**FTP服务器与客户端项目文档**

文档信息管理表

|  |  |
| --- | --- |
| 作者 | 周锐淇 |
| 学号 | 201858080232 |
| 班级 | 网络工程1802 |
| 完成时间 | 2020.06.30 |
| 项目名称 | FTP服务器与客户端服务系统 |
| 开发工具 | Intellij IDEA 2019 |
| 开发语言 | JAVA |
| JDK版本 | 12.0.2 |
| 版本 | 1.0 |

目录

[1. 需求分析报告 2](#_Toc44445790)

[2. 系统详细设计 2](#_Toc44445791)

[3. 系统测试报告 7](#_Toc44445792)

[4. 系统使用说明 14](#_Toc44445793)

[5. 源码展示 14](#_Toc44445794)

# 需求分析报告

将文件从一台计算机复制到另一台计算机是网络环境中的一项基本应用，而因特网上使用的最广泛的文件传输协议就是FTP。FTP协议是计算机网络中的应用层协议,基于c/s架构，提供了可靠的文件传输功能,并且具有平台特性,两个不同类型的操作系统只要都支持FTP协议就可以进行文件传输工作。

FTP系统与其他C/S系统不一样的地方在于它在客户端和服务器之间同时建立了两条连接来实现文件的传输，分别是控制连接和数据连接。控制连接用于客户端和服务器之间的命令和响应的传递；数据连接则主要用于传送数据信息。当用户通过FTP客户端向服务器发起一个会话的时候，客户端会通过熟知端口21与FTP服务器建立一个TCP连接，即控制连接。客户端使用此连接向FTP服务器发送所有FTP命令并读取应答。而对于大批量的数据，如数据文件或详细目录列表等，FTP系统会建立一个独立的数据连接去传送相关数据。

本项目就是基于java语言Socket编程设计一个FTP服务器和客户端系统。整个文件传输系统在Windows平台和Intellij IDEA 2019开发环境下进行开发，JDK版本为12.0.2。用户通过FTP客户端向服务器发送指令，服务器根据指令内容执行相关操作，并将执行结果返回给客户端。

FTP客户端操作界面应该有充分的提示，指引用户进行操作。要实现get、put、open等基本命令，完成客户端与服务端连接、文件传输、消息发送等功能。

FTP服务器需要执行来自客户端的指令。并返回结果。

# 系统详细设计

系统分为两个部分，分别为客户端以及服务端。

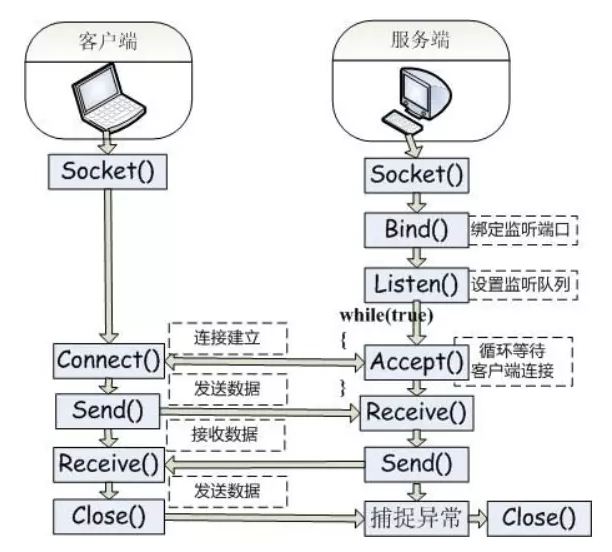


图1.客户端与服务器端连接示意图

socket是基于应用服务与TCP/IP通信之间的一个抽象，它将TCP/IP协议里面复杂的通信逻辑进行封装。

在开发本系统时使用的包如下

客户端：

import java.io.\*;

import java.net.Socket;

import java.util.StringTokenizer;

import java.util.Scanner;

服务端：

import java.io.\*;

import java.net.InetAddress;

import java.net.ServerSocket;

import java.net.Socket;

**FTP客户端**

客户端首先要创建一个与服务器连接的套接字，创建套接字使用java.net.Socket类的构造方法public Socket(String host, int port)，该构造方法有两个参数，分别为host：字符串类型的服务器ip地址和port：端口。使用该构造方法创建Socket对象时，会根据参数去连接在指定地址和端口上运行的服务器程序。

java流是一组有顺序的，有起点和终点的字节集合，是对数据传输的总称或抽象。它的特性是进行数据传输；本系统使用IO流对客户端与服务端的数据交互进行处理。

客户端在正确输入用户名、用户密码与服务端取得连接后通过getInputStream（）方法可以得到一个输入流，客户端的Socket对象上的getInputStream（）方法得到输入流其实就是从服务器端发回的数据。通过getOutputStream（）方法得到的是一个输出流，客户端的Socket对象上的getOutputStream（）方法得到的输出流其实就是发送给服务器端的数据。但是InputStream和OutputStream是java中可以按照最小字节单位读取的流，即每次读写一个字节，字节流是直接连接到输入源的流，可以读取任何数据。但是为了显示需要转将其换为字符流（通过InputStreamReader），为了提升读取速度，又转换为BufferReader与BufferWritter。这样就可以实现客户端与服务端的数据传输。

客户端的包结构如图所示：

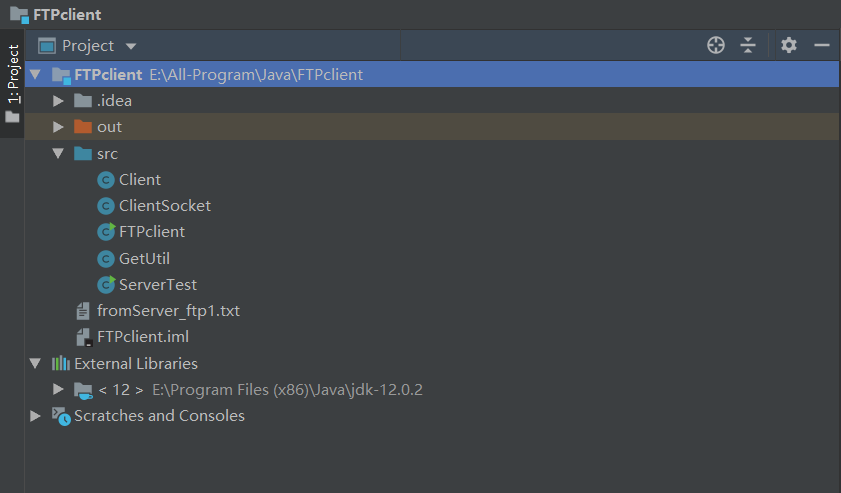


图2.客户端包结构

ClientSocket：对Socket进行简单的封装

Client和GetUtil：可实现连接服务器、查看目录、文件传输（put和get）、发送指令、接收响应、进行对话等功能；

ServerTest：对客户端功能进行测试；

FTPclient：进行流程控制以及用户交互。

**FTP服务器**

服务器用来响应客户端的请求，并返回相关执行结果。

Socket类代表一个客户端套接字，即任何时候连接到一个远程服务器应用时构建所需的socket。要实现一个服务器应用，则需要不同的做法。服务器需随时待命，因为不知道客户端什么时候会发来请求，此时，我们需要使用ServerSocket，对应的是java.net.ServerSocket类。

