

华东师范大学数据科学与工程学院实验报告

课程名称：计算机网络与编程

年级：21 级

上机实践成绩：

指导教师：张召

姓名：杨茜雅

学号：10215501435

上机实践名称：Lab09

上机实践日期：2023.5.5

上机实践编号：

组号：

上机实践时间：5.5-5.12

一、实验目的

学习使用Datagram Socket 实现 UDP 通信

二、实验任务

使用DatagramSocket 和 DatagramPacket 编写代码

三、使用环境

IntelliJ IDEA

JDK 版本: Java 19

四、实验过程

Task 1: 完善UDPPProvider和UDPSearcher，使得接受端在接受到发送端发送的信息后，将该信息向发送端回写，发送端将接收到的信息打印在控制台上，将修改后的代码和运行结果附在实验报告中

修改后的UDPPProvider

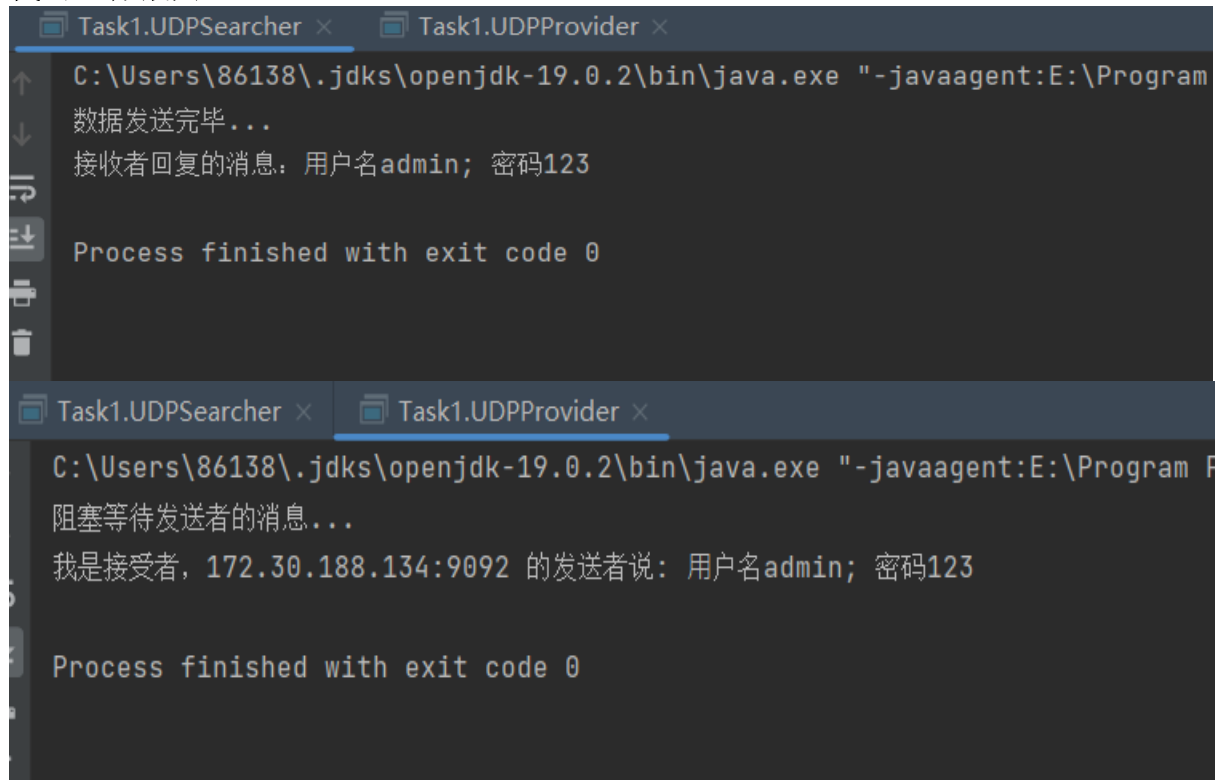
```
package Task1;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class UDPPProvider {
    public static void main(String[] args) throws IOException {
        // 1. 创建接受者端的DatagramSocket，并指定端口
        DatagramSocket datagramSocket = new DatagramSocket( port: 9091);
        // 2. 创建数据报，用于接受客户端发来的数据
        byte[] buf = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(buf, offset: 0, buf.length);
        // 3. 接受客户端发送的数据，此方法在收到数据报之前会一直阻塞
        System.out.println("阻塞等待发送者的消息...");
        datagramSocket.receive(receivePacket);
        // 4. 解析数据
        String ip = receivePacket.getAddress().getHostAddress();
        int port = receivePacket.getPort();
        int len = receivePacket.getLength();
        String data = new String(receivePacket.getData(), offset: 0, len);
        System.out.println("我是接受者, " + ip + ":" + port + " 的发送者说: " + data);
        // Task 1 TODO: 准备回送数据; 创建数据报, 用于发回给发送端; 发送数据报
        String responseData = data;
        byte[] responseBytes = responseData.getBytes();
        DatagramPacket responsePacket = new DatagramPacket(responseBytes, responseBytes.length,
            receivePacket.getAddress(), receivePacket.getPort());
        datagramSocket.send(responsePacket);
        // 5. 关闭datagramSocket
        datagramSocket.close();
    }
}
```

