

## 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

```
1 user-management/
2 |   └── src/
3 |     └── main/
4 |       └── java/io/codescience
5 |         └── model/
6 |           └── User.java
7 |         └── service/
8 |           └── UserService.java
9 |         └── ui/
10 |           └── cliUI.java
11 └── pom.xml
```

### 核心代码

#### 01. 领域模型层

```
1 // User.java
2 package io.codescience.model;
3
4
5 public class User {
6     private int id;
7     private String name;
8     private String gender;
9     private int age;
10    private String email;
11    private String phone;
12
13    public User(String name, String gender, int age, String email, String phone) {
14        validateInput(name, "姓名");
15        validateInput(gender, "性别");
16        validateAge(age);
17        this.name = name;
18        this.gender = gender;
19        this.age = age;
20        this.email = email;
21        this.phone = phone;
22    }
23
24    private void validateInput(String value, String field) {
25        if (value == null || value.trim().isEmpty()) {
26            throw new IllegalArgumentException(field + "不能为空");
27        }
28    }
29
30    private void validateAge(int age) {
31        if (age < 0) {
32            throw new IllegalArgumentException("年龄不能为负数");
33        }
34    }
35    // Getter/Setter methods...
36 }
```

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

## 02. 业务逻辑层

```
2 package io.codescience.service;
3
4 import io.codescience.model.User;
5 import java.util.ArrayList;
6 import java.util.HashMap;
7 import java.util.List;
8 import java.util.Map;
9
10 public class UserService {
11     private Map<Integer, User> users = new HashMap<>();
12     private int idCounter = 1;
13
14     public User addUser(User user) {
15         user.setId(idCounter++);
16         users.put(user.getId(), user);
17         return user;
18     }
19
20     public User deleteUser(int id) {
21         return users.remove(id);
22     }
23
24     public User updateUser(int id, User newData) {
25         User existing = users.get(id);
26         if (existing != null) {
27             existing.setName(newData.getName());
28             existing.setGender(newData.getGender());
29             existing.setAge(newData.getAge());
30             existing.setEmail(newData.getEmail());
31             existing.setPhone(newData.getPhone());
32         }
33         return existing;
34     }
35
36     public User getUser(int id) {
37         return users.get(id);
38     }
39 }
```

```
40     public List<User> listUsers() {
41         return new ArrayList<>(users.values());
42     }
43 }
```

### 03. 命令行界面 (CLI)

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

```
2 import io.codescience.model.User;
3 import io.codescience.service.UserService;
4
5 import java.util.HashMap;
6 import java.util.Map;
7 import java.util.Scanner;
8
9 public class CliUI {
10     private final UserService service = new UserService();
11
12     public CliUI() {
13         initSampleData();
14     }
15
16     private void initSampleData() {
17         // 添加示例用户数据
18         User user1 = new User("张三", "男", 25, "zhangsan@example.com", "13800138001");
19         User user2 = new User("李四", "女", 28, "lisi@example.com", "13800138002");
20         User user3 = new User("王五", "男", 30, "wangwu@example.com", "13800138003");
21
22         service.addUser(user1);
23         service.addUser(user2);
24         service.addUser(user3);
25
26         System.out.println("示例数据初始化完成");
27     }
28
29     public void start() {
30         System.out.println("用户管理系统启动（输入help查看命令帮助）");
31         Scanner scanner = new Scanner(System.in);
32
33         while (true) {
34             System.out.print("\n> ");
35             String input = scanner.nextLine().trim();
36             if (input.isEmpty())
37                 continue;
38
39             String[] args = input.split(" ");
```

```

40     try {
41         switch (args[0]) {
42             case "user":
43                 handleUserCommand(args);
44                 break;
45             case "exit":
46                 System.out.println("系统退出");
47                 return;
48             case "help":
49                 printHelp();
50                 break;
51             default:
52                 System.out.println("未知命令");
53             }
54         } catch (Exception e) {
55             System.out.println("错误: " + e.getMessage());
56         }
57     }
58 }
59
60 private void handleUserCommand(String[] args) {
61     if (args.length < 2) {
62         throw new IllegalArgumentException("命令不完整");
63     }
64
65     switch (args[1]) {
66         case "add":
67             addUserFlow(args);
68             break;
69         case "delete":
70             deleteUserFlow(args);
71             break;
72         case "list":
73             listUsersFlow(args);
74             break;
75         case "update":
76             updateUserFlow(args);
77             break;
78         default:
79             throw new IllegalArgumentException("未知命令: " + args[1]);
}

```

```

80         }
81     }
82
83     private void addUserFlow(String[] args) {
84         Map<String, String> params = parseParams(args);
85         User user = new User(
86             params.get("name"),
87             params.get("gender"),
88             Integer.parseInt(params.get("age")),
89             params.get("email"),
90             params.get("phone"));
91         service.addUser(user);
92         System.out.println("用户添加成功, ID: " + user.getId());
93     }
94
95     private Map<String, String> parseParams(String[] args) {
96         Map<String, String> params = new HashMap<>();
97         for (int i = 2; i < args.length; i++) {
98             if (args[i].startsWith("--")) {
99                 String key = args[i].substring(2);
100                params.put(key, args[++i]);
101            }
102        }
103        return params;
104    }
105
106    private void printHelp() {
107        System.out.println("可用命令: ");
108        System.out.println("user list");
109        System.out.println("user add --name <姓名> --gender <性别> --age <年龄> --email <邮箱> --phone <电话>");
110        System.out.println("user delete --id <用户ID>");
111        System.out.println("user update --id <用户ID> [--name <姓名>] [--gender <性别>] [--age <年龄>] [--email <邮箱>] [--phone <电话>]");
112        System.out.println("exit");
113    }
114
115    private void deleteUserFlow(String[] args) {
116        Map<String, String> params = parseParams(args);
117        String userId = params.get("id");

```

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

```

119         throw new IllegalArgumentException("缺少用户ID参数");
120     }
121
122     try {
123         int id = Integer.parseInt(userId);
124         service.deleteUser(id);
125         System.out.println("用户删除成功");
126     } catch (NumberFormatException e) {
127         throw new IllegalArgumentException("用户ID必须是数字");
128     }
129
130     private void listUsersFlow(String[] args) {
131         var users = service.listUsers();
132         if (users.isEmpty()) {
133             System.out.println("当前没有用户");
134             return;
135         }
136         System.out.println("用户列表: ");
137         for (User user : users) {
138             System.out.printf("ID: %d, 姓名: %s, 性别: %s, 年龄: %d, 邮箱: %s, 电话:
139 %s%n",
140                 user.getId(),
141                 user.getName(),
142                 user.getGender(),
143                 user.getAge(),
144                 user.getEmail(),
145                 user.getPhone());
146         }
147     }
148
149     private void updateUserFlow(String[] args) {
150         Map<String, String> params = parseParams(args);
151         String userId = params.get("id");
152         if (userId == null) {
153             throw new IllegalArgumentException("缺少用户ID参数");
154         }
155
156         try {
157             int id = Integer.parseInt(userId);
158             User user = service.getUser(id);

```

```

158     if (user == null) {

```

```

159         throw new IllegalArgumentException("用户不存在");
160     }
161
162     // 更新用户信息
163     if (params.containsKey("name"))
164         user.setName(params.get("name"));
165     if (params.containsKey("gender"))
166         user.setGender(params.get("gender"));
167     if (params.containsKey("age"))
168         user.setAge(Integer.parseInt(params.get("age")));
169     if (params.containsKey("email"))
170         user.setEmail(params.get("email"));
171     if (params.containsKey("phone"))
172         user.setPhone(params.get("phone"));
173
174     service.updateUser(id, user);
175     System.out.println("用户信息更新成功");
176 } catch (NumberFormatException e) {
177     throw new IllegalArgumentException("用户ID必须是数字");
178 }
179
180 }
```

## 运行示例

```

1 > user add --name "张三" --gender Male --age 30 --email z@example.com --phone
13800138000
2 用户添加成功, ID: 1
3
4 > user list
5 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

```

1 ┌── repository/
2   ├── UserRepository.java
3   └── UserRepositoryImpl.java
4 └── config/
5   └── DatabaseConfig.java
```

### 核心代码

#### 01. 数据库配置

```

1 package io.codescience;
2
3 import io.codescience.ui.CliUI;
4
5 public class Application {
6     public static void main(String[] args) {
7         CliUI cli = new CliUI();
8         cli.start();
9     }
10 }
```

## 04. 程序入口

```

1 package io.codescience.config;
2
3 public class DatabaseConfig {
4     public static final String URL = "jdbc:mysql://localhost:3306/user_management?
useSSL=false&serverTimezone=UTC";
5     public static final String USER = "root";
6     public static final String PASSWORD = "Craftsman@2025";
7
8     static {
9         try {
10             Class.forName("com.mysql.cj.jdbc.Driver");
11         } catch (ClassNotFoundException e) {
12             throw new RuntimeException("MySQL JDBC Driver not found", e);
13         }
14     }
15 }
16

```

## 02. 数据访问层

添加POM引用

```

1 <dependencies>
2     <!-- MySQL JDBC Driver -->
3     <dependency>
4         <groupId>mysql</groupId>
5         <artifactId>mysql-connector-java</artifactId>
6         <version>8.0.33</version>
7     </dependency>
8 </dependencies>

```

UserRepository.java

```

1 // UserRepository.java
2 package io.codescience.repository;
3
4 import io.codescience.model.User;
5 import java.util.List;
6 import java.util.Optional;
7
8 public interface UserRepository {
9     User save(User user);
10    Optional<User> findById(int id);
11    List<User> findAll();
12    void deleteById(int id);
13    void update(User user);
14 }
15
16
17
18 }

```

UserRepositoryImpl

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

```
2
3 import io.codescience.config.DatabaseConfig;
4 import io.codescience.model.User;
5 import java.sql.*;
6 import java.util.ArrayList;
7 import java.util.List;
8 import java.util.Optional;
9
10 public class UserRepositoryImpl implements UserRepository {
11
12     @Override
13     public User save(User user) {
14         String sql = "INSERT INTO users(name, gender, age, email, phone) VALUES
15             (?, ?, ?, ?, ?)";
16
17         try (Connection conn = DriverManager.getConnection(
18             DatabaseConfig.URL,
19             DatabaseConfig.USER,
20             DatabaseConfig.PASSWORD));
21
22             PreparedStatement stmt = conn.prepareStatement(sql,
23                 Statement.RETURN_GENERATED_KEYS)) {
24
25                 stmt.setString(1, user.getName());
26                 stmt.setString(2, user.getGender());
27                 stmt.setInt(3, user.getAge());
28                 stmt.setString(4, user.getEmail());
29                 stmt.setString(5, user.getPhone());
30
31                 stmt.executeUpdate();
32
33                 try (ResultSet rs = stmt.getGeneratedKeys()) {
34                     if (rs.next()) {
35                         user.setId(rs.getInt(1));
36                     }
37
38                 }
39
40             return user;
41         } catch (SQLException e) {
42             throw new RuntimeException("保存用户失败", e);
43         }
44     }
45 }
```

