摘 要

计算机技术的快速发展促进了各个行业的多方面发展，而网络技术的发展促进了各行各业的信息融合，这两门技术相辅相成。计算机技术中各种高级编程语言的发展，使得普通人也有机会模拟网络的传输过程，并且利用现有的工具直接利用网络编程实现自己想要的功能，编写自己的软件。在JAVA中，就有网络编程的相关类，程序设计人员可以利用该类设计自己的协议，搭建自己的服务器，开发出一套简单的文件管理系统，其他使用者可以使用相对应的客户端对服务端上的文件进行远程管理。同时，利用JAVA提供的GUI工具与多线程技术，编程人员可以实现多用户链接，并且为服务端和客户端设计比较优美的操作界面。

**关键词：**JAVA，网络编程，协议，GUI，多线程

目 录

[摘 要 1](#_Toc18673)

[第一章 绪 论 4](#_Toc15768)

[1.1 服务端与客户端概述 4](#_Toc12882)

[1.2 网络概述 4](#_Toc12547)

[1.3 JAVA概述 4](#_Toc24451)

[第二章 系统分析 5](#_Toc20663)

[2.1 功能分析 5](#_Toc20916)

[2.1.1 需求功能 5](#_Toc12519)

[2.1.2 基础操作 5](#_Toc3326)

[2.1.3 功能流程图 5](#_Toc27412)

[2.2 类关系分析 6](#_Toc7021)

[2.2.1 类概述 6](#_Toc1411)

[2.2.2 工具类-MyStreamMethod与MyActionMethod 6](#_Toc1292)

[2.2.3 服务端与客户端类-MySever、MyClient 7](#_Toc32645)

[2.2.4 特殊实现类-MyRunnable、InputWindow 7](#_Toc19858)

[2.3网络协议简述 7](#_Toc28912)

[第三章 详细设计及实现 8](#_Toc19572)

[3.1 视窗及双端初始化与连接 8](#_Toc24189)

[3.2 简单文件传输协议实现 8](#_Toc6742)

[3.3 流式工具类实现-MyStreamMethod 9](#_Toc12767)

[3.4 监听类实现-MyActionMethod 12](#_Toc26422)

[3.5 服务端线程类-MyRunnable实现 13](#_Toc31698)

[第四章 测试 14](#_Toc26310)

[4.1 程序运行环境 14](#_Toc6012)

[4.1.1程序运行的软件环境 14](#_Toc11478)

[4.1.2程序运行的硬件环境 14](#_Toc16836)

[4.2 客户端与服务端界面 14](#_Toc25015)

[4.3遍历文件测试 15](#_Toc3225)

[4.4 上传文件测试 16](#_Toc14072)

[4.5 下载文件测试 17](#_Toc5436)

[4.6 下一级及进入按钮测试 18](#_Toc1788)

[4.7 创建及删除文件测试 19](#_Toc24072)

[4.8 下一级按钮的预览功能测试 19](#_Toc7725)

[第五章 总结 20](#_Toc28497)

[5.1功能完整性总结 20](#_Toc25308)

[5.2程序不足 20](#_Toc16456)

[5.3功能展望 20](#_Toc30989)

[参考文献 21](#_Toc12304)

第一章 绪 论

1.1 服务端与客户端概述

客户端（Client）是发送请求（request），服务器端（Service）是响应请求（response），返回相应的资源数据。服务端是为客户端服务的，服务的内容诸如向客户端提供资源，保存客户端数据。

客户端或称为用户端，是指与服务器相对应，为客户提供本地服务的程序。除了一些只在本地运行的应用程序之外，一般安装在普通的客户机上，需要与服务端互相配合运行[1]。

客户端与服务端相互联系，相互配合，完成服务端能提供和客户端想实现的功能。客户端与服务端一般通过网络发送指令完成功能。

1.2 网络概述

计算机网络是指将不同地理位置，具有独立功能的多台计算机及网络设备通过通信线路(包括传输介质和网络设备连接起来)，在网络操作系统、网络管理软件及网络通信协议的共同管理和协调下实现资源共享和信息传递的计算机系统。[2]

通过网络，用户可以跨地区传达信息，大大方便了人类的生活，提高了信息传达效率，实现了天涯若比邻的实时传输。

网络通信协议是根据网络上的节点进行通信的一组规则，每种设备都可以根据通信协议识别其他设备的信息。使用相同协议的加算计可以识别彼此之间传递的信息。[3]方便了用户与多个用户之间信息传递，是网络中的一门通用语言。

1.3 JAVA概述

Java是一种通用的，基于类的，面向对象的编程语言。它是用于应用程序开发的计算平台。因此，Java是快速，安全和可靠的。它被广泛用于在笔记本电脑，数据中心，游戏机，科学超级计算机，手机等中开发Java应用程序。

