《Java 设计模式》期终考试

软件设计文档

题　　目	LinkUp 通讯录
老　　师	吴育峰
院　　系	数学与计算机学院
专　　业	计算机科学与技术
班　　级	计科 223
学　　号	22103220111
姓　　名	丁泉鸿

PPT 讲解 50%	设计文档 50%	得分

二〇二四年 12 月 24 日

目录

第1章 概述

1.1 课题简介

LinkUp 通讯录一款为个人设计的高效、便捷的联系人管理工具。它不仅提供基础的注册登录功能，还支持详细的分组管理和全面的通讯录管理，帮助用户轻松整理和查找联系人信息。此外，该产品具备多功能查询和数据导入导出功能，旨在满足用户的多样化需求，提高日常沟通效率。

图 1 软件 logo

1.2 模式应用概述

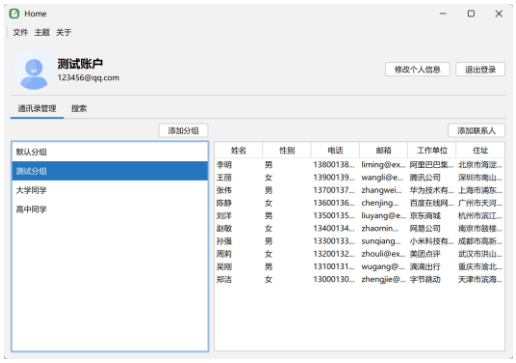
该项目在架构上借鉴了 Spring 的三层架构设计，融入了单例、备忘录、策略、观察者、外观、责任链共 6 种模式，使得项目的可拓展性和可维护性得到了极大的提高。具体使用概述如下：

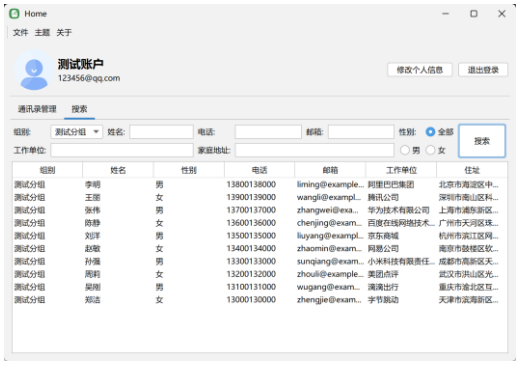
模 式	使用场景
单例模式	用于数据库连接、存储用户数据
备忘录模式	用于实现记住密码、自动登录和主题切换功能
策略模式	用于数据库操作封装，简化数据库操作
外观模式	用于实现导入、导出功能
责任链模式	用于在用户删除分组时将分组中的联系人移动至默认分组
观察者模式	用于实现界面的响应式组件

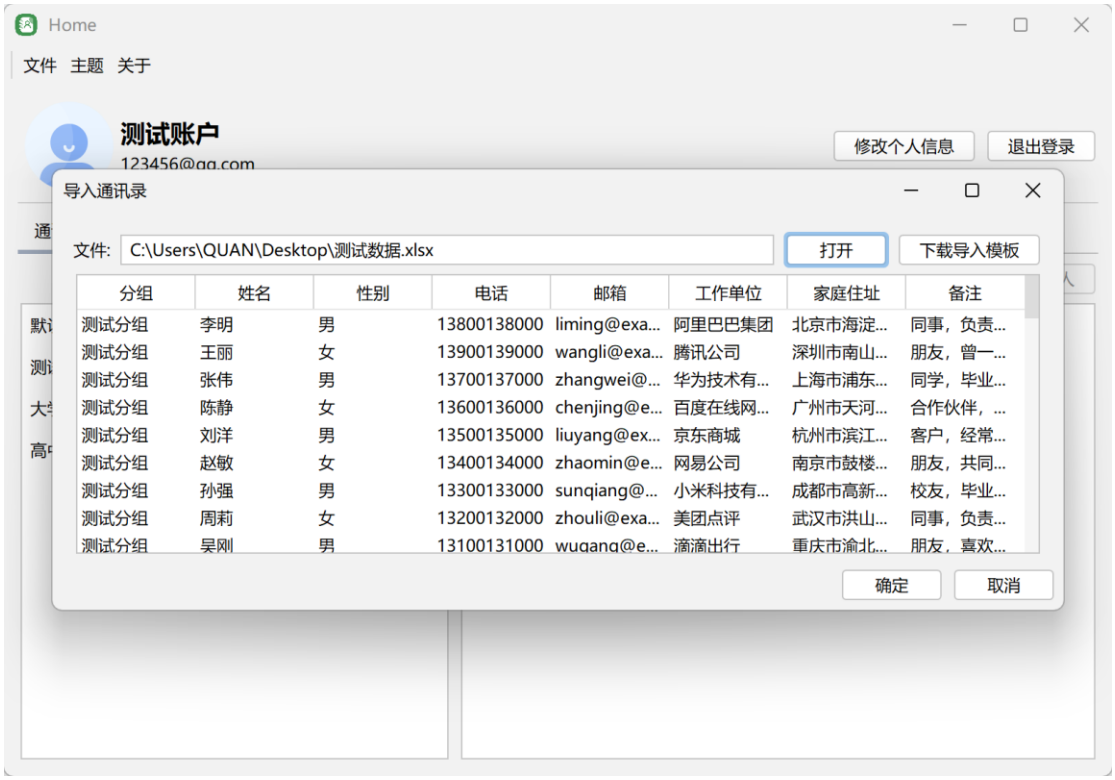
表 1 模式应用概述

1.3 软件界面


登录
LinkUp通讯录
邮箱:
密码:
5天内自动登录
记住密码
登录
注册


Home
文件 主题 关于
测试账户
123456@qq.com
修改个人信息
退出登录
通讯录管理
搜索
添加分组
添加联系人
默认分组
测试分组
大学同学
高中同学