```

39     }
40
41     @Override
42     public Optional<User> findById(int id) {
43         String sql = "SELECT * FROM users WHERE id = ?";
44
45         try (Connection conn = DriverManager.getConnection(
46             DatabaseConfig.URL,
47             DatabaseConfig.USER,
48             DatabaseConfig.PASSWORD);
49             PreparedStatement stmt = conn.prepareStatement(sql)) {
50
51             stmt.setInt(1, id);
52
53             try (ResultSet rs = stmt.executeQuery()) {
54                 if (rs.next()) {
55                     return Optional.of(mapResultSetToUser(rs));
56                 }
57             }
58             return Optional.empty();
59         } catch (SQLException e) {
60             throw new RuntimeException("查询用户失败", e);
61         }
62     }
63
64     @Override
65     public List<User> findAll() {
66         String sql = "SELECT * FROM users";
67         List<User> users = new ArrayList<>();
68
69         try (Connection conn = DriverManager.getConnection(
70             DatabaseConfig.URL,
71             DatabaseConfig.USER,
72             DatabaseConfig.PASSWORD);
73             PreparedStatement stmt = conn.prepareStatement(sql);
74             ResultSet rs = stmt.executeQuery()) {
75
76             while (rs.next()) {
77                 users.add(mapResultSetToUser(rs));
78             }
79
80         } catch (SQLException e) {
81             throw new RuntimeException("查询用户列表失败", e);
82         }
83     }
84
85     @Override
86     public void deleteById(int id) {
87         String sql = "DELETE FROM users WHERE id = ?";
88
89         try (Connection conn = DriverManager.getConnection(
90             DatabaseConfig.URL,
91             DatabaseConfig.USER,
92             DatabaseConfig.PASSWORD);
93             PreparedStatement stmt = conn.prepareStatement(sql)) {
94
95             stmt.setInt(1, id);
96             stmt.executeUpdate();
97         } catch (SQLException e) {
98             throw new RuntimeException("删除用户失败", e);
99         }
100    }
101
102    @Override
103    public void update(User user) {
104        String sql = "UPDATE users SET name = ?, gender = ?, age = ?, email = ?, phone = ? WHERE id = ?";
105
106        try (Connection conn = DriverManager.getConnection(
107            DatabaseConfig.URL,
108            DatabaseConfig.USER,
109            DatabaseConfig.PASSWORD);
110            PreparedStatement stmt = conn.prepareStatement(sql)) {
111
112            stmt.setString(1, user.getName());
113            stmt.setString(2, user.getGender());
114            stmt.setInt(3, user.getAge());
115            stmt.setString(4, user.getEmail());
116            stmt.setString(5, user.getPhone());
117            stmt.setInt(6, user.getId());
118
119        } catch (SQLException e) {
120            throw new RuntimeException("更新用户失败", e);
121        }
122    }
123
124    private User mapResultSetToUser(ResultSet rs) {
125        User user = new User();
126
127        try {
128            user.setId(rs.getInt("id"));
129            user.setName(rs.getString("name"));
130            user.setGender(rs.getString("gender"));
131            user.setAge(rs.getInt("age"));
132            user.setEmail(rs.getString("email"));
133            user.setPhone(rs.getString("phone"));
134        } catch (SQLException e) {
135            throw new RuntimeException("映射结果集失败", e);
136        }
137
138        return user;
139    }
140
141    private static final Map<String, Integer> columnIndexMap = new HashMap<>();
142
143    static {
144        columnIndexMap.put("id", 1);
145        columnIndexMap.put("name", 2);
146        columnIndexMap.put("gender", 3);
147        columnIndexMap.put("age", 4);
148        columnIndexMap.put("email", 5);
149        columnIndexMap.put("phone", 6);
150    }
151
152    private static User mapResultSetToUser(ResultSet rs) {
153        User user = new User();
154
155        try {
156            user.setId(columnIndexMap.get("id").intValue());
157            user.setName(rs.getString("name"));
158            user.setGender(rs.getString("gender"));
159            user.setAge(rs.getInt("age"));
160            user.setEmail(rs.getString("email"));
161            user.setPhone(rs.getString("phone"));
162        } catch (SQLException e) {
163            throw new RuntimeException("映射结果集失败", e);
164        }
165
166        return user;
167    }
168
169    private static final Map<String, Integer> columnIndexMap2 = new HashMap<>();
170
171    static {
172        columnIndexMap2.put("name", 1);
173        columnIndexMap2.put("gender", 2);
174        columnIndexMap2.put("age", 3);
175        columnIndexMap2.put("email", 4);
176        columnIndexMap2.put("phone", 5);
177    }
178
179    private static User mapResultSetToUser(ResultSet rs) {
180        User user = new User();
181
182        try {
183            user.setName(columnIndexMap2.get("name").toString());
184            user.setGender(columnIndexMap2.get("gender").toString());
185            user.setAge(columnIndexMap2.get("age").intValue());
186            user.setEmail(columnIndexMap2.get("email").toString());
187            user.setPhone(columnIndexMap2.get("phone").toString());
188        } catch (SQLException e) {
189            throw new RuntimeException("映射结果集失败", e);
190        }
191
192        return user;
193    }
194
195    private static final Map<String, Integer> columnIndexMap3 = new HashMap<>();
196
197    static {
198        columnIndexMap3.put("name", 1);
199        columnIndexMap3.put("gender", 2);
200        columnIndexMap3.put("age", 3);
201        columnIndexMap3.put("email", 4);
202        columnIndexMap3.put("phone", 5);
203    }
204
205    private static User mapResultSetToUser(ResultSet rs) {
206        User user = new User();
207
208        try {
209            user.setName(columnIndexMap3.get("name").toString());
210            user.setGender(columnIndexMap3.get("gender").toString());
211            user.setAge(columnIndexMap3.get("age").intValue());
212            user.setEmail(columnIndexMap3.get("email").toString());
213            user.setPhone(columnIndexMap3.get("phone").toString());
214        } catch (SQLException e) {
215            throw new RuntimeException("映射结果集失败", e);
216        }
217
218        return user;
219    }
220
221    private static final Map<String, Integer> columnIndexMap4 = new HashMap<>();
222
223    static {
224        columnIndexMap4.put("name", 1);
225        columnIndexMap4.put("gender", 2);
226        columnIndexMap4.put("age", 3);
227        columnIndexMap4.put("email", 4);
228        columnIndexMap4.put("phone", 5);
229    }
230
231    private static User mapResultSetToUser(ResultSet rs) {
232        User user = new User();
233
234        try {
235            user.setName(columnIndexMap4.get("name").toString());
236            user.setGender(columnIndexMap4.get("gender").toString());
237            user.setAge(columnIndexMap4.get("age").intValue());
238            user.setEmail(columnIndexMap4.get("email").toString());
239            user.setPhone(columnIndexMap4.get("phone").toString());
240        } catch (SQLException e) {
241            throw new RuntimeException("映射结果集失败", e);
242        }
243
244        return user;
245    }
246
247    private static final Map<String, Integer> columnIndexMap5 = new HashMap<>();
248
249    static {
250        columnIndexMap5.put("name", 1);
251        columnIndexMap5.put("gender", 2);
252        columnIndexMap5.put("age", 3);
253        columnIndexMap5.put("email", 4);
254        columnIndexMap5.put("phone", 5);
255    }
256
257    private static User mapResultSetToUser(ResultSet rs) {
258        User user = new User();
259
260        try {
261            user.setName(columnIndexMap5.get("name").toString());
262            user.setGender(columnIndexMap5.get("gender").toString());
263            user.setAge(columnIndexMap5.get("age").intValue());
264            user.setEmail(columnIndexMap5.get("email").toString());
265            user.setPhone(columnIndexMap5.get("phone").toString());
266        } catch (SQLException e) {
267            throw new RuntimeException("映射结果集失败", e);
268        }
269
270        return user;
271    }
272
273    private static final Map<String, Integer> columnIndexMap6 = new HashMap<>();
274
275    static {
276        columnIndexMap6.put("name", 1);
277        columnIndexMap6.put("gender", 2);
278        columnIndexMap6.put("age", 3);
279        columnIndexMap6.put("email", 4);
280        columnIndexMap6.put("phone", 5);
281    }
282
283    private static User mapResultSetToUser(ResultSet rs) {
284        User user = new User();
285
286        try {
287            user.setName(columnIndexMap6.get("name").toString());
288            user.setGender(columnIndexMap6.get("gender").toString());
289            user.setAge(columnIndexMap6.get("age").intValue());
290            user.setEmail(columnIndexMap6.get("email").toString());
291            user.setPhone(columnIndexMap6.get("phone").toString());
292        } catch (SQLException e) {
293            throw new RuntimeException("映射结果集失败", e);
294        }
295
296        return user;
297    }
298
299    private static final Map<String, Integer> columnIndexMap7 = new HashMap<>();
300
301    static {
302        columnIndexMap7.put("name", 1);
303        columnIndexMap7.put("gender", 2);
304        columnIndexMap7.put("age", 3);
305        columnIndexMap7.put("email", 4);
306        columnIndexMap7.put("phone", 5);
307    }
308
309    private static User mapResultSetToUser(ResultSet rs) {
310        User user = new User();
311
312        try {
313            user.setName(columnIndexMap7.get("name").toString());
314            user.setGender(columnIndexMap7.get("gender").toString());
315            user.setAge(columnIndexMap7.get("age").intValue());
316            user.setEmail(columnIndexMap7.get("email").toString());
317            user.setPhone(columnIndexMap7.get("phone").toString());
318        } catch (SQLException e) {
319            throw new RuntimeException("映射结果集失败", e);
320        }
321
322        return user;
323    }
324
325    private static final Map<String, Integer> columnIndexMap8 = new HashMap<>();
326
327    static {
328        columnIndexMap8.put("name", 1);
329        columnIndexMap8.put("gender", 2);
330        columnIndexMap8.put("age", 3);
331        columnIndexMap8.put("email", 4);
332        columnIndexMap8.put("phone", 5);
333    }
334
335    private static User mapResultSetToUser(ResultSet rs) {
336        User user = new User();
337
338        try {
339            user.setName(columnIndexMap8.get("name").toString());
340            user.setGender(columnIndexMap8.get("gender").toString());
341            user.setAge(columnIndexMap8.get("age").intValue());
342            user.setEmail(columnIndexMap8.get("email").toString());
343            user.setPhone(columnIndexMap8.get("phone").toString());
344        } catch (SQLException e) {
345            throw new RuntimeException("映射结果集失败", e);
346        }
347
348        return user;
349    }
350
351    private static final Map<String, Integer> columnIndexMap9 = new HashMap<>();
352
353    static {
354        columnIndexMap9.put("name", 1);
355        columnIndexMap9.put("gender", 2);
356        columnIndexMap9.put("age", 3);
357        columnIndexMap9.put("email", 4);
358        columnIndexMap9.put("phone", 5);
359    }
360
361    private static User mapResultSetToUser(ResultSet rs) {
362        User user = new User();
363
364        try {
365            user.setName(columnIndexMap9.get("name").toString());
366            user.setGender(columnIndexMap9.get("gender").toString());
367            user.setAge(columnIndexMap9.get("age").intValue());
368            user.setEmail(columnIndexMap9.get("email").toString());
369            user.setPhone(columnIndexMap9.get("phone").toString());
370        } catch (SQLException e) {
371            throw new RuntimeException("映射结果集失败", e);
372        }
373
374        return user;
375    }
376
377    private static final Map<String, Integer> columnIndexMap10 = new HashMap<>();
378
379    static {
380        columnIndexMap10.put("name", 1);
381        columnIndexMap10.put("gender", 2);
382        columnIndexMap10.put("age", 3);
383        columnIndexMap10.put("email", 4);
384        columnIndexMap10.put("phone", 5);
385    }
386
387    private static User mapResultSetToUser(ResultSet rs) {
388        User user = new User();
389
390        try {
391            user.setName(columnIndexMap10.get("name").toString());
392            user.setGender(columnIndexMap10.get("gender").toString());
393            user.setAge(columnIndexMap10.get("age").intValue());
394            user.setEmail(columnIndexMap10.get("email").toString());
395            user.setPhone(columnIndexMap10.get("phone").toString());
396        } catch (SQLException e) {
397            throw new RuntimeException("映射结果集失败", e);
398        }
399
400        return user;
401    }
402
403    private static final Map<String, Integer> columnIndexMap11 = new HashMap<>();
404
405    static {
406        columnIndexMap11.put("name", 1);
407        columnIndexMap11.put("gender", 2);
408        columnIndexMap11.put("age", 3);
409        columnIndexMap11.put("email", 4);
410        columnIndexMap11.put("phone", 5);
411    }
412
413    private static User mapResultSetToUser(ResultSet rs) {
414        User user = new User();
415
416        try {
417            user.setName(columnIndexMap11.get("name").toString());
418            user.setGender(columnIndexMap11.get("gender").toString());
419            user.setAge(columnIndexMap11.get("age").intValue());
420            user.setEmail(columnIndexMap11.get("email").toString());
421            user.setPhone(columnIndexMap11.get("phone").toString());
422        } catch (SQLException e) {
423            throw new RuntimeException("映射结果集失败", e);
424        }
425
426        return user;
427    }
428
429    private static final Map<String, Integer> columnIndexMap12 = new HashMap<>();
430
431    static {
432        columnIndexMap12.put("name", 1);
433        columnIndexMap12.put("gender", 2);
434        columnIndexMap12.put("age", 3);
435        columnIndexMap12.put("email", 4);
436        columnIndexMap12.put("phone", 5);
437    }
438
439    private static User mapResultSetToUser(ResultSet rs) {
440        User user = new User();
441
442        try {
443            user.setName(columnIndexMap12.get("name").toString());
444            user.setGender(columnIndexMap12.get("gender").toString());
445            user.setAge(columnIndexMap12.get("age").intValue());
446            user.setEmail(columnIndexMap12.get("email").toString());
447            user.setPhone(columnIndexMap12.get("phone").toString());
448        } catch (SQLException e) {
449            throw new RuntimeException("映射结果集失败", e);
450        }
451
452        return user;
453    }
454
455    private static final Map<String, Integer> columnIndexMap13 = new HashMap<>();
456
457    static {
458        columnIndexMap13.put("name", 1);
459        columnIndexMap13.put("gender", 2);
460        columnIndexMap13.put("age", 3);
461        columnIndexMap13.put("email", 4);
462        columnIndexMap13.put("phone", 5);
463    }
464
465    private static User mapResultSetToUser(ResultSet rs) {
466        User user = new User();
467
468        try {
469            user.setName(columnIndexMap13.get("name").toString());
470            user.setGender(columnIndexMap13.get("gender").toString());
471            user.setAge(columnIndexMap13.get("age").intValue());
472            user.setEmail(columnIndexMap13.get("email").toString());
473            user.setPhone(columnIndexMap13.get("phone").toString());
474        } catch (SQLException e) {
475            throw new RuntimeException("映射结果集失败", e);
476        }
477
478        return user;
479    }
480
481    private static final Map<String, Integer> columnIndexMap14 = new HashMap<>();
482
483    static {
484        columnIndexMap14.put("name", 1);
485        columnIndexMap14.put("gender", 2);
486        columnIndexMap14.put("age", 3);
487        columnIndexMap14.put("email", 4);
488        columnIndexMap14.put("phone", 5);
489    }
490
491    private static User mapResultSetToUser(ResultSet rs) {
492        User user = new User();
493
494        try {
495            user.setName(columnIndexMap14.get("name").toString());
496            user.setGender(columnIndexMap14.get("gender").toString());
497            user.setAge(columnIndexMap14.get("age").intValue());
498            user.setEmail(columnIndexMap14.get("email").toString());
499            user.setPhone(columnIndexMap14.get("phone").toString());
500        } catch (SQLException e) {
501            throw new RuntimeException("映射结果集失败", e);
502        }
503
504        return user;
505    }
506
507    private static final Map<String, Integer> columnIndexMap15 = new HashMap<>();
508
509    static {
510        columnIndexMap15.put("name", 1);
511        columnIndexMap15.put("gender", 2);
512        columnIndexMap15.put("age", 3);
513        columnIndexMap15.put("email", 4);
514        columnIndexMap15.put("phone", 5);
515    }
516
517    private static User mapResultSetToUser(ResultSet rs) {
518        User user = new User();
519
520        try {
521            user.setName(columnIndexMap15.get("name").toString());
522            user.setGender(columnIndexMap15.get("gender").toString());
523            user.setAge(columnIndexMap15.get("age").intValue());
524            user.setEmail(columnIndexMap15.get("email").toString());
525            user.setPhone(columnIndexMap15.get("phone").toString());
526        } catch (SQLException e) {
527            throw new RuntimeException("映射结果集失败", e);
528        }
529
530        return user;
531    }
532
533    private static final Map<String, Integer> columnIndexMap16 = new HashMap<>();
534
535    static {
536        columnIndexMap16.put("name", 1);
537        columnIndexMap16.put("gender", 2);
538        columnIndexMap16.put("age", 3);
539        columnIndexMap16.put("email", 4);
540        columnIndexMap16.put("phone", 5);
