的用户
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: 用户删除成功
        '404':
          description: 用户不存在

components:
  schemas:
    User:
      type: object
      required:
        - name
        - gender
        - age
        - email
        - phone
      properties:
        id:
          type: integer
          description: 用户ID
        name:
          type: string
          description: 用户姓名
          maxLength: 50
        gender:
          type: string
```



```
          description: 用户性别
          maxLength: 10
        age:
          type: integer
          description: 用户年龄
          minimum: 0
        email:
          type: string
          description: 用户邮箱
          format: email
          maxLength: 100
        phone:
          type: string
          description: 用户电话
          maxLength: 20
```

一、实验项目需求：企业运营数据仪表盘

核心功能

1. **实时数据展示区**
   - 顶部显示实时更新的公司关键指标（用户总数、订单量、营收额）
2. **核心指标卡片（Ant Design Card）**
   - 以卡片形式展示环比增长率（如用户增长+12%）、完成率（如季度目标85%）
   - 不同状态使用颜色区分：绿色（达标）、橙色（警告）、红色（异常）
3. **图表展示（Ant Design Charts）**
   - **折线图**：展示近30天订单趋势（X轴为日期，Y轴为数量）

二、实现步骤（关键步骤与代码示例）

1. 环境搭建

```
# 创建脚手架程序
npx create-react-app project-dashboard
cd project-dashboard

# 安装依赖
npm install antd @ant-design/charts axios

# 确保脚手架程序可以执行
npm start
```

2. 项目结构

```
src/
├── components/
│   ├── KpiCards.js        # 指标卡片
│   ├── OrderChart.js      # 订单折线图
├── services/
│   └── api.js             # 数据请求封装
├── App.js                 # 主组件
└── index.js
```

3. 数据请求封装（services/api.js）

```
import axios from "axios";

export const fetchData = async () => {
  // 模拟API数据（实际替换为真实接口）
  return {
    users: 12480,
    orders: 1567,
    revenue: 285430,
    serverLoad: 65,
    dailyOrders: [
      { date: "06-01", count: 120 },
      { date: "06-02", count: 135 },
      { date: "06-03", count: 150 },
      { date: "06-04", count: 125 },
      { date: "06-05", count: 140 },
      { date: "06-06", count: 130 },
      { date: "06-07", count: 155 },
      { date: "06-08", count: 145 },
      { date: "06-09", count: 160 },
      { date: "06-10", count: 150 },
      { date: "06-11", count: 165 },
      { date: "06-12", count: 170 },
    ],
  };
};
```

4. 指标卡片组件（components/KpiCards.js）

```jsx
import { Card, Statistic, Tag } from "antd";

export default ({ data }) => {
  if (!data) {
    return null;
  }

  return (
    <div
      className="kpi-cards"
      style={{
        display: "flex",
        flexWrap: "wrap",
        gap: "16px", // 设置卡片间距
      }}
    >
      <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
        <Statistic title="用户总数" value={data.users} />
        <Tag color="green">+12%周环比</Tag>
      </Card>
      <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
        <Statistic title="订单量" value={data.orders} />
        <Tag color="red">+12%周环比</Tag>
      </Card>
      <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
        <Statistic title="营收额" value={data.revenue} />
        <Tag color="green">+12%周环比</Tag>
      </Card>
      <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
        <Statistic title="完成率" value={data.serverLoad} />
        <Tag color="orange">+12%周环比</Tag>
      </Card>
    </div>
  );
};

```


5. 图表组件集成（components/OrderChart.js）

```jsx
import { Line } from "@ant-design/charts";

export default ({ data }) => {
  if (!data || !data.dailyOrders) {
    return null;
  }

  const config = {
    data: data.dailyOrders,
    xField: "date",
    yField: "count",
    point: { size: 5 },
  };
  return <Line {...config} />;
};

```

6. 主组件逻辑（App.js）

```jsx
import { Row, Col, Button, DatePicker, Spin } from "antd";
```



```jsx
import { useEffect, useState } from "react";
import { fetchData } from "./services/api";
import KpiCards from "./components/KpiCards";
import OrderChart from "./components/OrderChart";

export default () => {
  const [stats, setStats] = useState(null);
  const [loading, setLoading] = useState(true);

  const loadData = async () => {
    setLoading(true);
    try {
      const data = await fetchData();
      setStats(data);
    } catch (error) {
      console.error("加载数据失败:", error);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    loadData();
  }, []);

  return (
    <div style={{ padding: 24 }}>
      <Row gutter={[16, 16]}>
        <Col span={24}>
          <DatePicker.RangePicker />
          <Button onClick={loadData} style={{ marginLeft: 16 }}>
            刷新数据
          </Button>
        </Col>

        <Col span={24}>
          <Spin spinning={loading}>
            <KpiCards data={stats} />
          </Spin>
        </Col>
```

```
        <Col span={24}>
          <Spin spinning={loading}>
            <OrderChart data={stats} />
          </Spin>
        </Col>
      </Row>
    </div>
  );
};
```

7. 导入Ant Design

```
import 'antd/dist/reset.css';
```

8. 测试验证

```
npm start
```

三、技术要点解析与教学知识点

1. 项目架构设计

- **模块化组织**：
  - `components/` 封装可复用UI组件（KPI卡片、图表）
  - `services/` 隔离数据逻辑（API请求封装）
  - 符合“单一职责原则”，便于协作维护
- **数据流管理**：
  - 使用React Hooks（ `useState` / `useEffect` ）实现单向数据流

  - 父组件（App.js）统一管理状态，子组件通过props接收数据

2. 核心组件实现

- **KPI卡片（Ant Design Card）**：