Home
文件 主题 关于
测试账户
123456@qq.com
修改个人信息
退出登录
通讯录管理
搜索


Home
文件 主题 关于
测试账户
123456@qq.com
修改个人信息
退出登录
导入通讯录
文件: C:\Users\QUAN\Desktop\测试数据.xlsx
打开
下载导入模板
分组 姓名 性别 电话 邮箱 工作单位 家庭住址 备注
测试分组 李明 男 13800138000 liming@exa... 阿里巴巴集团 北京市海淀... 同事，负责...
测试分组 王丽 女 13900139000 wangli@exa... 腾讯公司 深圳市南山... 朋友，曾一...
测试分组 张伟 男 13700137000 zhangwei@... 华为技术有... 上海市浦东... 同学，毕业...
测试分组 陈静 女 13600136000 chenjing@e... 百度在线网... 广州市天河... 合作伙伴，...
测试分组 刘洋 男 13500135000 liuyang@ex... 京东商城 杭州市滨江... 客户，经常...
测试分组 赵敏 女 13400134000 zhaomin@e... 网易公司 南京市鼓楼... 朋友，共同...
测试分组 孙强 男 13300133000 sunqiang@... 小米科技有... 成都市高新... 校友，毕业...
测试分组 周莉 女 13200132000 zhouli@exa... 美团点评 武汉市洪山... 同事，负责...
测试分组 吴刚 男 13100131000 wugang@e... 滴滴出行 重庆市渝北... 朋友，喜欢...
确定
取消

第2章 需求分析

2.1 功能需求

2.1.1 用户帐号管理

(1) 注册

支持通过邮箱进行注册，要求用户提供用户名、邮箱等信息，并设置密码

(2) 登录

支持通过邮箱和密码进行登录，实现了记住密码和自动登录的功能，方便用户快速登录

2.1.2 分组管理

(1) 默认分组

系统预设默认分组，所有未指定分组的新增联系人自动归入此分组，默认分组不可删除，但用户可以重命名。

(2) 自定义分组

用户可以根据需要创建多个自定义分组，例如家人、朋友、同学等。支持对分组进行编辑、重命名和删除操作。当删除某个分组时，其下所有联系人自动转移至默认分组。

2.1.3 通讯录管理

(1) 增加联系人

用户可以录入详尽的联系人信息，包括但不限于姓名、电话、电子邮件、地址、生日、职业、工作单位等。支持批量添加联系人，以提升工作效率。

(2) 修改联系人

提供便捷的界面允许用户更新联系人的任何信息，保持数据的时效性和准确性。

(3) 删除联系人

支持单独移除联系人，也支持选择多个联系人进行批量删除。

(4) 多功能查询

- 按名字查询：输入部分或完整的名字，迅速定位目标联系人。
- 按地域查询：支持省、市、县三级地域筛选，方便找到特定区域的联系人。
- 按电话号码查询：通过电话号码快速检索联系人。
- 按工作单位查询：根据工作单位名称查找相关联系人。
- 其他查询条件：支持依据电子邮件等更多条件进行搜索，满足不同场景下的查询需求。

(5) 导入导出

支持从Excel格式文件中批量导入联系人信息，简化初始数据录入过程。在导入过程中，提供错误提示和处理建议，确保数据正确无误。允许用户将当前通讯录中的数据导出为标准格式文件，便于备份或与其他应用共享。提供选择导出特定分组或全部联系人的选项。