541    }
542
543    private static User mapResultSetToUser(ResultSet rs) {
544        User user = new User();
545
546        try {
547            user.setName(columnIndexMap16.get("name").toString());
548            user.setGender(columnIndexMap16.get("gender").toString());
549            user.setAge(columnIndexMap16.get("age").intValue());
550            user.setEmail(columnIndexMap16.get("email").toString());
551            user.setPhone(columnIndexMap16.get("phone").toString());
552        } catch (SQLException e) {
553            throw new RuntimeException("映射结果集失败", e);
554        }
555
556        return user;
557    }
558
559    private static final Map<String, Integer> columnIndexMap17 = new HashMap<>();
560
561    static {
562        columnIndexMap17.put("name", 1);
563        columnIndexMap17.put("gender", 2);
564        columnIndexMap17.put("age", 3);
565        columnIndexMap17.put("email", 4);
566        columnIndexMap17.put("phone", 5);
567    }
568
569    private static User mapResultSetToUser(ResultSet rs) {
570        User user = new User();
571
572        try {
573            user.setName(columnIndexMap17.get("name").toString());
574            user.setGender(columnIndexMap17.get("gender").toString());
575            user.setAge(columnIndexMap17.get("age").intValue());
576            user.setEmail(columnIndexMap17.get("email").toString());
577            user.setPhone(columnIndexMap17.get("phone").toString());
578        } catch (SQLException e) {
579            throw new RuntimeException("映射结果集失败", e);
580        }
581
582        return user;
583    }
584
585    private static final Map<String, Integer> columnIndexMap18 = new HashMap<>();
586
587    static {
588        columnIndexMap18.put("name", 1);
589        columnIndexMap18.put("gender", 2);
590        columnIndexMap18.put("age", 3);
591        columnIndexMap18.put("email", 4);
592        columnIndexMap18.put("phone", 5);
593    }
594
595    private static User mapResultSetToUser(ResultSet rs) {
596        User user = new User();
597
598        try {
599            user.setName(columnIndexMap18.get("name").toString());
600            user.setGender(columnIndexMap18.get("gender").toString());
601            user.setAge(columnIndexMap18.get("age").intValue());
602            user.setEmail(columnIndexMap18.get("email").toString());
603            user.setPhone(columnIndexMap18.get("phone").toString());
604        } catch (SQLException e) {
605            throw new RuntimeException("映射结果集失败", e);
606        }
607
608        return user;
609    }
610
611    private static final Map<String, Integer> columnIndexMap19 = new HashMap<>();
612
613    static {
614        columnIndexMap19.put("name", 1);
615        columnIndexMap19.put("gender", 2);
616        columnIndexMap19.put("age", 3);
617        columnIndexMap19.put("email", 4);
618        columnIndexMap19.put("phone", 5);
619    }
620
621    private static User mapResultSetToUser(ResultSet rs) {
622        User user = new User();
623
624        try {
625            user.setName(columnIndexMap19.get("name").toString());
626            user.setGender(columnIndexMap19.get("gender").toString());
627            user.setAge(columnIndexMap19.get("age").intValue());
628            user.setEmail(columnIndexMap19.get("email").toString());
629            user.setPhone(columnIndexMap19.get("phone").toString());
630        } catch (SQLException e) {
631            throw new RuntimeException("映射结果集失败", e);
632        }
633
634        return user;
635    }
636
637    private static final Map<String, Integer> columnIndexMap20 = new HashMap<>();
638
639    static {
640        columnIndexMap20.put("name", 1);
641        columnIndexMap20.put("gender", 2);
642        columnIndexMap20.put("age", 3);
643        columnIndexMap20.put("email", 4);
644        columnIndexMap20.put("phone", 5);
645    }
646
647    private static User mapResultSetToUser(ResultSet rs) {
648        User user = new User();
649
650        try {
651            user.setName(columnIndexMap20.get("name").toString());
652            user.setGender(columnIndexMap20.get("gender").toString());
653            user.setAge(columnIndexMap20.get("age").intValue());
654            user.setEmail(columnIndexMap20.get("email").toString());
655            user.setPhone(columnIndexMap20.get("phone").toString());
656        } catch (SQLException e) {
657            throw new RuntimeException("映射结果集失败", e);
658        }
659
660        return user;
661    }
662
663    private static final Map<String, Integer> columnIndexMap21 = new HashMap<>();
664
665    static {
666        columnIndexMap21.put("name", 1);
667        columnIndexMap21.put("gender", 2);
668        columnIndexMap21.put("age", 3);
669        columnIndexMap21.put("email", 4);
670        columnIndexMap21.put("phone", 5);
671    }
672
673    private static User mapResultSetToUser(ResultSet rs) {
674        User user = new User();
675
676        try {
677            user.setName(columnIndexMap21.get("name").toString());
678            user.setGender(columnIndexMap21.get("gender").toString());
679            user.setAge(columnIndexMap21.get("age").intValue());
680            user.setEmail(columnIndexMap21.get("email").toString());
681            user.setPhone(columnIndexMap21.get("phone").toString());
682        } catch (SQLException e) {
683            throw new RuntimeException("映射结果集失败", e);
684        }
685
686        return user;
687    }
688
689    private static final Map<String, Integer> columnIndexMap22 = new HashMap<>();
690
691    static {
692        columnIndexMap22.put("name", 1);
693        columnIndexMap22.put("gender", 2);
694        columnIndexMap22.put("age", 3);
695        columnIndexMap22.put("email", 4);
696        columnIndexMap22.put("phone", 5);
697    }
698
699    private static User mapResultSetToUser(ResultSet rs) {
700        User user = new User();
701
702        try {
703            user.setName(columnIndexMap22.get("name").toString());
704            user.setGender(columnIndexMap22.get("gender").toString());
705            user.setAge(columnIndexMap22.get("age").intValue());
706            user.setEmail(columnIndexMap22.get("email").toString());
707            user.setPhone(columnIndexMap22.get("phone").toString());
708        } catch (SQLException e) {
709            throw new RuntimeException("映射结果集失败", e);
710        }
711
712        return user;
713    }
714
715    private static final Map<String, Integer> columnIndexMap23 = new HashMap<>();
716
717    static {
718        columnIndexMap23.put("name", 1);
719        columnIndexMap23.put("gender", 2);
720        columnIndexMap23.put("age", 3);
721        columnIndexMap23.put("email", 4);
722        columnIndexMap23.put("phone", 5);
723    }
724
725    private static User mapResultSetToUser(ResultSet rs) {
726        User user = new User();
727
728        try {
729            user.setName(columnIndexMap23.get("name").toString());
730            user.setGender(columnIndexMap23.get("gender").toString());
731            user.setAge(columnIndexMap23.get("age").intValue());
732            user.setEmail(columnIndexMap23.get("email").toString());
733            user.setPhone(columnIndexMap23.get("phone").toString());
734        } catch (SQLException e) {
735            throw new RuntimeException("映射结果集失败", e);
736        }
737
738        return user;
739    }
740
741    private static final Map<String, Integer> columnIndexMap24 = new HashMap<>();
742
743    static {
744        columnIndexMap24.put("name", 1);
745        columnIndexMap24.put("gender", 2);
746        columnIndexMap24.put("age", 3);
747        columnIndexMap24.put("email", 4);
748        columnIndexMap24.put("phone", 5);
749    }
750
751    private static User mapResultSetToUser(ResultSet rs) {
752        User user = new User();
753
754        try {
755            user.setName(columnIndexMap24.get("name").toString());
756            user.setGender(columnIndexMap24.get("gender").toString());
757            user.setAge(columnIndexMap24.get("age").intValue());
758            user.setEmail(columnIndexMap24.get("email").toString());
759            user.setPhone(columnIndexMap24.get("phone").toString());
760        } catch (SQLException e) {
761            throw new RuntimeException("映射结果集失败", e);
762        }
763
764        return user;
765    }
766
767    private static final Map<String, Integer> columnIndexMap25 = new HashMap<>();
768
769    static {
770        columnIndexMap25.put("name", 1);
771        columnIndexMap25.put("gender", 2);
772        columnIndexMap25.put("age", 3);
773        columnIndexMap25.put("email", 4);
774        columnIndexMap25.put("phone", 5);
775    }
776
777    private static User mapResultSetToUser(ResultSet rs) {
778        User user = new User();
779
780        try {
781            user.setName(columnIndexMap25.get("name").toString());
782            user.setGender(columnIndexMap25.get("gender").toString());
783            user.setAge(columnIndexMap25.get("age").intValue());
784            user.setEmail(columnIndexMap25.get("email").toString());
785            user.setPhone(columnIndexMap25.get("phone").toString());
786        } catch (SQLException e) {
787            throw new RuntimeException("映射结果集失败", e);
788        }
789
790        return user;
791    }
792
793    private static final Map<String, Integer> columnIndexMap26 = new HashMap<>();
794
795    static {
796        columnIndexMap26.put("name", 1);
797        columnIndexMap26.put("gender", 2);
798        columnIndexMap26.put("age", 3);
799        columnIndexMap26.put("email", 4);
800        columnIndexMap26.put("phone", 5);
801    }
802
803    private static User mapResultSetToUser(ResultSet rs) {
804        User user = new User();
805
806        try {
807            user.setName(columnIndexMap26.get("name").toString());
808            user.setGender(columnIndexMap26.get("gender").toString());
809            user.setAge(columnIndexMap26.get("age").intValue());
810            user.setEmail(columnIndexMap26.get("email").toString());
811            user.setPhone(columnIndexMap26.get("phone").toString());
812        } catch (SQLException e) {
813            throw new RuntimeException("映射结果集失败", e);
814        }
815
816        return user;
817    }
818
819    private static final Map<String, Integer> columnIndexMap27 = new HashMap<>();
820
821    static {
822        columnIndexMap27.put("name", 1);
823        columnIndexMap27.put("gender", 2);
824        columnIndexMap27.put("age", 3);
825        columnIndexMap27.put("email", 4);
826        columnIndexMap27.put("phone", 5);
827    }
828
829    private static User mapResultSetToUser(ResultSet rs) {
830        User user = new User();
831
832        try {
833            user.setName(columnIndexMap27.get("name").toString());
834            user.setGender(columnIndexMap27.get("gender").toString());
835            user.setAge(columnIndexMap27.get("age").intValue());
836            user.setEmail(columnIndexMap27.get("email").toString());
837            user.setPhone(columnIndexMap27.get("phone").toString());
838        } catch (SQLException e) {
839            throw new RuntimeException("映射结果集失败", e);
840        }
841
842        return user;
843    }
844
845    private static final Map<String, Integer> columnIndexMap28 = new HashMap<>();
846
847    static {
848        columnIndexMap28.put("name", 1);
849        columnIndexMap28.put("gender", 2);
850        columnIndexMap28.put("age", 3);
851        columnIndexMap28.put("email", 4);
852        columnIndexMap28.put("phone", 5);
853    }
854
855    private static User mapResultSetToUser(ResultSet rs) {
856        User user = new User();
857
858        try {
859            user.setName(columnIndexMap28.get("name").toString());
860            user.setGender(columnIndexMap28.get("gender").toString());
861            user.setAge(columnIndexMap28.get("age").intValue());
862            user.setEmail(columnIndexMap28.get("email").toString());
863            user.setPhone(columnIndexMap28.get("phone").toString());
864        } catch (SQLException e) {
865            throw new RuntimeException("映射结果集失败", e);
866        }
867
868        return user;
869    }
870
871    private static final Map<String, Integer> columnIndexMap29 = new HashMap<>();
872
873    static {
874        columnIndexMap29.put("name", 1);
875        columnIndexMap29.put("gender", 2);
876        columnIndexMap29.put("age", 3);
877        columnIndexMap29.put("email", 4);
878        columnIndexMap29.put("phone", 5);
879    }
880
881    private static User mapResultSetToUser(ResultSet rs) {
882        User user = new User();
883
884        try {
885            user.setName(columnIndexMap29.get("name").toString());
886            user.setGender(columnIndexMap29.get("gender").toString());
887            user.setAge(columnIndexMap29.get("age").intValue());
888            user.setEmail(columnIndexMap29.get("email").toString());
889            user.setPhone(columnIndexMap29.get("phone").toString());
890        } catch (SQLException e) {
891            throw new RuntimeException("映射结果集失败", e);
892        }
893
894        return user;
895    }
896
897    private static final Map<String, Integer> columnIndexMap30 = new HashMap<>();
898
899    static {
900        columnIndexMap30.put("name", 1);
901        columnIndexMap30.put("gender", 2);
902        columnIndexMap30.put("age", 3);
903        columnIndexMap30.put("email", 4);
904        columnIndexMap30.put("phone", 5);
905    }
906
907    private static User mapResultSetToUser(ResultSet rs) {
908        User user = new User();
909
910        try {
911            user.setName(columnIndexMap30.get("name").toString());
912            user.setGender(columnIndexMap30.get("gender").toString());
913            user.setAge(columnIndexMap30.get("age").intValue());
914            user.setEmail(columnIndexMap30.get("email").toString());
915            user.setPhone(columnIndexMap30.get("phone").toString());
916        } catch (SQLException e) {
917            throw new RuntimeException("映射结果集失败", e);
918        }
919
920        return user;
921    }
922
923    private static final Map<String, Integer> columnIndexMap31 = new HashMap<>();
924
925    static {
926        columnIndexMap31.put("name", 1);
927        columnIndexMap31.put("gender", 2);
928        columnIndexMap31.put("age", 3);
929        columnIndexMap31.put("email", 4);
930        columnIndexMap31.put("phone", 5);
931    }
932
933    private static User mapResultSetToUser(ResultSet rs) {
934        User user = new User();
935
936        try {
937            user.setName(columnIndexMap31.get("name").toString());
938            user.setGender(columnIndexMap31.get("gender").toString());
939            user.setAge(columnIndexMap31.get("age").intValue());
940            user.setEmail(columnIndexMap31.get("email").toString());
941            user.setPhone(columnIndexMap31.get("phone").toString());
942        } catch (SQLException e) {
943            throw new RuntimeException("映射结果集失败", e);
944        }
945
946        return user;
947    }
948
949    private static final Map<String, Integer> columnIndexMap32 = new HashMap<>();
950
951    static {
952        columnIndexMap32.put("name", 1);
953        columnIndexMap32.put("gender", 2);
954        columnIndexMap32.put("age", 3);
955        columnIndexMap32.put("email", 4);
956        columnIndexMap32.put("phone", 5);
957    }
958
959    private static User mapResultSetToUser(ResultSet rs) {
960        User user = new User();
961
962        try {
963            user.setName(columnIndexMap32.get("name").toString());
964            user.setGender(columnIndexMap32.get("gender").toString());
965            user.setAge(columnIndexMap32.get("age").intValue());
966            user.setEmail(columnIndexMap32.get("email").toString());
967            user.setPhone(columnIndexMap32.get("phone").toString());
968        } catch (SQLException e) {
969            throw new RuntimeException("映射结果集失败", e);
970        }
971
972        return user;
973    }
974
975    private static final Map<String, Integer> columnIndexMap33 = new HashMap<>();
976
977    static {
978        columnIndexMap33.put("name", 1);
979        columnIndexMap33.put("gender", 2);
980        columnIndexMap33.put("age", 3);
981        columnIndexMap33.put("email", 4);
982        columnIndexMap33.put("phone", 5);
983    }
984
985    private static User mapResultSetToUser(ResultSet rs) {
986        User user = new User();
987
988        try {
989            user.setName(columnIndexMap33.get("name").toString());
990            user.setGender(columnIndexMap33.get("gender").toString());
991            user.setAge(columnIndexMap33.get("age").intValue());
992            user.setEmail(columnIndexMap33.get("email").toString());
993            user.setPhone(columnIndexMap33.get("phone").toString());
994        } catch (SQLException e) {
995            throw new RuntimeException("映射结果集失败", e);
996        }
997
998        return user;
999    }
1000
1001    private static final Map<String, Integer> columnIndexMap34 = new HashMap<>();
1002
1003    static {
1004        columnIndexMap34.put("name", 1);
1005        columnIndexMap34.put("gender", 2);
1006        columnIndexMap34.put("age", 3);
1007        columnIndexMap34.put("email", 4);
1008        columnIndexMap34.put("phone", 5);
1009    }
1010
1011    private static User mapResultSetToUser(ResultSet rs) {
1012        User user = new User();
1013
1014        try {
1015            user.setName(columnIndexMap34.get("name").toString());
1016            user.setGender(columnIndexMap34.get("gender").toString());
1017            user.setAge(columnIndexMap34.get("age").intValue());
1018            user.setEmail(columnIndexMap34.get("email").toString());
1019            user.setPhone(columnIndexMap34.get("phone").toString());
1020        } catch (SQLException e) {
1021            throw new RuntimeException("映射结果集失败", e);
1022        }
1023
1024        return user;
1025    }
1026
1027    private static final Map<String, Integer> columnIndexMap35 = new HashMap<>();
1028

```