```
<Statistic title="用户总数" value={data.users} />
<Tag color="green">+12%周环比</Tag>  // 状态颜色动态映射
```

  - 知识点： `Statistic` 数值格式化、 `Tag` 状态标识逻辑、响应式布局（ `flex` ）
- **折线图（Ant Design Charts）**：

```
const config = {
  xField: "date",  // 日期轴映射
  yField: "count", // 数值轴映射
  point: { size: 5 }, // 交互增强
};
```

  - 知识点：数据映射原理、可视化最佳实践（交叉提示框）

3. 数据层关键技术

- **API服务封装**：

```
export const fetchData = async () => {
  return { ... };  // 模拟数据层
};
```

  - 知识点：异步请求处理（ `async/await` ）、错误边界处理（ `try/catch` ）
- **加载状态管理**：

```
<Spin spinning={loading}>  // 全局加载指示器
  <KpiCards data={stats} />
</Spin>
```

  - 知识点：用户体验优化、异步状态反馈机制

4. 交互与扩展能力

- **用户操作响应**：
  - 日期选择器（`DatePicker.RangePicker`）过滤数据
  - 手动刷新按钮（`onClick={loadData}`）触发数据重载
- **可扩展性设计**：
  - 组件化架构支持快速添加新图表类型
  - API模块可无缝切换真实数据源

四、掌握的知识点

1. **React核心技术**：
   - Hooks状态管理（`useState`，`useEffect`）
   - 组件化开发模式（Props传递、模块拆分）
2. **Ant Design生态**：
   - 基础组件（Card, Statistic, Tag）
   - 可视化图表库（Line, 柱状图，饼图扩展）
3. **工程化实践**：
   - 项目脚手架（`create-react-app`）
   - 依赖管理（`npm install`）
   - 目录规范设计
4. **数据处理能力**：
   - 异步数据获取（Axios/Fetch）
   - 数据结构映射（API→组件）
5. **UI/UX原则**：
   - 状态反馈（Spin加载）
   - 响应式布局（Flex/Grid）
   - 数据可视化最佳实践

五、页面组件加载活动图（Mermaid）


```mermaid
graph TD
    A[用户访问仪表盘] --> B[App组件挂载]
    B --> C[useEffect触发loadData]
    C --> D[显示Spin加载状态]
    D --> E[调用fetchData API]
    E --> F{请求成功?}
    F -- 是 --> G[更新stats状态]
    F -- 否 --> H[控制台报错]
    G --> I[隐藏Spin]
    I --> J[渲染KPI卡片]
    J --> K[渲染折线图]
    K --> L[显示完整仪表盘]
    H --> M[显示错误提示] --> I
```

> **流程图说明**：
> 1. **初始化阶段**：组件挂载时自动加载数据
> 2. **状态管理**：通过Spin组件实现加载态/完成态切换
> 3. **异常处理**：捕获API错误并反馈（不影响UI渲染）
> 4. **渲染流程**：数据就绪后依次渲染卡片和图表

四、扩展学习建议

1. **进阶方向**：
   - 接入真实API（替换`fetchData`模拟数据）
   - 添加权限控制（路由守卫）
   - 集成Redux状态管理
2. **优化方向**：
   - 数据缓存策略（减少重复请求）
   - 图表细节优化（双Y轴、趋势线）
   - 移动端适配

通过本项目的实践，掌握企业级仪表盘开发的完整技术链条，从工程架构到数据可视化实现，培养全栈开发的核心能力。

下面我将基于Ant Design + React实现一个用户管理UI，分两个阶段进行开发：第一阶段使用Mock API实现功能，第二阶段连接真实API。

一、项目搭建与配置

1. **创建React项目**：

```
npx create-react-app user-management
cd user-management
```

1. **安装依赖**：

```
npm install antd axios @ant-design/icons
npm install @ant-design/v5-patch-for-react-19 --save
```

1. **配置Ant Design**（在 `src/index.js` 中）：

```
import 'antd/dist/reset.css';
import '@ant-design/v5-patch-for-react-19';
```

二、第一阶段：Mock API实现

1. Mock服务配置（ `src/mock/userApi.js` ）


```
import { v4 as uuidv4 } from 'uuid';

let users = [
  { id: '1', name: '张三', gender: '男', age: 25, email: 'zhang@example.com', phone: '13800138000' },
  { id: '2', name: '李四', gender: '女', age: 30, email: 'li@example.com', phone: '13900139000' }
];

export const mockUserApi = {
  getUsers: () => new Promise(resolve =>
    setTimeout(() => resolve({ data: users }), 500)),

  createUser: (user) => new Promise(resolve => {
    setTimeout(() => {
      const newUser = { ...user, id: uuidv4() };
      users = [...users, newUser];
      resolve({ data: newUser });
    }, 500);
  }),

  updateUser: (id, user) => new Promise(resolve => {
    setTimeout(() => {
      users = users.map(u => u.id === id ? { ...u, ...user } : u);
      resolve({ data: { id, ...user } });
    }, 500);
  }),

  deleteUser: (id) => new Promise(resolve => {
    setTimeout(() => {
      users = users.filter(u => u.id !== id);
      resolve({ data: { id } });
    }, 500);
  })
};
```

2. 用户管理界面（ `src/components/UserManagement.jsx` ）

