1 实验目的

运用 Java 语言，以迭代方式逐步编程实现一个小型档案管理系统。由此了解软件开发的一般过程，深入理解面向对象语言的基本概念和基本原理，理解和掌握继承与多态、异常处理、输入输出流、GUI 设计、JDBC 数据库操作、网络编程、多线程等技术；熟练掌握在 Java 语言环境下，上述技术的具体实现方法。

2 系统功能与描述

2.1 总体结构

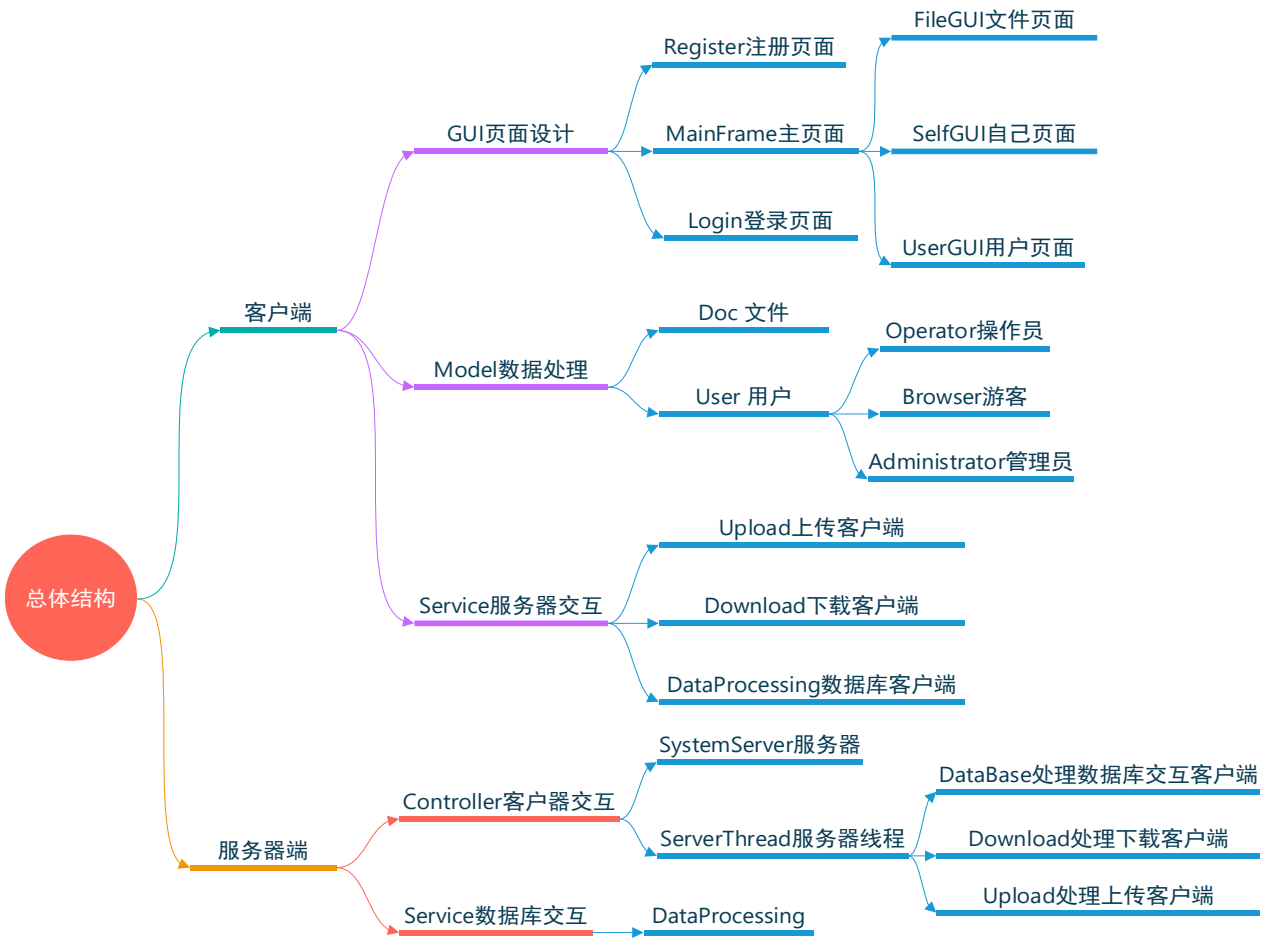

图 1：总体结构

如图 1 为该程序总体结构，分为服务器端与客户端，客户端分 Model，GUI，

Service 三层结构，服务器端分为 Controller,Service 两层结构。

2.2 基础功能介绍

2.2.1 登录

图 1 登录页面

功能介绍 如图 1,该页面主要用于用户登录,下面两个按钮分别是登录与注册，同时密码采用密码框，使密码输入更加安全。

正确返回 当可以连接服务器而且账号与密码匹配，就会进入主页面。

错误返回

1. 客户端会尝试连接服务器，如果连接成功就进入登录页面，反之就跳出的展示框，提示用户网络连接不成功。
2. 如果没有输入账号与密码就直接点击登录也会报错，提示用户需要输入。
3. 如果账号与密码不匹配，也会报错。

2.2.2 主页面

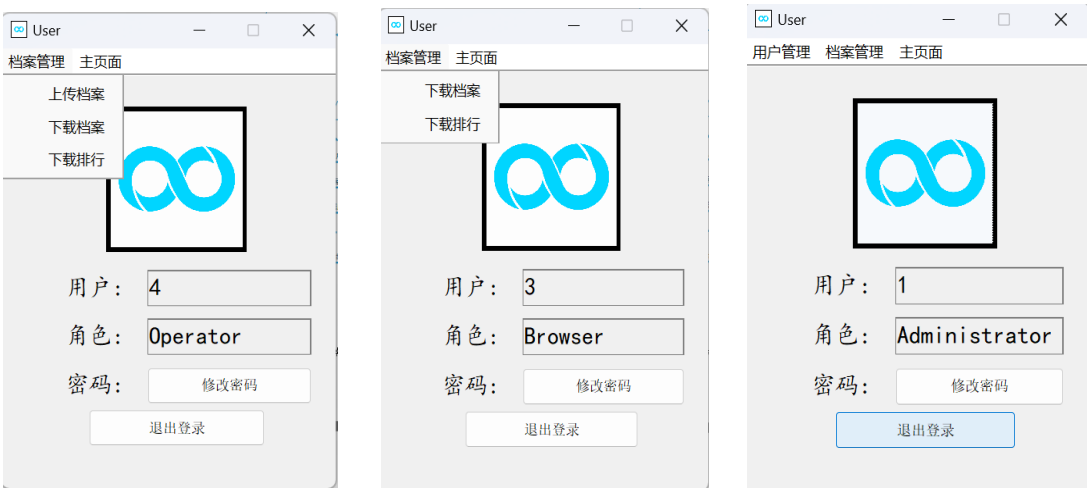

图 2 主页面

功能介绍 输入正确的密码与账号，可以进入主页面，主页面展示用户信息，主页面的菜单为用户的功能，根据不同的用户，我们提供了不同的功能，如 Operator 提供了上传档案的功能，Browser 的功能最少，只能下载档案，Administrator 用管理员，功能最多，可以对用户进行增删查改管理，但是不能上传用户