```
119         int affectedRows = stmt.executeUpdate();
120
121         if (affectedRows == 0) {
122             throw new IllegalArgumentException("用户不存在");
123         }
124     } catch (SQLException e) {
125         throw new RuntimeException("更新用户失败", e);
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
137         return user;
138     }
```

## 服务层改造

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

```
2 package io.codescience.service;
3
4 import io.codescience.model.User;
5 import io.codescience.repository.UserRepository;
6 import io.codescience.repository.UserRepositoryImpl;
7
8 import java.util.List;
9
10 public class UserService {
11     private final UserRepository repository;
12
13     public UserService() {
14         this.repository = new UserRepositoryImpl();
15     }
16
17     public User addUser(User user) {
18         return repository.save(user);
19     }
20
21     public User getUser(int id) {
22         return repository.findById(id)
23             .orElseThrow(() -> new IllegalArgumentException("用户不存在"));
24     }
25
26     public List<User> listUsers() {
27         return repository.findAll();
28     }
29
30     public void deleteUser(int id) {
31         if (!repository.findById(id).isPresent()) {
32             throw new IllegalArgumentException("用户不存在");
33         }
34         repository.deleteById(id);
35     }
36
37     public void updateUser(int id, User user) {
38         if (!repository.findById(id).isPresent()) {
39             throw new IllegalArgumentException("用户不存在");
```

```

40     }
41     user.setId(id);
42     repository.update(user);
43 }
44 }
```

## 数据库初始化脚本

### #04. Docker 数据管理 (Volumes) (可选)

```

1 CREATE DATABASE IF NOT EXISTS user_management;
2 USE user_management;
3
4 CREATE TABLE users (
5   id INT AUTO_INCREMENT PRIMARY KEY,
6   name VARCHAR(100) NOT NULL,
7   gender ENUM('Male','Female','Other') NOT NULL,
8   age INT,
9   email VARCHAR(100) UNIQUE,
10  phone VARCHAR(15)
11 );
```

## 第三阶段：换底层数据库访问逻辑

### 一、MyBatis Repository的实现

```

1 <dependencies>
2   <!-- MySQL JDBC Driver -->
3   <dependency>
4     <groupId>mysql</groupId>
5     <artifactId>mysql-connector-java</artifactId>
6     <version>8.0.33</version>
7   </dependency>
8
9   <!-- MyBatis -->
10  <dependency>
11    <groupId>org.mybatis</groupId>
12    <artifactId>mybatis</artifactId>
13    <version>3.5.13</version>
14  </dependency>
15 </dependencies>
16
```

### 2. 创建数据库配置

创建 `DatabaseConfig.java` :

```

1 package io.codescience.config;
2
3 public class DatabaseConfig {
4   public static final String URL = "jdbc:mysql://localhost:3306/user_management?";
5   useSSL=false&serverTimezone=UTC";
6   public static final String USER = "root";
7   public static final String PASSWORD = "your_password";
8
9   static {
10     try {
11       Class.forName("com.mysql.cj.jdbc.Driver");
12     } catch (ClassNotFoundException e) {
13       throw new RuntimeException("MySQL JDBC Driver not found", e);
14     }
15 }
```

### 3. 创建 MyBatis 配置文件

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

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3   PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6   <environments default="development">
7     <environment id="development">
8       <transactionManager type="JDBC"/>
9       <dataSource type="POOLED">
10        <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
11        <property name="url"
12          value="jdbc:mysql://localhost:3306/user_management?
13          useSSL=false&serverTimezone=UTC"/>
14        <property name="username" value="root"/>
15        <property name="password" value="your_password"/>
16      </dataSource>
17    </environment>
18  </environments>
19  <mappers>
20    <mapper resource="mapper/UserMapper.xml"/>
21  </mappers>
22 </configuration>
23

```

### 4. 创建 Mapper 接口

创建 `UserMapper.java` :

```
1 package io.codescience.repository;
2
3 import io.codescience.model.User;
4 import java.util.List;
5 import java.util.Optional;
6
7 public interface UserMapper {
8     void insert(User user);
9     Optional<User> findById(int id);
10    List<User> findAll();
11    void deleteById(int id);
12    void update(User user);
13 }
14
```

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

## 5. 创建 Mapper XML

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

```

2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="io.codescience.repository.UserMapper">
6      <resultMap id="userMap" type="io.codescience.model.User">
7          <id property="id" column="id"/>
8          <result property="name" column="name"/>
9          <result property="gender" column="gender"/>
10         <result property="age" column="age"/>
11         <result property="email" column="email"/>
12         <result property="phone" column="phone"/>
13     </resultMap>
14
15     <insert id="insert" parameterType="io.codescience.model.User"
useGeneratedKeys="true" keyProperty="id">
16         INSERT INTO users (name, gender, age, email, phone)
17         VALUES (#{name}, #{gender}, #{age}, #{email}, #{phone})
18     </insert>
19
20     <select id="findById" resultMap="userMap">
21         SELECT * FROM users WHERE id = #{id}
22     </select>
23
24     <select id="findAll" resultMap="userMap">
25         SELECT * FROM users
26     </select>
27
28     <delete id="deleteById">
29         DELETE FROM users WHERE id = #{id}
30     </delete>
31
32     <update id="update" parameterType="io.codescience.model.User">
33         UPDATE users
34         SET name = #{name},
35             gender = #{gender},
36             age = #{age},
37             email = #{email},
38             phone = #{phone}
39         WHERE id = #{id}

```

```

40     </update>
41     </mapper>
42

```

## 6. 实现 Repository

创建 `MyBatisUserRepository.java` :

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

```
2
3 import io.codescience.model.User;
4 import org.apache.ibatis.io.Resources;
5 import org.apache.ibatis.session.SqlSession;
6 import org.apache.ibatis.session.SqlSessionFactory;
7 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
8
9 import java.io.IOException;
10 import java.io.InputStream;
11 import java.util.List;
12 import java.util.Optional;
13
14 public class MyBatisUserRepository implements UserRepository {
15     private final SqlSessionFactory sqlSessionFactory;
16
17     public MyBatisUserRepository() {
18         try {
19             String resource = "mybatis-config.xml";
20             InputStream inputStream = Resources.getResourceAsStream(resource);
21             sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
22         } catch (IOException e) {
23             throw new RuntimeException("初始化 MyBatis 失败", e);
24         }
25     }
26
27     @Override
28     public User save(User user) {
29         try (SqlSession session = sqlSessionFactory.openSession()) {
30             UserMapper mapper = session.getMapper(UserMapper.class);
31             mapper.insert(user);
32             session.commit();
33             return mapper.findById(user.getId())
34                 .orElseThrow(() -> new RuntimeException("保存用户后无法获取用户信息"));
35         }
36     }
37
38     @Override
39     public Optional<User> findById(int id) {
```

```

40     try (SqlSession session = sqlSessionFactory.openSession()) {
41         UserMapper mapper = session.getMapper(UserMapper.class);
42         return mapper.findById(id);
43     }
44 }
45
46 @Override
47 public List<User> findAll() {
48     try (SqlSession session = sqlSessionFactory.openSession()) {
49         UserMapper mapper = session.getMapper(UserMapper.class);
50         return mapper.findAll();
51     }
52 }
53
54 @Override
55 public void deleteById(int id) {
56     try (SqlSession session = sqlSessionFactory.openSession()) {
57         UserMapper mapper = session.getMapper(UserMapper.class);
58         mapper.deleteById(id);
59         session.commit();
60     }
61 }
62
63 @Override
64 public void update(User user) {
65     try (SqlSession session = sqlSessionFactory.openSession()) {
66         UserMapper mapper = session.getMapper(UserMapper.class);
67         mapper.update(user);
68         session.commit();
69     }
70 }
71 }
72 
```

```

1 public class UserService {
2     private final UserRepository repository;
3
4     public UserService() {
5         this.repository = new MyBatisUserRepository();
6     }
7     // ... 其他方法保持不变
8 }
9 
```