JAVA编程语言发展迅速，技术人员编写了许多功能强大且实用的库，供后来的学习者学习使用。

第二章 系统分析

2.1 功能分析

2.1.1需求功能

1.在客户端远程创建单层目录

2.在客户端远程上传多个或单个文件（夹）

3.支持删除某个文件或文件夹

4.在客户端浏览服务端文件及文件夹

2.1.2基础操作

1.客户端与服务端窗口实现及按钮监听实现

2.客户端与服务端通过网络连接

3.进行操作后双端进行日志打印

4.双端对本地文件进行操作

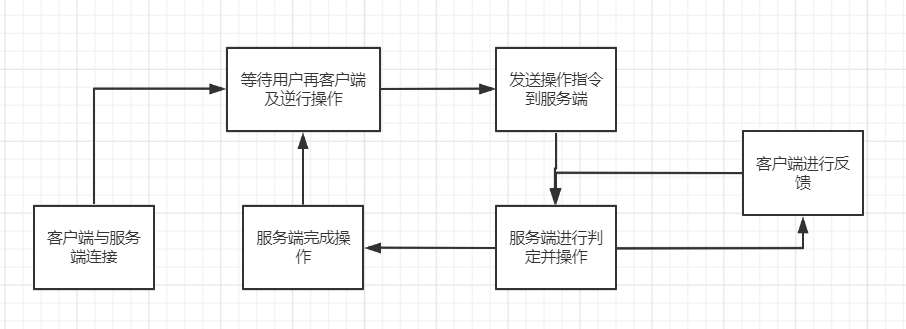
5.双端通过网络信道发送接收byte流

6.协议实现，规定byte流格式，长度及信息

7.服务端通过网络传输byte流后分析操作指令

8.服务端通过指令进行操作并发回反馈

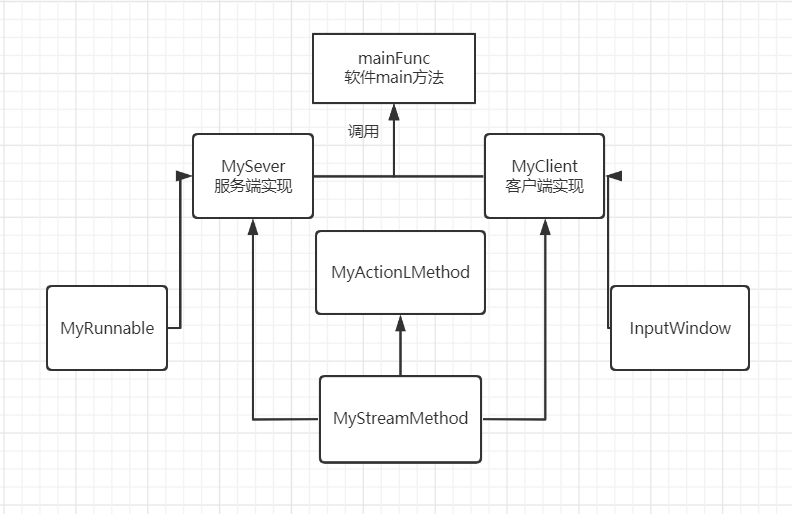
9.服务端对本地文件进行基本操作

2.1.3 功能流程图

2.2 类关系分析

2.2.1 类概述

本设计的软件实现分为客户端与服务端，而两个软件的实现中有许多共通之处，因此在将软件功能实现后，将两个软件中使用的相同功能以及复杂功能分化出来，分类放到其他源文件中，以此来减少单个文件的代码行数，也降低了代码阅读的难度和时间。下图是有关本软件内部类的关系图：

图2.2.1 类的调用关系图

2.2.2 工具类-MyStreamMethod与MyActionMethod

在软件代码实现中，工具类主要分为两类，一类是对本地文件及网络进行操作的代码，主要包括对文件的遍历、读取、删除、复制，以及利用网络信道发送及接收byte流。这一类方法的本质都是利用JAVA中的流进行操作,因此将这一类方法归到同一工具类中。另一类则是两个软件中对按钮监听的方法实现，这一类代码的特征是类型多，占用的代码量比较大，因此将其单独归类，阅读者只需要知道方法的用途，分析时再去寻找对应的源码即可。

MyActionMethod类中的方法，大多数在客户端MyClient类中被调用，也调用了MyStreamMethod中的方法，因此MyStreamMethod在整个软件中为最底层实现。

2.2.3 服务端与客户端类-MySever、MyClient

这两个类都继承于JFrame类，实现了ActionListener接口方法。

在这两个类中，主要包括了初始化方法，在初始化方法中，为窗口设置大小参数，对按钮进行排列，对按钮实现监听功能，监听的具体实现被分到MyActionMethod工具类中，监听到按钮操作后调用对应函数实行功能。

在客户端初始化之前，需要弹出小窗口，让用户输入服务器IP地址，此软件因在本地，未获得公网静态IP，因此避免出现更换网络后无法连接的情况，需要客户获取服务器IP再进行连接。

初始化之后，客户端申请与服务端连接，服务端等待申请连接并阻塞直到接收下一个申请。之后等待用户在客户端进行操作。

2.2.4 特殊实现类-MyRunnable、InputWindow

这两个类都只调用过一次，分别在服务端与客户端。

MyRunnable类继承Runnable接口，实现run（）方法，是服务端的具体功能实现方法，当有一个新的客户端进行连接，便在服务端的线程池中加入一个新的MyRunnable类等待对应客户操作。其阻塞等待接收客户端发出的byte流，根据byte流的信息对本地文件进行操作，以此实现客户端对服务端内文件的操作。