2.2.3 密码修改

图 3 密码修改

功能介绍 如图 3，为密码修改的功能页面，用户需要输入两次密码，而且两次密码一致就可以修改自己的密码。

正确返回 修改成功就会跳出修改成功的提示，并且重新登录为修改后的密码。

错误返回 如果两次输入密码不一致，就会输入提示，让用户重新输入密码。

2.2.4 用户管理

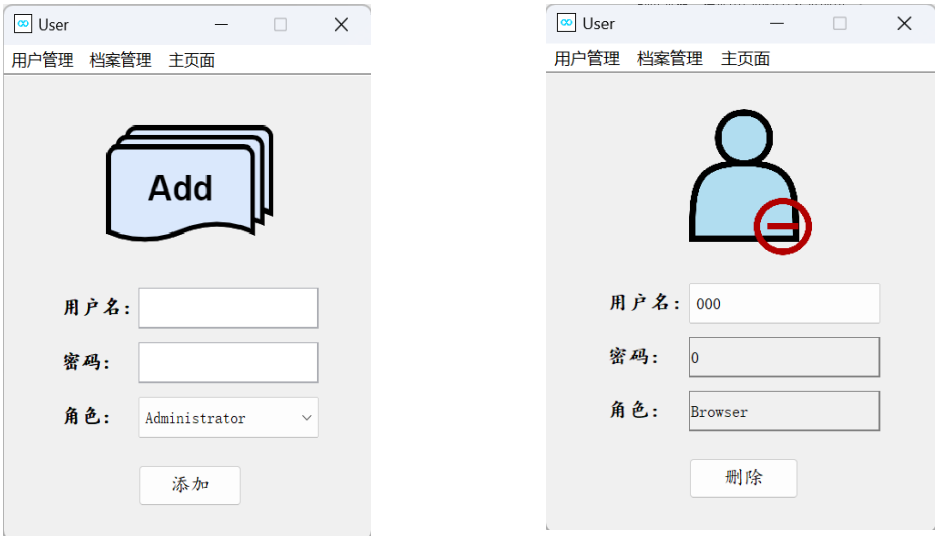

图 4 增加与删除

2.2.4.1 新增用户

功能介绍 如图 4，为增加用户的功能页面，管理员用户输入新增用户的用户名与用户密码，选定合适的角色就可以添加用户。

正确返回 添加成功后提示添加成功，同时数据库增加了这条用户信息。

错误返回

1. 如果用户重名会提示用户重名，提示用户修改用户名。
2. 用户没有填写所有信息就按下添加就会提示用户输入完整信息。

2.2.4.2 删除用户

功能介绍 如图 4，为删除用户的功能页面，管理员用户通过选择要删除的用户进行删除。

正确返回 删除成功就会返回删除成功信息，同时数据库删除了这条用户信息。

错误返回 如果未选择用户，则会提示未选择用户。

图 5 查询与更新

2.2.4.3 更新用户

功能介绍 如图 5，为更新用户的功能页面，管理员用户通过选择要求更新的用

户名，然后可以对密码与角色进行修改，最后点击更新就可以更新用户。

正确返回 更新成功数据库相关信息就会更新。

错误返回

1. 如果没有选择用户会提示未选择用户。
2. 如果没有输入要更新的用户，也会提示输入完整信息。

2.2.4.4 查询用户

功能介绍 如图 5，为查询用户页面，该页面展示了用户账号，密码与角色，管理员可以非常清晰的看用户信息。

2.2.5 文件管理

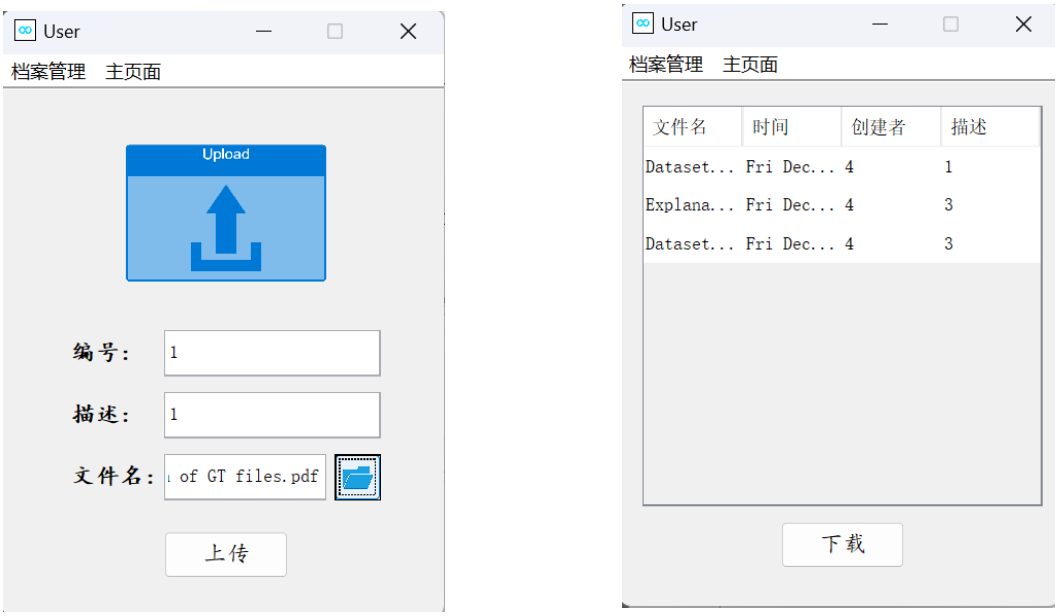

图 6 文件管理主页面

2.2.5.1 文件上传

功能介绍 如图 6，为文件上传主页面，输入编号与描述，再点击文件名旁边的按钮跳出文件选择框选择想上传的文件，点击上传，跳出上传的进度栏，进度栏结束即为上传成功。

正确返回 上传成功会跳出提示栏提示上传成功。

错误返回

1. 如果用户重名会提示用户重名，提示用户修改用户名。
2. 用户没有填写所有信息就按下添加就会提示用户输入完整信息。

2.2.5.2 文件下载

功能介绍 如图 4，为增加用户的功能页面，管理员用户输入新增用户的用户名与用户密码，选定合适的角色就可以添加用户。

正确返回 添加成功后提示添加成功，同时数据库增加了这条用户信息。

错误返回

1. 如果用户重名会提示用户重名，提示用户修改用户名。
2. 用户没有填写所有信息就按下添加就会提示用户输入完整信息。

2.3 拓展功能

2.3.1 Docker 与 properties 参数配置

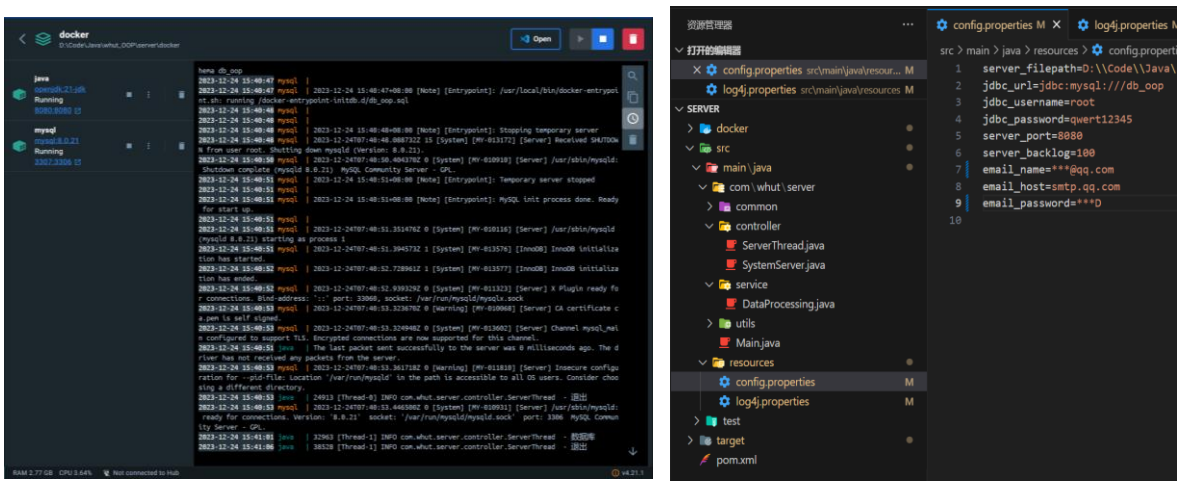

图 7Docker 与 properties 参数配置

功能介绍

Docker 与 properties 提高程序的灵活性，方便服务器程序快速部署。

1. server 服务端使用了容器技术，将服务器做成 Docker 应用，只需运行 docker-compose.yml 就可以快速在其他云服务器运行该应用，减少服务端配置数据库的时间，同时也减少后期运维成本。

2. properties 可以修改程序的一些重要参数，相比直接写死在代码里面，这种方式修改参数无需重复编译程序，用户只需修改 properties 文件，运行时指定采用哪个 properties 文件的参数就可以，这样使程序灵活性提高。

2.3.2 注册与邮箱验证

图 8 注册与邮箱验证

功能介绍 如图 8，为注册用户的页面，想要注册的用户可以输入自己的账号与密码，还有邮箱，获取验证码，输入邮箱验证即可注册一个 Browser 账号，注册功能使用户可以自己注册用户，无需通过管理员增加用户，同时增加了邮箱验证功能，通过验证码判断用户留下的邮箱是否有效。