### 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

## 7. 修改 Service 层

修改 `UserService.java` :

### 3. 运行程序

### 4. 测试基本操作:

```
1 user add --name Jessica --gender Female --age 27 --email 1111 --phone 11111111
2 user list
3 user update --id 1 --name Jessica2
4 user delete --id 1
5
```

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

```
1 User user = new User();
2 user.setName("张三");
3 user.setEmail("zhangsan@example.com");
4 userRepository.save(user);
5
```

#### 4.2 查询用户

```
1 Optional<User> user = userRepository.findById(1);
2 user.ifPresent(u -> System.out.println(u.getName()));
3
```

#### 4.3 更新用户

```
1 User user = userRepository.findById(1).orElseThrow();
2 user.setName("李四");
3 userRepository.update(user);
4
```

#### 4.4 删除用户

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

## 分层架构设计 (仅仅是介绍)

### 完整项目结构

```

1 src/
2   └── main/
3     ├── java/
4       ├── model/User.java
5       ├── repository/UserRepository.java
6       ├── service/UserService.java
7       └── ui/CliUI.java
8     └── exception/
9       ├── DataAccessException.java
10      └── ValidationException.java
11   └── resources/
12     └── db/
13       └── migration/
14         └── V1__Create_users_table.sql

```

## 分层调用流程

```

1 sequenceDiagram
2   participant UI as CLI界面
3   participant Service as UserService
4   participant Repository as UserRepository
5   participant DB as MySQL
6
7   UI->>Service: 执行user add命令
8   Service->>Repository: save(user)
9   Repository->>DB: INSERT INTO users...
10  DB-->>Repository: 返回生成的ID
11  Repository-->>Service: 返回User对象
12  Service-->>UI: 显示操作结果

```

## 关键设计原则

### 1. 依赖倒置

```

1 // Service层通过接口依赖Repository
2 public interface UserRepository {
3   User save(User user);
4   User findById(int id);
5   // 其他方法...
6 }
7
8 // 实现类
9 public class JdbcUserRepository implements UserRepository {
10   // JDBC具体实现...
11 }

```

### 1. 异常处理分层

```

1 // 自定义异常类
2 public class DataAccessException extends RuntimeException {
3   public DataAccessException(String message, Throwable cause) {
4     super(message, cause);
5   }
6 }
7
8 // Service层抛出业务异常
9 public class ValidationException extends RuntimeException {
10   public ValidationException(String message) {
11     super(message);
12   }
13 }

```

### 1. 输入验证增强

```

1 public User addUser(User user) {
2     validateEmailFormat(user.getEmail());
3     validatePhoneFormat(user.getPhone());
4     return repository.save(user);
5 }
6
7 private void validateEmailFormat(String email) {
8     if (!email.matches("^[A-Za-z0-9+_.-]+@[.]+$"))
9         { throw new ValidationException("邮箱格式不正确");}
10    }
11 }
```

## 扩展功能建议

### 1. 日志记录

```

1 // 添加日志依赖
2 <dependency>
3     <groupId>org.slf4j</groupId>
4     <artifactId>slf4j-api</artifactId>
5     <version>1.7.36</version>
6 </dependency>
7
8 // 在关键位置添加日志
9 private static final Logger logger = LoggerFactory.getLogger(UserService.class);
10
11 public User addUser(User user) {
12     logger.info("添加用户: {}", user.getName());
13     // ...
14 }
```

### 1. 分页查询优化

```

1 public List<User> listUsers(int page, int pageSize) {
2     String sql = "SELECT * FROM users LIMIT ? OFFSET ?";
3     int offset = (page - 1) * pageSize;
4     // 执行分页查询...
5 }
```

### 1. 连接池配置

```

1 // 使用HikariCP连接池
2 HikariConfig config = new HikariConfig();
3 config.setJdbcUrl(DatabaseConfig.URL);
4 config.setUsername(DatabaseConfig.USER);
5 config.setPassword(DatabaseConfig.PASSWORD);
6 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：

```
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。

## ✓ 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)

## 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）。

```
2  public class User {  
3      private Integer id;  
4      private String name;  
5      private String gender;  
6      private Integer age;  
7      private String email;  
8      private String phone;  
9  
10     // 构造方法  
11     public User() {}  
12  
13     public User(Integer id, String name, String gender, Integer age, String email,  
14         String phone) {  
15         this.id = id;  
16         this.name = name;  
17         this.gender = gender;  
18         this.age = age;  
19         this.email = email;  
20         this.phone = phone;  
21     }  
22  
23     // Getter & Setter  
24     public Integer getId() { return id; }  
25     public void setId(Integer id) { this.id = id; }  
26  
27     public String getName() { return name; }  
28     public void setName(String name) { this.name = name; }  
29  
30     public String getGender() { return gender; }  
31     public void setGender(String gender) { this.gender = gender; }  
32  
33     public Integer getAge() { return age; }  
34     public void setAge(Integer age) { this.age = age; }  
35  
36     public String getEmail() { return email; }  
37     public void setEmail(String email) { this.email = email; }  
38  
39     public String getPhone() { return phone; }  
40     public void setPhone(String phone) { this.phone = phone; }  
41 }
```

## 2. 创建 Controller (controller/UserController.java)

```

2 import io.codescience.model.User;
3 import org.springframework.http.ResponseEntity;
4 import org.springframework.web.bind.annotation.*;
5
6 import java.util.*;
7
8 @RestController
9 @RequestMapping("/api/users")
10 public class UserController {
11
12     // 模拟内存数据存储
13     private final Map<Integer, User> userMap = new HashMap<>();
14     private int currentId = 1;
15
16     // 获取所有用户
17     @GetMapping
18     public ResponseEntity<List<User>> getAllUsers() {
19         return ResponseEntity.ok(new ArrayList<>(userMap.values()));
20     }
21
22     // 创建用户
23     @PostMapping
24     public ResponseEntity<User> createUser(@RequestBody User user) {
25         user.setId(currentId++);
26         userMap.put(user.getId(), user);
27         return ResponseEntity.ok(user);
28     }
29
30     // 获取指定 ID 用户
31     @GetMapping("/{id}")
32     public ResponseEntity<User> getUserById(@PathVariable Integer id) {
33         User user = userMap.get(id);
34         if (user != null) {
35             return ResponseEntity.ok(user);
36         }
37         return ResponseEntity.notFound().build();
38     }
39

```

```

40     // 更新用户
41     @PutMapping("/{id}")
42     public ResponseEntity<?> updateUser(@PathVariable Integer id, @RequestBody User updatedUser) {
43         User existing = userMap.get(id);
44         if (existing == null) {
45             return ResponseEntity.notFound().build();
46         }
47         updatedUser.setId(id);
48         userMap.put(id, updatedUser);
49         return ResponseEntity.ok().build();
50     }
51
52     // 删除用户
53     @DeleteMapping("/{id}")
54     public ResponseEntity<?> deleteUser(@PathVariable Integer id) {
55         if (userMap.remove(id) != null) {
56             return ResponseEntity.ok().build();
57         }
58         return ResponseEntity.notFound().build();
59     }
60 }
61

```

### 3. 主程序入口 (UserManagementApplication.java)

```
1  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class UserManagementApplication {  
8     public static void main(String[] args) {  
9         SpringApplication.run(UserManagementApplication.class, args);  
10    }  
11 }  
12
```

#### 4. application.properties (任选其一)

```
1 server:  
2 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:**

```

1 [ 
2   {
3     "id": 1,
4     "name": "张三",
5     "gender": "男",
6     "age": 30,
7     "email": "zhangsan@example.com",
8     "phone": "13800000000"
9   }
10 ]
11

```

## 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	用户名	最长 50 字符
gender	string	是	性别	最长 10 字符
age	integer	是	年龄	$\geq 0$
email	string	是	邮箱	email 格式
phone	string	是	电话	最长 20 字符

1 openapi: 3.0.0

## 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

```

2 info:
3   title: 用户管理系统 API
4   version: 1.0.0
5   description: 用户管理系统的RESTful API接口文档
6
7 servers:
8   - url: http://localhost:8080
9   description: 本地开发服务器
10
11 paths:
12   /api/users:
13     get:
14       summary: 获取所有用户
15       description: 返回所有用户的列表
16       responses:
17         '200':
18           description: 成功获取用户列表
19           content:
20             application/json:
21               schema:
22                 type: array
23                 items:
24                   $ref: '#/components/schemas/User'
25     post:
26       summary: 创建用户
27       description: 创建一个新的用户
28       requestBody:
29         required: true
30         content:
31           application/json:
32             schema:
33               $ref: '#/components/schemas/User'
34       responses:
35         '200':
36           description: 用户创建成功
37           content:
38             application/json:
39               schema:

```

```

40           $ref: '#/components/schemas/User'
41           '400':
42             description: 请求参数验证失败
43
44   /api/users/{id}:
45     get:
46       summary: 获取用户
47       description: 根据ID获取用户信息
48       parameters:
49         - name: id
50           in: path
51           required: true
52           schema:
53             type: integer
54       responses:
55         '200':
56           description: 成功获取用户信息
57           content:
58             application/json:
59               schema:
60                 $ref: '#/components/schemas/User'
61         '404':
62           description: 用户不存在
63     put:
64       summary: 更新用户
65       description: 更新指定ID的用户信息
66       parameters:
67         - name: id
68           in: path
69           required: true
70           schema:
71             type: integer
72       requestBody:
73         required: true
74         content:
75           application/json:
76             schema:
77               $ref: '#/components/schemas/User'
78       responses:
79         '200':

```

```

80     description: 用户更新成功
81     '400':
82         description: 请求参数验证失败
83     '404':
84         description: 用户不存在
85 delete:
86     summary: 删除用户
87     description: 删除指定ID的用户
88     parameters:
89         - name: id
90             in: path
91             required: true
92             schema:
93                 type: integer
94     responses:
95         '200':
96             description: 用户删除成功
97         '404':
98             description: 用户不存在
99
100 components:
101 schemas:
102     User:
103         type: object
104         required:
105             - name
106             - gender
107             - age
108             - email
109             - phone
110         properties:
111             id:
112                 type: integer
113                 description: 用户ID
114             name:
115                 type: string
116                 description: 用户姓名
117                 maxLength: 50
118             gender:
119                 type: string

```

```

120         description: 用户性别
121         maxLength: 10
122         age:
123             type: integer
124             description: 用户年龄
125             minimum: 0
126         email:
127             type: string
128             description: 用户邮箱
129             format: email
130             maxLength: 100
131         phone:
132             type: string
133             description: 用户电话
134             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

```
1 # 创建脚手架程序
2 npx create-react-app project-dashboard
3 cd project-dashboard
4
5 # 安装依赖
6 npm install antd @ant-design/charts axios
7
8 # 确保脚手架程序可以执行
9 npm start
```

### 2. 项目结构

```
1 src/
2   ├── components/
3   |   ├── KpiCards.js      # 指标卡片
4   |   ├── OrderChart.js    # 订单折线图
5   ├── services/
6   |   └── api.js          # 数据请求封装
7   └── App.js              # 主组件
8     └── index.js
```

### 3. 数据请求封装 (services/api.js)

```
1 import axios from "axios";
2
3 export const fetchData = async () => {
4   // 模拟API数据（实际替换为真实接口）
5   return {
6     users: 12480,
7     orders: 1567,
8     revenue: 285430,
9     serverLoad: 65,
10    dailyOrders: [
11      { date: "06-01", count: 120 },
12      { date: "06-02", count: 135 },
13      { date: "06-03", count: 150 },
14      { date: "06-04", count: 125 },
15      { date: "06-05", count: 140 },
16      { date: "06-06", count: 130 },
17      { date: "06-07", count: 155 },
18      { date: "06-08", count: 145 },
19      { date: "06-09", count: 160 },
20      { date: "06-10", count: 150 },
21      { date: "06-11", count: 165 },
22      { date: "06-12", count: 170 },
23    ],
24  };
25};
```

## 4. 指标卡片组件 (components/KpiCards.js)

```
1 import { Card, Statistic, Tag } from "antd";
2
3 export default ({ data }) => {
4   if (!data) {
5     return null;
6   }
7
8   return (
9     <div
10       className="kpi-cards"
11       style={{
12         display: "flex",
13         flexWrap: "wrap",
14         gap: "16px", // 设置卡片间距
15       }}>
16
17     <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
18       <Statistic title="用户总数" value={data.users} />
19       <Tag color="green">+12%周环比</Tag>
20     </Card>
21
22     <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
23       <Statistic title="订单量" value={data.orders} />
24       <Tag color="red">+12%周环比</Tag>
25     </Card>
26
27     <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
28       <Statistic title="营收额" value={data.revenue} />
29       <Tag color="green">+12%周环比</Tag>
30     </Card>
31
32     <Card variant="borderless" style={{ flex: 1, minWidth: "200px" }}>
33       <Statistic title="完成率" value={data.serverLoad} />
34       <Tag color="orange">+12%周环比</Tag>
35     </Card>
36   </div>
37 );
38 };
39 
```