```jsx
import React, { useState, useEffect } from 'react';
```



```jsx
import { Table, Button, Modal, Form, Input, InputNumber, message } from 'antd';
import { PlusOutlined } from '@ant-design/icons';
import { mockUserApi } from '../mock/userApi';

const UserManagement = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(false);
  const [visible, setVisible] = useState(false);
  const [currentUser, setCurrentUser] = useState(null);
  const [form] = Form.useForm();

  const columns = [
    { title: '姓名', dataIndex: 'name', key: 'name' },
    { title: '性别', dataIndex: 'gender', key: 'gender' },
    { title: '年龄', dataIndex: 'age', key: 'age' },
    { title: '邮箱', dataIndex: 'email', key: 'email' },
    { title: '电话', dataIndex: 'phone', key: 'phone' },
    {
      title: '操作',
      key: 'action',
      render: (_, record) => (
        <>
          <Button type="link" onClick={() => handleEdit(record)}>编辑</Button>
          <Button type="link" danger onClick={() => handleDelete(record.id)}>删除
</Button>
        </>
      ),
    },
  ];

  const fetchUsers = async () => {
    setLoading(true);
    try {
      const response = await mockUserApi.getUsers();
      setUsers(response.data);
    } catch (error) {
      message.error('获取用户列表失败');
    } finally {
      setLoading(false);
```

```
    }
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  const handleCreate = () => {
    form.resetFields();
    setCurrentUser(null);
    setVisible(true);
  };

  const handleEdit = (user) => {
    form.setFieldsValue(user);
    setCurrentUser(user);
    setVisible(true);
  };

  const handleDelete = async (id) => {
    Modal.confirm({
      title: '确认删除',
      content: '确定要删除该用户吗？',
      onOk: async () => {
        await mockUserApi.deleteUser(id);
        message.success('删除成功');
        fetchUsers();
      },
    });
  };

  const handleSubmit = async () => {
    try {
      const values = await form.validateFields();
      if (currentUser) {
        await mockUserApi.updateUser(currentUser.id, values);
        message.success('更新成功');
      } else {
        await mockUserApi.createUser(values);
        message.success('创建成功');
```



```
      }
      setVisible(false);
      fetchUsers();
    } catch (error) {
      console.error('提交失败:', error);
    }
  };

  return (
    <div style={{ padding: 24 }}>
      <div style={{ marginBottom: 16 }}>
        <Button
          type="primary"
          icon={<PlusOutlined />}
          onClick={handleCreate}
        >
          添加用户
        </Button>
      </div>

      <Table
        columns={columns}
        dataSource={users}
        rowKey="id"
        loading={loading}
        pagination={{ pageSize: 10 }}
      />

      <Modal
        title={currentUser ? "编辑用户" : "创建用户"}
        open={visible}
        onOk={handleSubmit}
        onCancel={() => setVisible(false)}
      >
        <Form form={form} layout="vertical">
          <Form.Item
            name="name"
            label="姓名"
            rules={[{ required: true, message: '请输入姓名' }]}
          >
```

```
            <Input placeholder="张三" maxLength={50} />
          </Form.Item>

          <Form.Item
            name="gender"
            label="性别"
            rules={[{ required: true, message: '请输入性别' }]}
          >
            <Input placeholder="男/女" maxLength={10} />
          </Form.Item>

          <Form.Item
            name="age"
            label="年龄"
            rules={[{ required: true, message: '请输入年龄' }]}
          >
            <InputNumber min={0} style={{ width: '100%' }} />
          </Form.Item>

          <Form.Item
            name="email"
            label="邮箱"
            rules={[
              { required: true, message: '请输入邮箱' },
              { type: 'email', message: '邮箱格式不正确' }
            ]}
          >
            <Input placeholder="example@domain.com" maxLength={100} />
          </Form.Item>

          <Form.Item
            name="phone"
            label="电话"
            rules={[{ required: true, message: '请输入电话' }]}
          >
            <Input placeholder="13800138000" maxLength={20} />
          </Form.Item>
        </Form>
      </Modal>
    </div>
```



```
  );
};

export default UserManagement;
```

3. 主应用入口（`src/App.js`）

```
import UserManagement from './components/UserManagement';

function App()
  { return (
    <div className="App">
      <UserManagement />
    </div>
  );
}

export default App;
```

三、第二阶段：连接真实API

1. 创建API服务（`src/services/userService.js`）

```
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/users';

export const userService = {
  getUsers: () => axios.get(API_URL),
  createUser: (user) => axios.post(API_URL, user),
  updateUser: (id, user) => axios.put(`${API_URL}/${id}`, user),
  deleteUser: (id) => axios.delete(`${API_URL}/${id}`)
};
```

2. 修改用户管理组件

更换 mockUserApi ==>userService

```
// 在UserManagement.jsx中替换：
// import { mockUserApi } from '../mock/userApi';
import { userService } from '../services/userService';

// 修改所有API调用方法：
const fetchUsers = async () => {
  setLoading(true);
  try {
    const response = await userService.getUsers();
    setUsers(response.data);
  } catch (error) {
    message.error('获取用户列表失败');
  } finally {
    setLoading(false);
  }
};

// 同样修改create/update/delete方法中的API调用
```

3. 添加请求拦截器（`src/utils/http.js`）


```
import axios from 'axios';

const instance = axios.create({
  baseURL: 'http://localhost:8080',
  timeout: 5000
});

// 请求拦截器
instance.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// 响应拦截器
instance.interceptors.response.use(
  response => response.data,
  error => {
    if (error.response) {
      switch (error.response.status) {
        case 401:
          // 处理未授权
          break;
        case 404:
          // 处理未找到
          break;
        default:
          console.error('API错误:', error);
      }
    }
    return Promise.reject(error);
  }
);

export default instance;
```

四、功能特点说明

1. **用户界面**:
   - 响应式表格展示用户数据
   - 表单验证（必填项、邮箱格式、年龄最小值）
   - 操作反馈（成功/错误提示）
2. **Mock API实现**:
   - 使用内存数据模拟CRUD操作
   - 添加500ms延迟模拟网络请求
   - UUID生成唯一ID
3. **Ant Design组件使用**:
   - Table展示数据
   - Modal+Form实现编辑/创建
   - 消息提示和确认对话框
4. **真实API集成**:
   - Axios HTTP客户端
   - 请求/响应拦截器
   - 错误统一处理

五、项目结构