2.2 功能模块

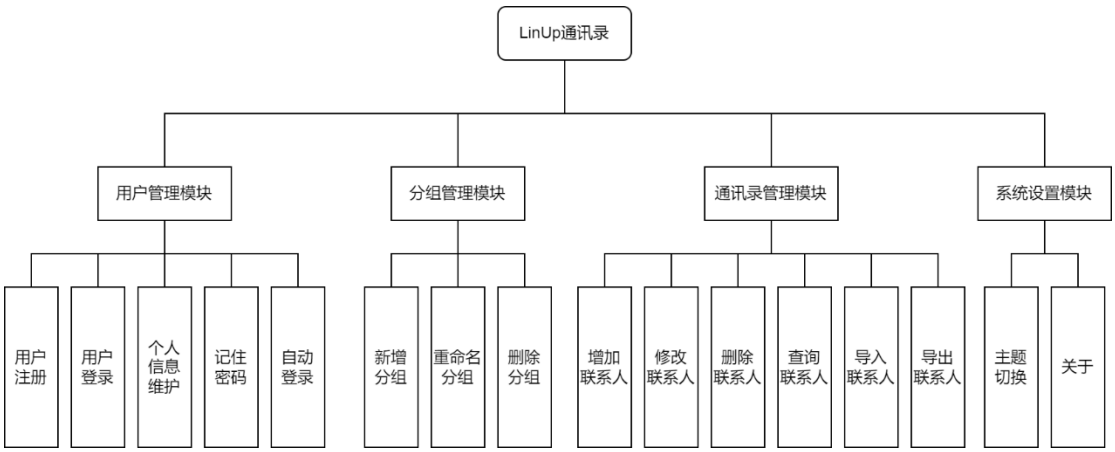


图 2 功能架构图

第3章 软件设计

3.1 架构设计

该项目在架构上借鉴了 Spring 的三层架构设计，分为数据库 Mappar 层、视图 View 层、控制层 Controler 层。

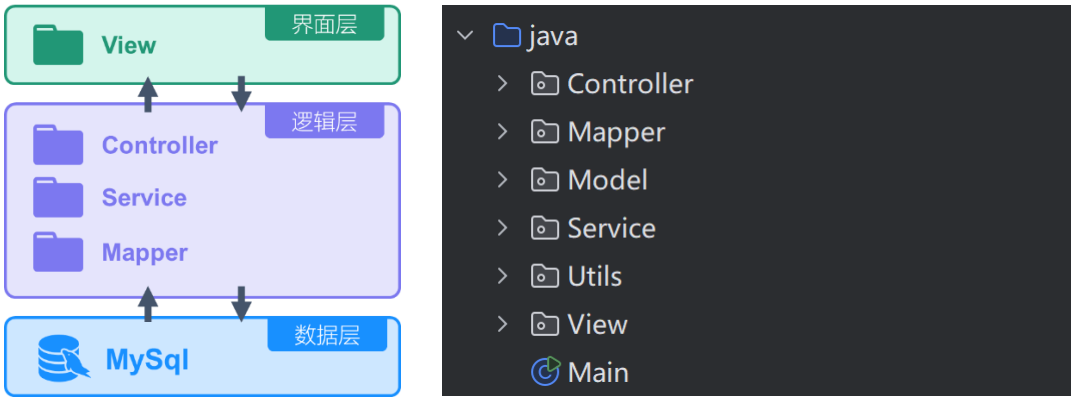


图 3 架构设计

3.2 数据库设计

为了支持通讯录产品的各项功能，我们需要设计一个高效、可扩展且安全的数据库结构。以下是针对各个功能模块的数据表设计及字段说明，以及一些关键的关系和索引。

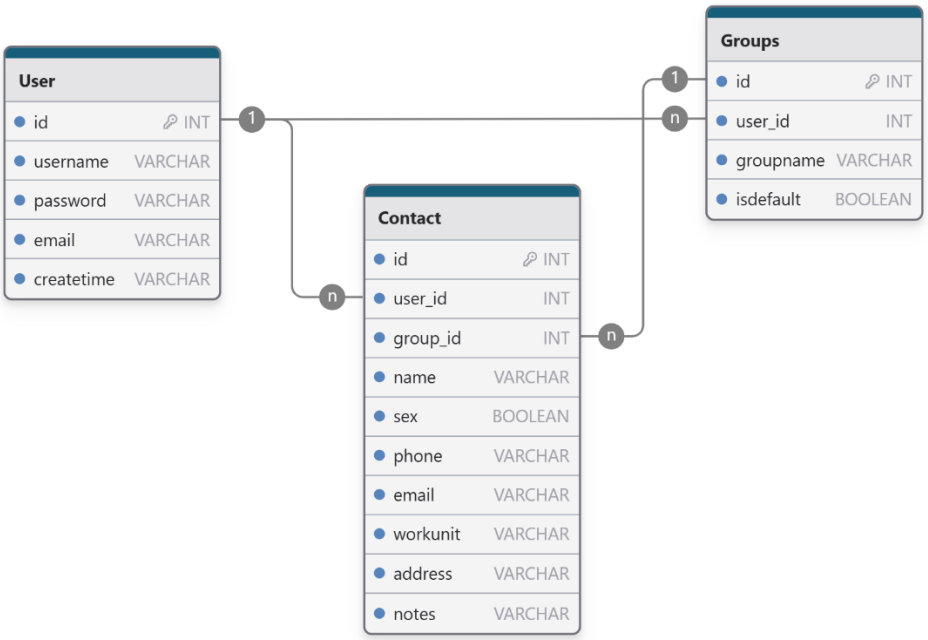


图 4 数据库设计

3.3 设计模式的应用

3.3.1 通过策略模式对数据库操作进行封装

在设计通讯录产品时，采用策略模式（Strategy Pattern）来封装数据库操作是一种非常有效的方法。这种方式不仅能够提高代码的可维护性和扩展性，还能让系统更加灵活地应对未来的需求变化。以下是具体实现思路及步骤。

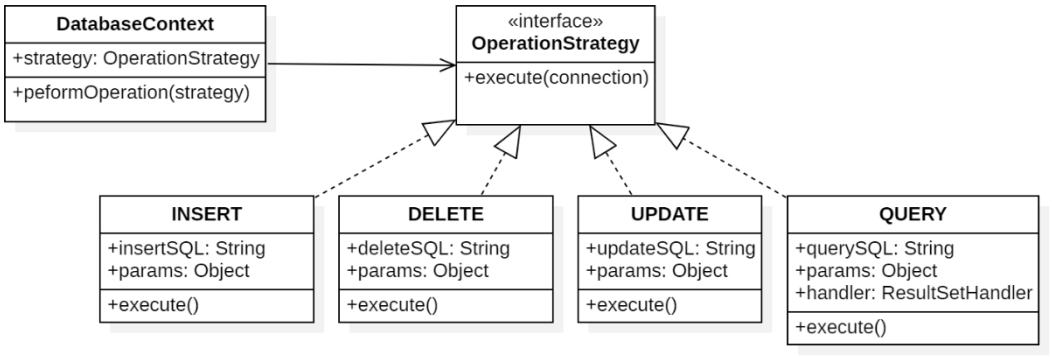


图 5 通过策略模式对数据库操作进行封装

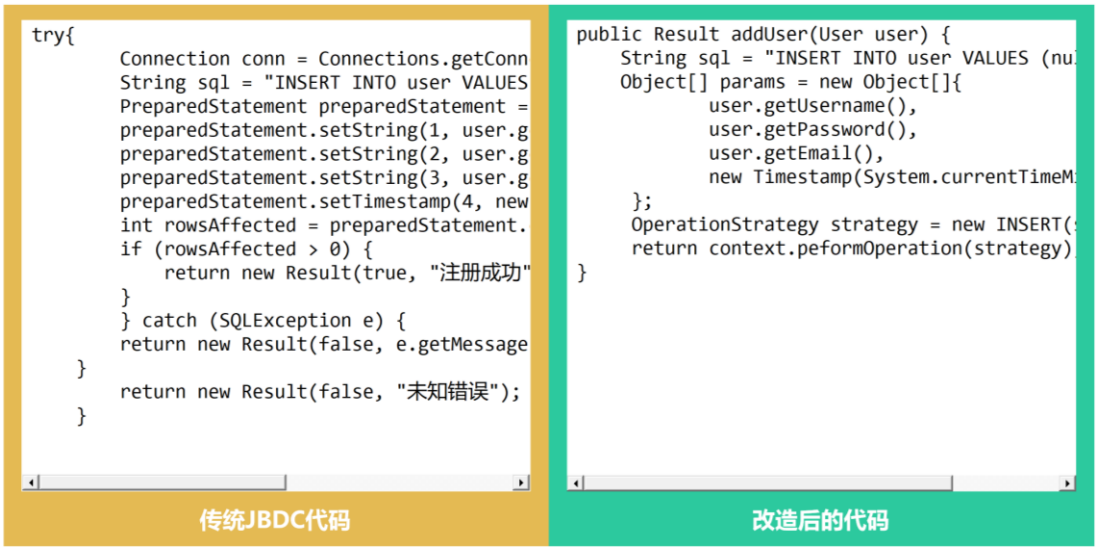


图 6 引入策略模式前后对比

3.3.2 通过备忘录模式实现记住密码和自动登录功能

在设计通讯录产品时，记住密码和自动登录功能是提升用户体验的重要特性之一。这些功能允许用户在一段时间内无需重新输入密码即可访问系统，从而简化了登录流程。为了实现这一目标，我们可以利用备忘录模式（Memento Pattern）来保存用户的登录状态，包括用户名和加密后的密码。以下是具体实现思路及步骤。

