

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：22级	上机实践成绩：
指导教师：张召	姓名：朱天祥	
上机实践名称：基于UDP的Socket编程	学号：10225501461	上机实践日期：23/5/5
上机实践编号：No.9	组号：	

一、目的

学习使用Datagram Socket实现UDP通信

二、实验内容

1. 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

Task1: 完善UDPProvider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

UDPProvider类：

```
public class UDPProvider {
    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(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, 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(), 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 msg = "我是接收方，我已经收到了发送方的消息";
        byte[] msgBytes = msg.getBytes(StandardCharsets.UTF_8);
        DatagramPacket sendPacket = new DatagramPacket(msgBytes, 0,
msgBytes.length, receivePacket.getAddress(), port);
        datagramSocket.send(sendPacket);
        System.out.println("数据发送完毕");
// 5. 关闭datagramSocket
        datagramSocket.close();
    }
}

```

UDPSearcher类:

```

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(9092);
// 3. 创建数据报，包含要发送的数据
        DatagramPacket sendPacket = new DatagramPacket(sendBytes, 0,
sendBytes.length,
            InetAddress.getLocalHost(), 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[] receiveBuf = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveBuf, 0, 1024);
        System.out.println("阻塞等待接受方的回写消息");
        datagramSocket.receive(receivePacket);

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

// 5. 关闭datagramSocket
        datagramSocket.close();
    }
}

```

UDPPProvider类的运行结果:

```
UDPPProvider x  UDPSearcher x
D:\Java\jdk1.8.0_40\bin\java.exe ...
阻塞等待发送者的消息...
我是接受者, 169.254.238.114:9092 的发送者说: 用户名admin; 密码123
数据发送完毕
Process finished with exit code 0
```

UDPSearcher类的运行结果:

```
UDPPProvider x  UDPSearcher x
D:\Java\jdk1.8.0_40\bin\java.exe ...
数据发送完毕...
阻塞等待接受方的回写消息
我是发送者, 169.254.238.114:9091 的接收者说: 我是接收方, 我已经收到了发送方的消息
Process finished with exit code 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:

UDPPProvider类:

```
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(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, 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
        int len = receivePacket.getLength();
        String data = new String(receivePacket.getData(), 0, len);
        int parsePort = MessageUtil.parsePort(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 msg = MessageUtil.buildWithTag(UUID.randomUUID().toString());
        byte[] msgBytes = msg.getBytes(StandardCharsets.UTF_8);
        DatagramPacket sendPacket = new DatagramPacket(msgBytes, 0,
msgBytes.length, receivePacket.getAddress(), parsePort);
        datagramSocket.send(sendPacket);
        System.out.println("数据发送完毕");
        // 5. 关闭datagramSocket
        datagramSocket.close();
    }
}
```

UDPSearcher类:

```
public class UDPSearcher {
    public static void main(String[] args) throws IOException {
```

```

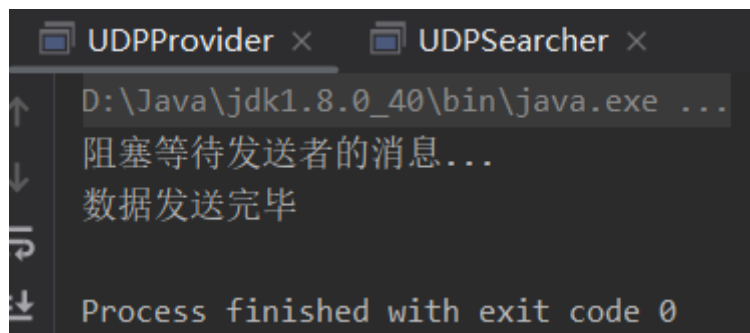
// 1. 定义要发送的数据
String sendData = MessageUtil.buildWithPort(30000);
byte[] sendBytes = sendData.getBytes(StandardCharsets.UTF_8);
// 2. 创建发送者端的DatagramSocket对象
DatagramSocket datagramSocket = new DatagramSocket(30000);
// 3. 创建数据报, 包含要发送的数据
DatagramPacket sendPacket = new DatagramPacket(sendBytes, 0,
sendBytes.length,
    InetAddress.getByName("255.255.255.255"), 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[] receiveBuf = new byte[1024];
DatagramPacket receivePacket = new DatagramPacket(receiveBuf, 0, 1024);
System.out.println("阻塞等待接受方的回写消息");
datagramSocket.receive(receivePacket);
int len = receivePacket.getLength();
String data = new String(receivePacket.getData(), 0, len);
System.out.println("tag为: " + MessageUtil.parseTag(data));