```
src/
├── components/
│   └── UserManagement.jsx
├── services/
│   └── userService.js
├── mock/
│   └── userApi.js
├── utils/
│   └── http.js
├── App.js
└── index.js
```

六、测试建议

1. **Mock阶段测试**:


```
// 在组件中添加测试按钮
<Button onClick={() => {
  mockUserApi.getUsers().then(res => console.log(res));
}}>
  测试Mock API
</Button>
```

1. **真实API测试**:

```
// 在浏览器控制台测试
userService.getUsers().then(console.log);
```

1. **错误处理测试**:

```
// 在userService.js中模拟错误
deleteUser: () => Promise.reject(new Error('模拟错误'))
```

个人博客页面开发 - 完整实现

下面是一个完整的个人博客网站实现，包含首页、文章展示页和关于我页面，具有响应式设计和所需的所有功能。

项目结构

```
personal-blog/
├── index.html          # 首页
├── article.html        # 文章展示页
├── about.html          # 关于我页面
├── css/
│   └── style.css       # 主样式文件
└── js/
    └── script.js       # 主JavaScript文件

```

1. index.html (首页)


```
<!DOCTYPE html>
```

```html
<html lang="zh-CN">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>我的个人博客</title>
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <header class="header">
      <div class="container">
        <h1 class="logo">我的博客</h1>
        <nav class="main-nav">
          <ul>
            <li><a href="index.html" class="active">首页</a></li>
            <li><a href="about.html">关于我</a></li>
            <li><a href="#contact">联系我</a></li>
          </ul>
        </nav>
      </div>
    </header>

    <div class="container main-content">
      <main class="content">
        <section class="blog-posts">
          <h2>最新文章</h2>

          <article class="post">
            <h3><a href="article.html">HTML5新特性探索</a></h3>
            <div class="post-meta">
              <span class="date">2023年5月15日</span>
              <span class="category">前端开发</span>
            </div>
            <p>
              HTML5带来了许多令人兴奋的新特性，如语义化标签、本地存储、Canvas绘图等。本文将详细介绍这些新特性及其应用场景...
            </p>
            <a href="article.html" class="read-more">阅读更多</a>
          </article>

          <article class="post">
            <h3><a href="article.html">CSS3动画入门</a></h3>
            <div class="post-meta">
              <span class="date">2023年5月10日</span>
              <span class="category">前端开发</span>
            </div>
            <p>
              CSS3动画可以让你的网页更加生动有趣。学习如何使用@keyframes、transition和transform属性来创建流畅的动画效果...
            </p>
            <a href="article.html" class="read-more">阅读更多</a>
          </article>

          <article class="post">
            <h3><a href="article.html">JavaScript基础教程</a></h3>
            <div class="post-meta">
              <span class="date">2023年5月5日</span>
              <span class="category">前端开发</span>
            </div>
            <p>
              JavaScript是网页交互的核心。本教程将带你从零开始学习JavaScript的基础语法、DOM操作和事件处理...
            </p>
            <a href="article.html" class="read-more">阅读更多</a>
          </article>
        </section>
      </main>

      <aside class="sidebar">
        <div class="widget">
          <h3>关于我</h3>
          <img src="./img/avatar-01.jpg" alt="我的头像" class="profile-img" />
          <p>我是前端开发爱好者，热爱学习新技术，喜欢分享知识。</p>
          <a href="about.html" class="btn">了解更多</a>
        </div>

        <div class="widget">
          <h3>分类</h3>
          <ul class="categories">
            <li><a href="#">前端开发</a></li>
```

```
            <li><a href="#">JavaScript</a></li>
```



```
            <li><a href="#">CSS技巧</a></li>
            <li><a href="#">HTML5</a></li>
          </ul>
        </div>

        <div class="widget">
          <h3>订阅</h3>
          <p>订阅我的博客，获取最新文章通知</p>
          <form class="subscribe-form">
            <input type="email" placeholder="你的邮箱地址" />
            <button type="submit" class="btn">订阅</button>
          </form>
        </div>
      </aside>
    </div>

    <footer class="footer">
      <div class="container">
        <p>&copy; 2023 我的个人博客. 保留所有权利.</p>
        <div class="social-links">
          <a href="#"><i class="fab fa-github"></i></a>
          <a href="#"><i class="fab fa-twitter"></i></a>
          <a href="#"><i class="fab fa-linkedin"></i></a>
        </div>
      </div>
    </footer>

    <button id="back-to-top" title="回到顶部">↑</button>
    <script src="js/script.js"></script>
  </body>
</html>
```

2. article.html (文章展示页)

```
<!DOCTYPE html>
```



```
<html lang="zh-CN">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>文章标题 - 我的个人博客</title>
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <header class="header">
      <div class="container">
        <h1 class="logo">我的博客</h1>
        <nav class="main-nav">
          <ul>
            <li><a href="index.html">首页</a></li>
            <li><a href="about.html">关于我</a></li>
            <li><a href="#contact">联系我</a></li>
          </ul>
        </nav>
      </div>
    </header>

    <div class="container article-container">
      <article class="article-content">
        <header>
          <h1>HTML5新特性探索</h1>
          <div class="article-meta">
            <span class="author">作者: 张三</span>
            <span class="date">发布时间: 2023年5月15日</span>
            <span class="category">分类: 前端开发</span>
          </div>
        </header>