InputWindow类在客户端初始化时调用，弹出窗口，接收用户输入的IP地址，传递到客户端，以便客户端进行连接。

2.3网络协议简述

在本软件实现中，需要频繁的在两端之间传递byte流，每次传递的信息都不尽相同，因此设计了一个简易协议帮助软件识别需要进行的操作，调用不同的方法。此简易协议规定了每次发送byte流的长度以及格式，在这里不仔细分析，具体讲述在第三章。

第三章 详细设计及实现

3.1 视窗及双端初始化与连接

MySever类、MyClient类与InputWindow类三个类继承自JFrame类，内部设置对应的参数，添加了对话框按钮，增加监听。用户可以在对应的部位添加内容，供软件运行。

客户端和服务端在进行连接时使用的是TCP协议，其是一种可靠的面向连接的传送服务。

初始化时服务端新初始化一个监听本地IP端口为5050的SocketSever，即TCP协议的服务端端口，等待服务端的连接，使用阻塞方法SocketSever.accept()方法，在未连接时一直等待。并在此时显示窗口，展示本机IP地址。当有客户端连接时，将连接的Socket加入线程池，调用MyRunnable类接口，等待客户端操作。

客户端初始化时先弹出客户端窗口，并初始化InputWindow类，弹出小窗口用于获取用户输入的IP地址。输入IP之后客户端会创建连接该IP的Socket，连接服务端。双端成功连接之后都会在窗口中打印成功连接的对话。

3.2 简单文件传输协议实现

在用户连接之后，客户端和服务端都进入静默状态，等待用户在客户端进行操作。客户端监听到按钮按下之后，需要服务端进行相对应的操作，服务端发送完成反馈。这些指令都蕴藏在双方发送的byte流中。

协议规定双方发送的单次byte流长度都为128。其中在操作指令中，首位即第一位是常驻操作数，其代表本次服务器需要进行的操作。服务端收到请求后会进行对应操作，在操作结束后返回一个长度为128的byte数组，作为本次操作的反馈。在进行操作过程中，可能会遇到一些情况需要用户判断，此时第二位也作为操作位，这些指令会在之后进行说明。

有些操作需要为服务端操作提供信息，其信息储存在操作数之后的位数，例如文件地址，文件名称等。这些信息占用的位数不能确定，无法详细规定多少位可以包含所有内容，但协议中所有有效信息全部包含在前127比特。

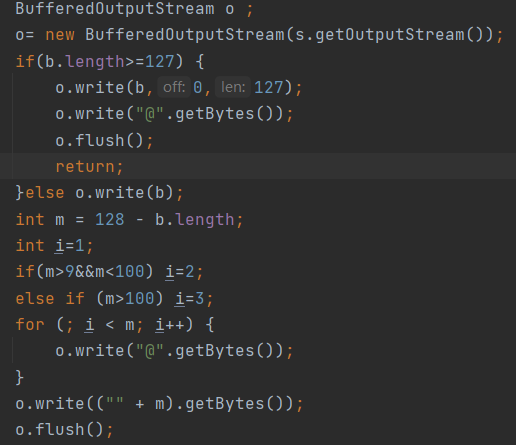
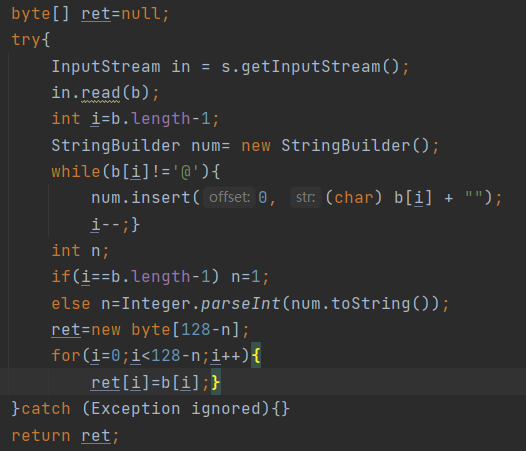
在一个byte串中加入操作数和信息位后，剩余部分会补齐，协议使用‘@’符号进行填充，并且为了避免信息混淆，最后几位不会填充符号而是将该byte串无效位的位数储存进去。例如有125位有效信息，则填充2位‘@’和一个数字3。除此之外，协议规定如果前127位都为有效位，则最后一位为‘@’，不需要填充数字1。

非操作指令中，舍弃了前面的操作数位，127比特全是有效信息位，后面一位填充‘@’，如果遇到不满127位的，按照前文所述的填充方法填充。使非操作指令和操作指令具有相似的格式，能够用同一种方法操作传送的byte流。

3.3 流式工具类实现-MyStreamMethod

该类定义了两种send()方法和两个receive()方法，相同名称的方法对应的参数和作用都不相同；还定义了获取本地文件夹内所有文件的方法、在对话框中添加文字的方法和弹出选择文件地址窗口的类。

send和receive方法分别对应发送和接收单个byte串和整个文件或者文件夹。对于单个byte串来讲，都是按照3.3介绍的自创协议进行处理。核心代码如下：

 图3.4.1 send方法实现代码 图3.4.2 receive方法实现代码

对于发送文件夹，send方法会判断文件是文件夹还是文件，若为文件夹，则发送储存文件夹内部的子文件的数量和文件夹名称的byte流。若为文件，先发送文件名称，之后的比特流发送文件内容，最后发送结束之后发送单个”@”字符作为文件发送完成的标志。