服务端先初始化ServerSocket，使用bind（）方法调用ServerSocket(int port, int backlog, InetAddress bindAddr)构造函数，参数port指定服务器要绑定的端口（服务器要监听的端口），参数backlog指定客户连接请求队列的长度，参数bindAddr指定服务器要绑定的IP地址。

在完成对指定的端口进行绑定后，服务器程序就开始对端口进行监听，通过调用accept（）方法阻塞，此时，如果客户端有一个Socket连接到服务端，那么服务端通过监听和accept（）方法就可以接收请求创建一个Socket示例来与客户端进行通信。

服务端同样使用IO流与客户端进行数据交互。

getInputStream（）方法得到的是一个输入流，服务端的Socket对象上的getInputStream（）方法得到的输入流其实就是从客户端发送给服务器端的数据流。

getOutputStream（）方法得到的是一个输出流，服务端的Socket对象上的getOutputStream（）方法得到的输出流其实就是发送给客户端的数据。

文件传输则使用DataInputStream**、**DataOutputStream数据流，将基础数据类型写入到文件中，或者读取出来。

通过While循环控制不断读取客户端消息。

服务器端包结构如图所示：

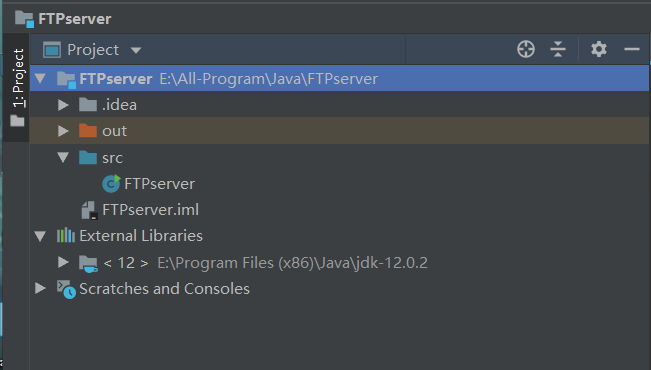


图3 . 服务器端包结构

FTPserver：进行端口绑定，接收客户端连接，实现文件传输、对话等功能。

# 系统测试报告

本系统能够稳定运行，完成相应的功能，以下为详细运行测试流程：

**测试准备：**

Serv-U服务器相关信息：



图4 . Serv-U相关信息

用户相关信息：

****

图5 . 用户相关信息

在E盘添加[ftp1.txt](ftp://ftp1.txt)文件用于测试文件传输

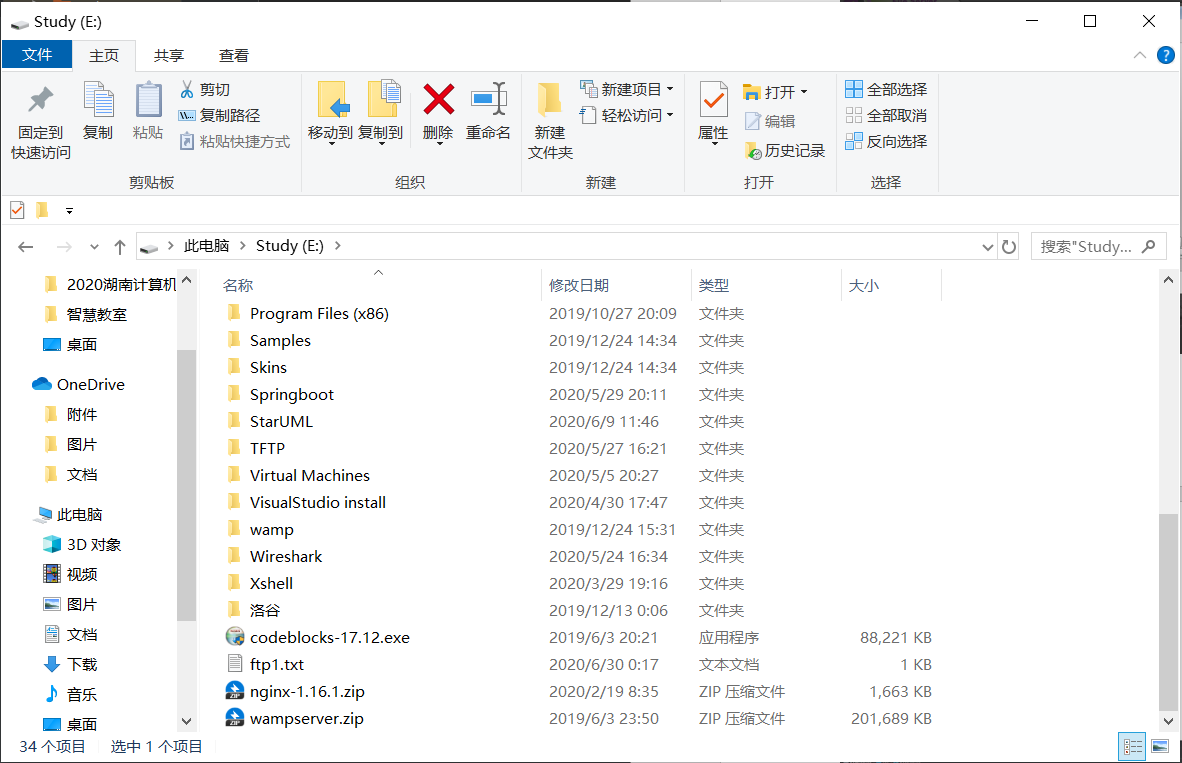


图6 . 创建测试文件

确认用户test2目录下没有该文件



图7 . 用户目录

确认D盘目录下没有该文件（从服务器下载文件默认存在D盘）



图 8 . D盘目录