若勾选了自动登录或记住密码，当用户成功登录后，数据库返回用户信息，系统将创建一个 UserToken,通过 UserManager 将其保存至当前目录下的 user.config 文件中。当用户下一次登录时，系统将自动读取该文件，并将其加载到系统中，实现记住密码或自动登录功能。

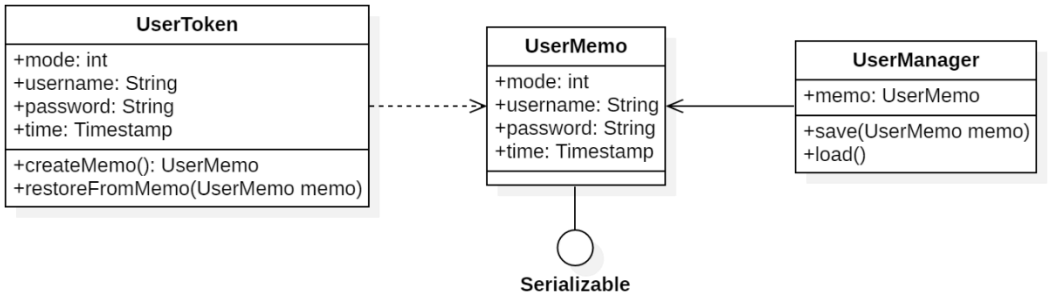


图 7 通过备忘录模式实现记住密码和自动登录功能

3.3.3 通过单例模式实现对用户信息的存储

在用户操作软件过程中，经常需要携带用户信息进行数据库操作。当涉及到用户信息时，通常希望在整个应用程序中使用相同的用户数据副本，以避免不同部分之间出现不一致的情况。此外，如果频繁创建和销毁用户信息对象，可能会导致不必要的性能开销。因此，采用单例模式可以有效地解决这些问题：

- 一致性：确保所有模块都引用同一个用户信息实例，从而保持数据的一致性。
- 资源优化：减少内存占用，因为只需要维护一个用户信息对象。
- 易于管理：提供了一个集中式的访问点，便于管理和更新用户信息。

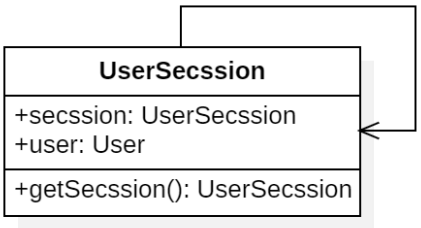


图 8 通过单例模式实现对用户信息的存储

3.3.4 通过外观模式实现导入导出功能

使用 Maven 引入 POI 依赖，使用外观模式对功能进行封装，使开发者在开发时不必在意 POI 的使用，仅需调用 Façade 层中的方法就可以完成功能的开发。

```xml
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>4.1.2</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>4.1.2</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml-schemas</artifactId>
    <version>4.1.2</version>
</dependency>
```

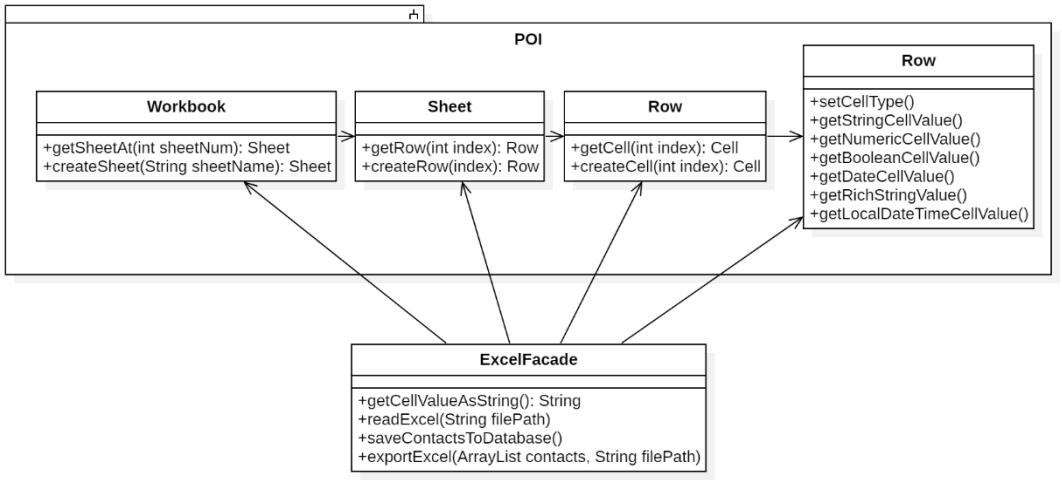


图 9 通过外观模式实现导入导出功能

3.3.5 通过责任链模式实现分组删除功能

为实现删除某分组时，属于该分组的通讯信息全部归入默认分组的功能。我将该功能拆解为①查询该分组下的所有联系人②移动这部分联系人至默认分组③删除该分组，通过责任链对其进行组装，降低功能的代码的耦合度。