正确返回 注册成功后提示注册成功，同时数据库增加了这条用户信息。

错误返回

1. 如果用户输入的验证码与发送的不一致，就会提示验证码不一致。
2. 如果用户填写验证码时间过于超过 5 分钟，则会提示用户重新获取。
3. 用户没有填写所有信息就按下注册就会提示用户输入完整信息才能注册。
4. 用户注册的名字与已有用户重名，也会提示用户换个名字。

2.3.3 下载排行榜

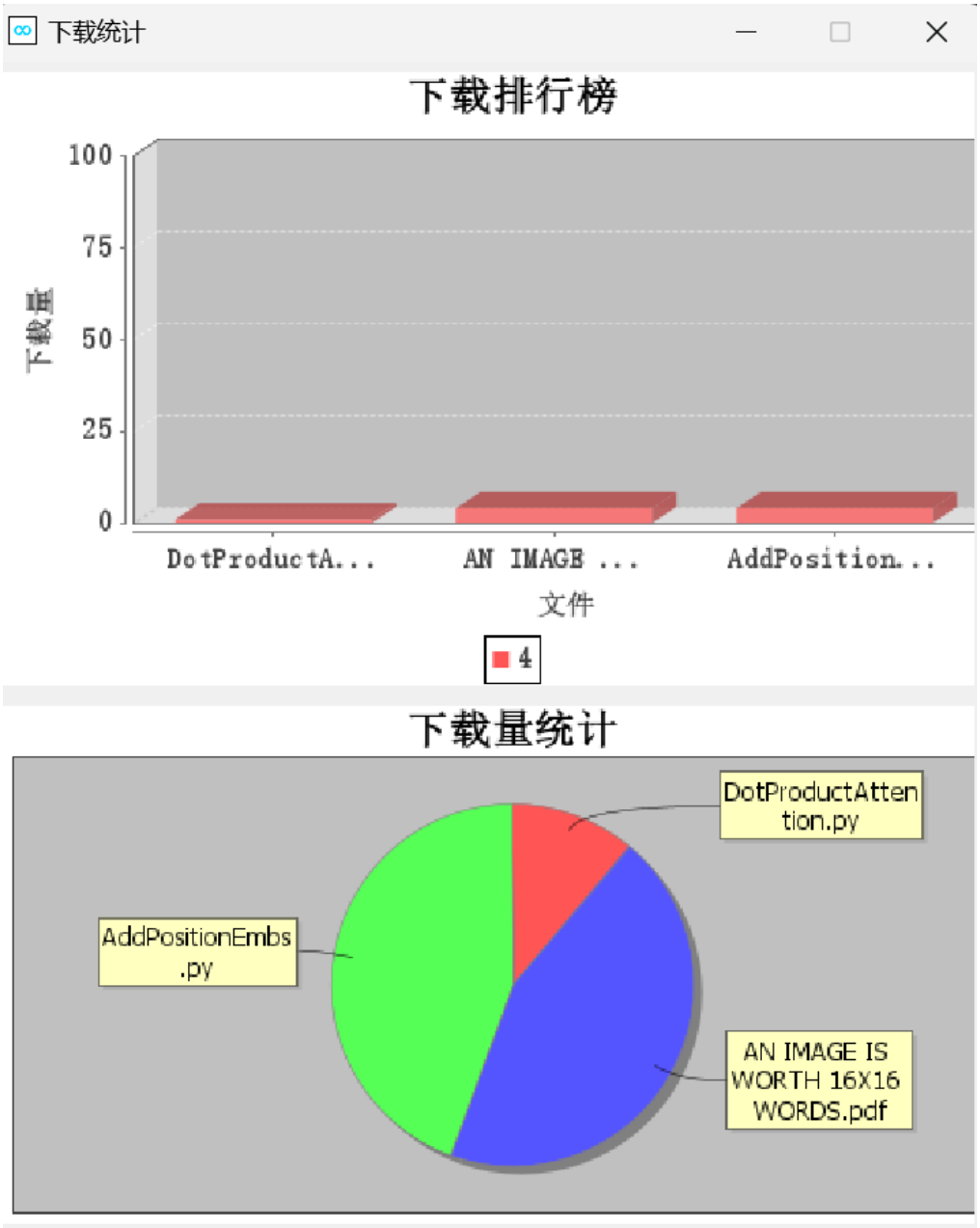

图 9 下载排行榜

功能介绍 如图 9 为下载排行榜，通过柱状图与扇形图，将下载量可视化，用户可以快速找到下载最多的档案，下载占比最多的档案，这些可视化功能可以帮助用户知道最近比较火的档案，操作员也可以根据排行删除一些下载量较少的文件。

2.3.4 版权说明

功能介绍

如图 10 点击主页面的图标，会跳出版权声明，这里也给出我的邮箱，也欢迎用户向我反映各种问题，以此不断完善该程序。

图 10 版权说明

3 模块设计

3.1 控制台模块

3.1.1 数据结构

Doc 类

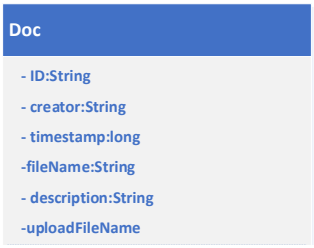

图 11Doc 类

User 类

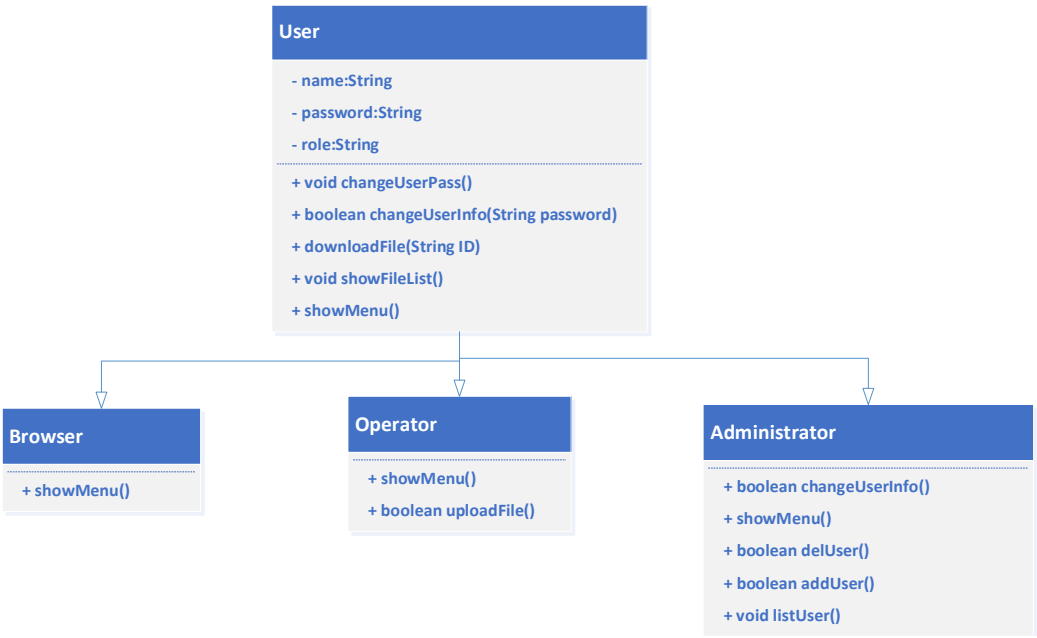

图 12User 类

图 11，图 12 为控制台程序的数据结构，分为 User 与 Doc 两个类，同时 User 类派生出 Broswer 游客类，Operator 操作员类，Administor 管理员类。