**启动服务：**

首先启动服务端程序

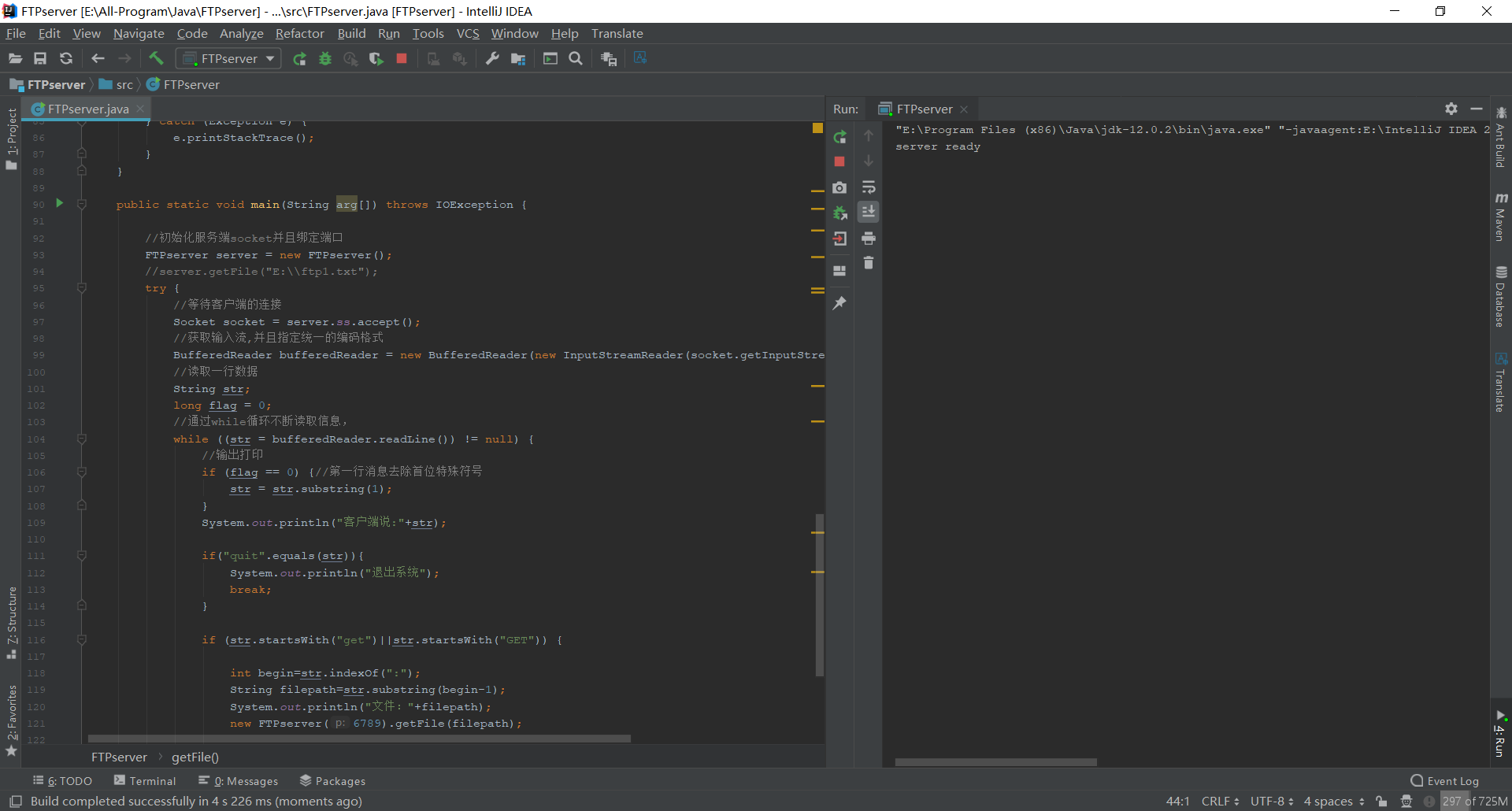


图9 . 启动服务端程序

接着启动客户端，依照提示输入服务器ip、用户名、密码。匹配成功，连接服务器，进行相关操作。

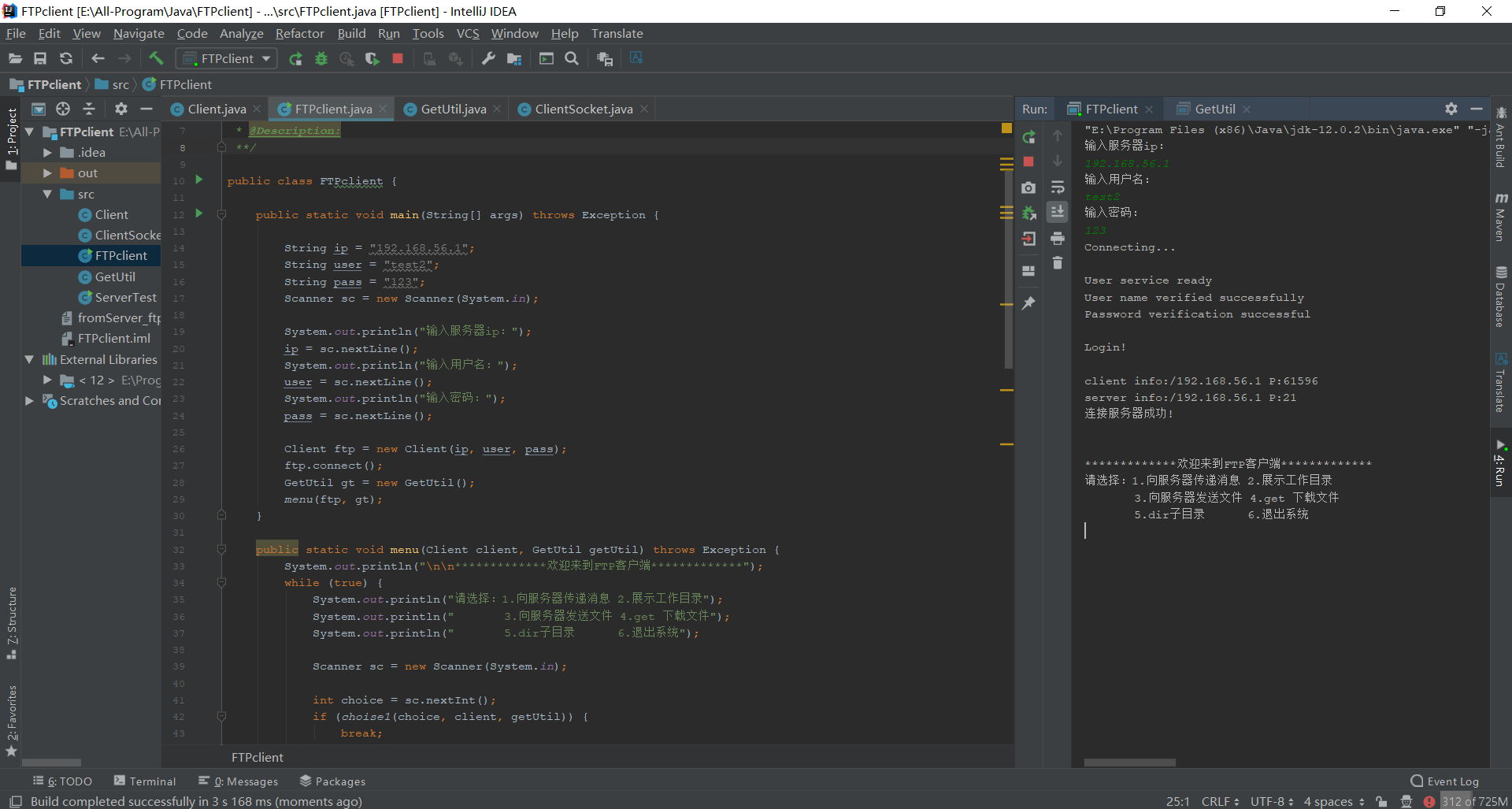


图10 . 启动客户端，连接成功

选择1与服务端进行对话,直到输入quit退出对话

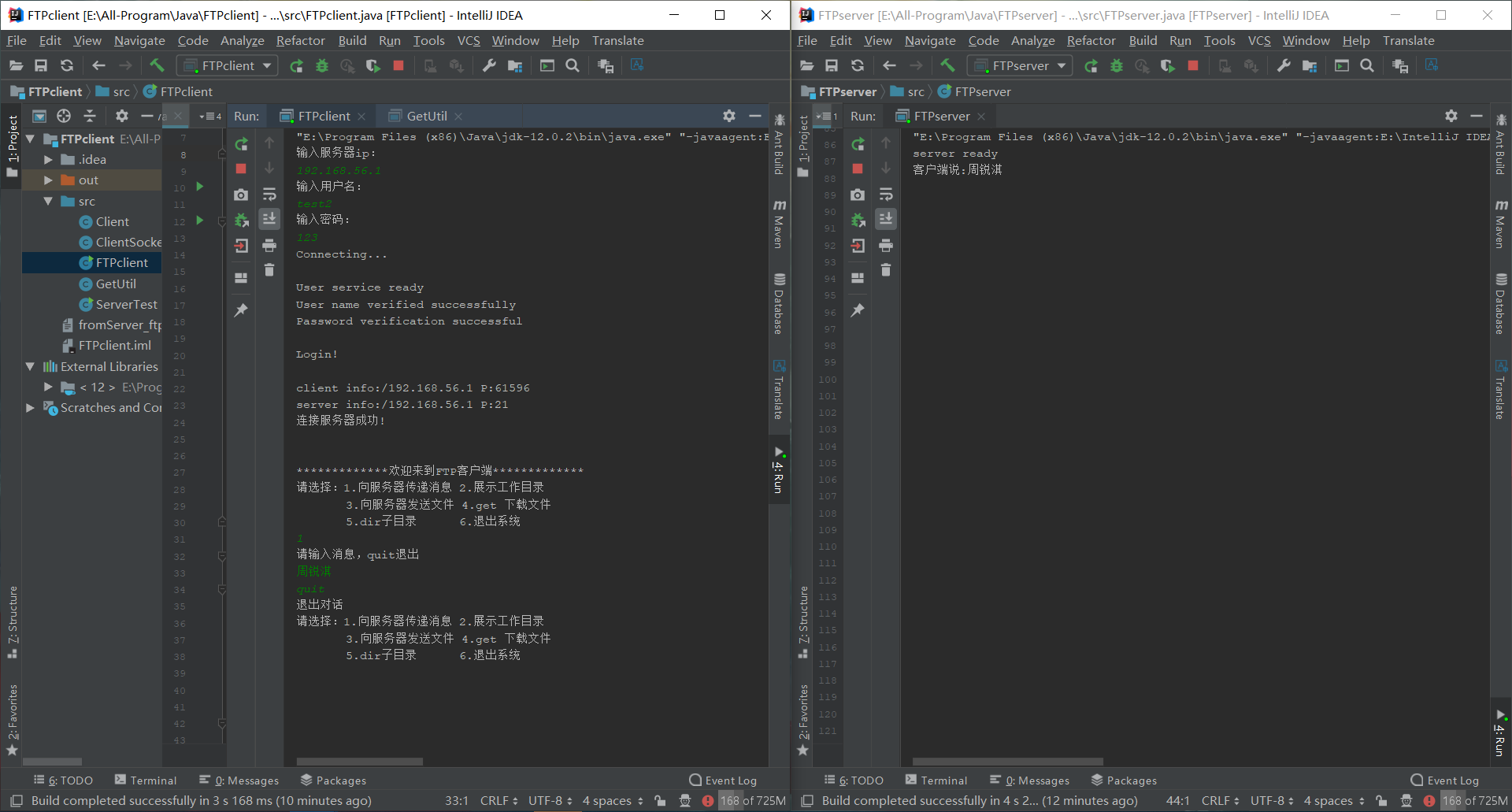


图11 . 与服务端对话

显示当前工作目录（根目录）

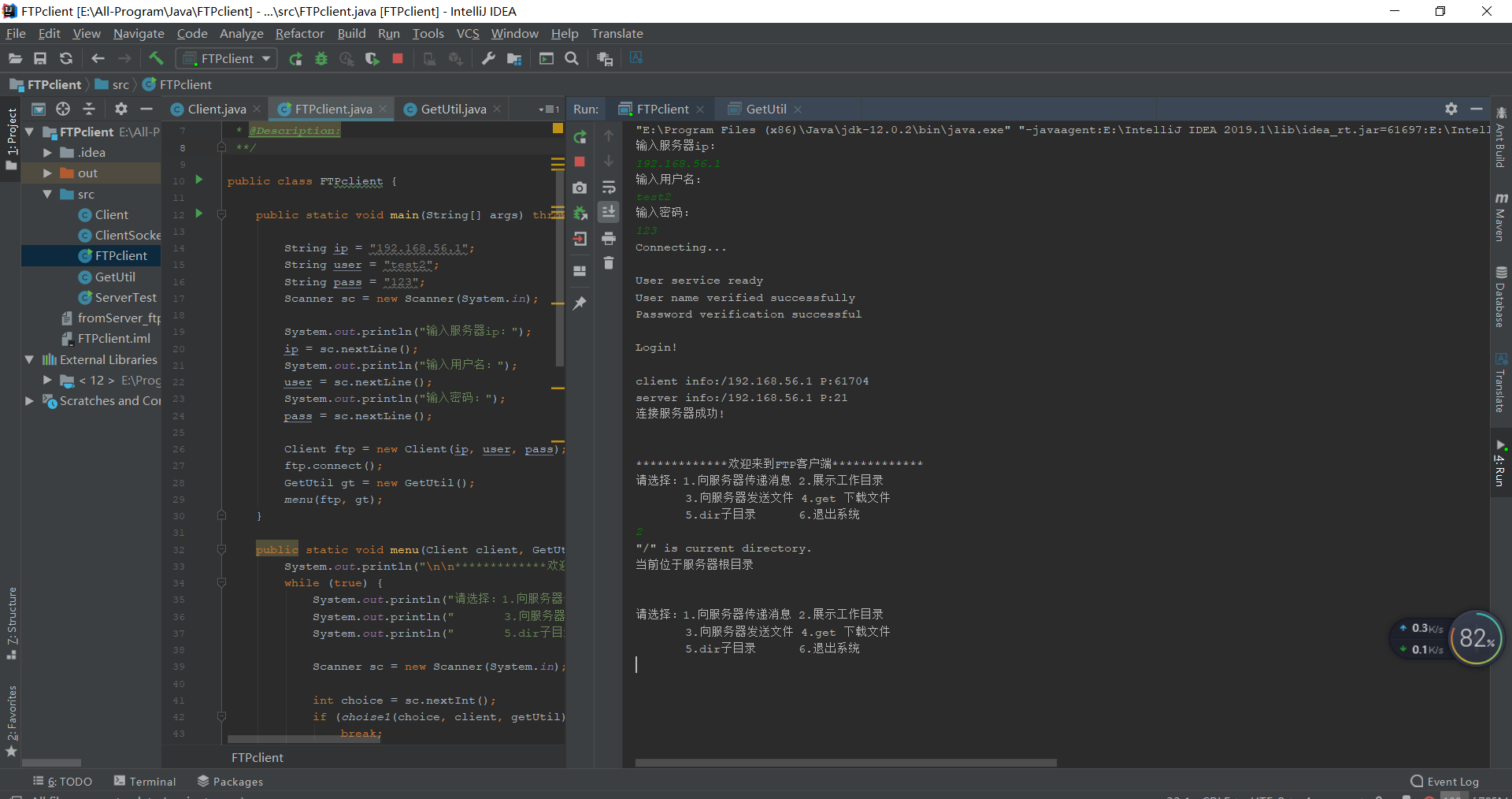


图 12 . 工作目录

向服务器发送文件

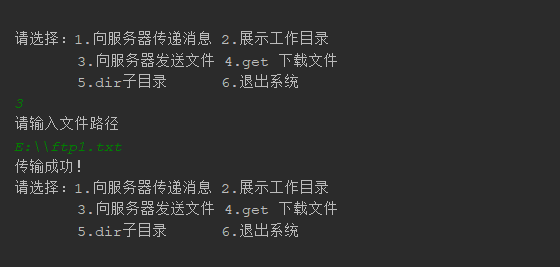


图 13 .发送文件

查看用户目录，传输成功。

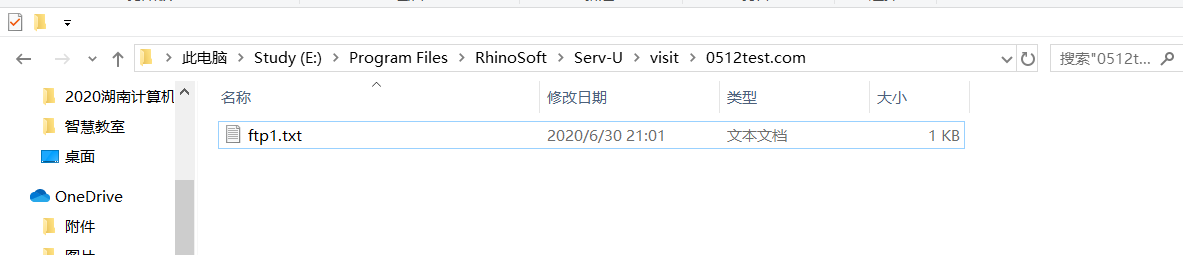


图14 . 用户目录文件上传成功

从服务器下载文件到D盘