        <img src="./img/html.png" alt="HTML5" class="featured-image" />

        <div class="article-body">
          <p>
            HTML5是HTML最新的修订版本，由万维网联盟（W3C）于2014年10月完成标准制定。HTML5的设计目的是为了在移动设备上支持多媒体。
          </p>
```

```
            <h2>语义化标签</h2>
            <p>
              HTML5引入了许多新的语义化元素，如&lt;header&gt;, &lt;footer&gt;,
              &lt;article&gt;,
              &lt;section&gt;等，这些标签让网页结构更加清晰，也便于搜索引擎理解网页内容。
            </p>

            <h2>多媒体支持</h2>
            <p>
              HTML5原生支持音频和视频播放，不再需要Flash等插件。使用&lt;audio&gt;和
&lt;video&gt;标签可以轻松地在网页中嵌入多媒体内容。
            </p>

            <pre><code>&lt;video width="320" height="240" controls&gt;
    &lt;source src="movie.mp4" type="video/mp4"&gt;
    &lt;source src="movie.ogg" type="video/ogg"&gt;
    您的浏览器不支持HTML5视频标签。
&lt;/video&gt;</code></pre>

            <h2>Canvas绘图</h2>
            <p>
              HTML5的&lt;canvas&gt;元素提供了一套JavaScript
              API，允许开发者在网页上绘制图形、动画和游戏。Canvas非常适合创建数据可视化、广告横
幅和游戏等。
            </p>

            <h2>本地存储</h2>
            <p>
              HTML5提供了localStorage和sessionStorage两种本地存储方式，可以在客户端存储数据，
减少与服务器的通信，提高应用性能。
            </p>
          </div>

          <footer class="article-footer">
            <div class="tags">
              <span>标签: </span>
              <a href="#">HTML5</a>, <a href="#">前端开发</a>,
              <a href="#">Web技术</a>
            </div>
          </footer>
```



```
        </article>

        <section class="comments-section">
          <h2>评论</h2>

          <div id="comments-list">
            <div class="comment">
              <div class="comment-author">
                <img src="./img/avatar-02.jpg" alt="用户头像" />
                <span>李四</span>
              </div>
              <div class="comment-content">
                <p>这篇文章写得很好，对HTML5的新特性介绍得很全面！</p>
                <div class="comment-meta">2023年5月16日</div>
              </div>
            </div>

            <div class="comment">
              <div class="comment-author">
                <img src="./img/avatar-03.jpg" alt="用户头像" />
                <span>王五</span>
              </div>
              <div class="comment-content">
                <p>Canvas部分可以再详细一些，期待后续文章！</p>
                <div class="comment-meta">2023年5月17日</div>
              </div>
            </div>
          </div>

          <form id="comment-form" class="comment-form">
            <h3>发表评论</h3>
            <div class="form-group">
              <label for="name">姓名:</label>
              <input type="text" id="name" name="name" required />
            </div>
            <div class="form-group">
              <label for="email">邮箱:</label>
              <input type="email" id="email" name="email" required />
            </div>
            <div class="form-group">
```

```
            <label for="comment">评论:</label>
            <textarea id="comment" name="comment" rows="5" required></textarea>
          </div>
          <button type="submit" class="btn">提交评论</button>
        </form>
      </section>
    </div>

    <footer class="footer">
      <div class="container">
        <p>&copy; 2023 我的个人博客. 保留所有权利.</p>
        <div class="social-links">
          <a href="#"><i class="fab fa-github"></i></a>
          <a href="#"><i class="fab fa-twitter"></i></a>
          <a href="#"><i class="fab fa-linkedin"></i></a>
        </div>
      </div>
    </footer>

    <button id="back-to-top" title="回到顶部">↑</button>

    <script src="js/script.js"></script>
  </body>
</html>
```

3. about.html (关于我页面)

```
<!DOCTYPE html>
```

```html
<html lang="zh-CN">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>关于我 - 我的个人博客</title>
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <header class="header">
      <div class="container">
        <h1 class="logo">我的博客</h1>
        <nav class="main-nav">
          <ul>
            <li><a href="index.html">首页</a></li>
            <li><a href="about.html" class="active">关于我</a></li>
            <li><a href="#contact">联系我</a></li>
          </ul>
        </nav>
      </div>
    </header>

    <div class="container about-container">
      <section class="about-section">
        <h2>关于我</h2>

        <div class="about-content">
          <img
            src="./img/avatar-06.jpg"
            style="width: 100px"
            alt="我的照片"
            class="about-img"
          />

          <div class="about-text">
            <h3>张三</h3>
            <p class="title">前端开发工程师</p>
            <p>
              我是一名热爱前端开发的技术爱好者，专注于HTML5、CSS3和JavaScript技术。拥有5年Web开发经验，喜欢探索新技术并分享知识。
            </p>
            <p>我的技术栈包括：</p>
            <ul class="skills">
              <li>HTML5 & CSS3</li>
              <li>JavaScript (ES6+)</li>
              <li>React.js</li>
              <li>Vue.js</li>
              <li>Node.js</li>
              <li>Git</li>
            </ul>
          </div>
        </div>
      </section>

      <section class="contact-section" id="contact">
        <h2>联系我</h2>

        <div class="contact-methods">
          <div class="contact-info">
            <h3>联系方式</h3>
            <ul>
              <li>
                <i class="fas fa-envelope"></i> Email: example@example.com
              </li>
              <li><i class="fas fa-phone"></i> 电话: 123-456-7890</li>
              <li><i class="fas fa-map-marker-alt"></i> 地址: 北京市海淀区</li>
            </ul>

