

华东师范大学数据科学与工程学院上机实践报告

课程名称：计算机网络与编程

年级：2022 级

上机实践成绩：

指导教师：张召

姓名：李芳

学号：10214602404

上机实践名称：基于 UDP 的 Socket 编程

上机实践日期：2024.04.26

上机实践编号：9

组号：

上机实践时间：

一、基于 UDP 的 Socket 编程---题目要求及实现情况

Task1: 完善UDPPProvider和UDPSeacher，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

储备知识：

一、Java.net.DatagramSocket 包

- 用于支持 UDP 协议的套接字类，提供了在网络上发送和接收数据报的功能。
- UDP：一种无连接的、不可靠的网络传输协议，适用于一些对数据传输延迟要求较低、允许一定数据丢失的场景（实时游戏、音视频传输）。
- 使用流程：
 - 创建 DatagramSocket：在服务端和客户端分别创建 DatagramSocket 对象（可以指定端口号）。
 - 创建数据报包：创建 DatagramPacket 对象来存储待发送或接收的数据以及对应的目标地址和端口号。
 - 发送数据（客户端）：通过 send()方法将数据报发送到目标地址。
 - 接收数据（服务端）：通过 receive()方法从指定端口接收数据报。
 - 处理数据：在服务端或客户端处理接收到的数据。
 - 关闭套接字：使用完毕后，通过 close()方法关闭 DatagramSocket 对象。
- 常用方法：
 - DatagramSocket(int port): 创建一个绑定到指定端口的 DatagramSocket 对象。
 - void send(DatagramPacket p): 发送数据报。
 - void receive(DatagramPacket p): 接收数据报。

- `void close()`: 关闭套接字。
- 服务端与客户端交互流程:
 - ① 服务端: 创建 `DatagramSocket` 实例并绑定到指定端口。
 - ② 服务端: 通过 `receive()` 方法接收客户端发送的数据报。
 - ③ 客户端: 创建 `DatagramSocket` 实例并指定目标地址和端口。
 - ④ 客户端: 通过 `send()` 方法向服务端发送数据报。
 - ⑤ 服务端: 接收到数据报后进行处理, 并可能向客户端发送响应数据。
 - ⑥ 客户端: 接收服务端的响应数据并进行处理。
 - ⑦ 服务端和客户端: 根据需求进行后续的交互操作, 直至通信结束。

二、*Java.net.DatagramPacket* 包

- 作用: 用于表示数据报的类, 包含了要发送或接收的数据以及数据的目标地址和端口号。可以在网络上发送和接收 UDP 数据报。
- 使用流程:
 - 创建 `DatagramPacket`: 创建一个 `DatagramPacket` 对象, 指定要发送或接收的数据以及目标地址和端口号。
 - 发送数据 (客户端): 将 `DatagramPacket` 对象作为参数传递给 `DatagramSocket` 的 `send()` 方法, 发送数据报到目标地址。
 - 接收数据 (服务端): 将 `DatagramPacket` 对象作为参数传递给 `DatagramSocket` 的 `receive()` 方法, 接收数据报。
 - 处理数据: 处理接收到的数据报中的数据。
 - 获取信息: `getData()` 方法获取数据报中的数据, `getAddress()` 获取发送数据报的地址, `getPort()` 获取发送数据报的端口号。
- 常用方法:
 - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`: 创建一个 `DatagramPacket` 实例, 指定数据、目标地址和端口号。

- `byte[] getData()`: 获取数据报中的数据。
- `int getLength()`: 获取数据报中数据的长度。
- `InetAddress getAddress()`: 获取发送数据报的地址。
- `int getPort()`: 获取发送数据报的端口号。

修改后代码& 运行结果:

➤ 代码:

■ UDPPProvider:

```

7
8 public class UDPPProvider {
9     public static void main(String[] args) throws IOException {
10         DatagramSocket datagramSocket = new DatagramSocket(port: 9091);
11         byte[] b = new byte[1024];
12         DatagramPacket receivePacket = new DatagramPacket(b, offset: 0, b.length);
13
14         System.out.println("阻塞等待发送者的消息...");
15         datagramSocket.receive(receivePacket);
16
17         String ip = receivePacket.getAddress().getHostAddress();
18         int port = receivePacket.getPort();
19         int length = receivePacket.getLength();
20         String data = new String(receivePacket.getData(), offset: 0, length);
21         System.out.println("我是接收者, "+ip+" "+port+"的发送者说: "+data);
22
23         String respon = "已收到您的消息. "+data;
24         byte[] responData = respon.getBytes(StandardCharsets.UTF_8);
25         DatagramPacket sendResponPacket = new DatagramPacket(responData, responData.length, receivePacket.getAddress(), receivePacket.getPort());
26         datagramSocket.send(sendResponPacket);
27
28         datagramSocket.close();
29     }
30 }

```

■ UDPSearcher:

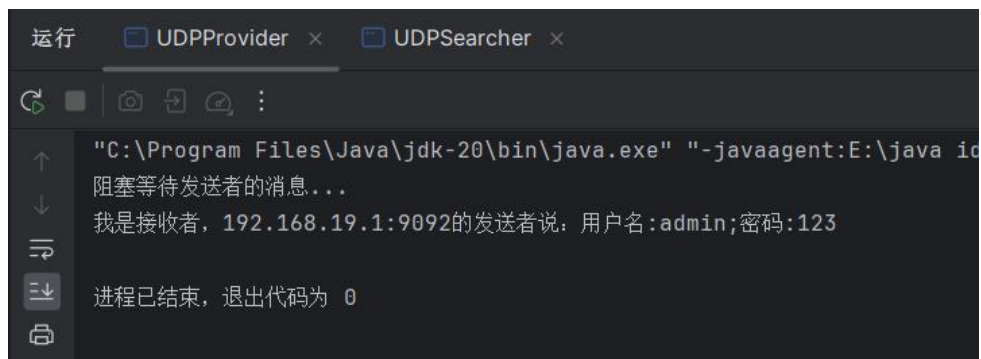
```

9 public class UDPSearcher {
10     public static void main(String[] args) throws IOException {
11         String sendData = "用户名:admin;密码:123";
12         byte[] sendB = sendData.getBytes(StandardCharsets.UTF_8);
13
14         DatagramSocket datagramSocket = new DatagramSocket(port: 9092);
15         DatagramPacket sendPacket = new DatagramPacket(sendB, offset: 0, sendB.length, InetAddress.getLocalHost(), port: 9091);
16
17         datagramSocket.send(sendPacket);
18         System.out.println("数据发送完毕...");
19
20         byte[] receiveResB = new byte[1024];
21         DatagramPacket receiveResonPacket = new DatagramPacket(receiveResB, receiveResB.length);
22         datagramSocket.receive(receiveResonPacket);
23         String receiveResData = new String(receiveResonPacket.getData(), offset: 0, receiveResonPacket.getLength());
24         System.out.println("接收到服务端回写消息: "+receiveResData);
25
26         datagramSocket.close();
27     }
28 }

```

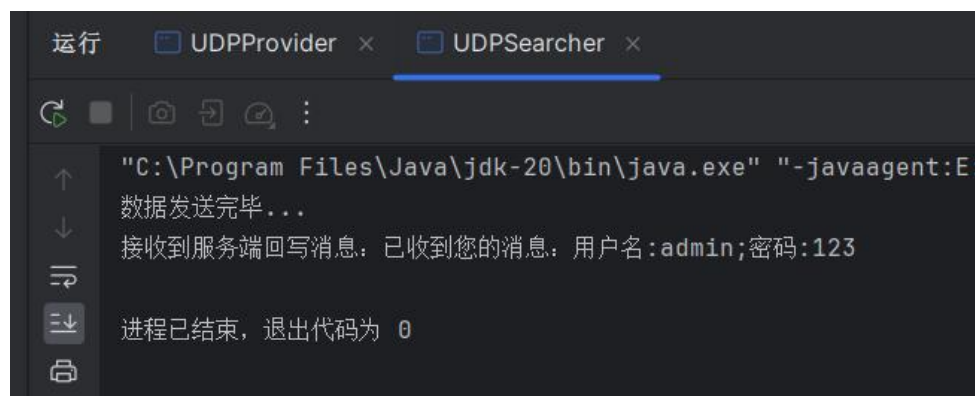
➤ 运行结果:

■ UDPPProvider:



```
运行  UDPPProvider x  UDPSearcher x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\java ic
阻塞等待发送者的消息...
我是接收者, 192.168.19.1:9092的发送者说: 用户名:admin;密码:123
进程已结束, 退出代码为 0
```

■ UDPSearcher:



```
运行  UDPPProvider x  UDPSearcher x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E
数据发送完毕...
接收到服务端回写消息: 已收到您的消息: 用户名:admin;密码:123
进程已结束, 退出代码为 0
```

Task2: 改写UDPPProvider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UDPPProvider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UDPPProvider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。

UDPPProvider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。

修改后代码& 运行结果:

➤ 代码:

■ UDPPProvider:

```

public class UDPPProvider {
    public static void main(String[] args) throws IOException {
        DatagramSocket datagramSocket = new DatagramSocket(port: 9091);
        byte[] buffer = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(buffer, offset: 0, buffer.length);

        System.out.println("阻塞等待发送者的消息...");
        datagramSocket.receive(receivePacket);

        String ip = receivePacket.getAddress().getHostAddress();
        int port = receivePacket.getPort();
        int length = receivePacket.getLength();
        String data = new String(receivePacket.getData(), offset: 0, length);
        System.out.println("我是接收者, " + ip + ":" + port + "的发送者说: " + data);

        int targetPort = MessageUtil.parsePort(data);
        String tag = UUID.randomUUID().toString();
        String sendMsg = MessageUtil.buildWithTag(tag);
        DatagramPacket sendData = new DatagramPacket(sendMsg.getBytes(), sendMsg.getBytes().length,
            InetAddress.getByName(ip), targetPort);
        datagramSocket.send(sendData);
        System.out.println("回应消息已发送到端口:" + targetPort);
        datagramSocket.close();
    }
}

```

■ UDPSearcher:

```

public class UDPSearcher {
    public static void main(String[] args) throws IOException {
        String sendData = MessageUtil.buildWithPort(30000);
        byte[] sendBuffer = sendData.getBytes(StandardCharsets.UTF_8);
        DatagramPacket sendPacket = new DatagramPacket(sendBuffer, offset: 0, sendBuffer.length,
            InetAddress.getByName(host: "255.255.255.255"), port: 9091);

        DatagramSocket datagramSocket = new DatagramSocket(port: 30000);
        //datagramSocket.setBroadcast(true);
        datagramSocket.send(sendPacket);
        System.out.println("数据已发送...");

        byte[] receiveBuffer = new byte[1024];
        DatagramPacket receiveResponsePacket = new DatagramPacket(receiveBuffer, offset: 0, receiveBuffer.length);
        System.out.println("阻塞等待接收者回写的消息...");
        datagramSocket.receive(receiveResponsePacket);

        String receiveData = new String(receiveResponsePacket.getData(), offset: 0, receiveResponsePacket.getLength());
        System.out.println("成功收到接收者回写的消息, 开始解析:");
        String tag = MessageUtil.parseTag(receiveData);
        if (tag != null) {
            System.out.println("接收到服务端回写消息: " + "ip是: " + receiveResponsePacket.getAddress().getHostAddress()
                + "、端口号是: " + receiveResponsePacket.getPort() + "、tag是: " + tag);
        } else {
            System.out.println("未能解析到有效的回写消息");
        }
        datagramSocket.close();
    }
}

```

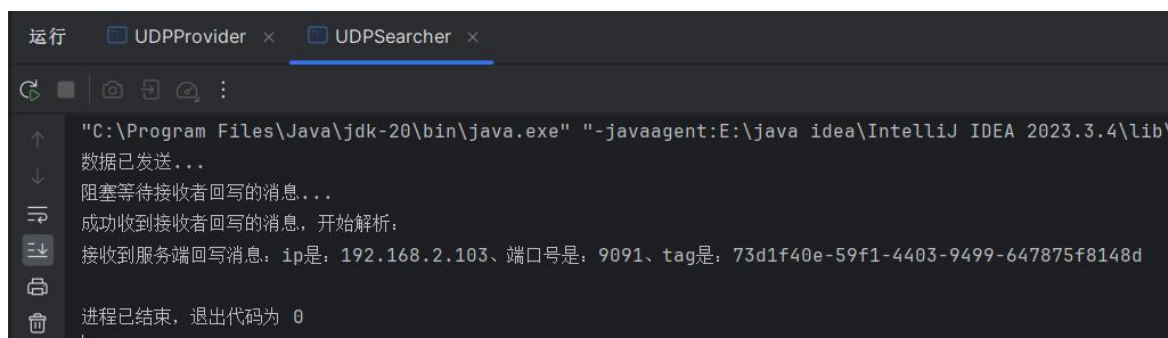
➤ 运行结果:

■ UDPPProvider:



```
运行  UDPPProvider x  UDPSearcher x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\java
阻塞等待发送者的消息...
我是接收者, 192.168.2.103:30000的发送者说: special port:30000
回应消息已发送到端口:30000
进程已结束, 退出代码为 0
```

■ UDPSearcher:



```
运行  UDPPProvider x  UDPSearcher x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\java_idea\IntelliJ IDEA 2023.3.4\lib\
数据已发送...
阻塞等待接收者回写的消息...
成功收到接收者回写的消息, 开始解析:
接收到服务端回写消息: ip是: 192.168.2.103、端口号是: 9091、tag是: 73d1f40e-59f1-4403-9499-647875f8148d
进程已结束, 退出代码为 0
```

➤ 出错与反思:

一开始理解题意有误, 以为要用广播模式, 也就是我在 UDPSearcher 中注释掉的

`//datagramSocket.setBroadcast(true);`这句话, 导致客户端接收不到服务端回写的信息。

设置套接字的广播属性为 `true`, 允许该套接字发送广播数据包。在 UDP 中, 通过广播的方式向同一局域网内的所有主机发送数据包, 而不需要知道目标主机的 IP 地址。因此, 可能会影响性能, 特别是在网络负载较高的情况下, 可能会导致网络延迟增加或者数据包丢失。

Task3: 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。

储备知识:

一、MD5 算法

- 一种常用的哈希算法, 用于生成数据的摘要或签名。
- MD5 算法将任意长度的数据作为输入, 经过计算生成一个固定长度 (128 位, 通常表示为 **32 个十六进制数字**) 的哈希值。这个哈希值通常用于验证数据的完整性 (检查文件是否被篡改等)。

- 原理：将输入的数据分成若干个固定大小的数据块，然后对每个数据块进行一系列的处理，最终生成一个 128 位的哈希值。
- 处理步骤：
 - 填充：对输入数据进行填充，使其长度符合算法的要求。
 - 初始化：初始化一个 128 位的缓冲区，称为状态变量（MD5 中有四个状态变量）。
 - 处理数据块：将每个数据块进行一系列的处理，包括多轮的位运算、逻辑函数以及状态变量的更新。
 - 输出结果：最终将处理后的状态变量连接起来，得到最终的 128 位哈希值。
- 应用：
 - **数据完整性校验：文件传输过程中，计算文件的 MD5 值并与接收方计算的值进行比较，以确保文件未被篡改。**
 - 密码存储：过去常用于密码存储，但由于其存在碰撞（两个不同的输入数据可以生成相同的哈希值）和加盐（在密码哈希过程中添加随机的额外数据）等安全性问题，现在更多地使用更安全的哈希算法（SHA-256 等）。
 - 数字签名：要签名的数据通过 MD5 算法生成摘要，然后使用私钥对摘要进行加密。
 - **消息验证：发送方可以对消息进行哈希运算并附加在消息中，接收方可以对接收到的消息进行相同的哈希运算并与附加的哈希值比较，从而验证消息的完整性和真实性。**

二、FileWriter 包

- 属于 Java I/O 包，用于向文件写入字符数据的类，继承自 Writer 类，提供一种便捷的方式写入字符到文件中。
- 方法：
 - 构造函数：接受一个字符串类型的文件名或一个 File 对象作为参数，用于指定要写入的文件路径。
 - 写入字符数据：使用 write() 方法的多个重载形式，可以写入单个字符、字符数组或字符串等数据。
 - 刷新：写入完数据后，调用 flush() 方法将缓冲区的数据刷新到文件中。
 - 关闭：调用 close() 方法关闭流，释放资源。
 - 异常处理：在写入文件时，可能会抛出 IOException 异常。