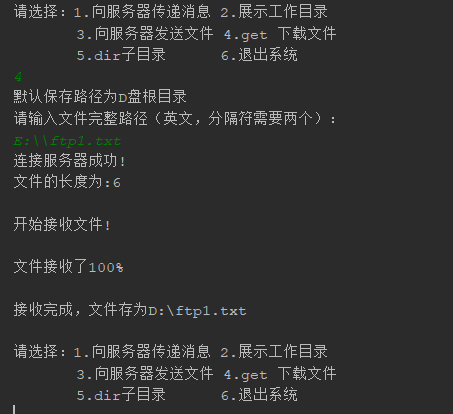


图 15 . 从服务器下载文件

服务器响应

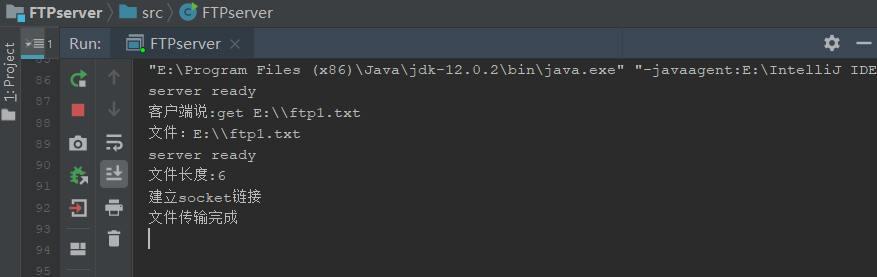


图 16 . 从服务器下载文件

查看D盘，刷新目录，显示文件下载成功



图17 . 查看D盘，文件下载成功

按照提示退出系统

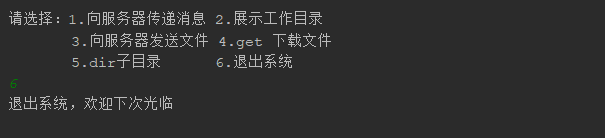


图 19 . 退出系统

# 系统使用说明

首先准备好相应使用条件接着先运行服务器程序，运行成功后弹出窗口并进入监听状态。打开FTP客户端程序运行，按照提示输入连接的主机ip地址，连接成功则进入下一步，依据提示进行相关操作。

相关输入严格按照提示进行。选择使用数字。文件路径使用英文，分隔符需要双份。

# 源码展示

**客户端**：

Client：

1. **import** java.io.\*;
2. **import** java.net.Socket;
3. **import** java.util.StringTokenizer;