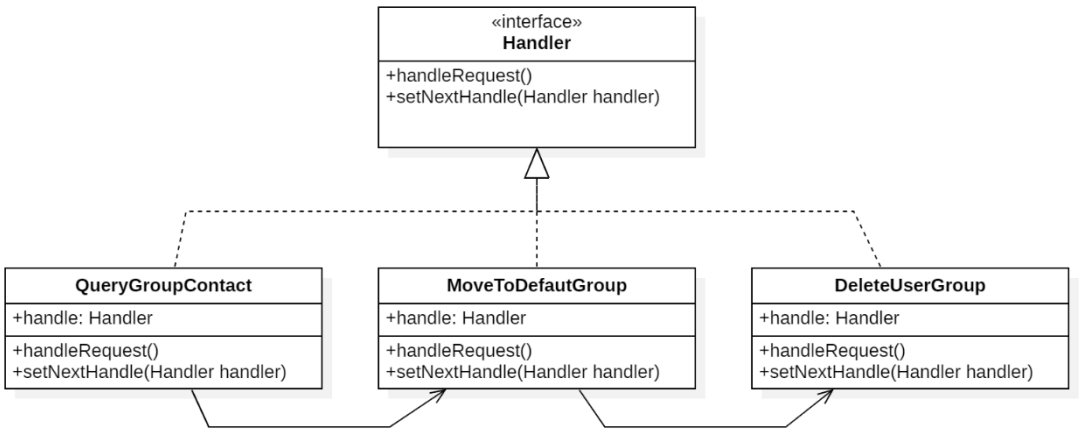
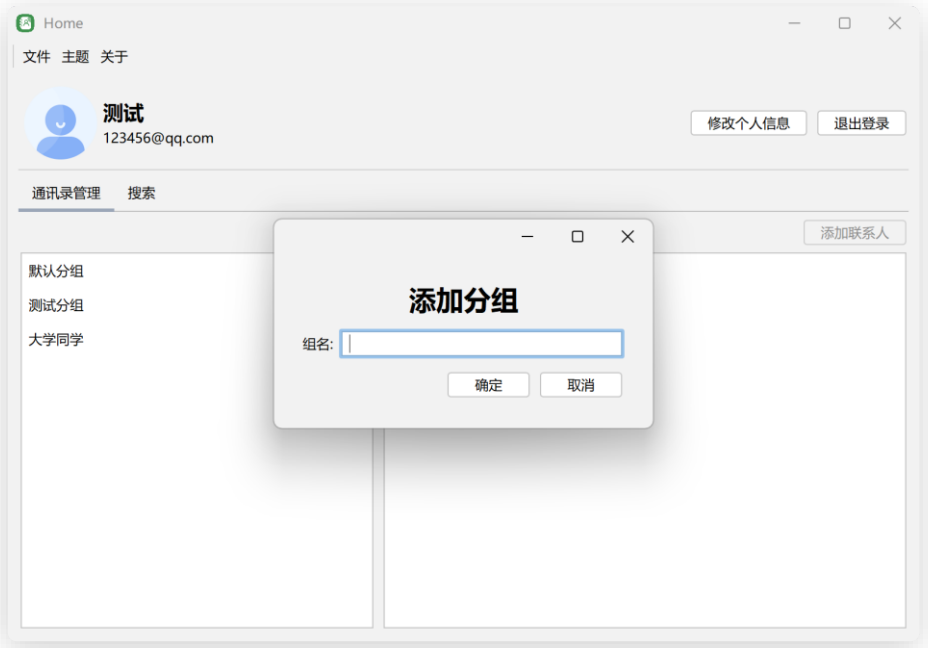


图 10 通过责任链模式实现分组删除功能

3.3.6 通过观察者模式实现响应式组件

在引入观察者模式前，用户对分组和联系人进行操作后，界面中的分组列表不会自动跟新，需要手动刷新。引入观察者模式后，可将组件在被观察对象中进行注册，当用户对分组和联系人进行操作后，通过观察者对页面元素进行自动更新。



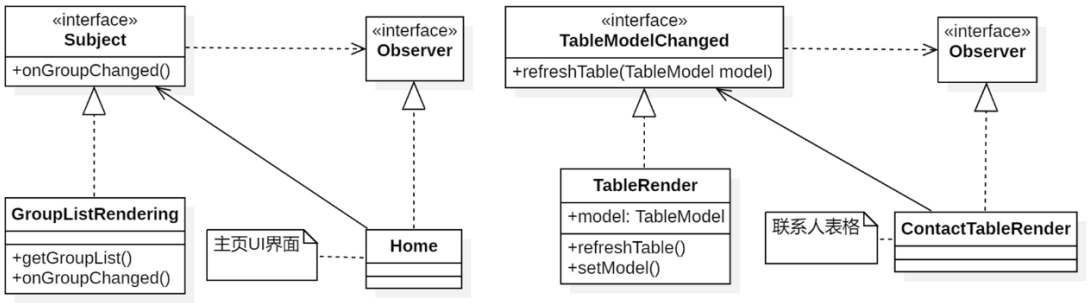


图 11 通过观察者模式实现响应式组件

3.3.7 构想：通过工厂模式实现响应式 Swing 组件库

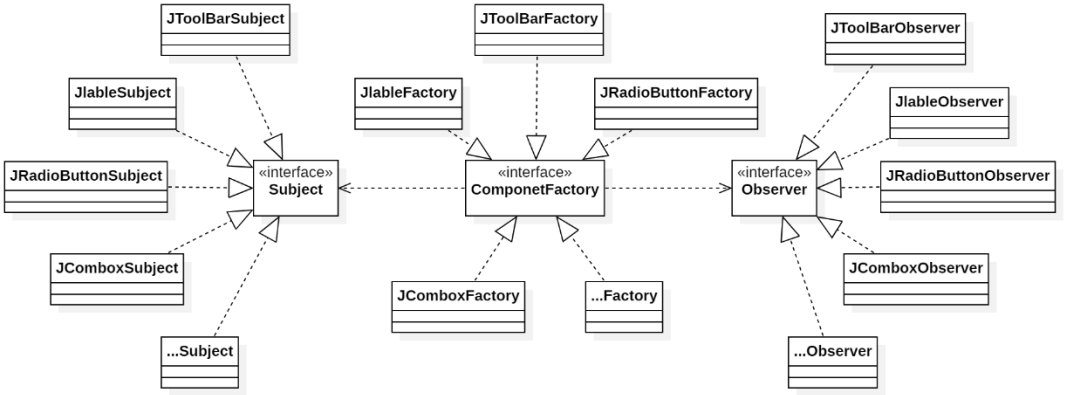


图 12 通过工厂模式实现响应式 Swing 组件库

第4章 源代码

4.1.1 通过策略模式对数据库操作进行封装

DatabaseContext

```
public class DatabaseContext {
    private final Connection connection;

    public DatabaseContext(Connection connection) {
        this.connection = connection;
    }

    public Result peformOperation(OperationStrategy strategy) {
        try {
            return strategy.execute(connection);
        }catch (SQLException e){
            return new Result(false, e.getMessage());
        }
    }
}
```