            <div class="social-links">
              <a href="#"><i class="fab fa-github"></i></a>
              <a href="#"><i class="fab fa-twitter"></i></a>
              <a href="#"><i class="fab fa-linkedin"></i></a>
              <a href="#"><i class="fab fa-weibo"></i></a>
            </div>
          </div>

          <form class="contact-form">
            <h3>发送消息</h3>
            <div class="form-group">
              <input type="text" placeholder="你的姓名" required />
```

```
            </div>
            <div class="form-group">
              <input type="email" placeholder="你的邮箱" required />
            </div>
            <div class="form-group">
              <input type="text" placeholder="主题" />
            </div>
            <div class="form-group">
              <textarea placeholder="你的消息" rows="5" required></textarea>
            </div>
            <button type="submit" class="btn">发送消息</button>
          </form>
        </div>
      </section>
    </div>

    <footer class="footer">
      <div class="container">
        <p>&copy; 2023 我的个人博客. 保留所有权利.</p>
        <div class="social-links">
          <a href="#"><i class="fab fa-github"></i></a>
          <a href="#"><i class="fab fa-twitter"></i></a>
          <a href="#"><i class="fab fa-linkedin"></i></a>
        </div>
      </div>
    </footer>

    <button id="back-to-top" title="回到顶部">↑</button>
    <script src="js/script.js"></script>
  </body>
</html>
```

4. css/style.css (样式文件)


```
/* 全局样式 */
:root {
    --primary-color: #3498db;
    --secondary-color: #2980b9;
    --dark-color: #2c3e50;
    --light-color: #ecf0f1;
    --success-color: #2ecc71;
    --danger-color: #e74c3c;
    --warning-color: #f39c12;
    --gray-color: #95a5a6;
    --white-color: #fff;
    --black-color: #333;
}

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    line-height: 1.6;
    color: var(--black-color);
    background-color: #f5f5f5;
}

a {
    text-decoration: none;
    color: var(--primary-color);
}

a:hover {
    color: var(--secondary-color);
}

img {
    max-width: 100%;
    height: auto;
```

```css
}

.container {
    width: 100%;
    max-width: 1200px;
    margin: 0 auto;
    padding: 0 15px;
}

.btn {
    display: inline-block;
    background: var(--primary-color);
    color: var(--white-color);
    padding: 8px 16px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: all 0.3s ease;
}

.btn:hover {
    background: var(--secondary-color);
    color: var(--white-color);
}

/* 头部样式 */
.header {
    background: var(--dark-color);
    color: var(--white-color);
    padding: 20px 0;
    position: sticky;
    top: 0;
    z-index: 100;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.header .container {
    display: flex;
    justify-content: space-between;
    align-items: center;
```



```css
}

.logo {
    font-size: 24px;
    font-weight: bold;
}

.main-nav ul {
    display: flex;
    list-style: none;
}

.main-nav ul li {
    margin-left: 20px;
}

.main-nav ul li a {
    color: var(--white-color);
    font-weight: 500;
    transition: color 0.3s ease;
}

.main-nav ul li a:hover {
    color: var(--primary-color);
}

.main-nav ul li a.active {
    color: var(--primary-color);
    font-weight: bold;
}

/* 主要内容布局 */
.main-content {
    display: flex;
    margin: 30px auto;
    gap: 30px;
}

.content {
    flex: 2;
```

```css
}

.sidebar {
    flex: 1;
}

/* 文章列表样式 */
.blog-posts {
    margin-bottom: 30px;
}

.blog-posts h2 {
    margin-bottom: 20px;
    padding-bottom: 10px;
    border-bottom: 1px solid #ddd;
}

.post {
    background: var(--white-color);
    padding: 20px;
    margin-bottom: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.post h3 {
    margin-bottom: 10px;
}

.post h3 a {
    color: var(--dark-color);
}

.post h3 a:hover {
    color: var(--primary-color);
}

.post-meta {
    font-size: 14px;
    color: var(--gray-color);
```



```css
    margin-bottom: 15px;
}

.post-meta .date {
    margin-right: 15px;
}

.post p {
    margin-bottom: 15px;
}

.read-more {
    display: inline-block;
    font-weight: bold;
    transition: all 0.3s ease;
}

.read-more:hover {
    color: var(--secondary-color);
}

/* 侧边栏样式 */
.widget {
    background: var(--white-color);
    padding: 20px;
    margin-bottom: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.widget h3 {
    margin-bottom: 15px;
    padding-bottom: 10px;
    border-bottom: 1px solid #ddd;
}

.profile-img {
    display: block;
    margin: 0 auto 15px;
    border-radius: 50%;
```

```
}

.categories {
    list-style: none;
}

.categories li {
    margin-bottom: 8px;
    padding-bottom: 8px;
    border-bottom: 1px dashed #eee;
}

.categories li:last-child {
    margin-bottom: 0;
    padding-bottom: 0;
    border-bottom: none;
}

.subscribe-form input {
    width: 100%;
    padding: 8px;
    margin-bottom: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
}

/* 页脚样式 */
.footer {
    background: var(--dark-color);
    color: var(--white-color);
    padding: 30px 0;
    text-align: center;
}

.footer .container {
    display: flex;
    flex-direction: column;
    align-items: center;
}

```



```
.social-links {
    margin-top: 15px;
}

.social-links a {
    color: var(--white-color);
    margin: 0 10px;
    font-size: 20px;
    transition: color 0.3s ease;
}

.social-links a:hover {
    color: var(--primary-color);
}

/* 回到顶部按钮 */
#back-to-top {
    display: none;
    position: fixed;
    bottom: 20px;
    right: 20px;
    background: var(--primary-color);
    color: var(--white-color);
    border: none;
    border-radius: 50%;
    width: 50px;
    height: 50px;
    font-size: 20px;
    cursor: pointer;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
    transition: all 0.3s ease;
    z-index: 99;
}