修改后的UDPSearcher:

```
package Task1;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.nio.charset.StandardCharsets;

public class UDPSearcher {
    public static void main(String[] args) throws IOException {
        // 1. 定义要发送的数据
        String sendData = "用户名admin; 密码123";
        byte[] sendBytes = sendData.getBytes(StandardCharsets.UTF_8);
        // 2. 创建发送者端的DatagramSocket对象
        DatagramSocket datagramSocket = new DatagramSocket(port: 9092);
        // 3. 创建数据报, 包含要发送的数据
        DatagramPacket sendPacket = new DatagramPacket(sendBytes, offset: 0, sendBytes.length,
            InetAddress.getLocalHost(), port: 9091);
        // 4. 向接受者端发送数据报
        datagramSocket.send(sendPacket);
        System.out.println("数据发送完毕...");
        // Task 1 TODO: 准备接收Provider的回送消息; 查看接受信息并打印
        byte[] buf = new byte[2048];
        DatagramPacket receivePacket = new DatagramPacket(buf, buf.length);
        datagramSocket.receive(receivePacket);
        String receiveData = new String(receivePacket.getData(), offset: 0, receivePacket.getLength());
        System.out.println("接收者回复的消息: " + receiveData);
        // 5. 关闭datagramSocket
        datagramSocket.close();
    }
}
```

代码运行截图:



Task 2:

改写UDPProvider和UDPSearcher代码完成以下功能，并将实验结果附在实验报告中：

广播地址：255.255.255.255

现需要设计完成如下场景：

UDPSearcher将UDP包发送至广播地址的9091号端口（这表示该UDP包将会被广播至局域网下所有主机的对应端口）。

如果有UDPProvider在监听，解析接受的UDP包，通过解析其中的data得到要回送的端口号，并将自己的一些信息写回，UDPSearcher接收到UDPProvider的消息后打印出来。

现提供发送消息的格式：

UDPSearcher请使用如下buildWithPort构建消息，port在实验中指定为30000。

UDPProvider请使用如下parsePort解析收到的消息并得到要回写的端口号，然后用buildWithTag构建消息，tag可以是 `String tag = UUID.randomUUID().toString()`，然后回写。