3.1.2 功能过程

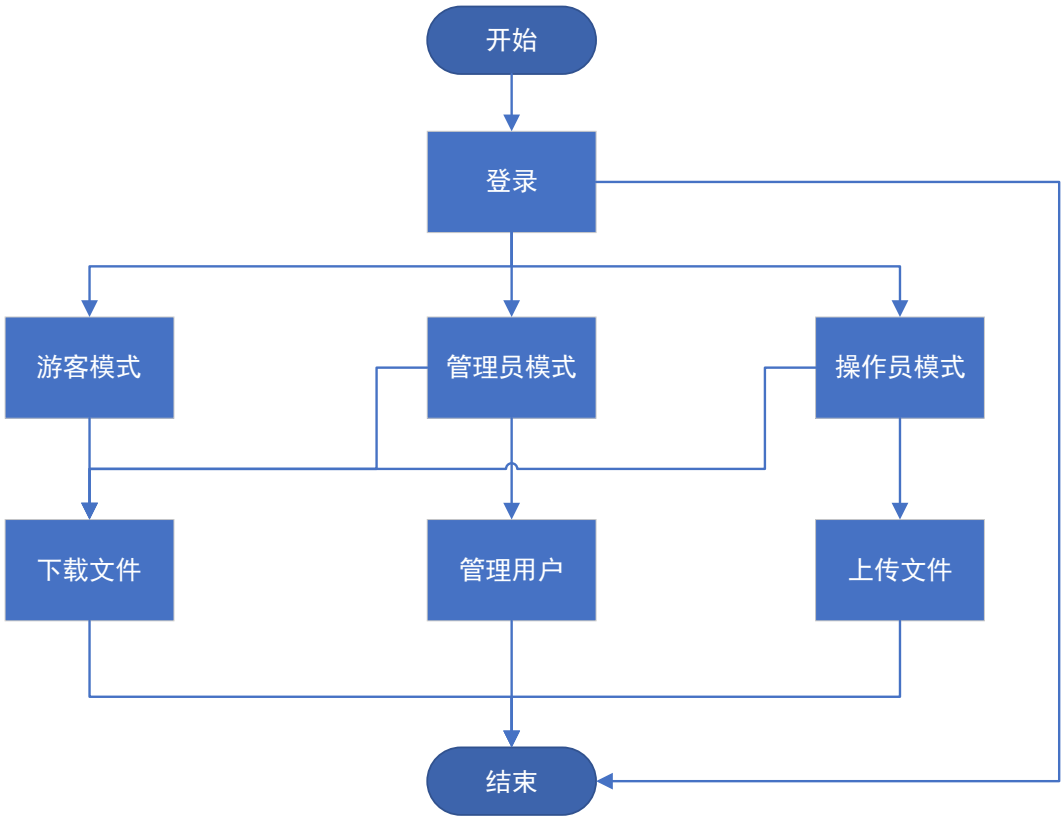

图 13 控制台功能过程

如图 13，这程序是一个用户权限管理系统，包含登录、用户模式切换、下载、上传和用户管理等关键功能。首先，用户通过提供用户名和密码进行登录，程序随后验证用户信息。验证成功后，系统创建一个用户对象，标识当前用户，并将其角色设置为默认的游客模式。

已登录用户可以切换用户模式，这包括游客、管理员和操作员。用户模式切换的权限仅限于已登录用户。这个特性允许用户根据需要访问系统中不同的功能，以适应其角色和职责。

系统支持文件的下载功能，对于所有已登录用户而言，无论是游客、管理员还是操作员，都具备下载文件的权限。这个功能执行文件下载操作，为用户提供了方便的资源获取途径。

上传文件的权限限制更为严格，只有以操作员身份登录的用户才能执行这一功能。这种设计确保了对上传功能的控制，只有拥有特定权限的用户才能向系统提

交新的文件。

用户管理是系统中的另一个核心功能，但其访问权限受到更为严格的控制。只有以管理员身份登录的用户才能执行用户管理操作。这包括添加、删除或修改用户信息等关键任务，以确保系统中的用户管理得以安全和有序进行。

3.1.3 模块接口

这个程序具备多个关键接口，以实现其各种功能。首先，登录接口承担着验证用户身份的任务，用户通过输入用户名和密码进行登录。登录接口的输出涵盖了登录状态的反馈，以及在成功登录的情况下可能返回的用户信息。这一接口的设计旨在实现对用户身份的有效验证，确保系统只允许合法用户进入。

用户模式切换接口提供了一种在已登录状态下调整用户权限的机制。通过输入目标用户模式，用户可以切换到游客、管理员或操作员模式。切换接口的输出反映了这一操作的结果，即切换是否成功。这个接口的存在使得用户可以根据其当前需要和角色要求，动态地调整系统中的功能访问权限。

文件下载接口是程序的另一个重要组成部分，其实现了用户获取系统资源的途径。由于下载通常是通过用户界面触发的，这一接口的输入较为简单，而其输出则传达了文件下载的状态，反映了操作的成功或失败。这为用户提供了便捷的方式来获取系统中所需的文件或信息。

文件上传接口具有更为严格的权限控制，只有以操作员身份登录的用户才能执行这一功能。输入包括上传文件的内容和相关信息，而输出则是上传状态的反馈，指示了上传操作的结果。这种设计确保了对于文件上传这一敏感操作的有序管理，有效防范了潜在的滥用。

用户管理接口是系统中另一个关键的功能接口，其任务是执行用户管理操作，包括添加、删除或修改用户信息等任务。这一接口的输入涵盖了用户管理操作和相关信息，而输出则传递了用户管理操作的状态，反映了操作的成功或失败。仅以管理员身份登录的用户才能执行用户管理操作，确保了对用户数据的安全管理。

3.2 通讯模块

3.2.1 数据结构

服务器

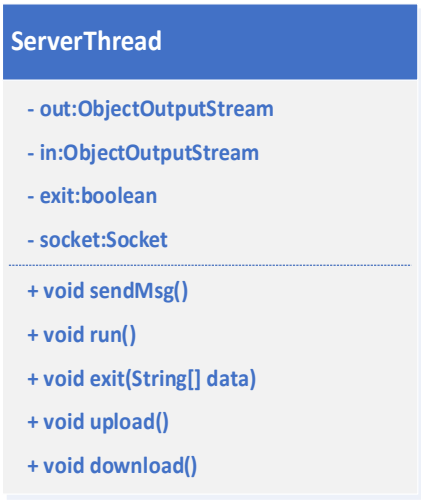

图 14 服务器线程类

客户端

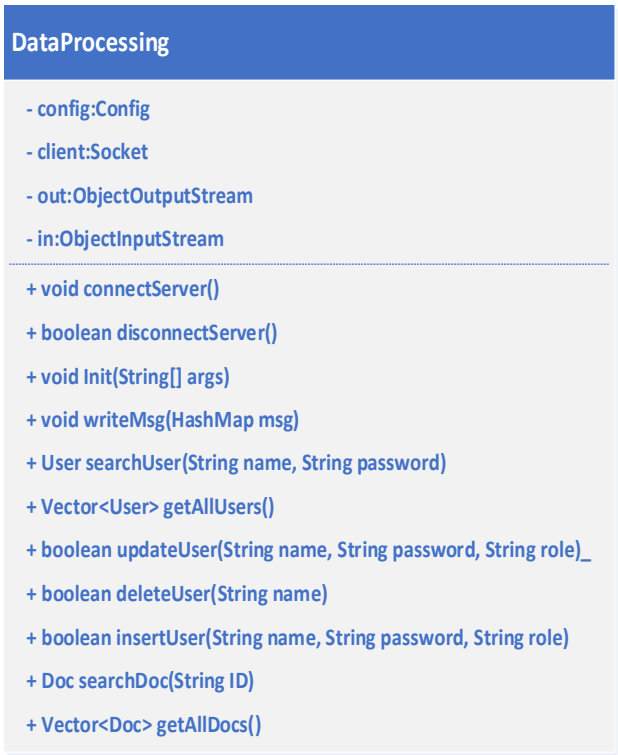

图 15 客户端线程类

传输数据

图 16 传输数据

如图 14 为服务器线程类的数据结构，图 15 为客户端类的数据结构，图 16 的 ClientMsg 为客户端传给服务器的数据的结构，Type 为消息类型，Data 为数据，ServerMsg 为服务器传给客户端的数据的结构，Status 为数据类型，Data 为数据。