OperationStrategy

```
public interface OperationStrategy {
  public Result execute(Connection connection) throws SQLException;
}
```

ResultSetHandler

```
@FunctionalInterface
public interface ResultSetHandler<T> {
    T handle(ResultSet rs) throws SQLException;
}
```

INSERT

```
public class INSERT implements OperationStrategy{
    private final String insertSQL;
    private final Object[] params;

    public INSERT(String insertSQL, Object[] params) {
        this.insertSQL = insertSQL;
        this.params = params;
    }
```

```
    @Override
    public Result execute(Connection connection) throws
SQLException {
        try {
            PreparedStatement pstmt =
connection.prepareStatement(insertSQL);
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
            pstmt.executeUpdate();
            return new Result(true, insertSQL+" inserted
successfully");
        }catch (SQLException e) {
            return new Result(false, e.getMessage());
        }
    }
}
```

DELETE

```
public class DELETE implements OperationStrategy{
    private final String deletesql;
    private final Object[] params;

    public DELETE(String deletesql, Object[] params) {
        this.deletesql = deletesql;
        this.params = params;
    }

    @Override
    public Result execute(Connection connection) throws
SQLException {
        try {
            PreparedStatement pstmt =
connection.prepareStatement(deletesql);
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
            int affected = pstmt.executeUpdate();
            if (affected == 0){
                return new Result(false, deletesql+" failed");
            }
            return new Result(true, deletesql+" successful");
        }catch (SQLException e){
            return new Result(false, e.getMessage());
```

```
        }
    }
}
```

UPDATE

```
public class UPDATE implements OperationStrategy{
    private final String updatasql;
    private final Object[] params;

    public UPDATE(String updatasql, Object[] params) {
        this.updatasql = updatasql;
        this.params = params;
    }

    @Override
    public Result execute(Connection connection) throws
SQLException {
        try {
            PreparedStatement pstmt =
connection.prepareStatement(updatasql);
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
            int affected =pstmt.executeUpdate();
            if (affected == 0) {
                return new Result(false, "No rows affected");
            }
            return new Result(true,affected+"条数据"+"更新成功!");
        }catch (SQLException e){
            return new Result(false, e.getMessage());
        }
    }
}
```

QUERY

```
public class QUERY<T> implements OperationStrategy {
    private final String querysql;
    private final Object[] params;
    private final ResultSetHandler<T> handler;

    public QUERY(String querysql, Object[] params,
ResultSetHandler<T> handler) {
        this.querysql = querysql;
        this.params = params;
```

```
        this.handler = handler;
    }

    @Override
    public Result execute(Connection connection) throws
SQLException {
        try {
            PreparedStatement pstmt =
connection.prepareStatement(querysql);
            if (params != null) {
                for (int i = 0; i < params.length; i++) {
                    pstmt.setObject(i + 1, params[i]);
                }
            }
            try {
                ResultSet rs = pstmt.executeQuery();
                T result = handler.handle(rs);
                return new Result(true,"查询成功！",result);
            }catch (SQLException e){
                return new Result(false,e.getMessage());
            }
        }catch (SQLException e){
            return new Result(false, e.getMessage());
        }
    }
}
```

4.1.2 通过备忘录模式实现记住密码和自动登录功能

UserToken

```
public class UserToken {
    int mode;
    private String username;
    private String password;
    private Timestamp time;
    public UserMemo createMemo() {
        UserMemo userMemo = new UserMemo();
        userMemo.setMode(mode);
        userMemo.setUsername(username);
        userMemo.setPassword(password);
        userMemo.setTime(time);
        return userMemo;
```

```
    }
    public void restoreFromMemo(UserMemo userMemo) {
        mode = userMemo.getMode();
        username = userMemo.getUsername();
        password = userMemo.getPassword();
        time = userMemo.getTime();
    }
   /*构造函数及 Getter 和 Setter*/
}
```

UserMemo

```
public class UserMemo implements Serializable {
    int mode;
    private String username;
    private String password;
    private Timestamp time;
    //其他构造函数及其 getter 和 setter
}
```

UserManager

```
public class UserManager {
    Path path = Paths.get("./user.config");
    public Result save(UserMemo memo)  {
        try {
            Files.createDirectories(path.getParent());
            if (Files.notExists(path)) {
                Files.createFile(path);
            }
            ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(path.toFile()));
            oos.writeObject(memo);
            oos.close();
            return new Result(true,"用户信息记录成功");
        } catch (IOException e) {
            return new Result(false,e.getMessage());
        }
    }
    public Result load() {

        UserMemo memo = null;
        try {
            Files.createDirectories(path.getParent());
            if (Files.notExists(path)) {
                return new Result(false,"未找到用户信息文件");
```

```
            }
            ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(path.toFile()));
            memo = (UserMemo) ois.readObject();
            return new Result(true,"读取用户信息成功!",memo);
        } catch (IOException e) {
            return new Result(false,e.getMessage());
        } catch (ClassNotFoundException e) {
            return new Result(false,e.getMessage());
        }
    }

}
```

4.1.3 通过单例模式实现对用户信息的存储

UserSecssion

```
public class UserSecssion {
    private static UserSecssion secssion;
    private User user;
    public static UserSecssion getSecssion() {
        if (UserSecssion.secssion == null) {
            secssion = new UserSecssion();
        }
        return secssion;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public User getUser() {
        return user;
    }

    public UserSecssion() {
    }
}
```

4.1.4 通过外观模式实现导入导出功能

ExcelContact

```
public class ExcelContact extends Contact {
    private String groupName;
    public ExcelContact(int id, int userId, int groupId, String
name, int sex, String phone, String email, String workunit, String
address, String notes) {
        super(id, userId, groupId, name, sex, phone, email,
workunit, address, notes);
    }
    public ExcelContact(int id, int userId, int groupId, String
name, int sex, String phone, String email, String workunit, String
address, String notes,String groupName) {
        super(id, userId, groupId, name, sex, phone, email,
workunit, address, notes);
        this.groupName = groupName;
    }
    public String getGroupName() {
        return groupName;
    }
}
```