UDPSearcher请使用parseTag得到Tag。

UDPProvider:

```
package Task2;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.util.UUID;
public class UDPProvider {
    public static void main(String[] args) throws IOException {
        // 创建接受者端的datagramsocket 并指定接口
        DatagramSocket datagramSocket = new DatagramSocket( port: 9091);
        // 创建数据报用于接收客户端传来的数据
        byte[] buf = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(buf, offset: 0, buf.length);
        // 接收客户端发来的数据此方法在收到数据报之前会一直阻塞
        System.out.println("阻塞等待发送者的消息.....");
        datagramSocket.receive(receivePacket);
        // 解析数据取得新的port
        int len = receivePacket.getLength();
        String newPort = new String(receivePacket.getData(), offset: 0, len);
        int port = MessageUtil.parsePort(newPort);
        System.out.println(port);
        String tag = UUID.randomUUID().toString();
        String data = MessageUtil.buildWithTag(tag);
        //取得新的ip
        String newIP = receivePacket.getAddress().getHostAddress();
        System.out.println("我是接受者," + newIP + ":" + port + "的发送者说:" +
            new String(receivePacket.getData(), offset: 0, len));
        //准备回送数据
        byte[] responseDataBytes = data.getBytes();
        // 创建数据报用于发回给发送端
        DatagramPacket responsePack = new DatagramPacket(responseDataBytes,
            responseDataBytes.length, receivePacket.getAddress(), port);
        datagramSocket.send(responsePack);
        // 关闭UDPProvider
        datagramSocket.close();
    }
}
```

UDPSearcher:

```
package Task2;

import java.io.IOException;
import java.net.*;

public class UDPSearcher {

    public static void main(String[] args) throws IOException {
        DatagramSocket datagramSocket = null;
        try {
            datagramSocket = new DatagramSocket();
        } catch (SocketException e) {
            e.printStackTrace();
        }

        // 定义要发送的数据
        String sendData = MessageUtil.buildWithPort(30000);
        byte[] sendBytes = sendData.getBytes();

        // 发送至localhost: 9091
        DatagramPacket sendPack = null;
        try {
            sendPack = new DatagramPacket(sendBytes, sendBytes.length,
                InetAddress.getByName(host: "255.255.255.255"), port: 9091);
            datagramSocket.send(sendPack);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } System.out.println("数据发送完毕.....");

        // 准备接收Provider 的回送数据
        try {
            datagramSocket = new DatagramSocket(port: 30000);
        } catch (SocketException e) {
            e.printStackTrace();
        } byte[] buf =
            new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(buf, buf.length);
        datagramSocket.receive(receivePacket);

        // 查看receiverPacket 的相关信息
        String ip = receivePacket.getAddress().getHostAddress();
        int port = receivePacket.getPort();
        int len = receivePacket.getLength();
        String data = new String(receivePacket.getData(), offset: 0, len);
        System.out.println("我是发送者," + "接受者" + ip + ":" + port + "说: " + MessageUtil.parseTag(data));

        // 关闭UDPSearcher
        datagramSocket.close();
    }
}
```

运行结果:

```

C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrains\I
阻塞等待发送者的消息.....
30000
我是接受者,192.168.184.1:30000的发送者说:special port:30000
Process finished with exit code 0

C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrains\In
数据发送完毕.....
我是发送者,接受者192.168.184.1:9091说: 43f15fe8-3ec1-4d2b-9c8c-d17da7f46ce3
Process finished with exit code 0

```

Task 3: 现使用UDP实现文件传输功能, 给出UDPFileSender类、请完善UDPFileReceiver类, 实现接收文件的功能。请测试在文件参数为1e3和1e8时的情况, 将修改后的代码和运行时截图附在实验报告中, 并对实验现象进行解释说明。