6. **public** **class** Client {
8. **private** Socket socket = **null**;
9. **private** BufferedReader reader = **null**;
10. **private** BufferedWriter writer = **null**;
11. **private** **static** **boolean** debug = **false**;
12. **private** String host;//连接主机
13. **private** String user;//用户名
14. **private** String pass;//用户密码
15. **private** **int** port;//端口
17. {
18. //默认赋值
19. host = "192.168.56.1";
20. user = "test2";
21. pass = "123";
22. port = 21;
23. }
25. **public** Client() {
26. }
28. //全参构造器
29. **public** Client(String host, String user, String pass) {
30. **this**.host = host;
31. **this**.user = user;
32. **this**.pass = pass;
34. }
36. //取得连接
37. **public** **synchronized** **boolean** connect() **throws** Exception {
38. **if** (socket != **null**) {
39. **throw** **new** Exception("already connect!");
40. } **else** {
41. System.out.println("Connecting...");
42. }
43. socket = **new** Socket(host, port);//创建套接字
44. reader = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));
45. writer = **new** BufferedWriter(**new** OutputStreamWriter(socket.getOutputStream()));
46. String response = readLine();
47. **if** (!response.startsWith("220")) {
48. //220： 新用户服务准备好了
49. **throw** **new** Exception("unknown response after connect!");
50. } **else** {
51. System.out.println("\nUser service ready");
52. }
53. sendLine("USER " + user);
54. response = readLine();
55. **if** (!response.startsWith("331")) {
56. //331： 用户名正确，需要口令
57. **throw** **new** Exception("unknown response after send user");
58. } **else** {
59. System.out.println("User name verified successfully");
60. }
61. sendLine("PASS " + pass);
62. response = readLine();
63. **if** (!response.startsWith("230")) {
64. //230： 用户登录
65. **throw** **new** Exception("unknown response after send pass");
66. } **else** {
67. System.out.println("Password verification successful");
68. }
69. System.out.println("\nLogin!\n");
71. //打印FTP客户端及服务器信息
72. System.out.println("client info:" + socket.getLocalAddress() + " P:" + socket.getLocalPort());//打印本地服务器地址和本地端口号
73. System.out.println("server info:" + socket.getInetAddress() + " P:" + socket.getPort());
74. **return** **true**;
75. }

78. /\*\*
79. \* 向服务器发送指令
80. \*/
81. **private** **void** sendLine(String line) **throws** Exception {
82. **if** (socket == **null**) {
83. **throw** **new** Exception("not connect!");
84. }
85. writer.write(line + "\r\n");
86. writer.flush();
87. **if** (debug) {
88. System.out.println(">" + line);
89. }
90. }
92. /\*\*
93. \* 接收服务器返回信息
94. \*/
95. **private** String readLine() **throws** IOException {
96. String line = reader.readLine();
97. **if** (debug) {
98. System.out.println("<" + line);
99. }
100. **return** line;
101. }
103. /\*\*
104. \* 返回ftp服务器工作目录
105. \*/
106. **public** **synchronized** String pwd() **throws** Exception {
107. sendLine("PWD");
108. String dir = **null**;
109. String response = readLine();
110. **if** (response.startsWith("257")) {
111. //257： 创建"PATHNAME"
112. **int** firstQuote = response.indexOf("/");
113. dir = response.substring(firstQuote - 1);
114. }
115. **return** dir;
116. }
118. /\*\*
119. \* 查看当前目录下的子目录
120. \*/
121. **public** **synchronized** String dir() **throws** Exception {
122. sendLine("dir");
123. //        StringBuffer fl = new StringBuffer();
124. //        String response;
125. //        do {
126. //            response = readLine();
127. //            fl.append(response);
128. //        }while((response=readLine())!=null);
129. String response = readLine();
130. **return** response;
131. }

134. /\*\*
135. \* 向服务器发送文件
136. \* 互斥访问
137. \*/
138. **public** **synchronized** **boolean** stor(File file) **throws** Exception {
139. **if** (file.isDirectory()) {
140. **throw** **new** Exception("cannot upload a directory!");
141. }
142. String fileName = file.getName();
143. **return** upload(**new** FileInputStream(file), fileName);
144. }
146. /\*\*
147. \* 上传文件
148. \*/
149. **public** **synchronized** **boolean** upload(InputStream inputStream, String fileName) **throws** Exception {
150. BufferedInputStream input = **new** BufferedInputStream(inputStream);
151. sendLine("PASV");
152. String response = readLine();
153. **if** (!response.startsWith("227")) {
154. //227： 进入被动模式
155. **throw** **new** Exception("not request passive mode!");
156. }
157. String ip = **null**;
158. **int** port = -1;
159. **int** opening = response.indexOf('(');
160. **int** closing = response.indexOf(')', opening + 1);
161. **if** (closing > 0) {
162. String dataLink = response.substring(opening + 1, closing);
163. StringTokenizer tokenzier = **new** StringTokenizer(dataLink, ",");
164. **try** {
165. ip = tokenzier.nextToken() + "." + tokenzier.nextToken() + "."
166. + tokenzier.nextToken() + "." + tokenzier.nextToken();
167. port = Integer.parseInt(tokenzier.nextToken()) \* 256 + Integer.parseInt(tokenzier.nextToken());
168. ;
169. } **catch** (Exception e) {
170. // TODO Auto-generated catch block
171. **throw** **new** Exception("bad data link after upload!");
172. }
173. }
174. sendLine("STOR " + fileName);
175. Socket dataSocket = **new** Socket(ip, port);
176. response = readLine();
177. **if** (!response.startsWith("150")) {
178. //150： 打开数据连接
179. **throw** **new** Exception("not allowed to send the file!");
180. }
181. BufferedOutputStream output = **new** BufferedOutputStream(dataSocket.getOutputStream());
182. **byte**[] buffer = **new** **byte**[4096];
183. **int** bytesRead = 0;
184. **while** ((bytesRead = input.read(buffer)) != -1) {
185. output.write(buffer, 0, bytesRead);
186. }
187. output.flush();
188. output.close();
189. input.close();
190. response = readLine();
191. **return** response.startsWith("226");
192. //226： 关闭数据连接，请求的文件操作成功
193. }
195. }

ClientSocket：

1. **import** java.io.BufferedInputStream;
2. **import** java.io.DataInputStream;
3. **import** java.io.DataOutputStream;
4. **import** java.net.Socket;
6. **public** **class** ClientSocket {
7. **private** String ip;
9. **private** **int** port;
11. **private** Socket socket = **null**;
13. DataOutputStream out = **null**;
15. DataInputStream getMessageStream = **null**;
17. **public** ClientSocket(String ip, **int** port) {
18. **this**.ip = ip;
19. **this**.port = port;
20. }
22. **public** Socket getSocket() {
23. **return** **this**.socket;
24. }
25. /\*\* \*/
26. /\*\*
27. \* 创建socket连接
28. \*
29. \* @throws Exception
30. \*             exception
31. \*/
32. **public** **void** CreateConnection() **throws** Exception {
33. **try** {
34. socket = **new** Socket(ip, port);
35. } **catch** (Exception e) {
36. e.printStackTrace();
37. **if** (socket != **null**)
38. socket.close();
39. **throw** e;
40. } **finally** {
41. }
42. }
44. **public** **void** sendMessage(String sendMessage) **throws** Exception {
45. **try** {
46. out = **new** DataOutputStream(socket.getOutputStream());
47. **if** (sendMessage.equals("Windows")) {
48. out.writeByte(0x1);
49. out.flush();
50. **return**;
51. }
52. **if** (sendMessage.equals("Unix")) {
53. out.writeByte(0x2);
54. out.flush();
55. **return**;
56. }
57. **if** (sendMessage.equals("Linux")) {
58. out.writeByte(0x3);
59. out.flush();
60. } **else** {
61. out.writeUTF(sendMessage);
62. out.flush();
63. }
64. } **catch** (Exception e) {
65. e.printStackTrace();
66. **if** (out != **null**)
67. out.close();
68. **throw** e;
69. } **finally** {
70. }
71. }
73. **public** DataInputStream getMessageStream() **throws** Exception {
74. **try** {
75. getMessageStream = **new** DataInputStream(**new** BufferedInputStream(socket.getInputStream()));
76. **return** getMessageStream;
77. } **catch** (Exception e) {
78. e.printStackTrace();
79. **if** (getMessageStream != **null**)
80. getMessageStream.close();
81. **throw** e;
82. } **finally** {
83. }
84. }
86. **public** **void** shutDownConnection() {
87. **try** {
88. **if** (out != **null**)
89. out.close();
90. **if** (getMessageStream != **null**)
91. getMessageStream.close();
92. **if** (socket != **null**)
93. socket.close();
94. } **catch** (Exception e) {
96. }
97. }
98. }