修改后代码& 运行结果:

➤ 初步完善 UDPFileReceiver 类

■ 代码:

```
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);

        while (true) {
            ds.receive(dp);
            int len = dp.getLength();
            if (len == 0) {
                break;
            }
            output.write(data, off: 0, len);
        }

        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));
    }
}
```

1. 增加while循环: 保证进行不断接收
2. 循环中接收数据报, 并将接收到的数据写入文件
3. 在接收到数据报长度为零时, 文件接收完毕, 退出循环

➤ 1e3 情况下, 运行结果:

◆ UDPFileReceiver 类



```
运行  UDPFileSender x  UDPFileReceiver x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\j...
接收来自 192.168.19.1 的文件已完成! 对方所使用的端口号为: 56785
接收文件的md5为: 28b8ac4cecb9a6f46071edd09c0cb12
进程已结束, 退出代码为 0
```


◆ UDPFileSender 类



```
运行  UDPFileSender x  UDPFileReceiver x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\j...
发送文件生成完毕
发送文件的md5为: 28b8ac4cecb9a6f46071edd09c0cb12
向LAPTOP-L9FRK1MI/192.168.19.1发送文件完毕! 端口号为: 9091
进程已结束, 退出代码为 0
```

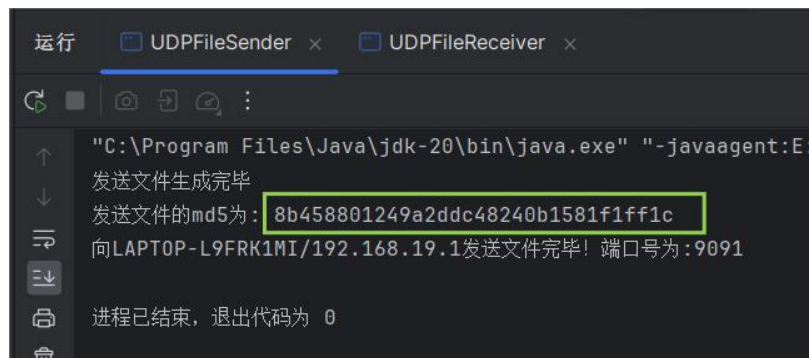

➤ 1e8 情况下，运行结果：

◆ UDPFileReceiver 类



```
运行  UDPFileSender x  UDPFileReceiver x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\
接收来自 /192.168.19.1 的文件已完成! 对方所使用的端口号为: 63625
接收文件的md5为: 24be290028c1720d832e50d3f1fa6137
进程已结束, 退出代码为 0
```

◆ UDPFileSender 类



```
运行  UDPFileSender x  UDPFileReceiver x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:
发送文件生成完毕
发送文件的md5为: 8b458801249a2ddc48240b1581f1ff1c
向LAPT0P-L9FRK1MI/192.168.19.1发送文件完毕! 端口号为: 9091
进程已结束, 退出代码为 0
```

➤ 打印结果分析：

■ 由截图可知：

1e3 情况下，运行时间很短，且发送端和接收端打印的 md5 值相同，证明文件接收正确；相反，1e8 情况下，运行时间较长，发送端和接收端的 md5 值不同，证明文件接收出现差错。

1e8 出现文件传输差错的原因可能是：**由于数据传输时发生了丢包、数据损坏或乱序等情况，导致接收到的文件与原始文件不一致，因此计算出的 MD5 值也不同。**

Bonus Task1: (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。

(1.选项) 修改后代码&运行结果：

➤ 代码：

■ TCPFileSender:

```
public class TCPFileReceiver {
    public static void main(String[] args) throws IOException, NoSuchAlgorithmException {
        String fileName = "checksum_recv1e8_tcp.txt";
        File file = new File(fileName);
        FileOutputStream output = new FileOutputStream(file);
        ServerSocket serverSocket = new ServerSocket(port: 9091);
        System.out.println("阻塞等待客户端连接中...");
        Socket socket = serverSocket.accept();

        InputStream is = socket.getInputStream();

        try {
            byte[] buffer = new byte[1024];
            int len;
            while ((len = is.read(buffer)) != -1) {
                output.write(buffer, off: 0, len);
            }
            System.out.println("文件接收完毕!");
            System.out.println("接收文件的md5为: " + MD5Util.getMD5(file));
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            output.close();
            is.close();
            socket.close();
            serverSocket.close();
        }
    }
}
```