ExcelFacade

```
public class ExcelFacade {
        public  String getCellValueAsString(Cell cell) {
            if (cell == null) {
                return "";
            }
            switch (cell.getCellType()) {
                case STRING:
                    return cell.getStringCellValue();
                case NUMERIC:
                    if (DateUtil.isCellDateFormatted(cell)) {
                        return cell.getDateCellValue().toString();
                    } else {
                        return String.valueOf((long)
cell.getNumericCellValue());
                    }
                case BOOLEAN:
                    return String.valueOf(cell.getBooleanCellValue());
                case FORMULA:
```

```
                return cell.getCellFormula();
            default:
                return "";
        }
    }
    public Result readExcel(String filePath) {
        try {
            File excelFile = new File(filePath);
            FileInputStream fileInputStream = new
FileInputStream(excelFile);
            Workbook workbook = null;
            if (filePath.endsWith(".xls")) {
                workbook = new HSSFWorkbook(fileInputStream);
            } else if (filePath.endsWith(".xlsx")) {
                workbook = new XSSFWorkbook(fileInputStream);
            } else {
                return new Result(false, "文件格式错误");
            }
            Sheet sheet = workbook.getSheetAt(0);
            int rowNum = sheet.getPhysicalNumberOfRows();
            if (rowNum == 0) {
                return new Result(false, "文件为空");
            }

            ArrayList<ExcelContact> excelContacts = new
ArrayList<>();

            for (int i = 0; i < rowNum; i++) {
                Row row = sheet.getRow(i);
                if (row == null) {
                    continue;
                }

                String groupname =
getCellValueAsString(row.getCell(0));
                String name =
getCellValueAsString(row.getCell(1));
                String sex = getCellValueAsString(row.getCell(2));
                String phone =
getCellValueAsString(row.getCell(3));
                String email =
getCellValueAsString(row.getCell(4));
                String workunit =
getCellValueAsString(row.getCell(5));
```

```
                String address =
getCellValueAsString(row.getCell(6));
                String notes =
getCellValueAsString(row.getCell(7));

                ExcelContact excelContact = new ExcelContact(
                        -1,

UserSecssion.getSecssion().getUser().getId(),
                        -1,
                        name,
                        "男".equals(sex) ? 1 : 0,
                        phone,
                        email,
                        workunit,
                        address,
                        notes,
                        groupname
                );
                excelContacts.add(excelContact);
            }

            return new Result(true, "读取成功", excelContacts);
        } catch (IOException e) {
            return new Result(false, e.getMessage());
        }
    }
    public Result saveContactsToDatabase(ArrayList<ExcelContact>
excelContacts) {
        GroupService groupService = new GroupService();
        ContactService contactService = new ContactService();
        User user = UserSecssion.getSecssion().getUser();
        for (ExcelContact excelContact : excelContacts){
            Result searchresult =
groupService.getUserGroupByName(excelContact.getGroupName(),user);
            Group group = (Group) searchresult.getData();
            if (searchresult.getStatus()){
                if (group==null){
                    Result addGroupResult =
groupService.addUserGroup(user.getId(),excelContact.getGroupName())
;
                    if (addGroupResult.getStatus()){
                        group = (Group)
groupService.getUserGroupByName(excelContact.getGroupName(),user).g
```

```
etData();
                    }else {
                        return new
Result(false,addGroupResult.getMessage());
                    }
                }
                Result addContactResult =
contactService.addContact(new Contact(
                        -1,
                        user.getId(),
                        group.getId(),
                        excelContact.getName(),
                        excelContact.getSex(),
                        excelContact.getPhone(),
                        excelContact.getEmail(),
                        excelContact.getWorkunit(),
                        excelContact.getAddress(),
                        excelContact.getNotes()
                ));
                if (!addContactResult.getStatus()){
                    return new
Result(false,addContactResult.getMessage());
                }

            }else {
                return new
Result(false,searchresult.getMessage());
            }
        }
        return new Result(true,"导入成功");
    }
    public Result exportExcel(ArrayList<Contact> contacts,
String filePath) {
        Workbook workbook = null;
        if (filePath.endsWith(".xls")) {
            workbook = new HSSFWorkbook();
        } else if (filePath.endsWith(".xlsx")) {
            workbook = new XSSFWorkbook();
        } else {
            return new Result(false, "文件格式错误");
        }
        Sheet sheet = workbook.createSheet("LinkUp");
        Row row = sheet.createRow(0);
        row.createCell(0).setCellValue("序号");
```

```
            row.createCell(1).setCellValue("组别");
            row.createCell(2).setCellValue("姓名");
            row.createCell(3).setCellValue("性别");
            row.createCell(4).setCellValue("电话");
            row.createCell(5).setCellValue("邮箱");
            row.createCell(6).setCellValue("工作单位");
            row.createCell(7).setCellValue("住址");
            row.createCell(8).setCellValue("备注");
            for (int i = 0; i < contacts.size(); i++) {
                Contact contact = contacts.get(i);
                row = sheet.createRow(i + 1);
                row.createCell(0).setCellValue(i + 1);
                GroupService groupService = new GroupService();
                Result groupResult =
groupService.getGroupByID(contact.getGroupId());
                if (groupResult.getStatus()){
                    row.createCell(1).setCellValue(((Group)
groupResult.getData()).getGroupName());
                }else {
                    row.createCell(1).setCellValue("");
                }
                row.createCell(2).setCellValue(contact.getName());
                row.createCell(3).setCellValue(contact.getSex() ==
1 ? "男" : "女");
                row.createCell(4).setCellValue(contact.getPhone());
                row.createCell(5).setCellValue(contact.getEmail());

row.createCell(6).setCellValue(contact.getWorkunit());
                row.createCell(7).setCellValue(contact.getAddress());
                row.createCell(8).setCellValue(contact.getNotes());
            }
            try (FileOutputStream outputStream = new
FileOutputStream(filePath)) {
                workbook.write(outputStream);
                return new Result(true, "导出成功");
            } catch (IOException e) {
                return new Result(false, e.getMessage());
            }
        }
}
```

4.1.5 通过责任链模式实现分组删除功能

Handler

```
public interface Handler {
    public abstract Result handleRequest(Group group, Group
defaultGroup,Result prevResult);
    public abstract void setNextHandler(Handler handler);
}
```