3.2.2 功能过程

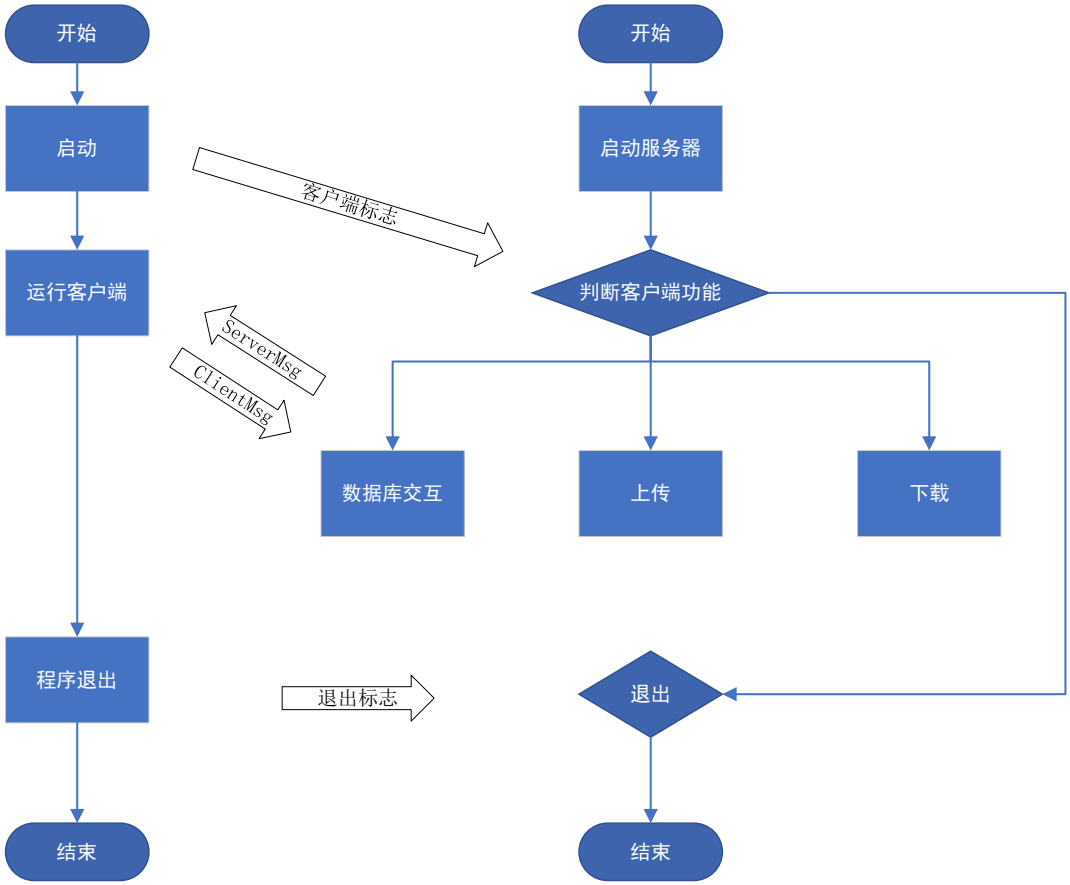

图 16 服务器与客户端交互过程

这是一个基于客户端-服务器模型的简单网络应用程序流程。在这个模型中，服务器首先启动并进入阻塞状态，等待客户端的连接。一旦有客户端连接，服务器会新开线程处理与该客户端的通信。

客户端启动后，会发送启动标志给服务器。服务器接收到标志后，通过解析

标志内容判断客户端的操作类型，可能是文件上传、文件下载或数据库操作。根据判断结果，服务器相应地进行处理。

对于文件上传和下载，客户端与服务器之间建立通信通道，通过该通道传输文件数据。服务器负责接收、存储或发送文件，并在完成后关闭相应的线程。这保证了多个客户端可以同时进行文件传输操作，而不会相互干扰。

对于数据库操作，客户端可能发送查询、插入、更新或删除等数据库指令。服务器接收这些指令后，与数据库进行交互，执行相应的操作，并将结果返回给客户端。这种架构使得多个客户端可以同时与服务器进行数据库交互，提高了系统的并发性能。

如果客户端发出退出标志，服务器会相应地关闭与该客户端相关的线程，确保资源的有效释放。这样，系统可以动态地处理多个客户端的请求，实现了高并发性和灵活性。

这个流程展示了一个简单而有效的客户端-服务器模型，适用于需要处理文件传输和数据库操作的网络应用程序。通过多线程的设计，系统能够同时处理多个客户端的请求，提高了整体的性能和响应速度。这种模型的灵活性和可扩展性使其成为许多网络应用程序的基础架构。

3.2.3 模块接口

这模型中，服务器端通过 SystemServer.start()函数启动，随后进入等待状态，准备接收客户端连接。一旦连接建立，服务器通过 ServerThread.run()解析客户端请求，确定操作类型。针对文件上传，服务器调用 upload()处理客户端上传的文件数据；对于文件下载，使用 download()处理客户端的下载请求。对数据库操作，服务器通过 DataProcessing 类处理客户端发送的数据库请求。最后，通过 close_connection(client_socket)可以关闭与特定客户端的连接。

在客户端方面，启动客户端应用。随后，通过 connectServer()向服务器发送启动标志，表示客户端已启动。对于文件上传，客户端可以使用 upload_file(file_path)函数将指定文件传输至服务器；而对于文件下载，通过 download_file(file_name)可以从服务器获取指定文件。对数据库操作，通过 DataProcessing 可以向服务器发送相应的数据库请求。当客户端需要关闭连接时，可使用 send_exit_flag()向服务器发送退出标志，通知服务器关闭与该客

户端的连接。

这些接口的设计使得服务器和客户端之间的通信和数据处理得以清晰分离，提高了系统的可维护性和扩展性。在实际应用中，可以根据具体需求添加额外的接口和功能，确保系统的健壮性和灵活性。

3.3 GUI 模块

3.3.1 数据结构

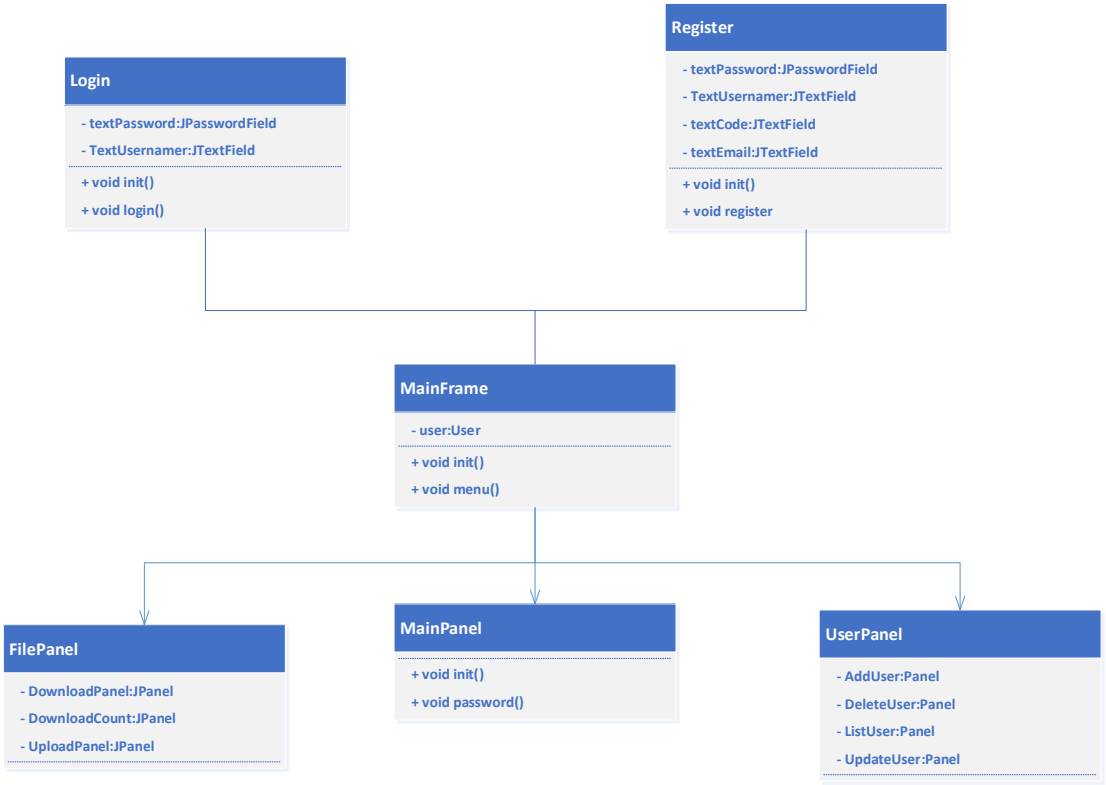

图 18GUI 模块数据结构

这 GUI 页面的类包括 Login、Register、MainFrame，以及其中的 FilePanel、MainPanel 和 UserPanel。Login 类包含属性：username_field（用于输入用户名的文本框）、password_field（用于输入密码的密码框）以及 login_button（登录按钮，用于触发登录操作）。Register 类包含属性：new_username_field（用于输入新用户名的文本框）、password_field（用于输入新密码的密码框）以及 register_button（注册按钮，用于触发注册操作）。MainFrame 类包含属性：file_panel（类型为 FilePanel 的文件管理面板）、main_panel（类型为 MainPanel 的主页面管理面板）和 user_panel（类型为 UserPanel 的用户管理面板）。在