UDPFileReceiver:

```

package Task3;
import java.io.*;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class UDPFileReceiver {
    public static void main(String[] args) throws IOException {
        File file = new File("checksum_recv.txt"); // 要接收的文件存放路径
        FileOutputStream output = new FileOutputStream(file);
        byte[] data=new byte[1024];
        DatagramSocket ds=new DatagramSocket( port: 9091);
        DatagramPacket dp=new DatagramPacket(data, data.length);

        // 不断接收数据报并将其通过文件输出流写入文件, 以数据报长度为零作为终止条件
        while (true) {
            ds.receive(dp);
            if (dp.getLength() == 0) {
                break;
            }
            output.write(dp.getData(), dp.getOffset(), dp.getLength());
        }
        output.close();
        ds.close();
        System.out.println("接收来自" + dp.getAddress().toString() + "的文件已完成! 对方所使用的端口号为: " + dp.getPort());

        file = new File("checksum_recv.txt");
        System.out.println("接收文件的md5为: " + MD5Util.getMD5(file));
    }
}

```

先发送文件参数为1e3的文件：





```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Pro
接收来自/172.30.188.134的文件已完成！对方所使用的端口号为：57008
接收文件的md5为：28b8ac4cecba9a6f46071edd09c0cb12

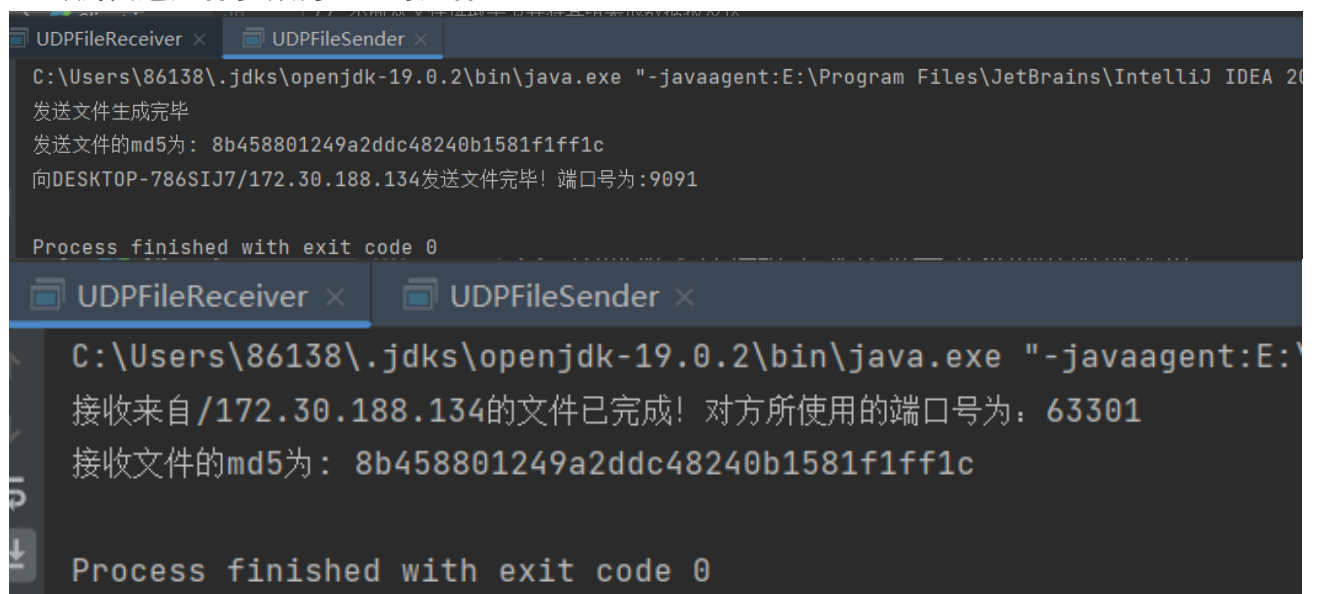
Process finished with exit code 0

C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E
发送文件生成完毕
发送文件的md5为：28b8ac4cecba9a6f46071edd09c0cb12
向DESKTOP-786SIJ7/172.30.188.134发送文件完毕！端口号为：9091

Process finished with exit code 0
```

 checksum.txt	2023-05-06 16:31	Text Document	3 KB
 checksum_recv.txt	2023-05-06 16:31	Text Document	3 KB