#back-to-top:hover {
    background: var(--secondary-color);
}

/* 文章页样式 */
.article-container {
```

```
    margin: 30px auto;
}

.article-content {
    background: var(--white-color);
    padding: 30px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    margin-bottom: 30px;
}

.article-content h1 {
    margin-bottom: 15px;
    font-size: 32px;
}

.article-meta {
    font-size: 14px;
    color: var(--gray-color);
    margin-bottom: 20px;
}

.article-meta span {
    margin-right: 15px;
}

.featured-image {
    display: block;
    margin: 0 auto 20px;
    max-height: 400px;
    object-fit: cover;
    border-radius: 5px;
}

.article-body h2 {
    margin: 25px 0 15px;
    font-size: 24px;
}

.article-body p {
```



```
    margin-bottom: 15px;
}

.article-body pre {
    background: #f5f5f5;
    padding: 15px;
    border-radius: 5px;
    margin: 15px 0;
    overflow-x: auto;
}

.article-body code {
    font-family: 'Courier New', Courier, monospace;
    font-size: 14px;
}

.article-footer {
    margin-top: 30px;
    padding-top: 15px;
    border-top: 1px solid #eee;
}

.tags a {
    display: inline-block;
    background: #f5f5f5;
    padding: 3px 8px;
    margin-right: 5px;
    border-radius: 3px;
    font-size: 14px;
    color: var(--gray-color);
}

.tags a:hover {
    background: var(--primary-color);
    color: var(--white-color);
}

/* 评论区域样式 */
.comments-section {
    background: var(--white-color);
```

```
    padding: 30px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.comments-section h2 {
    margin-bottom: 20px;
    padding-bottom: 10px;
    border-bottom: 1px solid #ddd;
}

.comment {
    display: flex;
    margin-bottom: 20px;
    padding-bottom: 20px;
    border-bottom: 1px solid #eee;
}

.comment:last-child {
    margin-bottom: 0;
    padding-bottom: 0;
    border-bottom: none;
}

.comment-author {
    flex: 0 0 100px;
    text-align: center;
}

.comment-author img {
    border-radius: 50%;
    margin-bottom: 5px;
}

.comment-content {
    flex: 1;
    padding-left: 20px;
}

.comment-meta {
```



```
    font-size: 12px;
    color: var(--gray-color);
    margin-top: 5px;
}

.comment-form {
    margin-top: 30px;
}

.comment-form h3 {
    margin-bottom: 15px;
}

.form-group {
    margin-bottom: 15px;
}

.form-group label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
}

.form-group input,
.form-group textarea {
    width: 100%;
    padding: 8px;
    border: 1px solid #ddd;
    border-radius: 4px;
}

.form-group textarea {
    resize: vertical;
    min-height: 100px;
}

/* 关于我页面样式 */
.about-container {
    margin: 30px auto;
}
```

```
.about-section {
    background: var(--white-color);
    padding: 30px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    margin-bottom: 30px;
}

.about-section h2 {
    margin-bottom: 20px;
    padding-bottom: 10px;
    border-bottom: 1px solid #ddd;
}

.about-content {
    display: flex;
    gap: 30px;
}

.about-img {
    flex: 0 0 300px;
    border-radius: 5px;
    object-fit: cover;
}

.about-text {
    flex: 1;
}

.about-text h3 {
    font-size: 24px;
    margin-bottom: 5px;
}

.title {
    font-size: 16px;
    color: var(--gray-color);
    margin-bottom: 15px;
}
```



```
.skills {
    display: flex;
    flex-wrap: wrap;
    list-style: none;
    margin: 15px 0;
}

.skills li {
    background: #f5f5f5;
    padding: 5px 10px;
    margin: 0 10px 10px 0;
    border-radius: 3px;
    font-size: 14px;
}

.contact-section {
    background: var(--white-color);
    padding: 30px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.contact-section h2 {
    margin-bottom: 20px;
    padding-bottom: 10px;
    border-bottom: 1px solid #ddd;
}

.contact-methods {
    display: flex;
    gap: 30px;
}

.contact-info {
    flex: 1;
}

.contact-info ul {
    list-style: none;
```

```
}

.contact-info li {
    margin-bottom: 10px;
    display: flex;
    align-items: center;
}

.contact-info i {
    margin-right: 10px;
    width: 20px;
    text-align: center;
}

.contact-form {
    flex: 1;
}

/* 响应式设计 */
@media (max-width: 992px) {
    .main-content {
        flex-direction: column;
    }

    .about-content {
        flex-direction: column;
    }

    .about-img {
        margin-bottom: 20px;
    }

    .contact-methods {
        flex-direction: column;
    }
}

@media (max-width: 768px) {
    .header .container {
        flex-direction: column;
    }

    .logo {
        margin-bottom: 15px;
    }

    .main-nav ul {
        flex-direction: column;
        align-items: center;
    }

    .main-nav ul li {
        margin: 5px 0;
    }

    .article-content,
    .comments-section,
    .about-section,
    .contact-section {
        padding: 20px;
    }
}
```


5. js/script.js (JavaScript文件)

```
document.addEventListener('DOMContentLoaded', function() {
    // 回到顶部按钮
    const backToTopButton = document.getElementById('back-to-top');

    window.addEventListener('scroll', function() {
        if (window.pageYOffset > 300) {
            backToTopButton.style.display = 'block';
        } else {
            backToTopButton.style.display = 'none';
        }
    });

    backToTopButton.addEventListener('click', function(e) {
        e.preventDefault();
        window.scrollTo({
            top: 0,
            behavior: 'smooth'
        });
    });