■ TCPFileReceiver:

```
public class TCPFileSender {
    public static void main(String[] args) throws IOException, NoSuchAlgorithmException {
        String fileName = "checksum1e8_tcp.txt";
        FileInputStream fis = new FileInputStream(fileName);
        FileWriter fileWriter = new FileWriter(fileName);
        Socket socket = new Socket(host: "127.0.0.1", port: 9091);
        BufferedInputStream bis = new BufferedInputStream(fis);
        OutputStream os = socket.getOutputStream();
        try {
            Random random = new Random(seed: 2023);
            for (int i = 0; i < 1e8; i++) {
                fileWriter.write(random.nextInt());
            }
            System.out.println("发送文件生成完毕");
            System.out.println("发送文件的md5为: " + MD5Util.getMD5(new File(fileName)));

            byte[] buffer = new byte[1024];
            int len;
            while ((len = bis.read(buffer)) != -1) {
                os.write(buffer, off: 0, len);
            }
            os.flush();
            System.out.println("文件发送完毕!");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            socket.close();
        }
    }
}
```

➤ 运行结果：

■ TCPFileSender:



```
运行 TCPFileSender x TCPFileReceiver x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\j...
发送文件生成完毕
发送文件的md5为: 5d168af99e62e47b5b49b6f1fe13bcbf
文件发送完毕!
进程已结束, 退出代码为 0
```

■ TCPFileReceiver:



```
运行 TCPFileSender x TCPFileReceiver x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:E:\j...
阻塞等待客户端连接中...
文件接收完毕!
接收文件的md5为: 5d168af99e62e47b5b49b6f1fe13bcbf
进程已结束, 退出代码为 0
```

- 结果分析：从打印结果中可以看出，**TCP 确实是可靠的文件传输方式**，发送的和接收的文件 md5 是完全相同的。

二、总结

这次实验主题是 UDP 通信和文件传输功能，在 IDEA 中通过使用 Java 代码的 Datagram Socket 实现了 UDP 通信，还实现了基本的消息发送、接收和文件传输功能。在实验过程中，我逐步完善了 UDPPProvider 和 UDPSeacher，以便于接收端接收到信息后向发送端回写、发送端打印接收到的信息、广播、文件传输等功能。

Task1: 我利用 DatagramSocket 和 DatagramPacket 编写了 UDPPProvider 和 UDPSeacher，实现了基本的 UDP 通信功能：UDPPProvider 接收到 UDP 包后，将接收到的信息向发送端回写、UDPSeacher 将接收到的信息打印在控制台上。

Task2: 我实现了 UDP 广播功能, 将 UDP 包发送至广播地址的指定端口, 并通过解析收到的 UDP 包进行回写, 实现了 UDPProvider 和 UDPSearcher 之间的消息交互。

Task3: 我实现了 UDPFileSender 和 UDPFileReceiver, 也就是通过 UDP 实现了文件的发送和接收功能, 还对不同大小的文件 (1e3 和 1e8) 进行了测试, 探讨了对应的实验现象及出现原因。

Bonus Task1: 我选择了 TCP 这一可靠的文件传输协议, 巩固了上次实验 TCP 的知识, 实现了它的文件传输功能。

在实验过程中, 我发现了一些问题, 并想办法去解决优化了它:

1. UDP 通信存在丢包和乱序的情况, 传输文件时, UDP 相比于 TCP 具有更低的传输效率和可靠性, 尤其对于大文件传输来说, 丢包的情况更加明显, 可以考虑实现有序发送和接收端细粒度校验来提高传输的可靠性, 实现数据压缩和分片传输等技术提高 UDP 的传输效率。

2. UDP 广播功能可以方便地实现局域网内的消息传递, 但需要注意广播地址的设置和消息的格式。

综上, 通过本次实验, 我对 UDP 通信和文件传输有了更深入的掌握和探究, 提高了我在 Java 网络编程方面的实践能力。通过对实验中遇到的问题进行分析和改进, 让我能够更加得心应手地应用 UDP 通信和文件传输技术。