对于接收文件夹，先由接收到的第一个byte流的第一位判断是文件还是文件夹，文件夹只需要创建该文件夹，然后将当前路径进入到该文件夹内，根据传输的文件数量重复操作。如果为文件，创建文件，并且一直接收数据写入文件，直到接收到结束byte串“@”后结束。以下为发送和接收文件的方法实现：

public static void send(File file, Socket s){

int flag=0;

if(file.exists()) {

if(file.isFile()) flag = 1;

if (file.isDirectory()) flag = 2;

}

//判断文件是否存在

StringBuilder string = new StringBuilder();

string.append(1).append(flag);

switch (flag) {

case 1 -> {

byte[] b = new byte[127];

string.append(file.getName());

send(string.toString().getBytes(), s);

try {

InputStream in;

in = new FileInputStream(file);

int j=0,i=0;

while (j!=-1){

for(i=0;i<127&&(j=in.read())!=-1;i++){

b[i]=(byte) j;

}

if(i!=127){

byte [] c=b;

b=new byte[i];

System.arraycopy(c, 0, b, 0, i);

}

send(b, s);

}

send("@".getBytes(),s);

in.close();

} catch (Exception ignore) {}

}

case 2 -> {

File[] list = file.listFiles();

int num=0;

if(list!=null)num = list.length;

if (num <= 999 && num > 99) string.append(num);

else if (num < 100 && num > 9) string.append(0).append(num);

else if (num < 10) string.append(0).append(0).append(num);

string.append(file.getName());

send(string.toString().getBytes(), s);

if(list!=null&&num!=0) for (File f : list) {

send(f, s);

}

}

}

}

除此之外，还有几个比较常规的方法，一个fileWindow方法弹出一个窗口选择文件目录，一个getAllFile方法，使用递归思想，获得文件夹内所有文件的名称，以下为核心代码：[3]

public static void receive(byte[] b, Socket s, String nowPath){

try{

if(b[0]=='0') return;

switch (b[1]) {

case '1' -> {

String string = nowPath + "\\" + new String(b, 2, b.length - 2);

File file = new File(string);

if (file.exists()) {

int i=1;

String last=string.substring(string.lastIndexOf("."));

string = string.substring(0,string.lastIndexOf("."))+"-副本";

while((file=new File(string+i+last)).exists()){

i++;

}

file.createNewFile();

}

FileOutputStream o;

o=new FileOutputStream(file);

while((b = receive(new byte[128], s)).length==127) {

o.write(b);

o.flush();

}

if(b[b.length-2]!='@') o.write(b);

o.flush();

o.close();

receive(new byte[128],s);

}

case '2' -> {

String string = nowPath + "\\" + new String(b, 5, b.length - 5);

File f = new File(string);

if (!f.exists()) f.mkdirs();

int num = 0;

for (int i = 2; i < 5; i++) {

num = num \* 10 + b[i] - '0';

}

for (int i = 0; i < num; i++) {

b = receive(new byte[128], s);

receive(b, s, string);

}

}

}

}catch (Exception ignored){}

}

3.4 监听类实现-MyActionMethod

for (File file : fileList) {

if (file.isDirectory()) {

// 递归处理文件夹

allFileList.add(string + file.getName() + "\\");

getAllFile(file, allFileList, n + 1);

} else {

// 如果是文件则将其加入到文件名数组中

allFileList.add(string + file.getName());

}

}

此类中的方法，在用户在客户端上进行操作后调用。为了客户端与服务端进行通信，客户端在进行操作后，向服务端发送包含操作数的byte流和相关参数，后接收服务端反馈进行操作。如无说明，则方法所说的发送和接收为MyStreamMethod的发送byte流方法。此类主要实现了8个方法，按窗口的按钮顺序介绍方法实现。

第一个方法show（）遍历服务端文件，反馈为文件的总数量n，如果当前文件夹为空则为0，否则再接收n次byte流，为n个文件的文件名，循环打印即可。

第二个方法upload（）上传文件或文件夹，首先调用MyStreamMethod类中的fileWindow方法，用户选取上传的文件，发送第一个含有操作位的byte流后调用MyStreamMethod类中的send方法发送文件。

第三个方法download（）下载文件，用户需要输入要下载的文件，然后将操作数，文件名打包发送给服务端，调用MyStreamMethod类中的receive方法接收文件，然后在本地循环判断并将接收到的byte数组按顺序写入文件。

第四个方法last（）返回上一级，向服务端发送包含操作位的byte头，接受反馈判断打印成功或失败。

第五个方法clear（）清空对话，将窗口显示的对话清空。调用了JTextArea的setText方法。

第六个方法create（）创建文件，用户输入文件名称后在服务端的当前目录创建新的文件夹，后返回反馈，创建成功和失败，或者存在同名文件夹都会打印在客户端本地。

第七个方法del（）删除文件，用户输入需要删除的文件名称，将操作位打包发送给服务端，接受反馈返回操作结果。

第八个方法cd（）进入，用户输入子文件夹或文件内容，服务端根据文件类型进行操作，文件夹则进入该目录，TXT文件则为服务端发送一串文件内容，用户可预览该文件内容，其他类型的文件则不支持预览。

3.5 服务端线程类-MyRunnable实现

此类的实现对应于MyActionMethod类的方法，每个客户端方法实现对应一个该类方法对应实现，完成客户端与服务端的互通。

当一个与客户端连接的Socket加入服务端线程之后，等待一个byte流，接收之后根据switch按照第一位的操作位进行操作，则相对应的MyRunnable中实现了7个主要方法和一个被调用的方法。