GetUtil：

1. **import** java.io.\*;
3. **public** **class** GetUtil {
4. **private** ClientSocket cs = **null**;
5. **private** String ip = "localhost";// 设置成服务器IP
6. **private** **int** port = 8821;
7. **private** String sendMessage = "Windows";

10. **public** GetUtil() {
11. **try** {
12. **if** (createConnection()) {
13. sendMessage();
14. }
16. } **catch** (Exception ex) {
17. ex.printStackTrace();
18. }
19. }
21. **public** GetUtil(**int** port) {
22. **this**.port=port;
23. **try** {
24. **if** (createConnection()) {
25. sendMessage();
26. }
28. } **catch** (Exception ex) {
29. ex.printStackTrace();
30. }
31. }

34. **private** **boolean** createConnection() {
35. cs = **new** ClientSocket(ip, port);
36. **try** {
37. cs.CreateConnection();
38. System.out.print("连接服务器成功!" + "\n");
39. **return** **true**;
40. } **catch** (Exception e) {
41. System.out.print("连接服务器失败!" + "\n");
42. **return** **false**;
43. }
45. }
47. **private** **void** sendMessage() {
48. **if** (cs == **null**)
49. **return**;
50. **try** {
51. cs.sendMessage(sendMessage);
52. } **catch** (Exception e) {
53. System.out.print("发送消息失败!" + "\n");
54. }
55. }
57. **public** **void** getMessage() {
58. **if** (cs == **null**) {
59. System.out.println("cs==null");
60. **return**;
61. }
62. DataInputStream inputStream = **null**;
63. **try** {
64. inputStream = cs.getMessageStream();
65. } **catch** (Exception e) {
66. System.out.print("接收消息缓存错误\n");
67. **return**;
68. }
70. **try** {
71. // 本地保存路径，文件名会自动从服务器端继承而来。
72. String savePath = "D:\\";//存在D盘根目录
73. **int** bufferSize = 8192;
74. **byte**[] buf = **new** **byte**[bufferSize];
75. **int** passedlen = 0;
76. **long** len = 0;

79. savePath += inputStream.readUTF();
80. DataOutputStream fileOut = **new** DataOutputStream(**new** BufferedOutputStream(**new** FileOutputStream(savePath)));
81. len = inputStream.readLong();
83. System.out.println("文件的长度为:" + len + "\n");
84. System.out.println("开始接收文件!" + "\n");
86. **while** (**true**) {
87. **int** read = 0;
88. **if** (inputStream != **null**) {
89. read = inputStream.read(buf);
90. }
91. passedlen += read;
92. **if** (read == -1) {
93. **break**;
94. }
95. // 下面进度条本为图形界面的prograssBar做的，这里如果是打文件，可能会重复打印出一些相同的百分比
96. System.out.println("文件接收了" + (passedlen \* 100 / len) + "%\n");
97. fileOut.write(buf, 0, read);
98. }
99. System.out.println("接收完成，文件存为" + savePath + "\n");
101. fileOut.flush();
102. } **catch** (Exception e) {
103. //            System.out.println("\*\*接收消息错误" + "\n");
104. //            return;
105. }
107. }

110. **public** **void** send(String line) **throws** Exception {
111. BufferedWriter writer = **null**;
112. **if** (cs.getSocket() == **null**) {
113. **throw** **new** Exception("not connect!");
114. }
115. writer = **new** BufferedWriter(**new** OutputStreamWriter(cs.getSocket().getOutputStream()));
116. writer.write(line + "\r\n");
117. writer.flush();
118. //        cs.getSocket().shutdownOutput();
119. }

122. **public** **void** communication() {
123. **try** {
124. //初始化一个socket
125. //            Socket socket = new Socket("127.0.0.1", 9999);
126. //通过socket获取字符流
127. BufferedWriter bufferedWriter = **new** BufferedWriter(**new** OutputStreamWriter(cs.getSocket().getOutputStream()));
128. //通过标准输入流获取字符流
129. BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.in, "UTF-8"));
130. **while** (**true**) {
131. String str = bufferedReader.readLine();
133. **if** ("quit".equals(str)) {
134. System.out.println("退出对话");
135. **return**;
136. }
137. bufferedWriter.write(str);
138. bufferedWriter.write("\n");
139. bufferedWriter.flush();

142. }
143. } **catch** (IOException e) {
144. e.printStackTrace();
145. }
146. }
148. //    public static void main(String arg[]) throws Exception {
149. ////        new GetUtil().getMessage();
150. //    GetUtil gt=new GetUtil();
151. //    gt.send("E:\\ftp1.txt");
152. //
153. //    GetUtil gt2=new GetUtil(6789);
154. //    gt2.getMessage();
155. //
156. //    }
157. }