    // 评论表单提交
    const commentForm = document.getElementById('comment-form');
    if (commentForm) {
        commentForm.addEventListener('submit', function(e) {
            e.preventDefault();

            const name = document.getElementById('name').value;
            const commentText = document.getElementById('comment').value;

            if (name && commentText) {
                const commentsList = document.getElementById('comments-list');
                const now = new Date();
                const dateStr = now.toLocaleDateString('zh-CN');

                const newComment = document.createElement('div');
                newComment.className = 'comment';
                newComment.innerHTML = `
                    <div class="comment-author">
                        <img src="https://via.placeholder.com/50" alt="用户头像">
```



```
                        <span>${name}</span>
                    </div>
                    <div class="comment-content">
                        <p>${commentText}</p>
                        <div class="comment-meta">${dateStr}</div>
                    </div>
                `;

                commentsList.appendChild(newComment);

                // 清空表单
                commentForm.reset();

                // 滚动到新评论
                newComment.scrollIntoView({ behavior: 'smooth' });
            }
        });
    }

    // 联系表单提交
    const contactForm = document.querySelector('.contact-form');
    if (contactForm) {
        contactForm.addEventListener('submit', function(e) {
            e.preventDefault();

            const name = contactForm.querySelector('input[type="text"]').value;
            const email = contactForm.querySelector('input[type="email"]').value;
            const message = contactForm.querySelector('textarea').value;

            if (name && email && message) {
                alert(`感谢您的留言，${name}！我们会尽快回复您。`);
                contactForm.reset();
            } else {
                alert('请填写所有必填字段！');
            }
        });
    }

    // 订阅表单提交
    const subscribeForm = document.querySelector('.subscribe-form');
```

```
    if (subscribeForm) {
        subscribeForm.addEventListener('submit', function(e) {
            e.preventDefault();

            const email = subscribeForm.querySelector('input[type="email"]').value;

            if (email) {
                alert(`感谢订阅！我们已将确认邮件发送至 ${email}`);
                subscribeForm.reset();
            } else {
                alert('请输入有效的邮箱地址！');
            }
        });
    }
});
```

项目特点

1. **响应式设计**：使用CSS媒体查询确保网站在不同设备上都能良好显示
2. **语义化HTML**：正确使用HTML5语义化标签
3. **现代CSS**：使用CSS变量、Flexbox等现代CSS特性
4. **交互功能**：
   - 滚动到顶部按钮
   - 文章评论功能
   - 联系表单和订阅表单处理

使用说明

1. 将上述代码分别保存到对应的文件中
2. 确保文件结构正确
3. 在浏览器中打开index.html即可查看网站

这个实现包含了所有要求的功能，并且具有良好的代码结构和样式设计。你可以根据需要进一步自定义内容和样式。

基本目标：锻炼学员基础知识的理解

项目主题：个人博客页面开发（HTML5 + CSS3 + JavaScript）

项目描述：学员需要设计并实现一个简单的个人博客页面，页面包含主页、文章展示页、以及简短的关于我部分。项目的重点是掌握HTML5、CSS3以及基础JavaScript的使用，主要完成以下任务：

任务要求：

1. **首页设计**
   - 创建一个包含导航栏、侧边栏、博客文章列表（包含标题、摘要、阅读更多链接）和页脚的主页。
   - 使用HTML5的语义化标签，如<header>、<footer>、<section>、<article>等，来组织页面结构。
   - 使用CSS3设置页面的基本样式（背景色、文本样式、布局等）。
   - JavaScript部分：添加一个滚动到顶部的按钮，当用户滚动页面到一定位置时，按钮会显示出来。
2. **文章展示页**
   - 创建单独的文章展示页，用户点击文章列表中的“阅读更多”链接后，跳转到对应的文章页面。
   - 使用HTML5来编写文章内容，包括标题、日期、正文、图片等。
   - 使用CSS3进行页面样式设计，特别是排版、图片样式、段落间距等。
   - JavaScript部分：实现“评论”功能，用户可以在文章下方提交评论，并显示在页面上。
3. **关于我页面**
   - 创建一个关于我的页面，展示个人简介、联系方式、社交媒体链接等。
   - 页面设计上突出个人特色，使用CSS3美化页面，包括字体样式、背景、边框等。

项目输出：

- 一个包含首页、文章展示页和关于我页面的简单个人博客网站。
- 页面必须具备响应式设计，能够适应不同设备的屏幕尺寸（使用CSS3媒体查询）。

进阶目标：个人博客页面开发（Bootstrap + JavaScript）

改造个人博客页面功能：

1. 使用Bootstrap重构页面
2. 使用组件化思想封装页面组件
3. 使用Webpack 构建项目
4. 添加图表分析

目录建议：

假设你的项目结构如下：

```
src/
└── main/
    └── java/
        └── com/
            └── example/
                └── demo/
                    ├── DemoApplication.java        <-- 启动类
                    ├── controller/
                    │   └── UserController.java
                    └── config/
                        └── GlobalCorsConfig.java  <-- CORS 配置类

```

> 将 `GlobalCorsConfig.java` 放在 `config/` 包下即可，只要它和 `DemoApplication.java` 启动类**在同一个包或子包下**，就能被 Spring 自动加载。

示例代码回顾：


```java
// src/main/java/com/example/demo/config/GlobalCorsConfig.java

package com.example.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class GlobalCorsConfig {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("http://localhost:3000")
                        .allowedMethods("GET", POST, PUT, DELETE, OPTIONS")
                        .allowedHeaders("*")
                        .allowCredentials(true);
            }
        };
    }
}

```