第一个方法showDir（）遍历服务端文件，调用getAllFile方法获取文件夹名称的数组序列，反馈文件夹数量并循环发送文件名称。

第二个方法upload（）上传文件或文件夹，直接调用MyStreamMethod的receive文件夹方法。

第三个方法download（）下载文件，从byte串中获取用户需要下载的文件名称，检测文件是否存在，若存在调用MyStreamMethod中的send文件方法发送。

第四个方法last（）返回上一级，服务端将类中当前路径设置为当前路径的父路径，发送反馈中包括父路径的名称。

第五个方法create（）创建文件，获取byte流中设置的文件名称，以该名称创建文件夹，成功或文件夹已存在时发送反馈。

第六个方法del（）删除文件，获取byte流中设置的文件名称，检测该路径中存在该文件，若文件不存在直接发送失败的反馈。若存在且为文件直接删除发送反馈，若为文件夹，调用类中另一个方法delAllFile（file），根据返回值发送反馈。

第七个方法cd（）进入，获取byte流中设置的文件名称，检测该路径中存在该文件，若文件不存在直接发送失败的反馈。若文件存在且为文件夹，将当前目录设置为该文件夹的目录，若为文件且为TXT文件，则读取文件部分内容作为反馈发送给客户端，其余文件则直接发回反馈。

最后一个方法时delAllFile（file）方法，该方法运用了递归的思想，当遇到子文件为文件时，直接删除；为文件夹时再调用该方法进行删除，之后删除该清空的文件夹。

第四章 测试

4.1 程序运行环境

4.1.1程序运行的软件环境

本程序依托于JAVA编写，源码编写时版本为JAVA 18，为使程序稳定运行，请使用JAVA 18及以上环境运行本程序，低版本可能出现报错。

4.1.2程序运行的硬件环境

程序运行需要联网，支持IPv4环境

支持在Windows 10、11系统下运行，未在Linux及MACos环境下进行测试

4.2 客户端与服务端界面

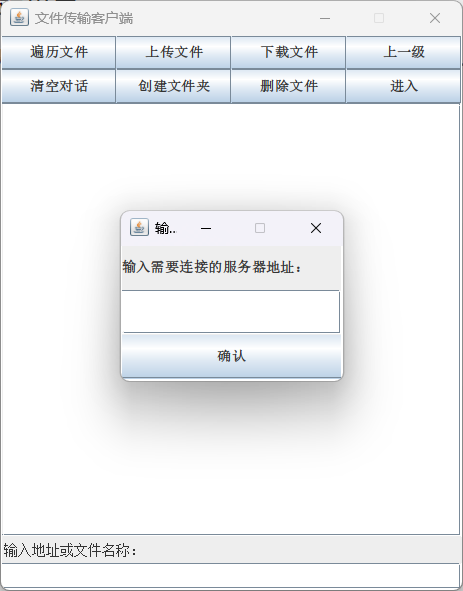
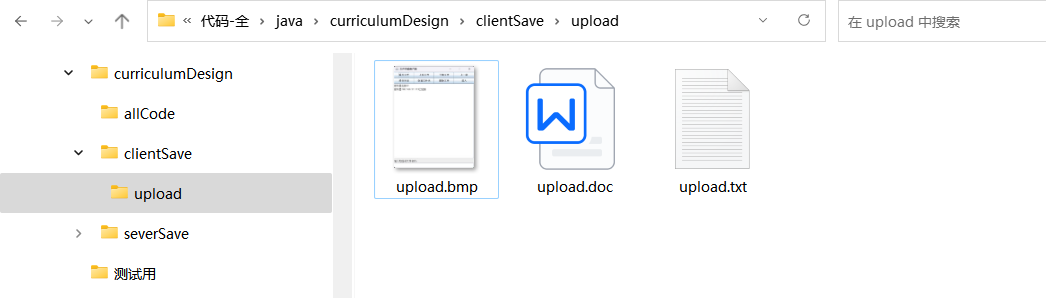
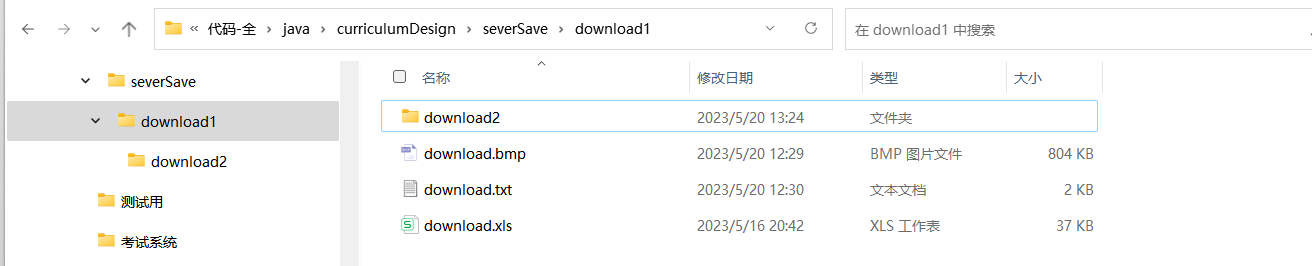
如下图，进入程序之后，客户端弹出输入IP地址的窗口，输入窗口及成功连接的窗口如下图：