// 5. 关闭datagramSocket
datagramSocket.close();
}
}

```

UDPProvider运行结果:

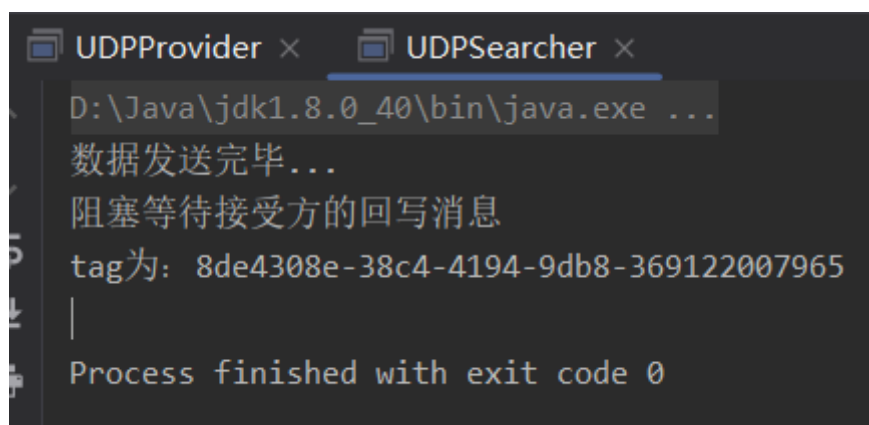


```

D:\Java\jdk1.8.0_40\bin\java.exe ...
阻塞等待发送者的消息...
数据发送完毕
Process finished with exit code 0

```

UDPSearcher运行结果:



```

D:\Java\jdk1.8.0_40\bin\java.exe ...
数据发送完毕...
阻塞等待接受方的回写消息
tag为: 8de4308e-38c4-4194-9db8-369122007965
|
Process finished with exit code 0

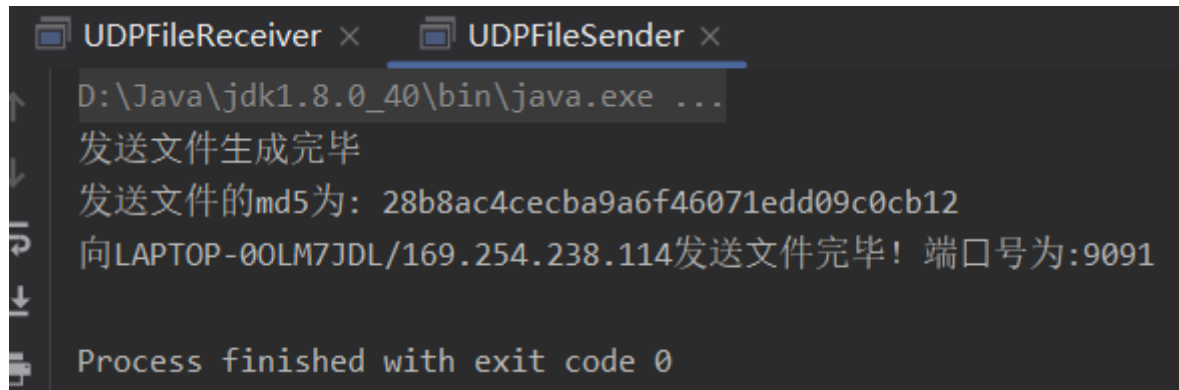
```

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。

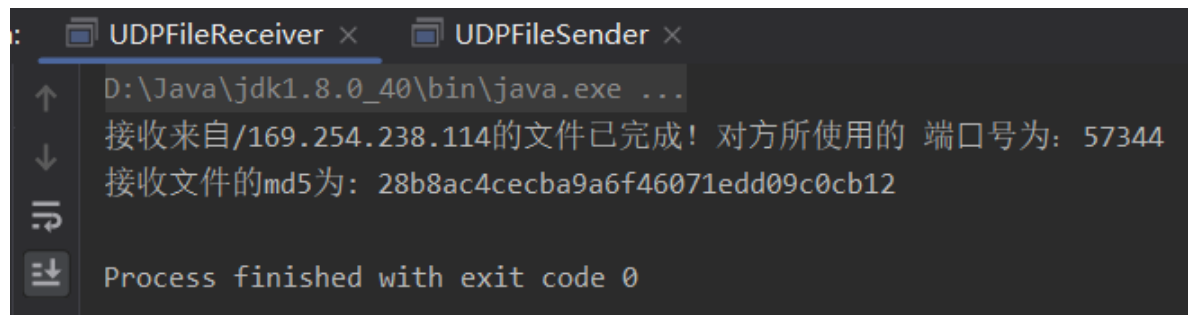
UDPFileReceiver的TODO代码段：

