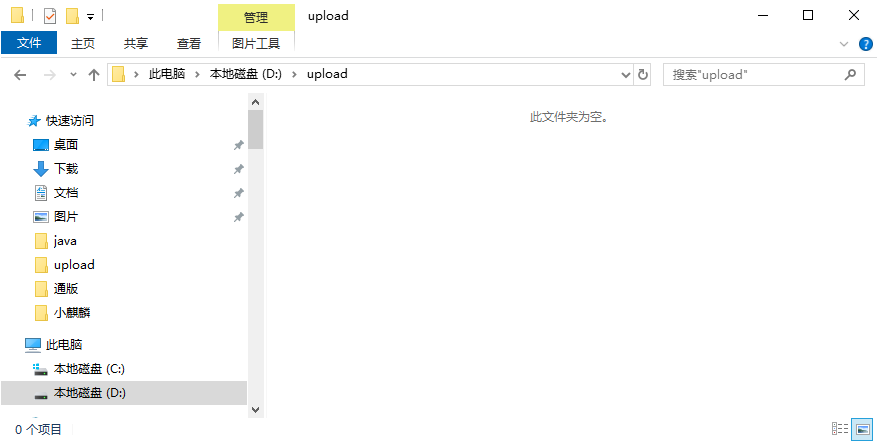
【案例9-4】 客户端向服务端上传文件

### 【案例介绍】

**1.案例描述**

编写一个客户端向服务端上传文件的程序，要求使用TCP通信的的知识，完成将本地机器输入的路径下的文件上传到D盘中名称为upload的文件夹中。并把客户端的IP地址加上count标识作为上传后文件的文件名，即IP（count）的形式。其中，count随着文件的增多而增大，例如127.0.0.(1).jpg、127.0.0.(2).jpg。

**2.效果显示**



上传文件之前



上传文件之后

### 【案例目标】

* 学会分析“文件上传”思路。
* 根据思路独立完成“文件上传”任务的源代码编写、编译及运行。
* 掌握网络通信中TCP协议的编程原理。
* 掌握TCP网络通信ServerSocket和Socket的使用。
* 掌握多线程的使用。

### 【案例分析】

（1）根据任务描述中使用TCP通信的知识实现文件上传功能可知，要实现此功能，需要定义一个服务器接收文件的程序和 一个客户端上传文件的程序。

（2）首先要编写服务器端程序来接收文件。服务器端需要使用ServerSocket对象的accept()方法接收客户端的请求，由于一个服务器可能对于多个客户端，所以当客户端与服务器端简历连接后，服务器需要单独开启一个新的线程来处理与客户端的交互，这时需要在服务器端编写开启新线程的方法。在新线程的方法中，需要获取客户端的端口号，并且使用输入输出流来传输文件到指定的目录中。

（3）编写客户端的功能代码，客户端功能的实现，因为是用户自己输入上传文件。所以要定义键盘录入。录入后需要使用Socket类来创建客户对象，并通过输入输出流来定义指定的文件。

（4）最后我们启动程序，先启动服务端程序，再运行客户端程序来测试上传的结果。

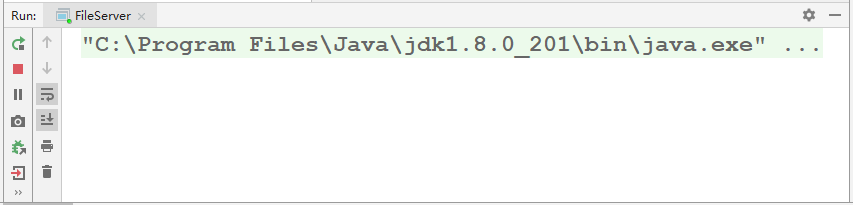
### 【案例实现】

（1）首先编写服务器端的程序，用来接收文件，其代码具体如下所示。

FileServer.java

1. import java.io.File;
2. import java.io.FileOutputStream;
3. import java.io.InputStream;
4. import java.io.OutputStream;
5. import java.net.ServerSocket;
6. import java.net.Socket;
7. public class FileServer {
8. public static void main(String[] args) throws Exception {
9. //创建ServerSocket对象
10. ServerSocket serverSocket = new ServerSocket(10001);
11. while (true) {
12. // 调用accept()方法接收客户端请求，得到Socket对象
13. Socket s = serverSocket.accept();
14. // 每当和客户端建立Socket连接后，单独开启一个线程处理和客户端的交互
15. new Thread(new ServerThread(s)).start();
16. }
17. }
18. }
19. class ServerThread implements Runnable {
20. // 持有一个Socket类型的属性
21. private Socket socket;
22. // 构造方法中把Socket对象作为实参传入
23. public ServerThread(Socket socket) {
24. this.socket = socket;
25. }
26. public void run() {
27. // 获取客户端的IP地址
28. String ip = socket.getInetAddress().getHostAddress();
29. // 上传图片个数
30. int count = 1;
31. try {
32. InputStream in = socket.getInputStream();
33. // 创建上传图片目录的File对象
34. File parentFile = new File("D:\\upload\\");
35. // 如果不存在，就创建这个目录
36. if (!parentFile.exists()) {
37. parentFile.mkdir();
38. }
39. // 把客户端的IP地址作为上传文件的文件名
40. File file = new File(parentFile, ip + "(" + count +
41. ").jpg");
42. while (file.exists()) {
43. // 如果文件名存在，则把count++
44. file = new File(parentFile, ip + "(" + (count++) +
45. ").jpg");
46. }
47. // 创建FileOutputStream对象
48. FileOutputStream fos = new FileOutputStream(file);
49. // 定义一个字节数组
50. byte[] buf = new byte[1024];
51. // 定义一个int类型的变量len，初始值为0
52. int len = 0;
53. // 循环读取数据
54. while ((len = in.read(buf)) != -1) {
55. fos.write(buf, 0, len);
56. }
57. // 获取服务端的输出流
58. OutputStream out = socket.getOutputStream();
59. // 上传成功后向客户端写出“上传成功”
60. out.write("上传成功".getBytes());
61. // 关闭输出流对象
62. fos.close();
63. // 关闭Socket对象
64. socket.close();
65. } catch (Exception e) {
66. throw new RuntimeException(e);
67. }
68. }