再尝试发送文件参数为1e8的文件：





```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrains\IntelliJ IDEA 20
发送文件生成完毕
发送文件的md5为：8b458801249a2ddc48240b1581f1ff1c
向DESKTOP-786SIJ7/172.30.188.134发送文件完毕！端口号为：9091

Process finished with exit code 0

C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\
接收来自/172.30.188.134的文件已完成！对方所使用的端口号为：63301
接收文件的md5为：8b458801249a2ddc48240b1581f1ff1c

Process finished with exit code 0
```

 checksum.txt	2023-05-06 16:29	Text Document	283,669 KB
 checksum_recv.txt	2023-05-06 16:29	Text Document	283,669 KB

对实验现象进行解释说明：

一、在使用UDP实现文件传输功能时，需要先开启receive端再开启sent端，是因为UDP是面向无连接的协议，即发送端发送数据时不需要建立连接，而是直接将数据包发送到目的地址。因此，如果先开启发送端，当发送的数据包到达接收端时，如果接收端还没有启动，则无法接收到这些数据包，导致文件传输失败。通过先启动接收端，确保接收端处于监听状态，可以在发送端开始发送数据之前准备好接收数据包，从而保证传输的数据可以成功接收。因此，在基于UDP的socket实现文件传输时，建议先启动接收端，再启动发送端。