FTPclient：

1. **import** java.io.\*;
2. **import** java.util.Scanner;
4. /\*\*
5. \* @Author: Z.Richard
6. \* @CreateTime: 2020/6/30 8:02
7. \* @Description:
8. \*\*/
10. **public** **class** FTPclient {
12. **public** **static** **void** main(String[] args) **throws** Exception {
14. String ip = "192.168.56.1";
15. String user = "test2";
16. String pass = "123";
17. Scanner sc = **new** Scanner(System.in);
19. System.out.println("输入服务器ip：");
20. ip = sc.nextLine();
21. System.out.println("输入用户名：");
22. user = sc.nextLine();
23. System.out.println("输入密码：");
24. pass = sc.nextLine();
26. Client ftp = **new** Client(ip, user, pass);
27. ftp.connect();
28. GetUtil gt = **new** GetUtil();
29. menu(ftp, gt);
30. }
32. **public** **static** **void** menu(Client client, GetUtil getUtil) **throws** Exception {
33. System.out.println("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*欢迎来到FTP客户端\*\*\*\*\*\*\*\*\*\*\*\*\*");
34. **while** (**true**) {
35. System.out.println("请选择：1.向服务器传递消息 2.展示工作目录");
36. System.out.println("       3.向服务器发送文件 4.get 下载文件");
37. System.out.println("       5.dir子目录      6.退出系统");
39. Scanner sc = **new** Scanner(System.in);
41. **int** choice = sc.nextInt();
42. **if** (choise1(choice, client, getUtil)) {
43. **break**;
44. }
45. }
46. }
48. **public** **static** **boolean** choise1(**int** cho, Client client, GetUtil getUtil) **throws** Exception {
49. **switch** (cho) {
50. **case** 1: {
51. System.out.println("请输入消息，quit退出");
52. getUtil.communication();
53. }
54. **break**;
55. **case** 2: {
56. String dir = client.pwd();
57. System.out.println(dir);
58. **if** (dir.startsWith("\"\\\"")) ;
59. System.out.println("当前位于服务器根目录\n\n");
60. }
61. **break**;
62. **case** 3: {
63. **while** (**true**) {
64. System.out.println("请输入文件路径");
65. String dir = **new** Scanner(System.in).nextLine();
66. File file = **new** File(dir);
67. **if** (file.isFile()) {
68. **if**(client.stor(file)){
69. System.out.println("传输成功！");
70. }**else** {
71. System.out.println("传输失败！");
72. }
73. **break**;
74. } **else** {
75. System.out.println("路径有误");
76. }
77. }
78. }
79. **break**;
80. **case** 4:{
81. System.out.println("默认保存路径为D盘根目录");
82. System.out.println("请输入文件完整路径（英文，分隔符需要两个）：");
83. String str=**new** Scanner(System.in).nextLine();
84. getUtil.send("get "+str);
85. **new** GetUtil(6789).getMessage();
86. }
87. **break**;
88. **case** 5:{
89. String str=client.dir();
90. **if** (str.startsWith("500")){
91. System.out.println("该功能暂时出现了bug");
92. }
93. System.out.println(str);
95. }**break**;
96. **case** 6:{
97. System.out.println("退出系统，欢迎下次光临");
98. getUtil.send("quit");
99. System.exit(0);
100. }
101. **default**:
102. System.out.println("选择错误请重新选择");
103. **return** **false**;
104. }
105. **return** **false**;
106. }
107. }

**服务器端：**

FTPserver：

1. **import** java.io.\*;
2. **import** java.net.InetAddress;
3. **import** java.net.ServerSocket;
4. **import** java.net.Socket;
6. **public** **class** FTPserver {
8. **private** **int** port = 8821;
9. **private** String ip = "localhost";//设置服务器ip
10. **private** ServerSocket ss;

13. **public** FTPserver() **throws** IOException {
14. bind();
15. }
17. **public** FTPserver(**int** p) **throws** IOException {
18. **this**.port = p;
19. bind();
20. }
22. **public** FTPserver(**int** port, String ip) **throws** IOException {
23. **this**.port = port;
24. **this**.ip = ip;
25. bind();
26. }
28. **public** **void** bind() **throws** IOException {
29. //初始化服务端socket并且绑定端口
30. ss = **new** ServerSocket(port, 10, InetAddress.getByName(ip));
31. System.out.println("server ready");
32. }

35. **void** getFile(String filePath) **throws** IOException {
36. Socket s = **null**;
37. **try** {
38. File fi = **new** File(filePath);
39. System.out.println("文件长度:" + (**int**) fi.length());
41. // public Socket accept() throws
42. // IOException侦听并接受到此套接字的连接。此方法在进行连接之前一直阻塞。
44. s = ss.accept();//与客户端取得连接
45. System.out.println("建立socket链接");
46. DataInputStream dis = **new** DataInputStream(
47. **new** BufferedInputStream(s.getInputStream()));
48. dis.readByte();
50. DataInputStream fis = **new** DataInputStream(
51. **new** BufferedInputStream(**new** FileInputStream(filePath)));
52. DataOutputStream ps = **new** DataOutputStream(s.getOutputStream());
53. // 将文件名及长度传给客户端。这里要真正适用所有平台，例如中文名的处理，还需要加工，具体可以参见Think In Java
54. // 4th里有现成的代码。
55. ps.writeUTF(fi.getName());
56. ps.flush();
57. ps.writeLong((**long**) fi.length());
58. ps.flush();
60. **int** bufferSize = 8192;
61. **byte**[] buf = **new** **byte**[bufferSize];
63. **while** (**true**) {
64. **int** read = 0;
65. **if** (fis != **null**) {
66. read = fis.read(buf);
67. // 从包含的输入流中读取一定数量的字节，并将它们存储到缓冲区数组 b
68. // 中。以整数形式返回实际读取的字节数。在输入数据可用、检测到文件末尾 (end of file)
69. // 或抛出异常之前，此方法将一直阻塞。
70. }
72. **if** (read == -1) {
73. **break**;
74. }
75. ps.write(buf, 0, read);
76. }
77. ps.flush();
78. // 注意关闭socket链接哦，不然客户端会等待server的数据过来，
79. // 直到socket超时，导致数据不完整。
80. fis.close();
81. s.close();
82. System.out.println("文件传输完成");

85. } **catch** (Exception e) {
86. e.printStackTrace();
87. }
88. }
90. **public** **static** **void** main(String arg[]) **throws** IOException {
92. //初始化服务端socket并且绑定端口
93. FTPserver server = **new** FTPserver();
94. //server.getFile("E:\\ftp1.txt");
95. **try** {
96. //等待客户端的连接
97. Socket socket = server.ss.accept();
98. //获取输入流,并且指定统一的编码格式
99. BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream(), "UTF-8"));
100. //读取一行数据
101. String str;
102. **long** flag = 0;
103. //通过while循环不断读取信息，
104. **while** ((str = bufferedReader.readLine()) != **null**) {
105. //输出打印
106. **if** (flag == 0) {//第一行消息去除首位特殊符号
107. str = str.substring(1);
108. }
109. System.out.println("客户端说:"+str);
111. **if**("quit".equals(str)){
112. System.out.println("退出系统");
113. **break**;
114. }
116. **if** (str.startsWith("get")||str.startsWith("GET")) {
118. **int** begin=str.indexOf(":");
119. String filepath=str.substring(begin-1);
120. System.out.println("文件："+filepath);
121. **new** FTPserver(6789).getFile(filepath);
123. }
124. flag++;
125. }
126. } **catch** (IOException e) {
127. e.printStackTrace();
128. }
130. }
131. }