图4.2.1客户端界面 图4.2.2服务端界面

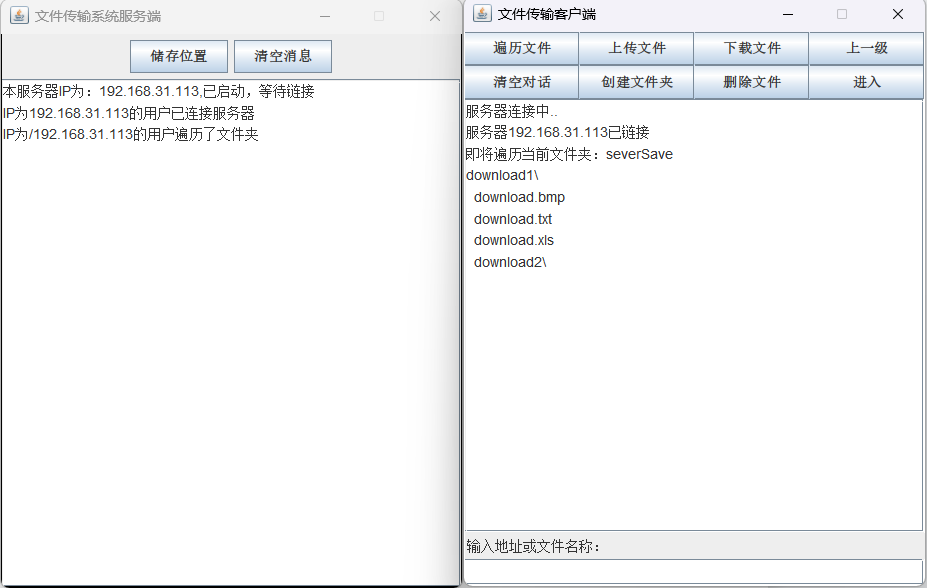
4.3遍历文件测试

首先展示初始状态下双端目录下的文件：

图4.3.1初始状态客户端中文件

图4.3.2初始状态服务端中文件

客户端点击遍历文件按钮，双端界面变为：

图4.3.3遍历双端后界面

4.4 上传文件测试

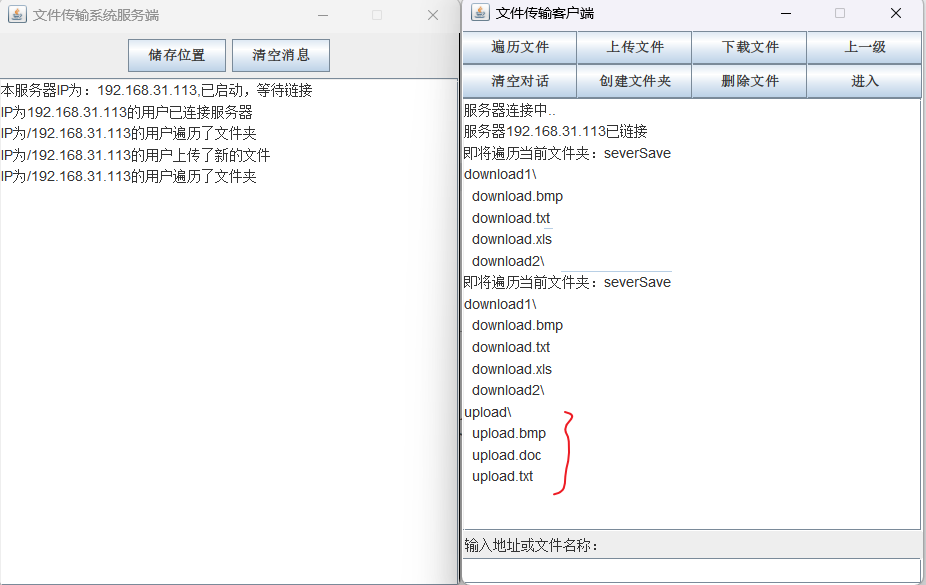
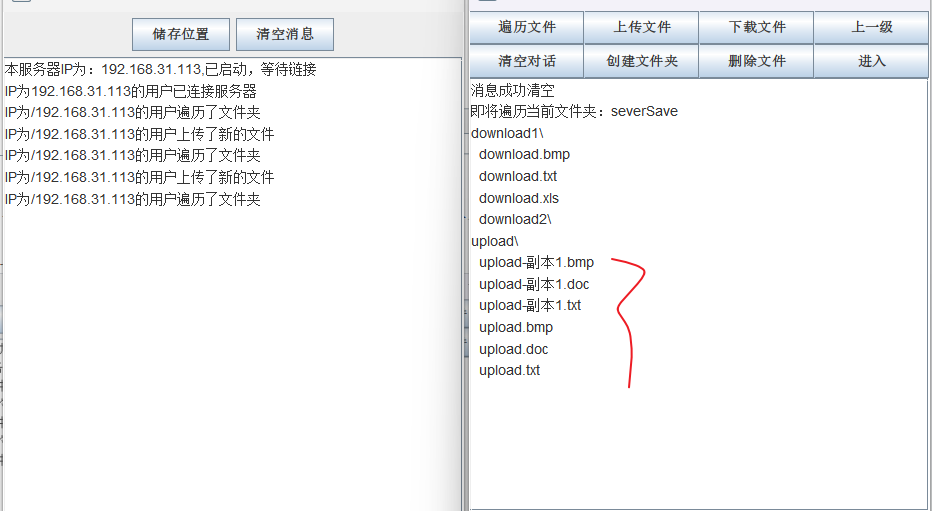
初始文件如4.3所示，点击客户端上传文件并选择“upload”文件夹后，并点击遍历文件按钮，根据页面显示，红色部分为新上传的文件，在服务端的文件管理中，新上传的文件都与用户上传的文件相同，可以正常打开。