QueryGroupContact

```
public class QueryGroupContact implements Handler{
    private Handler handler;
    @Override
    public Result handleRequest(Group group, Group defaultGroup,
Result prevResult) {
        ContactService service = new ContactService();
        Result result = service.getContactsByGroupID(group.getId());
        if (handler!=null){
            handler.handleRequest(group,defaultGroup,result);
        }
        return result;
    }

    @Override
    public void setNextHandler(Handler handler) {
        this.handler=handler;
    }
}
```

MoveToDefautGroup

```
public class MoveToDefautGroup implements Handler{
    private Handler handler;
    @Override
    public Result handleRequest(Group group, Group defaultGroup,
Result prevResult) {
        if (prevResult!=null&&prevResult.getStatus()){
            ContactService service = new ContactService();
            ArrayList<Contact> contacts = (ArrayList<Contact>)
prevResult.getData();
            for (Contact contact:contacts){
                Result moveResult =
```

```
service.moveContact(contact,defaultGroup);
                if (!moveResult.getStatus()){
                    return moveResult;
                }
            }
            Result result=new Result(true);
            if (handler!=null){
                handler.handleRequest(group,defaultGroup,result);
            }
            return result;
        }
        return new Result(false,"move 责任链配置错误");
    }

    @Override
    public void setNextHandler(Handler handler) {
        this.handler=handler;
    }
}
```

DeleteUserGroup

```
public class DeleteUserGroup implements Handler{
    private Handler handler;
    @Override
    public Result handleRequest(Group group, Group defaultGroup,
Result prevResult) {
        if (prevResult!=null&&prevResult.getStatus()){
            GroupMapper mapper = new GroupMapper();
            Result result = mapper.deleteUserGroup(group);
            if (handler!=null){
                handler.handleRequest(group,defaultGroup,result);
            }
            return result;
        }
        return new Result(false,"delete 责任链配置错误");
    }

    @Override
    public void setNextHandler(Handler handler) {
        this.handler = handler;
    }
}
```

deleteUserGroup

```
@Override
public Result deleteUserGroup(Group chosenGroup,Group defaultGroup)
{
    QueryGroupContact queryGroupContact = new QueryGroupContact();
    MoveToDefautGroup moveToDefautGroup = new MoveToDefautGroup();
    DeleteUserGroup deleteUserGroup = new DeleteUserGroup();
    queryGroupContact.setNextHandler(moveToDefautGroup);
    moveToDefautGroup.setNextHandler(deleteUserGroup);
    return
queryGroupContact.handleRequest(chosenGroup,defaultGroup,null);
}
```

4.1.6 通过观察者模式实现响应式组件

TableModelChanged

```
public interface TableModelChanged {
    public void refreshTable(TableModel model);
}
```

TableRender

```
public class TableRender implements TableModelChanged {
    private static JTable theTable;
    private static TableModel model;

    public TableRender(JTable Table) {
        theTable=Table;
    }

    public void setModel(TableModel model) {
        TableRender.model = model;
        refreshTable(model);
    }

    @Override
    public void refreshTable(TableModel model) {
        theTable.setModel(model);
    }
}
```

ContactTableRender

```
public class ContactTableRender {
    public static void refresh(JTable table){
        TableRender render = new TableRender(table);
        ContactService service = new ContactService();
        if (Home.getChosengroup()==null){
            return;
        }
        Result result =
service.getContactsByGroupID(Home.getChosengroup().getId());
        String[] tabletitle = {"姓名","性别","电话","邮箱","工作单位","
住址"};
        DefaultTableModel model = new
DefaultTableModel(tabletitle,0){
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        ArrayList<Contact> contacts = (ArrayList<Contact>)
result.getData();
        Home.setContacts(contacts);
        for (Contact contact : contacts) {
            String sex = "男";
            if (contact.getSex()==0){
                sex="女";
            }
            Object[] row = {
                    contact.getName(),
                    sex,
                    contact.getPhone(),
                    contact.getEmail(),
                    contact.getWorkunit(),
                    contact.getAddress(),
            };
            model.addRow(row);
        }
        render.refreshTable(model);
    }
}
```

第5章 总结

在 LinkUp 通讯录项目的开发过程中，我不仅成功构建了一款高效、便捷的个人联系人管理工具，还通过实践深入学习并应用了多种设计模式，显著提升了系统的可扩展性和可维护性。这款通讯录提供了基础的注册登录功能，支持详细的分组管理和全面的通讯录管理，帮助用户轻松整理和查找联系人信息。此外，它集成了多功能查询和数据导入导出功能，旨在满足用户的多样化需求，提高日常沟通效率。

为了确保系统的稳定性和灵活性，在架构上借鉴了 Spring 的三层架构设计，并巧妙地融入了单例、备忘录、策略、外观、责任链和观察者共六种设计模式：

- **单例模式**被用于确保数据库连接和用户数据存储的唯一实例化，有效减少了资源开销，保证了系统资源的有效利用。
- **备忘录模式**实现了记住密码、自动登录和主题切换等功能，为用户提供更加个性化的使用体验，同时保存了系统的状态快照以便于恢复。
- **策略模式**对数据库操作进行了封装，使得不同的数据库操作可以通过定义一组算法互相替换，简化了数据库交互逻辑，提高了代码的灵活性和可读性。
- **外观模式**为复杂的数据导入导出功能提供了一个简单的接口，隐藏了背后的复杂性，使这些功能更加易于使用和维护，同时也增强了系统的模块化。
- **责任链模式**在用户删除一个分组时，负责将分组中的联系人按照预设规则移动到默认分组，避免了数据丢失，保持了联系人信息的完整性，体现了良好的错误处理机制。
- **观察者模式**实现了界面组件之间的松耦合，使得界面元素能够响应其他组件的变化，增强了用户体验的流畅性和即时性，促进了 UI 与业务逻辑的分离。

通过这个项目，我不仅掌握了如何在实际开发中应用设计模式来解决具体问题，还提升了面向对象编程的能力，学会了如何设计结构清晰、易于维护的代码。此外，此次项目开发也为我积累了宝贵的项目开发经验。从需求分析、架构设计到编码实现和测试，整个项目流程让我对软件开发有了更全面的理解，为未来的项目打下了坚实的基础。