FilePanel 类中，可以定义文件管理相关的组件，如文件列表、上传按钮、下载按钮等。在 MainPanel 类中，可以定义主页面相关的组件，可能包括通知区域、快捷操作按钮等。UserPanel 类包含用户管理相关的组件，如用户列表、增加用户按钮、删除用户按钮等。每个类都可以包含适当的方法来处理用户交互，例如登录、注册、文件管理、用户管理等操作。这样的结构使得 GUI 页面清晰、模块化，方便维护和扩展。通过调用不同类的方法或触发相应的事件，可以实现各个功能模块之间的协同工作。

3.2.2 功能过程

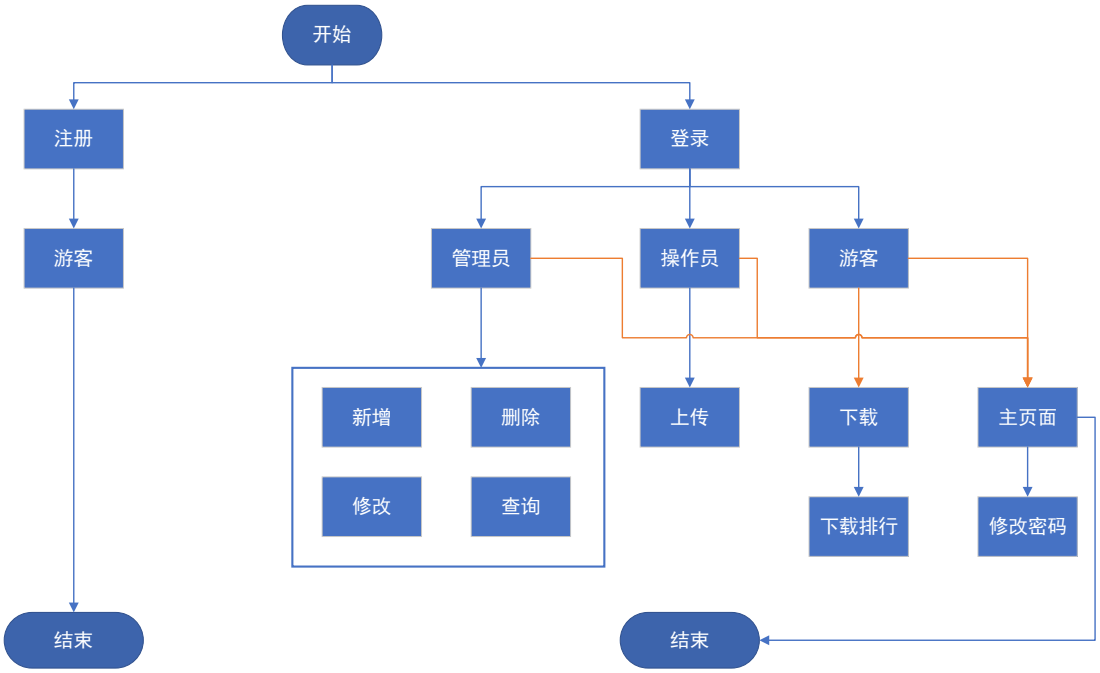

图 19GUI 页面

GUI 页面的流程按照以下步骤进行。首先，用户启动应用程序，触发应用程序的入口点。随后，应用程序呈现登录界面，其中包括用户名输入框、密码输入框以及登录按钮，用户通过输入用户名和密码并点击登录按钮进行登录。在登录阶段，系统调用登录类的方法，验证用户提供的用户名和密码的有效性。如果验证成功，系统切换到主页面，该页面包括文件管理、主要操作区域和用户管理等模块。

在主页面中，用户可以执行文件管理操作，如上传、下载、删除等。文件管理模块的逻辑由 FilePanel 类处理。同时，用户还可以在主要操作区域执行应用程序的核心功能，这可能涉及到业务逻辑和数据处理，主要操作区域的逻辑由

MainPanel 类处理。此外，用户可以进行用户管理，包括添加、删除、编辑用户等操作，UserPanel 类负责处理用户管理模块的逻辑。

用户还有注册新用户的选项，点击注册按钮后，应用程序显示注册界面，用户填写新用户信息。注册阶段，系统调用 Register 类的方法，验证新用户信息的有效性。如果验证通过，系统添加新用户并允许进行登录；否则，系统显示错误消息或保持在注册界面。

用户可以选择退出应用程序，触发应用程序的退出流程。这个流程将用户从登录到主要功能的整个交互过程清晰地划分为不同的步骤，每个步骤由相应的类和模块处理，确保了 GUI 页面的逻辑结构清晰可维护。

3.2.3 模块接口

在这个 GUI 应用程序中，我们设计了六个主要的类：Login、Register、MainFrame、FilePanel、MainPanel 以及 UserPanel。每个类都有其独特的数据结构和相应的接口，共同构建了一个模块化、清晰分层的 GUI 系统。

Login 类负责处理用户登录的逻辑。该类包含了用户名输入框、密码输入框和登录按钮等组件。其接口主要包括 login(username, password)方法，用于验证用户身份。这个接口实现了用户在登录界面输入用户名和密码后，系统对其进行验证，确保用户的合法登录。

Register 类专注于用户注册的功能。它包含了新用户名输入框、新密码输入框和注册按钮等组件。注册类的接口主要有 register(username, password)方法，负责将新用户的信息添加到系统中。这个接口实现了用户在注册界面填写新用户信息后，系统将其注册并加入用户列表。

MainFrame 类是整个应用程序的主页面管理类，其数据结构包括 FilePanel、MainPanel 和 UserPanel。主页面的切换和整体管理都由 MainFrame 来处理。接口可能包括 FilePanel()和 UserPanel()等方法，使用户可以在不同功能模块之间进行切换。

在文件管理方面，FilePanel 类负责处理文件上传、下载和删除等操作。该类包含了文件列表、上传按钮、下载按钮等组件。其接口包括 uploadFile(filePath)和 downloadFile(fileName)等方法，实现用户对文件的管理操作。这种模块化的设计使得文件管理独立于其他功能，便于维护和扩展。

MainPanel 类处理主页面的核心功能。主页面可能包括通知区域、快捷操作按钮等组件。用于执行主要操作区域的核心功能。这个接口实现了用户在主页面执行核心功能时，系统相应的逻辑处理。

UserPanel 类是用户管理的核心，负责用户的增加、删除、修改和查询操作。其数据结构包括用户列表、用户名输入框、密码输入框和相应的按钮。接口有 addUser(username,password),deleteUser(username),updateUser(username) 以及 searchUsers(keyword)等方法，实现用户管理功能的完整性。

这个 GUI 应用程序通过合理的类的数据结构和接口设计，使得用户可以在一个统一的界面中完成登录、注册、文件管理、主要操作以及用户管理等功能。这种模块化的设计理念有助于提高代码的可读性和可维护性，为 GUI 应用程序的开发和维护提供了有效的指导。

4 开发难点与体会

4.1 代码管理

难点

1. 代码迭代快，改变大，导致编写错误无法回溯。在这个项目中，一开始是一个控制台应用，但随着时间的推移，需求不断变化，项目不断演变。首先，由于用户体验的要求，决定将应用改写成 GUI 形式，这就引入了大规模的代码改变。而后，为了实现网络功能，又加入了网络编程模块，这使得项目的改动更加剧烈。这样的快速迭代和大规模的改变，导致了编写错误难以回溯的问题。例如，在 GUI 和网络编程的引入过程中，可能出现了一些隐藏的错误，而这些错误的定位和修复将变得相当耗时。

2. 涉及引入大量第三方 jar 包，导致不便管理。这次项目需要引入 jdbc 与 jfreechart 画图与 email，log4j 日志的 jar 包，这使得项目变得臃肿，难以维护。同时，版本控制和协同开发也可能受到依赖管理不善的影响。

3. 模块较多，导致代码结构容易混乱。在这次开发中，当项目逐渐扩大，GUI 模块,网络编程等模块增多时，代码结构可能会变得难以理解和维护。这可能阻碍新成员加入项目或导致团队成员在代码库中迷失方向。