图4.4.1上传文件后双端页面

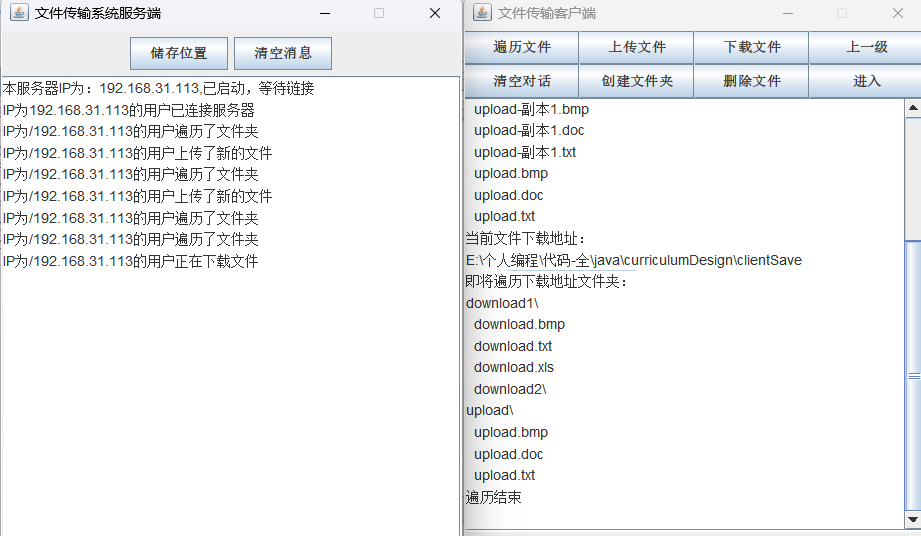
当用户再次上传该文件或文件夹后，会在该文件夹内新建另一个文件并加入副本的后缀表明是新上传的文件，即进行存在性检测，界面如下：