二、同时在传输文件参数为1e8的文件时，会很明显地发现文件传输和接收的时间比文件参数为1e3时要长。

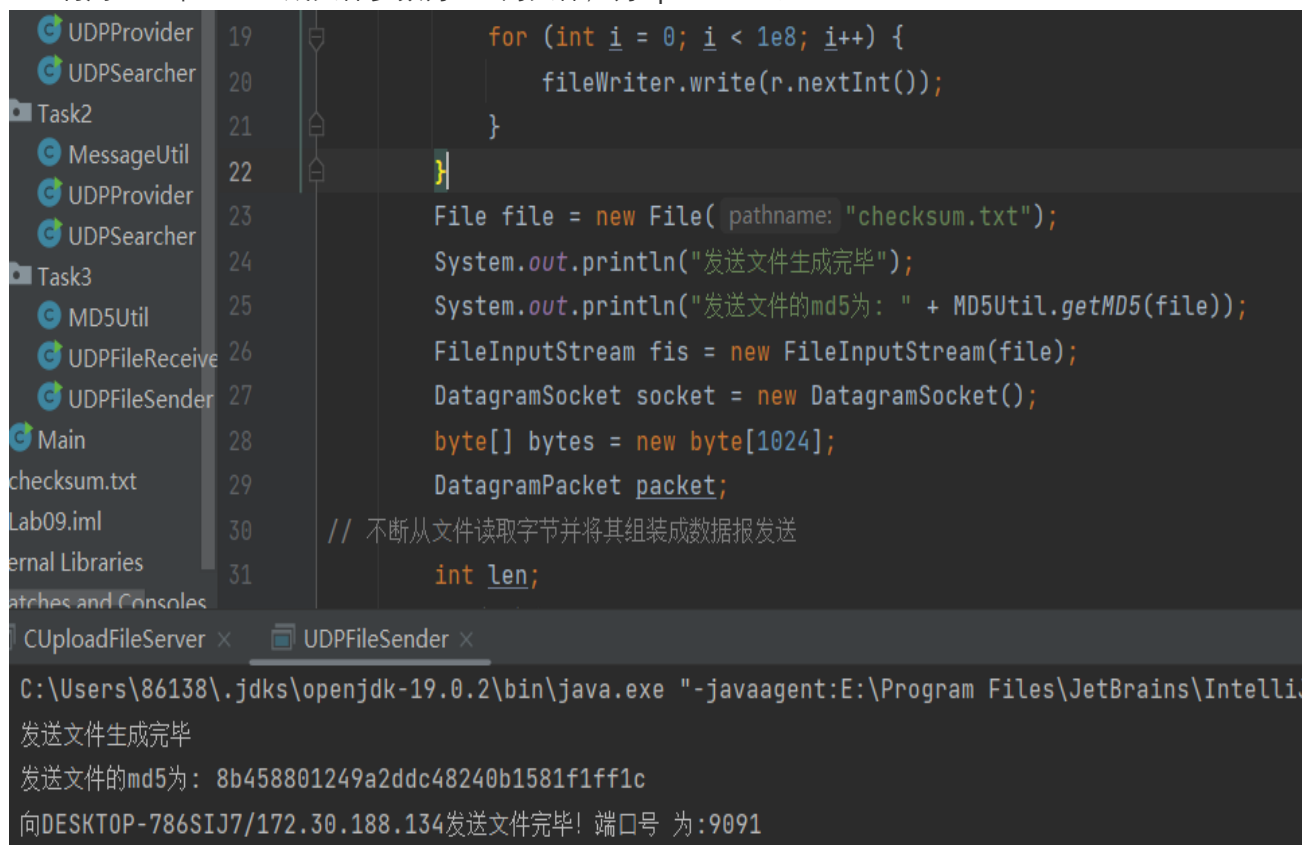
除了文件本身大小的影响外，还可能有以下原因：

- 1、网络带宽和延迟：传输大文件需要较大的网络带宽，如果网络带宽较低，则会导致传输速度变慢。此外，UDP协议本身也不会对网络延迟进行优化，如果网络延迟较高，则会导致数据包传输的延迟增加，从而影响传输速度。
- 2、传输方式的选择：基于UDP的socket实现文件传输时，传输方式的选择也会影响传输速度。如果采用逐个数据包传输的方式，每个数据包传输完毕后再发送下一个数据包，则会造成传输的开销较大。而如果采用批量传输的方式，一次性发送多个数据包，则可以减少传输开销，提高传输效率。
- 3、UDP协议的特点：UDP是一种无连接协议，发送端发送数据时不需要建立连接，也不会对数据进行确认和重传等机制，因此在传输大文件时容易发生数据包丢失或重复等问题。而且UDP传输数据包的大小受到网络MTU（Maximum Transmission Unit，最大传输单元）的限制，如果数据包超过MTU的大小，则需要分片传输，这也会增加传输的开销。

Bouns Task : (2选1) 试完善文件传输功能，可选择 1.使用基于TCP的Socket进行改写；2.优化基于UDP文件传输，包括有序发送、接收端细粒度校验和发送端数据重传。请测试在文件参数为1e8时的情况，将修改后的代码和运行时截图附在实验报告中。

选择基于TCP的Socket进行改写：

1、利用上一个Task生成文件参数为1e8的文件，为upload.txt。



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes folders for Task2, Task3, and Main, along with files like checksum.txt, Lab09.iml, and various utility classes. The code editor displays the implementation of the UDPFileSender class, which generates a file named 'checksum.txt' and sends its MD5 hash and the file content over a DatagramSocket. The console output at the bottom shows the execution of the program, including the MD5 hash and the destination address and port.

```
19      for (int i = 0; i < 1e8; i++) {
20          fileWriter.write(r.nextInt());
21      }
22
23      File file = new File( pathname: "checksum.txt");
24      System.out.println("发送文件生成完毕");
25      System.out.println("发送文件的md5为: " + MD5Util.getMD5(file));
26      FileInputStream fis = new FileInputStream(file);
27      DatagramSocket socket = new DatagramSocket();
28      byte[] bytes = new byte[1024];
29      DatagramPacket packet;
30      // 不断从文件读取字节并将其组装成数据报发送
31      int len;
```

Console Output:

```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrains\IntelliJ
发送文件生成完毕
发送文件的md5为: 8b458801249a2ddc48240b1581f1ff1c
向DESKTOP-786SIJ7/172.30.188.134发送文件完毕! 端口号 为:9091
```