体会

1. 使用 Git 管理

采用 Git 版本控制系统可以有效解决由于代码迭代快、改变大而导致编写错误无法回溯的问题。通过将每一次代码变动都提交到 Git 中，项目的整个演进历程都得以清晰记录。在实际应用中，我经历了将项目从控制台应用改写成 GUI 并引入网络编程的情况。通过 Git 的提交历史，我能够轻松地追溯到不同阶段的代码状态。这在调试和问题排查时提供了极大的便利，避免了在修改频繁的情况下迷失在代码堆栈中。

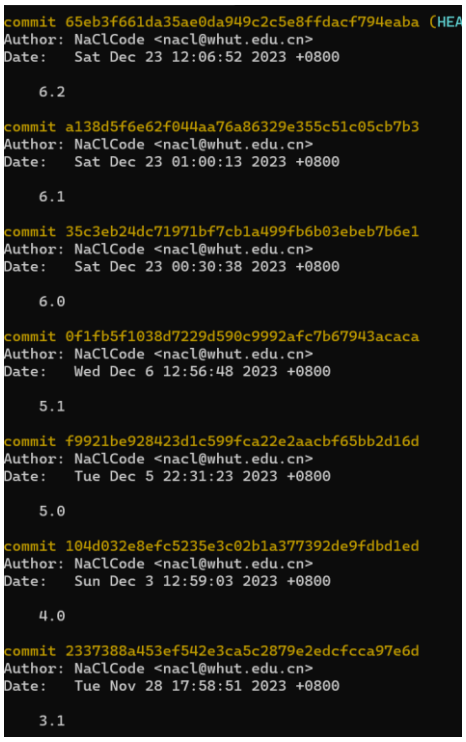

图 20 Git 管理项目

Git 的分支管理对于处理不同阶段的开发也非常有帮助。为控制台版本创建一个分支，为 GUI 版本创建另一个分支，这使得不同阶段的开发可以并行进行，而不会相互干扰。

2. 使用 Maven

在项目中运用 Maven 解决依赖管理问题，我个人深感 Maven 的便利和价值。通过在项目的 pom.xml 文件中明确声明对关键库的依赖，我将项目的依赖关系变得清晰可见。Maven 的自动下载和管理库的特性使得整个过程更加高效，无需手动处理繁琐的依赖项。

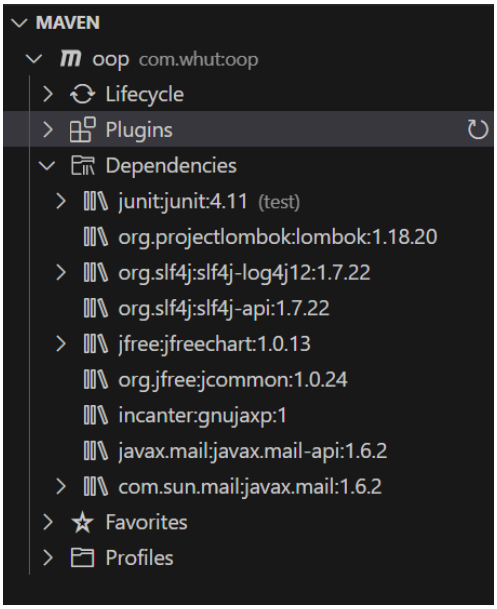

图 21Maven 工程

采用 Maven 的依赖管理机制，我体会到集中管理库版本的重要性。确保项目中所有库使用相同版本有助于避免潜在的版本兼容性问题，使项目在整个生命周期中保持稳定和一致。

我发现 Maven 排除不必要的传递性依赖的功能也非常实用。这使得项目能够保持轻量化。

2. 使用 MVC

在构建客户端与服务端的档案管理系统时，采用 MVC 架构对于解决模块较多导致代码结构混乱的问题起到了显著的作用。通过将系统划分为模型、视图和控制器这三个独立的组件，我能够更清晰地组织和管理不同层次的代码，提高了整个系统的可维护性和可扩展性。

在模型层，我将档案数据的存储和处理独立出来，实现了数据操作和业务逻辑的模块化。这种设计使得模型层更易于扩展和维护，同时也方便进行单元测试，确保数据处理的正确性。

视图层的引入使得用户界面的设计和展示与底层的业务逻辑分离开来。这种分层设计使得前端开发更为清晰和灵活，不同视图的变化不会对底层业务逻辑产生过大影响。我发现这样的结构在应对不同用户交互需求时更加灵活，有助于提高用户体验。

控制器层负责处理用户的输入，将用户请求传递给模型进行数据处理，并更新视图以展示最新的结果。这种分离用户输入处理和业务逻辑的方式提高了系统的可维护性和可测试性，使得团队能够更容易理解和调试代码。

整个系统通过 MVC 架构的独立模块划分，使得每个模块都能够独立开发、测试和维护。

4.2 数据库配置

难点

1. 参数设置与拼写错误： 在配置数据库时，需要填写各种参数，如数据库类型、主机地址、端口号、用户名、密码等。由于这些参数通常都是字符串形式，一小处的拼写错误或者不正确的参数值都可能导致连接失败。

2. 版本兼容性问题：选择正确版本的数据库驱动对于 JDBC 连接至关重要。不同的数据库版本可能需要不同版本的驱动程序，而且驱动程序本身也会有更新。如果数据库版本与驱动程序版本不兼容，可能会导致连接失败或出现意想不到的问题。

3. 调试与日志： 当连接失败时，定位问题也是一个挑战。数据库连接的错误信息可能并不总是清晰明了，需要通过查看日志或者启用更详细的调试信息来

定位问题。

体会

我使用 Docker 解决上面问题，使用 Docker 的时候，我发现它在解决数据库配置和 JDBC 连接方面提供了很多便利。

Docker 允许将整个应用程序及其所有依赖项打包成一个容器，包括数据库配置、驱动程序等。这种方式确保了在不同的环境中（开发、测试、生产）使用相同的容器，减少了由于环境差异导致的问题。此外，通过在 Docker 镜像中明确定义应用程序和依赖项的版本，包括数据库驱动的版本，有助于解决版本兼容性问题。

在网络配置方面，Docker 允许开发人员定义容器之间的网络设置，包括端口映射，从而简化了与数据库之间的网络通信，减少了网络问题的可能性。同时，Docker 容器生成的日志提供了关于应用程序运行状态和错误的详细信息，通过查看容器日志，开发人员可以更容易地定位数据库连接问题，识别拼写错误或其他配置问题。

在安全性设置方面，Docker 提供了一些机制，比如通过环境变量或 Docker Secrets 等方式安全地传递敏感信息，如数据库密码。这有助于在不暴露敏感信息的前提下进行数据库连接的安全配置。

通过使用 Docker Compose 等工具，可以轻松地一键启动整个应用栈，包括数据库和应用程序。这简化了开发人员在新环境中快速部署和调试的过程，减少了连接问题的出现。综合而言，Docker 提供了一种更加便捷、可控和可重复的部署方式，能够有效减少数据库配置和 JDBC 连接的难点，提高开发和部署的效率。

4.3 页面编写

难点

1. 在使用 Swing 构建档案管理系统 GUI 时，会面临一系列困难。需要考虑合适的布局管理器以及如何设计直观、用户友好的界面。选择合适的组件、颜色和图标，以及设计有效的布局，都需要一定的用户界面设计知识。此外，确保系统在不同环境下的一致性，需要细致处理数据模型与显示的分离，以及事件处理

的逻辑。

2. 在编写过程中避免重复创建对象导致的资源浪费。重复创建对象可能导致系统内存的不必要占用，从而增加了系统的内存负担。这不仅可能导致系统性能下降，还可能引发内存不足的问题，最终导致系统崩溃或变得异常缓慢。大量的无效对象可能会使代码变得混乱不堪，增加了后续维护的难度。同时，当系统需要进行扩展或优化时，资源浪费问题可能会成为系统架构的瓶颈，制约了系统的进一步发展。

体会

1. 自定义图标与事件处理

在界面设计上，选择合适的布局管理器、组件、颜色和图标并不是一件容易的事情。结合功能介绍，我自己设计一套图标，取代 java 原本的图标，确保用户能够轻松理解和使用系统。这需要一些实践和尝试，通过不断调整和反馈，逐渐找到最佳的设计方案。