图4.4.2再次上传文件后双端页面

4.5 下载文件测试

输入需要下载的文件或文件夹名称，点击下载按钮，弹出对话框选择文件下载地址，选择地址后，点击下载文件按钮下载文件到本地，如下：

图4.5.1下载测试前

图4.5.2下载测试后

4.6 下一级及进入按钮测试

测试进入根目录中的download1文件夹，遍历文件夹后返回根本目录：

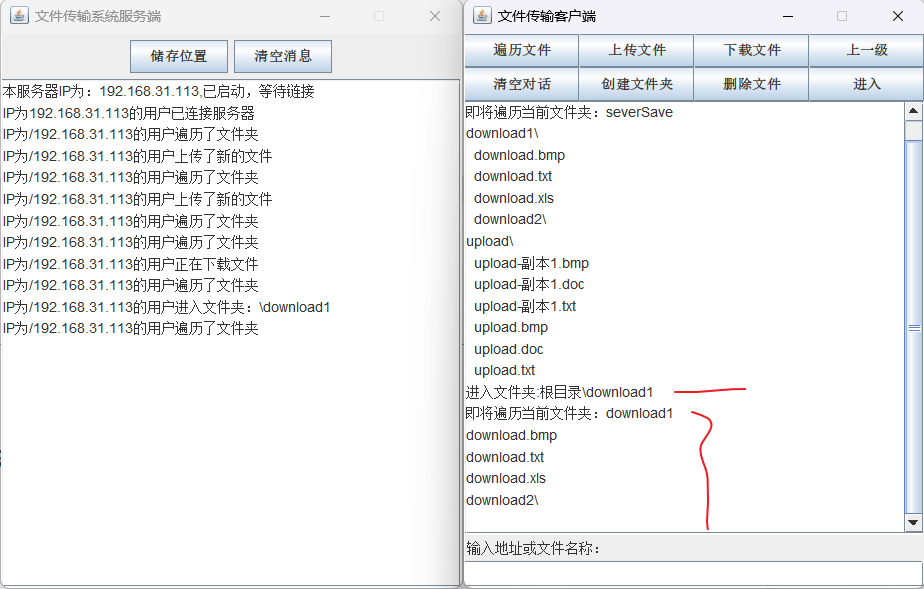
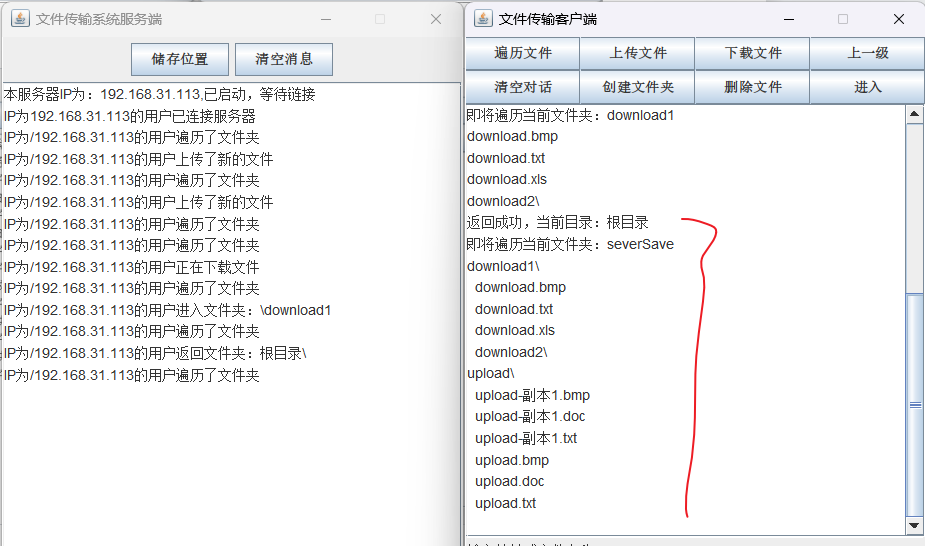
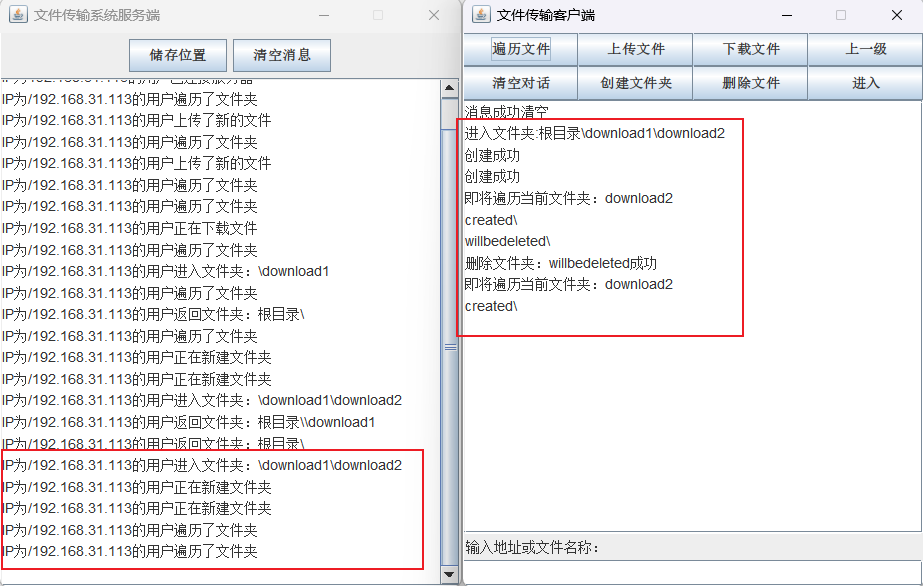


图4.6.1进入按钮测试

图4.6.1上一级按钮测试

4.7 创建及删除文件测试

首先进入服务端的download1\download2文件夹中，创建created和willbedeleted文件夹，后删除willbedeleted文件夹,在服务端本地显示操作正常，界面上创建及删除都成功进行：

图4.7.1创建及删除文件测试

4.8 下一级按钮的预览功能测试

选择预览服务端根目录下的download1\download.txt文件，进入路径并输入该文件名称后，可以得到一段该文件的开头一段内容，功能正常：

图4.8预览功能

第五章 总结

5.1功能完整性总结

在本次课程设计中，复习巩固了JAVA的编程知识，计算机网络的相关概念，练习使用了VScode和IDEA编程工具，熟悉了JAVA的GUI工具和网络编程的类库，完成了一个简单的文件传输服务器和客户端。最终完成了设计的基本功能：

1、支持创建单层目录

2、支持一次上传单份文件

3、支持一次上传多个文件或文件夹

4、支持删除某个文件

5、支持删除某个目录

6、支持浏览文件和文件

5.2程序不足

本程序实现基本功能，完成了一个简单的图形界面窗口，使得用户可以在窗口上完成想要完成的操作。但是还存在一些问题需要解决：

1、窗口不够美观，按钮排列较紧凑，功能排列不美观

2、有些功能需要用户输入参数，不够便捷

3、程序的服务端没有考虑多用户的情况

4、协议设计不是很完美，传输数据有冗余情况

5、多用户设计不够明显，若有多名用户同时登录可能会出现文件冲突

5.3功能展望

1、将图形界面设计的更加完美，使得服务端的文件可以直观完整的展示在客户端，并且实现除了文件创建功能，客户端完全用鼠标操作，实现对文件的各种操作。

2、考虑多用户情况，对每个用户和IP单独进行文件管理，设计简单的客户端登陆界面，在服务端连接数据库进行用户识别和登录。

3、设计文件移动功能，将服务端内的文件移动到另一个文件目录内。

4、完善协议，使得网络信道的利用率更高，耗费的流量数据更少，传递速度更快。

参考文献

1. CSDN.服务端和客户端的区别及介绍.

https://blog.csdn.net/manerzi/article/details/127599455

1. 知乎.计算机网络概述. https://zhuanlan.zhihu.com/p/52244814.
2. CSDN.java读取路径文件夹的所有文件名

https://blog.csdn.net/qq\_45749688/article/details/120233270