## 5. 图表组件集成 (components/OrderChart.js)

```
1 import { Line } from "@ant-design/charts";
2
3 export default ({ data }) => {
4   if (!data || !data.dailyOrders) {
5     return null;
6   }
7
8   const config = {
9     data: data.dailyOrders,
10    xField: "date",
11    yField: "count",
12    point: { size: 5 },
13  };
14  return <Line {...config} />;
15 };
16 
```

## 6. 主组件逻辑 (App.js)

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

```
2 import { useEffect, useState } from "react";
3 import { fetchData } from "./services/api";
4 import KpiCards from "./components/KpiCards";
5 import OrderChart from "./components/OrderChart";
6
7 export default () => {
8   const [stats, setStats] = useState(null);
9   const [loading, setLoading] = useState(true);
10
11   const loadData = async () => {
12     setLoading(true);
13     try {
14       const data = await fetchData();
15       setStats(data);
16     } catch (error) {
17       console.error("加载数据失败:", error);
18     } finally {
19       setLoading(false);
20     }
21   };
22
23   useEffect(() => {
24     loadData();
25   }, []);
26
27   return (
28     <div style={{ padding: 24 }}>
29       <Row gutter={[16, 16]}>
30         <Col span={24}>
31           <DatePicker.RangePicker />
32           <Button onClick={loadData} style={{ marginLeft: 16 }}>
33             刷新数据
34           </Button>
35         </Col>
36
37         <Col span={24}>
38           <Spin spinning={loading}>
39             <KpiCards data={stats} />
40           </Spin>
41         </Col>
42     </div>
43   );
44 }
```

```
42
43     <Col span={24}>
44         <Spin spinning={loading}>
45             <OrderChart data={stats} />
46         </Spin>
47     </Col>
48 </Row>
49 </div>
50 );
51 };
```

## 7. 导入Ant Design

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

## 8. 测试验证

```
1 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) :

```
1 <Statistic title="用户总数" value={data.users} />
2 <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) :

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

◦ 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:

```
1 export const fetchData = async () => {
2   return { ... }; // 模拟数据层
3 };
```

◦ 知识点: 异步请求处理 (`async/await`) 、错误边界处理 ( )

- 加载状态管理:

```
1 <Spin spinning={loading}> // 全局加载指示器
2   <KpiCards data={stats} />
3 </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)

```
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：

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

### 1. 安装依赖：

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

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

```
1 import 'antd/dist/reset.css';
2 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`)

```
1 import { v4 as uuidv4 } from 'uuid';
2
3 let users = [
4   { id: '1', name: '张三', gender: '男', age: 25, email: 'zhang@example.com', phone: '13800138000' },
5   { id: '2', name: '李四', gender: '女', age: 30, email: 'li@example.com', phone: '13900139000' }
6 ];
7
8 export const mockUserApi = {
9   getUsers: () => new Promise(resolve =>
10     setTimeout(() => resolve({ data: users }), 500),
11
12   createUser: (user) => new Promise(resolve => {
13     setTimeout(() => {
14       const newUser = { ...user, id: uuidv4() };
15       users = [...users, newUser];
16       resolve({ data: newUser });
17     }, 500);
18   }),
19
20   updateUser: (id, user) => new Promise(resolve => {
21     setTimeout(() => {
22       users = users.map(u => u.id === id ? { ...u, ...user } : u);
23       resolve({ data: { id, ...user } });
24     }, 500);
25   }),
26
27   deleteUser: (id) => new Promise(resolve => {
28     setTimeout(() => {
29       users = users.filter(u => u.id !== id);
30       resolve({ data: { id } });
31     }, 500);
32   })
33};
```

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

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

```
2 import { Table, Button, Modal, Form, Input, InputNumber, message } from 'antd';
3 import { PlusOutlined } from '@ant-design/icons';
4 import { mockUserApi } from '../mock/userApi';
5
6 const UserManagement = () => {
7   const [users, setUsers] = useState([]);
8   const [loading, setLoading] = useState(false);
9   const [visible, setVisible] = useState(false);
10  const [currentUser, setCurrentUser] = useState(null);
11  const [form] = Form.useForm();
12
13  const columns = [
14    { title: '姓名', dataIndex: 'name', key: 'name' },
15    { title: '性别', dataIndex: 'gender', key: 'gender' },
16    { title: '年龄', dataIndex: 'age', key: 'age' },
17    { title: '邮箱', dataIndex: 'email', key: 'email' },
18    { title: '电话', dataIndex: 'phone', key: 'phone' },
19    {
20      title: '操作',
21      key: 'action',
22      render: (_, record) => (
23        <>
24          <Button type="link" onClick={() => handleEdit(record)}>编辑</Button>
25          <Button type="link" danger onClick={() => handleDelete(record.id)}>删除</Button>
26        </>
27      ),
28    },
29  ];
30
31  const fetchUsers = async () => {
32    setLoading(true);
33    try {
34      const response = await mockUserApi.getUsers();
35      setUsers(response.data);
36    } catch (error) {
37      message.error('获取用户列表失败');
38    } finally {
39      setLoading(false);
40    }
41  };
42
43  const handleEdit = (record) => {
44    setCurrentUser(record);
45    setVisible(true);
46  };
47
48  const handleDelete = (id) => {
49    const confirm = Modal.confirm({
50      title: '确认删除',
51      content: '您确定要删除该用户吗？',
52      okText: '确定',
53      cancelText: '取消',
54      onOk: () => {
55        mockUserApi.deleteUser(id).then(() => {
56          setUsers(users.filter((user) => user.id !== id));
57        });
58      },
59    });
60  };
61
62  return (
63    <Table
64      columns={columns}
65      dataSource={users}
66      loading={loading}
67      rowKey="id"
68      onRow={(record) => ({
69        onClick: () => handleEdit(record),
70        onContextMenu: () => handleDelete(record.id),
71      })}
72    />
73  );
74}
```

```

40     }
41   };
42
43   useEffect(() => {
44     fetchUsers();
45   }, []);
46
47   const handleCreate = () => {
48     form.resetFields();
49     setCurrentUser(null);
50     setVisible(true);
51   };
52
53   const handleEdit = (user) => {
54     form.setFieldsValue(user);
55     setCurrentUser(user);
56     setVisible(true);
57   };
58
59   const handleDelete = async (id) => {
60     Modal.confirm({
61       title: '确认删除',
62       content: '确定要删除该用户吗？',
63       onOk: async () => {
64         await mockUserApi.deleteUser(id);
65         message.success('删除成功');
66         fetchUsers();
67       },
68     });
69   };
70
71   const handleSubmit = async () => {
72     try {
73       const values = await form.validateFields();
74       if (currentUser) {
75         await mockUserApi.updateUser(currentUser.id, values);
76         message.success('更新成功');
77       } else {
78         await mockUserApi.createUser(values);
79         message.success('创建成功');
80       }
81     } catch (error) {
82       console.error('提交失败:', error);
83     }
84   };

```

```

80     }
81     setVisible(false);
82     fetchUsers();
83   } catch (error) {
84     console.error('提交失败:', error);
85   }
86 };
87
88 return (
89   <div style={{ padding: 24 }}>
90     <div style={{ marginBottom: 16 }}>
91       <Button
92         type="primary"
93         icon={<PlusOutlined />}
94         onClick={handleCreate}
95       >
96         添加用户
97       </Button>
98     </div>
99
100    <Table
101      columns={columns}
102      dataSource={users}
103      rowKey="id"
104      loading={loading}
105      pagination={{ pageSize: 10 }}
106    />
107
108    <Modal
109      title={currentUser ? "编辑用户" : "创建用户"}
110      open={visible}
111      onOk={handleSubmit}
112      onCancel={() => setVisible(false)}
113    >
114      <Form form={form} layout="vertical">
115        <Form.Item
116          name="name"
117          label="姓名"
118          rules={[{ required: true, message: '请输入姓名' }]}
119        >

```

```

120     <Input placeholder="张三" maxLength={50} />
121   </Form.Item>
122
123   <Form.Item
124     name="gender"
125     label="性别"
126     rules={[{ required: true, message: '请输入性别' }]}
127   >
128     <Input placeholder="男/女" maxLength={10} />
129   </Form.Item>
130
131   <Form.Item
132     name="age"
133     label="年龄"
134     rules={[{ required: true, message: '请输入年龄' }]}
135   >
136     <InputNumber min={0} style={{ width: '100%' }} />
137   </Form.Item>
138
139   <Form.Item
140     name="email"
141     label="邮箱"
142     rules={[
143       { required: true, message: '请输入邮箱' },
144       { type: 'email', message: '邮箱格式不正确' }
145     ]}
146   >
147     <Input placeholder="example@domain.com" maxLength={100} />
148   </Form.Item>
149
150   <Form.Item
151     name="phone"
152     label="电话"
153     rules={[{ required: true, message: '请输入电话' }]}
154   >
155     <Input placeholder="13800138000" maxLength={20} />
156   </Form.Item>
157
158 </Form>
159 </Modal>

```

```

160   );
161 };
162
163 export default UserManagement;

```

### 3. 主应用入口 ( `src/App.js` )

```

1 import UserManagement from './components/UserManagement';
2
3 function App()
4 {
5   return (
6     <div className="App">
7       <UserManagement />
8     </div>
9   );
10
11 export default App;

```

## 三、第二阶段：连接真实API

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

```

1 import axios from 'axios';
2
3 const API_URL = 'http://localhost:8080/api/users';
4
5 export const userService = {
6   getUsers: () => axios.get(API_URL),
7   createUser: (user) => axios.post(API_URL, user),
8   updateUser: (id, user) => axios.put(` ${API_URL} / ${id}` , user),
9   deleteUser: (id) => axios.delete(` ${API_URL} / ${id}` )
10 };

```

### 2. 修改用户管理组件

## 更换 mockUserApi ==>userService

```
1 // 在UserManagement.jsx中替换:  
2 // import { mockUserApi } from '../mock/userApi';  
3 import { userService } from '../services/userService';  
4  
5 // 修改所有API调用方法:  
6 const fetchUsers = async () => {  
7   setLoading(true);  
8   try {  
9     const response = await userService.getUsers();  
10    setUsers(response.data);  
11  } catch (error) {  
12    message.error('获取用户列表失败');  
13  } finally {  
14    setLoading(false);  
15  }  
16};  
17  
18 // 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`)

```
1 import axios from 'axios';  
2  
3 const instance = axios.create({  
4   baseURL: 'http://localhost:8080',  
5   timeout: 5000  
6 });  
7  
8 // 请求拦截器  
9 instance.interceptors.request.use(config => {  
10   const token = localStorage.getItem('token');  
11   if (token) {  
12     config.headers.Authorization = `Bearer ${token}`;  
13   }  
14   return config;  
15 });  
16  
17 // 响应拦截器  
18 instance.interceptors.response.use(  
19   response => response.data,  
20   error => {  
21     if (error.response) {  
22       switch (error.response.status) {  
23         case 401:  
24           // 处理未授权  
25           break;  
26         case 404:  
27           // 处理未找到  
28           break;  
29         default:  
30           console.error('API错误:', error);  
31       }  
32     }  
33     return Promise.reject(error);  
34   }  
35 );  
36  
37 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

```
1 src/
2   └── components/
3     └── UserManagement.jsx
4   ├── services/
5     └── userService.js
6   └── mock/
7     └── userApi.js
8   ├── utils/
9     └── http.js
10  └── App.js
11  └── index.js
```

## 六、测试建议

### 1. Mock阶段测试:

```
1 // 在组件中添加测试按钮
2 <Button onClick={() => {
3   mockUserApi.getUsers().then(res => console.log(res));
4 }}>
5   测试Mock API
6 </Button>
```

### 1. 真实API测试:

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

### 1. 错误处理测试:

```
1 // 在userService.js中模拟错误
2 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

```
1 personal-blog/
2   ├── index.html      # 首页
3   ├── article.html    # 文章展示页
4   ├── about.html      # 关于我页面
5   ├── css/
6     └── style.css      # 主样式文件
7   └── js/
8     └── script.js       # 主JavaScript文件
9
```

```
1 <!DOCTYPE html>
```

### 1. index.html (首页)

```

2 <html lang="zh-CN">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>我的个人博客</title>
7     <link rel="stylesheet" href="css/style.css" />
8   </head>
9   <body>
10    <header class="header">
11      <div class="container">
12        <h1 class="logo">我的博客</h1>
13        <nav class="main-nav">
14          <ul>
15            <li><a href="index.html" class="active">首页</a></li>
16            <li><a href="about.html">关于我</a></li>
17            <li><a href="#contact">联系我</a></li>
18          </ul>
19        </nav>
20      </div>
21    </header>
22
23    <div class="container main-content">
24      <main class="content">
25        <section class="blog-posts">
26          <h2>最新文章</h2>
27
28          <article class="post">
29            <h3><a href="article.html">HTML5新特性探索</a></h3>
30            <div class="post-meta">
31              <span class="date">2023年5月15日</span>
32              <span class="category">前端开发</span>
33            </div>
34            <p>
35              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...
36            </p>
37            <a href="article.html" class="read-more">阅读更多</a>
38          </article>
39