客户端：

> 此电脑 > 新加卷 (E:) > java project > UploadFileClient >

名称	修改日期	类型	大小
.idea	2023-05-06 15:34	文件夹	
out	2023-05-06 15:25	文件夹	
src	2023-05-06 15:27	文件夹	
upload.txt	2023-05-06 15:30	Text Document	283,669 KB
UploadFileClient.iml	2023-05-06 15:22	IML 文件	1 KB


```
package upload.file.client;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
//客户端：上传文件
public class CUploadFileClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket( host: "127.0.0.1", port: 8686);
        BufferedReader bufr = new BufferedReader(new FileReader( fileName: "upload.txt"));
        PrintWriter out = new PrintWriter(s.getOutputStream(), autoFlush: true);
        String line = null;
        //狂读数据
        while ((line = bufr.readLine()) != null)
        {
            //读一行，发送一行
            out.println(line);
        }
        //关闭客户端输出流，相当于给流中加结束标记
        s.shutdownOutput();
        //接受服务端消息
        BufferedReader bufIn = new BufferedReader(new InputStreamReader(s.getInputStream()));
        String str = bufIn.readLine();
        //打印服务器发送过来的消息
        System.out.println(str);
        bufr.close();
        bufIn.close();
        s.close();
    }
}
```

服务端：

此电脑 > 新加卷 (E:) > java project > UploadFileServer >					▼	☆	↺	↻
名称	^	修改日期	类型	大小				
文件夹	.idea	2023-05-06 15:35	文件夹					
文件夹	out	2023-05-06 15:25	文件夹					
文件夹	src	2023-05-06 15:26	文件夹					
文件	UploadFileServer.iml	2023-05-06 15:24	IML 文件	1 KB				

```
package upload.file.server;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

//服务端：接受文件
public class CUploadFileServer {

    public static void main(String[] args) throws Exception{
        //监听8686端口
        ServerSocket ss = new ServerSocket( port: 8686);
        System.out.println("服务器已启动，正在监听8686端口...");
        //进入消息等待接受中...
        Socket s = ss.accept();
        //显示客户端ip地址
        String ip = s.getInetAddress().getHostAddress();
        System.out.println(ip+"已连接");
        //这里是getInputStream流
        BufferedReader bufIn = new BufferedReader(new InputStreamReader(s.getInputStream()));
        //这里是FileWriter
        PrintWriter out = new PrintWriter(new FileWriter( fileName: "Rece.txt"), autoFlush: true);
        String line = null;
        //狂读数据，往文件里写一行
        while ((line = bufIn.readLine()) != null){
            {
                out.println(line);
            }
            //获得输出流，给客户端发送一条消息
            PrintWriter pw = new PrintWriter(s.getOutputStream(), autoFlush: true);
            pw.println("上传成功");
            //关闭资源
            bufIn.close();
            out.close();
            pw.close();
            s.close();ss.close();
        }
    }
}
```

运行代码：

```
upload.file.server.CUploadFileServer x
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe
服务器已启动，正在监听8686端口...
```

```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe
上传成功
```

```
Process finished with exit code 0
```

```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe
服务器已启动，正在监听8686端口...
127.0.0.1已连接
```

```
Process finished with exit code 0
```

名称	修改日期	类型	大小
.idea	2023-05-06 15:41	文件夹	
out	2023-05-06 15:25	文件夹	
src	2023-05-06 15:26	文件夹	
UploadFileServer.iml	2023-05-06 15:24	IML 文件	1 KB
Rece.txt	2023-05-06 15:41	Text Document	283,672 KB

五、总结

通过本次实验掌握了使用DatagramSocket 和DatagramPacket 编写代码，实现了基于UDP 的Socket 通信。了解了一些UDP常用的API及其作用，学习了DatagramSocket的交互过程，实现了使用UDP的文件传输功能以及在不同文件参数下程序运行状态不同的原理原因，同时还对之前基于TCP的socket进行优化，测试了文件参数为1e8时的情况。