运行结果如图所示。



服务器端运行结果

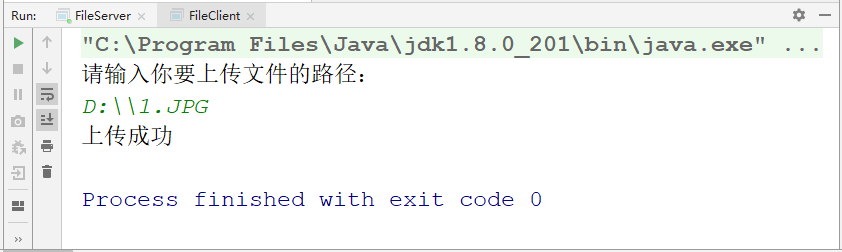
上述代码中，第10行代码用于创建一个ServerSocket对象，第11~16行代码用于在while(true)无限循环中调用ServerSocket的accept()方法来接收客户端的请求，没当和一个客户端建立Socket连接后，就开启一个新的线程和这个客户端进行交互，开启的新线程是通过实现Runnable接口创建的，重写的run()方法中实现了服务端接收并保存客户端上传文件的功能在第34行代码上对文件的保存目录用一个File对象进行封装，如果这个目录不存在就调用File的mkdir()方法创建这个目录，为了避免存放的图片名重复而导致的新上传的文件将已经存在的文件覆盖，在第30行代码定义了一个整型变量count，用于统计文件的数目，使用“IP地址(count).jpg”作为上传文件的名称。在第42~46行代码用于对表示文件名的File对象进行循环判断，如果文件名存在则一直执行count++。最后将从客户端接收的文件信息写入到指定的目录中，在第58~60行代码用于获取服务端的输出流，向客户端输出“上传成功”信息。通过图中运行结果可以看出，服务器端进入阻塞状态，等待客户端连接。

（2）创建客户端用于上传文件，其代码如下所示。

FileClient.java

1. import java.io.FileInputStream;
2. import java.io.InputStream;
3. import java.io.OutputStream;
4. import java.net.Socket;
5. import java.util.Scanner;
6. public class FileClient {
7. public static void main(String[] args) throws Exception {
8. // 创建客户端Socket
9. Socket socket = new Socket("127.0.0.1", 10001);
10. // 获取Socket的输出流对象
11. OutputStream out = socket.getOutputStream();
12. // 创建FileInputStream对象
13. System.out.println("请输入你要上传文件的路径：");
14. Scanner sc =new Scanner(System.in);
15. String upload = sc.nextLine();
16. if(!upload.isEmpty()){
17. FileInputStream fis = new FileInputStream(upload);
18. // 定义一个字节数组
19. byte[] buf = new byte[1024];
20. // 定义一个int类型的变量len
21. int len;
22. // 循环读取数据
23. while ((len = fis.read(buf)) != -1) {
24. out.write(buf, 0, len);
25. }
26. // 关闭客户端输出流
27. socket.shutdownOutput();
28. // 获取Socket的输入流对象
29. InputStream in = socket.getInputStream();
30. // 定义一个字节数组
31. byte[] bufMsg = new byte[1024];
32. // 接收服务端的信息
33. int num = in.read(bufMsg);
34. String Msg = new String(bufMsg, 0, num);
35. System.out.println(Msg);
36. // 关键输入流对象
37. fis.close();
38. // 关闭Socket对象
39. socket.close();
40. }else {
41. System.out.println("对不起请您输入文件路径后再上传！！！");
42. }
43. }
44. }
45. }

客户端运行结果如图所示。



客户端运行结果

上述代码中，第9行代码用于创建一个Socket对象，指定连接服务器的IP地址和端口号，然后获取Socket的输出流对象。第17~25行代码用于创建FileInputStream对象读取我们键盘录入的文件名称，并通过Socket的输出流对象向服务端发送文件。发送完毕后调用Socket的shutDownOutput()方法关闭客户端的输出流。需要注意的是shutDownOutput()方法非常重要，因为服务器端程序在while循环中读取客户端发送的数据时，如果读取不到数据，fis.read(buf)方法会返回-1。也就是说，只要返回的值不是-1。就说明还有数据，需要一直读取，只有fis.read(buf)方法返回的值是-1时循环才会结束。如果客户端不调用shutDownOutput()方法关闭输出流，服务端的fis.read(buf)方法就不会返回-1，而会一直执行while循环，同时客户端读取服务端数据的read(byte [])方法也是一个阻塞方法，这样服务端和客户端程序就进入了一个“死锁”状态。两个程序都不能结束。客户端上传图片成功后，会读取服务端发送的“上传成功”信息，至此，客户端的程序编写完成。