```

```

40          <article class="post">
41            <h3><a href="article.html">CSS3动画入门</a></h3>
42            <div class="post-meta">
43              <span class="date">2023年5月10日</span>
44              <span class="category">前端开发</span>
45            </div>
46            <p>
47              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...
48            </p>
49            <a href="article.html" class="read-more">阅读更多</a>
50          </article>
51
52          <article class="post">
53            <h3><a href="article.html">JavaScript基础教程</a></h3>
54            <div class="post-meta">
55              <span class="date">2023年5月5日</span>
56              <span class="category">前端开发</span>
57            </div>
58            <p>
59              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...
60            </p>
61            <a href="article.html" class="read-more">阅读更多</a>
62          </article>
63        </section>
64      </main>
65
66      <aside class="sidebar">
67        <div class="widget">
68          <h3>关于我</h3>
69          
70          <p>我是前端开发爱好者，热爱学习新技术，喜欢分享知识。</p>
71          <a href="about.html" class="btn">了解更多</a>
72        </div>
73
74        <div class="widget">
75          <h3>分类</h3>
76          <ul class="categories">
77            <li><a href="#">前端开发</a></li>
78          </ul>
79        </div>
80
81      </aside>
82
83      <div class="content">
84        <h2>最新文章</h2>
85        <ul class="list-group">
86          <li><a href="article.html">HTML5新特性探索</a></li>
87          <li><a href="article.html">CSS3动画入门</a></li>
88          <li><a href="article.html">JavaScript基础教程</a></li>
89        </ul>
90      </div>
91
92    </div>
93
94  </body>
95
96</html>

```

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

```

79      <li><a href="#">CSS技巧</a></li>
80      <li><a href="#">HTML5</a></li>
81      </ul>
82      </div>
83
84      <div class="widget">
85          <h3>订阅</h3>
86          <p>订阅我的博客，获取最新文章通知</p>
87          <form class="subscribe-form">
88              <input type="email" placeholder="你的邮箱地址" />
89              <button type="submit" class="btn">订阅</button>
90          </form>
91      </div>
92      </aside>
93  </div>
94
95  <footer class="footer">
96      <div class="container">
97          <p>© 2023 我的个人博客。保留所有权利。</p>
98          <div class="social-links">
99              <a href="#"><i class="fab fa-github"></i></a>
100             <a href="#"><i class="fab fa-twitter"></i></a>
101             <a href="#"><i class="fab fa-linkedin"></i></a>
102         </div>
103     </div>
104  </footer>
105
106  <button id="back-to-top" title="回到顶部">↑</button>
107  <script src="js/script.js"></script>
108 </body>
109 </html>
```

## 2. article.html (文章展示页)

```
1 <!DOCTYPE html>
```

```
2 <html lang="zh-CN">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>文章标题 - 我的个人博客</title>
7     <link rel="stylesheet" href="css/style.css" />
8   </head>
9   <body>
10    <header class="header">
11      <div class="container">
12        <h1 class="logo">我的博客</h1>
13        <nav class="main-nav">
14          <ul>
15            <li><a href="index.html">首页</a></li>
16            <li><a href="about.html">关于我</a></li>
17            <li><a href="#contact">联系我</a></li>
18          </ul>
19        </nav>
20      </div>
21    </header>
22
23    <div class="container article-container">
24      <article class="article-content">
25        <header>
26          <h1>HTML5新特性探索</h1>
27          <div class="article-meta">
28            <span class="author">作者: 张三</span>
29            <span class="date">发布时间: 2023年5月15日</span>
30            <span class="category">分类: 前端开发</span>
31          </div>
32        </header>
33
34        
35
36        <div class="article-body">
37          <p>
38            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。
39          </p>

```

```

40
41      <h2>语义化标签</h2>
42
43      <p>
44          HTML5引入了许多新的语义化元素，如<header>，<footer>，
45          <article>，<section>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。
46
47
48      <h2>多媒体支持</h2>
49
50      <p>
51          HTML5原生支持音频和视频播放，不再需要Flash等插件。使用<audio>和
52          <video>标签可以轻松地在网页中嵌入多媒体内容。
53
54      </p>
55
56          <pre><code><video width="320" height="240" controls>
57              &lt;source src="movie.mp4" type="video/mp4"&gt;
58              &lt;source src="movie.ogg" type="video/ogg"&gt;
59          您的浏览器不支持HTML5视频标签。
60
61          &lt;/video&gt;</code></pre>
62
63
64
65      <h2>Canvas绘图</h2>
66
67      <p>
68          HTML5的<canvas>元素提供了一套JavaScript
69          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
70          幅和游戏等。
71
72
73      <h2>本地存储</h2>
74
75      <p>
76          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，  

77          减少与服务器的通信，提高应用性能。
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117

```

```

78      </article>
79
80      <section class="comments-section">
81          <h2>评论</h2>
82
83          <div id="comments-list">
84              <div class="comment">
85                  <div class="comment-author">
86                      
87                      <span>李四</span>
88                  </div>
89                  <div class="comment-content">
90                      <p>这篇文章写得很好，对HTML5的新特性介绍得很全面！</p>
91                  <div class="comment-meta">2023年5月16日</div>
92              </div>
93          </div>
94
95          <div class="comment">
96              <div class="comment-author">
97                  
98                  <span>王五</span>
99              </div>
100             <div class="comment-content">
101                 <p>Canvas部分可以再详细一些，期待后续文章！</p>
102             <div class="comment-meta">2023年5月17日</div>
103         </div>
104     </div>
105
106
107     <form id="comment-form" class="comment-form">
108         <h3>发表评论</h3>
109         <div class="form-group">
110             <label for="name">姓名:</label>
111             <input type="text" id="name" name="name" required />
112         </div>
113         <div class="form-group">
114             <label for="email">邮箱:</label>
115             <input type="email" id="email" name="email" required />
116         </div>
117         <div class="form-group">

```

```
118     <label for="comment">评论:</label>
119     <textarea id="comment" name="comment" rows="5" required></textarea>
120   </div>
121   <button type="submit" class="btn">提交评论</button>
122 </form>
123 </section>
124 </div>
125
126 <footer class="footer">
127   <div class="container">
128     <p>&copy; 2023 我的个人博客. 保留所有权利.</p>
129     <div class="social-links">
130       <a href="#"><i class="fab fa-github"></i></a>
131       <a href="#"><i class="fab fa-twitter"></i></a>
132       <a href="#"><i class="fab fa-linkedin"></i></a>
133     </div>
134   </div>
135 </footer>
136
137 <button id="back-to-top" title="回到顶部">↑</button>
138
139 <script src="js/script.js"></script>
140 </body>
141 </html>
```

```
1 <!DOCTYPE html>
```

### 3. about.html (关于我页面)

```

2 <html lang="zh-CN">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>关于我 - 我的个人博客</title>
7     <link rel="stylesheet" href="css/style.css" />
8   </head>
9   <body>
10    <header class="header">
11      <div class="container">
12        <h1 class="logo">我的博客</h1>
13        <nav class="main-nav">
14          <ul>
15            <li><a href="index.html">首页</a></li>
16            <li><a href="about.html" class="active">关于我</a></li>
17            <li><a href="#contact">联系我</a></li>
18          </ul>
19        </nav>
20      </div>
21    </header>
22
23    <div class="container about-container">
24      <section class="about-section">
25        <h2>关于我</h2>
26
27        <div class="about-content">
28          
34
35          <div class="about-text">
36            <h3>张三</h3>
37            <p class="title">前端开发工程师</p>
38            <p>
39              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。

```

40            </p>
41            <p>我的技术栈包括：</p>
42            <ul class="skills">
43              <li>HTML5 & CSS3</li>
44              <li>JavaScript (ES6+)</li>
45              <li>React.js</li>
46              <li>Vue.js</li>
47              <li>Node.js</li>
48              <li>Git</li>
49            </ul>
50          </div>
51        </div>
52      </section>
53
54      <section class="contact-section" id="contact">
55        <h2>联系我</h2>
56
57        <div class="contact-methods">
58          <div class="contact-info">
59            <h3>联系方式</h3>
60            <ul>
61              <li>
62                <i class="fas fa-envelope"></i> Email: example@example.com
63              </li>
64              <li><i class="fas fa-phone"></i> 电话: 123-456-7890</li>
65              <li><i class="fas fa-map-marker-alt"></i> 地址: 北京市海淀区</li>
66            </ul>
67
68            <div class="social-links">
69              <a href="#"><i class="fab fa-github"></i></a>
70              <a href="#"><i class="fab fa-twitter"></i></a>
71              <a href="#"><i class="fab fa-linkedin"></i></a>
72              <a href="#"><i class="fab fa-weibo"></i></a>
73            </div>
74          </div>
75
76          <form class="contact-form">
77            <h3>发送消息</h3>
78            <div class="form-group">
79              <input type="text" placeholder="你的姓名" required />

```

```

80      </div>
81
82      <div class="form-group">
83          <input type="email" placeholder="你的邮箱" required />
84      </div>
85
86      <div class="form-group">
87          <input type="text" placeholder="主题" />
88      </div>
89
90      <div class="form-group">
91          <textarea placeholder="你的消息" rows="5" required></textarea>
92      </div>
93
94      <button type="submit" class="btn">发送消息</button>
95
96  </form>
97  </div>
98
99  </section>
100 </div>
101
102 <footer class="footer">
103     <div class="container">
104         <p>&copy; 2023 我的个人博客. 保留所有权利.</p>
105
106         <div class="social-links">
107             <a href="#"><i class="fab fa-github"></i></a>
108             <a href="#"><i class="fab fa-twitter"></i></a>
109             <a href="#"><i class="fab fa-linkedin"></i></a>
110         </div>
111
112     </div>
113
114 </footer>
115
116
117 <button id="back-to-top" title="回到顶部">↑</button>
118
119 <script src="js/script.js"></script>
120
121 </body>
122
123 </html>

```

## 4. css/style.css (样式文件)

```

1  /* 全局样式 */
2  :root {
3      --primary-color: #3498db;
4      --secondary-color: #2980b9;
5      --dark-color: #2c3e50;
6      --light-color: #ecf0f1;
7      --success-color: #2ecc71;
8      --danger-color: #e74c3c;
9      --warning-color: #f39c12;
10     --gray-color: #95a5a6;
11     --white-color: #fff;
12     --black-color: #333;
13 }
14
15  * {
16      margin: 0;
17      padding: 0;
18      box-sizing: border-box;
19 }
20
21 body {
22     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
23     line-height: 1.6;
24     color: var(--black-color);
25     background-color: #f5f5f5;
26 }
27
28 a {
29     text-decoration: none;
30     color: var(--primary-color);
31 }
32
33 a:hover {
34     color: var(--secondary-color);
35 }
36
37 img {
38     max-width: 100%;
39     height: auto;

```

```

40 }
41
42 .container {
43   width: 100%;
44   max-width: 1200px;
45   margin: 0 auto;
46   padding: 0 15px;
47 }
48
49 .btn {
50   display: inline-block;
51   background: var(--primary-color);
52   color: var(--white-color);
53   padding: 8px 16px;
54   border: none;
55   border-radius: 4px;
56   cursor: pointer;
57   transition: all 0.3s ease;
58 }
59
60 .btn:hover {
61   background: var(--secondary-color);
62   color: var(--white-color);
63 }
64
65 /* 头部样式 */
66 .header {
67   background: var(--dark-color);
68   color: var(--white-color);
69   padding: 20px 0;
70   position: sticky;
71   top: 0;
72   z-index: 100;
73   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
74 }
75
76 .header .container {
77   display: flex;
78   justify-content: space-between;
79   align-items: center;

```

```

80 }
81
82 .logo {
83   font-size: 24px;
84   font-weight: bold;
85 }
86
87 .main-nav ul {
88   display: flex;
89   list-style: none;
90 }
91
92 .main-nav ul li {
93   margin-left: 20px;
94 }
95
96 .main-nav ul li a {
97   color: var(--white-color);
98   font-weight: 500;
99   transition: color 0.3s ease;
100 }
101
102 .main-nav ul li a:hover {
103   color: var(--primary-color);
104 }
105
106 .main-nav ul li a.active {
107   color: var(--primary-color);
108   font-weight: bold;
109 }
110
111 /* 主要内容布局 */
112 .main-content {
113   display: flex;
114   margin: 30px auto;
115   gap: 30px;
116 }
117
118 .content {
119   flex: 2;

```