事件处理的逻辑也是一个需要深思熟虑的问题。在处理用户交互时，需要确保事件响应的准确性和效率。我学到了如何合理使用事件监听器，以及在需要时进行异步处理，以提高系统的响应速度。这方面的体会主要来自于在实际项目中不断调试和优化事件处理代码的过程。

2. 使用单例模式

在 Swing 构建档案管理系统 GUI 时，采用单例模式是一种有力的解决方案，用于应对对象重复创建的问题，尤其是在页面或组件的生成过程中。通过使用单例模式，可以确保特定页面或组件的类仅有一个实例，并通过全局访问点提供统一的访问方式，从而避免在不同地方多次创建相同的对象。

通过设计一个单例页面类，可以防止外部直接实例化，并确保类的构造函数是私有的。通过在类内部提供一个静态方法，如 getInstance()，来获取类的唯一实例。这样一来，系统中的任何地方都可以通过该方法获取相同的页面实例。在单例页面类中，可以添加一些方法来管理页面的生命周期，例如初始化页面内容和刷新页面数据。这有助于确保页面在整个应用生命周期内保持一致，而不会发生多次创建相同页面的情况。

通过在需要使用页面的地方调用单例页面类的静态方法，即可获取页面实例。这样可以保证在整个应用中使用的始终是同一个页面对象，避免了对象重复创建的问题。

使用单例模式的好处不仅在于避免了资源浪费，还确保了页面状态的一致性。在 Swing 应用中，特别是对于需要在全局范围内共享的页面或组件，采用单例模式是一种可靠的设计选择。通过这种方式，可以提高系统性能，优化资源利用，同时确保页面的统一管理和一致性。

4.4 网络编程

难点

1. 连接管理： TCP 是面向连接的协议，因此需要有效地处理连接的建立和断开。在档案管理系统中，可能需要考虑如何优雅地处理客户端与服务器的连接，以及在连接断开时如何进行适当的清理和资源释放。

2. 并发处理： 当多个客户端同时连接到服务器时，需要考虑并发处理的问题。有效的并发管理是确保系统性能和响应速度的关键。使用多线程或异步编程模型来处理并发请求可能是一种解决方案。

3. 数据序列化和反序列化： 在 TCP 通信中，需要将数据进行序列化发送，并在接收端进行反序列化。选择合适的序列化机制，并考虑数据的版本兼容性，是确保通信正常进行的关键。

体会

1. 连接管理

在处理连接的建立和断开时，我认识到优雅的连接管理对系统的健壮性至关重要。采用连接池是一个有效的方式，能够降低连接的创建和销毁开销，提高系统的性能。使用心跳机制可以实时检测连接状态，帮助快速发现连接断开的情况，进而采取相应的措施。在断线重连策略方面，我学到了如何在网络不稳定的情况下，通过自动重新建立连接，保持系统的稳定性。适当设置连接超时时间是在保证及时释放资源的同时，避免不必要等待的关键。

2. 并发处理：

面对多个客户端同时连接的挑战，我发现采用多线程或异步编程模型是一种有效的解决方案。通过合理的并发管理，可以充分利用系统资源，提高响应速度。然而，需要注意处理并发操作时的线程安全性问题，以防止数据竞争和状态不一致。我还学到了如何设计合适的同步机制，确保多线程环境下的数据一致性。

3. 数据序列化和反序列化：

在 TCP 通信中，数据序列化和反序列化是确保数据正确传输的关键。我体会到选择合适的序列化机制是非常重要的，根据具体情况选择性能高效且适用的序列化方式。考虑数据的版本兼容性是防止未来升级时出现兼容性问题的重要考虑因素。在实践中，我学到了如何使用序列化框架，并通过版本控制等手段确保数据的正确传输和解析。

5 实验总结

在构建档案管理系统的实验中，我涉及了多个关键技术领域，从 Java Swing 页面设计到网络编程、注册验证 Email，再到柱形图展示下载量，以及使用 Git 和 Maven 进行项目管理，最终将服务端运行在 Docker 容器中。这个全面的实验为我提供了在实际项目中应用多种技术的机会，以下是我在不同方面的个人总结和体会：

通过使用 Java Swing 进行页面设计，我深刻认识到了用户界面设计的重要性。学会如何选择合适的布局管理器、组件和颜色，以及如何处理用户交互事件，是确保系统界面直观友好的关键。这个经验让我更加自信地处理 Swing 框架的 GUI 设计。

实现网络编程中的文件上传和下载涉及到了 TCP 通信、数据传输可靠性以及文件 IO 等方面的知识。解决并发性和可靠性问题，以确保文件传输的稳定性，是在实际网络应用中至关重要的。这个经验增强了我的网络编程技能。

在用户管理方面，注册验证 Email 功能的实现让我深入了解了用户认证的流程。通过使用 JavaMail API，我成功实现了注册时的邮件验证，提高了系统的安全性和用户管理的可靠性。

数据可视化方面，通过柱形图展示下载量，我学到了如何使用图表库来呈现数据。这种直观的可视化方式为用户提供了清晰的反馈，是提升系统用户体验的重要手段。

在团队协作和版本控制方面，我通过使用 Git 进行代码管理，学到了如何有效地进行分支管理、合并代码以及解决冲突。

项目管理方面，使用 Maven 简化了依赖管理和构建过程。我了解了如何配置 Maven 项目，管理依赖关系，以及如何使用 Maven 的生命周期和插件来管理项目的构建和打包。

最后，将服务端运行在 Docker 容器中是提高系统可移植性和环境隔离性的有效手段。我了解了 Docker 的基本概念，学会了如何编写 Dockerfile、构建镜像、运行容器，以及如何使用 Docker Compose 来管理多个容器的协同工作。

为了测试程序，使用腾讯云服务器进行测试，下面是运行结果图：

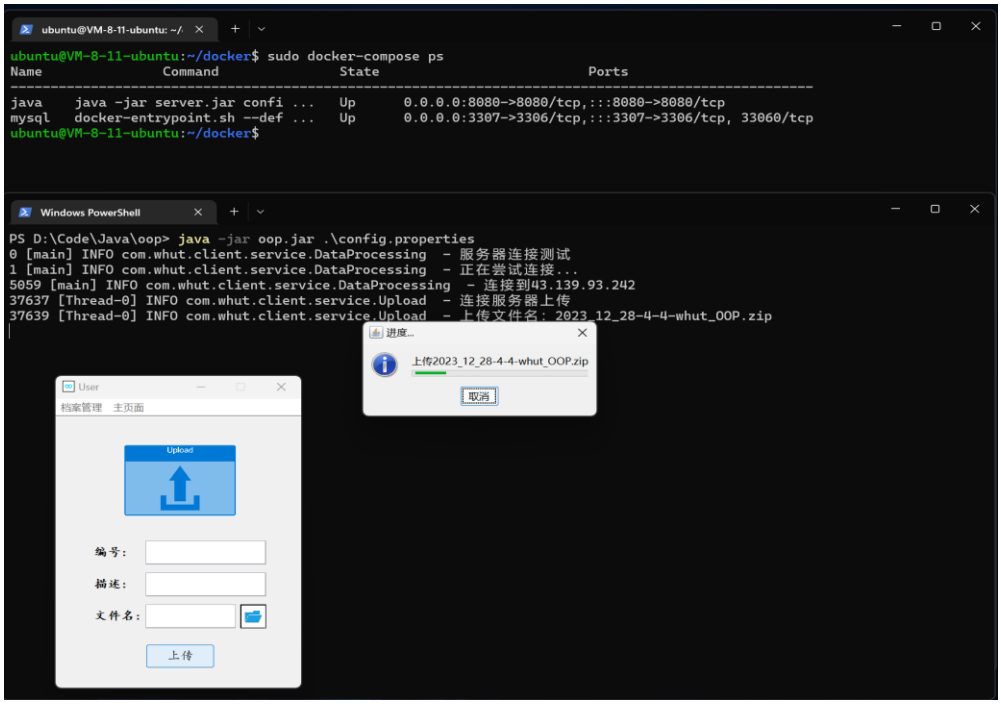

图 22 云服务器测试图

总体而言，这次实验是一个全面而深入的项目实践，涵盖了软件开发的各个方面。通过解决实际问题，我不仅掌握了具体的技术和工具的使用，还提高了问题解决和综合应用不同领域知识的能力。这个实验为我搭建了一个完整的项目实践框架，使我更加自信地应对未来的项目挑战。