```
120 }
121
122 .sidebar {
123     flex: 1;
124 }
125
126 /* 文章列表样式 */
127 .blog-posts {
128     margin-bottom: 30px;
129 }
130
131 .blog-posts h2 {
132     margin-bottom: 20px;
133     padding-bottom: 10px;
134     border-bottom: 1px solid #ddd;
135 }
136
137 .post {
138     background: var(--white-color);
139     padding: 20px;
140     margin-bottom: 20px;
141     border-radius: 5px;
142     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
143 }
144
145 .post h3 {
146     margin-bottom: 10px;
147 }
148
149 .post h3 a {
150     color: var(--dark-color);
151 }
152
153 .post h3 a:hover {
154     color: var(--primary-color);
155 }
156
157 .post-meta {
158     font-size: 14px;
159     color: var(--gray-color);
```

```
160     margin-bottom: 15px;
161 }
162
163 .post-meta .date {
164     margin-right: 15px;
165 }
166
167 .post p {
168     margin-bottom: 15px;
169 }
170
171 .read-more {
172     display: inline-block;
173     font-weight: bold;
174     transition: all 0.3s ease;
175 }
176
177 .read-more:hover {
178     color: var(--secondary-color);
179 }
180
181 /* 侧边栏样式 */
182 .widget {
183     background: var(--white-color);
184     padding: 20px;
185     margin-bottom: 20px;
186     border-radius: 5px;
187     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
188 }
189
190 .widget h3 {
191     margin-bottom: 15px;
192     padding-bottom: 10px;
193     border-bottom: 1px solid #ddd;
194 }
195
196 .profile-img {
197     display: block;
198     margin: 0 auto 15px;
199     border-radius: 50%;
```

```
200 }
201
202 .categories {
203   list-style: none;
204 }
205
206 .categories li {
207   margin-bottom: 8px;
208   padding-bottom: 8px;
209   border-bottom: 1px dashed #eee;
210 }
211
212 .categories li:last-child {
213   margin-bottom: 0;
214   padding-bottom: 0;
215   border-bottom: none;
216 }
217
218 .subscribe-form input {
219   width: 100%;
220   padding: 8px;
221   margin-bottom: 10px;
222   border: 1px solid #ddd;
223   border-radius: 4px;
224 }
225
226 /* 脚部样式 */
227 .footer {
228   background: var(--dark-color);
229   color: var(--white-color);
230   padding: 30px 0;
231   text-align: center;
232 }
233
234 .footer .container {
235   display: flex;
236   flex-direction: column;
237   align-items: center;
238 }
239
```

```
240 .social-links {
241   margin-top: 15px;
242 }
243
244 .social-links a {
245   color: var(--white-color);
246   margin: 0 10px;
247   font-size: 20px;
248   transition: color 0.3s ease;
249 }
250
251 .social-links a:hover {
252   color: var(--primary-color);
253 }
254
255 /* 回到顶部按钮 */
256 #back-to-top {
257   display: none;
258   position: fixed;
259   bottom: 20px;
260   right: 20px;
261   background: var(--primary-color);
262   color: var(--white-color);
263   border: none;
264   border-radius: 50%;
265   width: 50px;
266   height: 50px;
267   font-size: 20px;
268   cursor: pointer;
269   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
270   transition: all 0.3s ease;
271   z-index: 99;
272 }
273
274 #back-to-top:hover {
275   background: var(--secondary-color);
276 }
277
278 /* 文章页样式 */
279 .article-container {
```

```
280     margin: 30px auto;
281 }
282
283 .article-content {
284     background: var(--white-color);
285     padding: 30px;
286     border-radius: 5px;
287     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
288     margin-bottom: 30px;
289 }
290
291 .article-content h1 {
292     margin-bottom: 15px;
293     font-size: 32px;
294 }
295
296 .article-meta {
297     font-size: 14px;
298     color: var(--gray-color);
299     margin-bottom: 20px;
300 }
301
302 .article-meta span {
303     margin-right: 15px;
304 }
305
306 .featured-image {
307     display: block;
308     margin: 0 auto 20px;
309     max-height: 400px;
310     object-fit: cover;
311     border-radius: 5px;
312 }
313
314 .article-body h2 {
315     margin: 25px 0 15px;
316     font-size: 24px;
317 }
318
319 .article-body p {
```

```
320     margin-bottom: 15px;
321 }
322
323 .article-body pre {
324     background: #f5f5f5;
325     padding: 15px;
326     border-radius: 5px;
327     margin: 15px 0;
328     overflow-x: auto;
329 }
330
331 .article-body code {
332     font-family: 'Courier New', Courier, monospace;
333     font-size: 14px;
334 }
335
336 .article-footer {
337     margin-top: 30px;
338     padding-top: 15px;
339     border-top: 1px solid #eee;
340 }
341
342 .tags a {
343     display: inline-block;
344     background: #f5f5f5;
345     padding: 3px 8px;
346     margin-right: 5px;
347     border-radius: 3px;
348     font-size: 14px;
349     color: var(--gray-color);
350 }
351
352 .tags a:hover {
353     background: var(--primary-color);
354     color: var(--white-color);
355 }
356
357 /* 评论区域样式 */
358 .comments-section {
359     background: var(--white-color);
```

```

360     padding: 30px;
361     border-radius: 5px;
362     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
363 }
364
365 .comments-section h2 {
366     margin-bottom: 20px;
367     padding-bottom: 10px;
368     border-bottom: 1px solid #ddd;
369 }
370
371 .comment {
372     display: flex;
373     margin-bottom: 20px;
374     padding-bottom: 20px;
375     border-bottom: 1px solid #eee;
376 }
377
378 .comment:last-child {
379     margin-bottom: 0;
380     padding-bottom: 0;
381     border-bottom: none;
382 }
383
384 .comment-author {
385     flex: 0 0 100px;
386     text-align: center;
387 }
388
389 .comment-author img {
390     border-radius: 50%;
391     margin-bottom: 5px;
392 }
393
394 .comment-content {
395     flex: 1;
396     padding-left: 20px;
397 }
398
399 .comment-meta {

```

```

400     font-size: 12px;
401     color: var(--gray-color);
402     margin-top: 5px;
403 }
404
405 .comment-form {
406     margin-top: 30px;
407 }
408
409 .comment-form h3 {
410     margin-bottom: 15px;
411 }
412
413 .form-group {
414     margin-bottom: 15px;
415 }
416
417 .form-group label {
418     display: block;
419     margin-bottom: 5px;
420     font-weight: bold;
421 }
422
423 .form-group input,
424 .form-group textarea {
425     width: 100%;
426     padding: 8px;
427     border: 1px solid #ddd;
428     border-radius: 4px;
429 }
430
431 .form-group textarea {
432     resize: vertical;
433     min-height: 100px;
434 }
435
436 /* 关于我页面样式 */
437 .about-container {
438     margin: 30px auto;
439 }

```

```

440
441 .about-section {
442   background: var(--white-color);
443   padding: 30px;
444   border-radius: 5px;
445   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
446   margin-bottom: 30px;
447 }
448
449 .about-section h2 {
450   margin-bottom: 20px;
451   padding-bottom: 10px;
452   border-bottom: 1px solid #ddd;
453 }
454
455 .about-content {
456   display: flex;
457   gap: 30px;
458 }
459
460 .about-img {
461   flex: 0 0 300px;
462   border-radius: 5px;
463   object-fit: cover;
464 }
465
466 .about-text {
467   flex: 1;
468 }
469
470 .about-text h3 {
471   font-size: 24px;
472   margin-bottom: 5px;
473 }
474
475 .title {
476   font-size: 16px;
477   color: var(--gray-color);
478   margin-bottom: 15px;
479 }

```

```

480
481 .skills {
482   display: flex;
483   flex-wrap: wrap;
484   list-style: none;
485   margin: 15px 0;
486 }
487
488 .skills li {
489   background: #f5f5f5;
490   padding: 5px 10px;
491   margin: 0 10px 10px 0;
492   border-radius: 3px;
493   font-size: 14px;
494 }
495
496 .contact-section {
497   background: var(--white-color);
498   padding: 30px;
499   border-radius: 5px;
500   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
501 }
502
503 .contact-section h2 {
504   margin-bottom: 20px;
505   padding-bottom: 10px;
506   border-bottom: 1px solid #ddd;
507 }
508
509 .contact-methods {
510   display: flex;
511   gap: 30px;
512 }
513
514 .contact-info {
515   flex: 1;
516 }
517
518 .contact-info ul {
519   list-style: none;

```

```

520 }
521
522 .contact-info li {
523     margin-bottom: 10px;
524     display: flex;
525     align-items: center;
526 }
527
528 .contact-info i {
529     margin-right: 10px;
530     width: 20px;
531     text-align: center;
532 }
533
534 .contact-form {
535     flex: 1;
536 }
537
538 /* 响应式设计 */
539 @media (max-width: 992px) {
540     .main-content {
541         flex-direction: column;
542     }
543
544     .about-content {
545         flex-direction: column;
546     }
547
548     .about-img {
549         margin-bottom: 20px;
550     }
551
552     .contact-methods {
553         flex-direction: column;
554     }
555 }
556
557 @media (max-width: 768px) {
558     .header .container {
559         flex-direction: column;

```

```

560     }
561
562     .logo {
563         margin-bottom: 15px;
564     }
565
566     .main-nav ul {
567         flex-direction: column;
568         align-items: center;
569     }
570
571     .main-nav ul li {
572         margin: 5px 0;
573     }
574
575     .article-content,
576     .comments-section,
577     .about-section,
578     .contact-section {
579         padding: 20px;
580     }
581 }
582

```

## 5. js/script.js (JavaScript文件)

```

1 document.addEventListener('DOMContentLoaded', function() {
2     // 回到顶部按钮
3     const backToTopButton = document.getElementById('back-to-top');
4
5     window.addEventListener('scroll', function() {
6         if (window.pageYOffset > 300) {
7             backToTopButton.style.display = 'block';
8         } else {
9             backToTopButton.style.display = 'none';
10        }
11    });
12
13    backToTopButton.addEventListener('click', function(e) {
14        e.preventDefault();
15        window.scrollTo({
16            top: 0,
17            behavior: 'smooth'
18        });
19    });
20
21    // 评论表单提交
22    const commentForm = document.getElementById('comment-form');
23    if (commentForm) {
24        commentForm.addEventListener('submit', function(e) {
25            e.preventDefault();
26
27            const name = document.getElementById('name').value;
28            const commentText = document.getElementById('comment').value;
29
30            if (name && commentText) {
31                const commentsList = document.getElementById('comments-list');
32                const now = new Date();
33                const dateStr = now.toLocaleDateString('zh-CN');
34
35                const newComment = document.createElement('div');
36                newComment.className = 'comment';
37                newComment.innerHTML =
38                    `

39                        


```

```

40                            <span>${name}</span>
41                        </div>
42                        <div class="comment-content">
43                            <p>${commentText}</p>
44                            <div class="comment-meta">${dateStr}</div>
45                        </div>
46                    `;
47
48                    commentsList.appendChild(newComment);
49
50                    // 清空表单
51                    commentForm.reset();
52
53                    // 滚动到新评论
54                    newComment.scrollIntoView({ behavior: 'smooth' });
55                }
56            );
57        }
58
59        // 联系表单提交
60        const contactForm = document.querySelector('.contact-form');
61        if (contactForm) {
62            contactForm.addEventListener('submit', function(e) {
63                e.preventDefault();
64
65                const name = contactForm.querySelector('input[type="text"]').value;
66                const email = contactForm.querySelector('input[type="email"]').value;
67                const message = contactForm.querySelector('textarea').value;
68
69                if (name && email && message) {
70                    alert(`感谢您的留言, ${name}! 我们会尽快回复您。`);
71                    contactForm.reset();
72                } else {
73                    alert('请填写所有必填字段!');
74                }
75            });
76        }
77
78        // 订阅表单提交
79        const subscribeForm = document.querySelector('.subscribe-form');

```

```

80 if (subscribeForm) {
81     subscribeForm.addEventListener('submit', function(e) {
82         e.preventDefault();
83
84         const email = subscribeForm.querySelector('input[type="email"]').value;
85
86         if (email) {
87             alert(`感谢订阅！我们已将确认邮件发送至 ${email}`);
88             subscribeForm.reset();
89         } else {
90             alert('请输入有效的邮箱地址！');
91         }
92     });
93 }
94 });

```

## 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：

```
1 src/
2   └── main/
3     └── java/
4       └── com/
5         └── example/
6           └── demo/
7             ├── DemoApplication.java      <-- 启动类
8             └── controller/
9               └── UserController.java
10            └── config/
11              └── GlobalCorsConfig.java  <-- CORS 配置类
12
```

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：

```
1 // src/main/java/com/example/demo/config/GlobalCorsConfig.java
2
3 package com.example.demo.config;
4
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7 import org.springframework.web.servlet.config.annotation.CorsRegistry;
8 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
9
10 @Configuration
11 public class GlobalCorsConfig {
12     @Bean
13     public WebMvcConfigurer corsConfigurer() {
14         return new WebMvcConfigurer() {
15             @Override
16             public void addCorsMappings(CorsRegistry registry) {
17                 registry.addMapping("/**")
18                     .allowedOrigins("http://localhost:3000")
19                     .allowedMethods("GET", POST, PUT, DELETE, OPTIONS)
20                     .allowedHeaders("*")
21                     .allowCredentials(true);
22             }
23         };
24     }
25 }
```