```
// TODO 实现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);
    int len = dp.getLength();
    if(len == 0)
        break;
    output.write(data,0,len);
}
```

当参数为1e3时Sender和Receiver的运行结果：

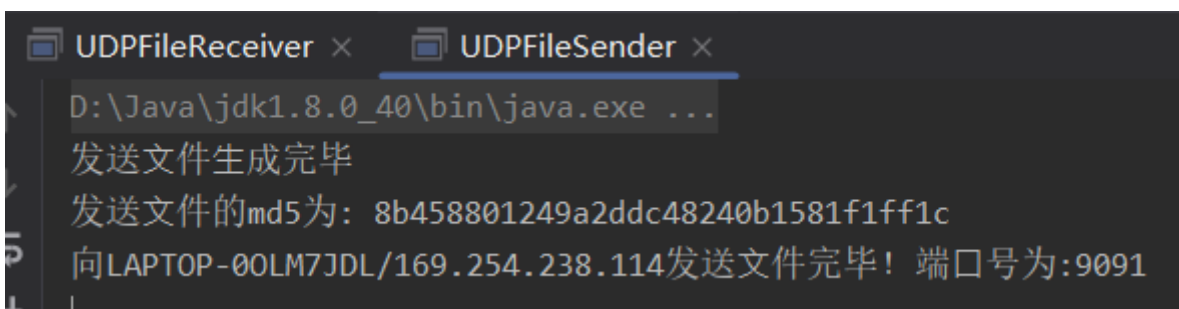


UDPFileReceiver x UDPFileSender x
D:\Java\jdk1.8.0_40\bin\java.exe ...
发送文件生成完毕
发送文件的md5为: 28b8ac4cecb9a6f46071edd09c0cb12
向LAPTOP-00LM7JDL/169.254.238.114发送文件完毕! 端口号为:9091
Process finished with exit code 0

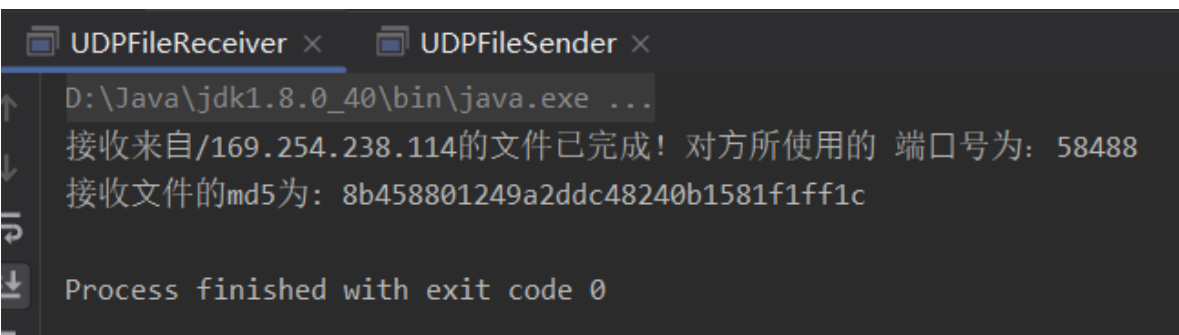


UDPFileReceiver x UDPFileSender x
D:\Java\jdk1.8.0_40\bin\java.exe ...
接收来自/169.254.238.114的文件已完成! 对方所使用的 端口号为: 57344
接收文件的md5为: 28b8ac4cecb9a6f46071edd09c0cb12
Process finished with exit code 0

当参数为1e8时Sender和Receiver的运行结果：



UDPFileReceiver x UDPFileSender x
D:\Java\jdk1.8.0_40\bin\java.exe ...
发送文件生成完毕
发送文件的md5为: 8b458801249a2ddc48240b1581f1ff1c
向LAPTOP-00LM7JDL/169.254.238.114发送文件完毕! 端口号为:9091



UDPFileReceiver x UDPFileSender x
D:\Java\jdk1.8.0_40\bin\java.exe ...
接收来自/169.254.238.114的文件已完成! 对方所使用的 端口号为: 58488
接收文件的md5为: 8b458801249a2ddc48240b1581f1ff1c
Process finished with exit code 0

运行了好多次没发现什么问题，md5都是一样的

但是我猜想可能出现的错误应该就是接收方和发送方md5不一样，原因大概率是因为UDP传输并不可靠，无法确保数据按顺序到达接收方，可能会存在先发出去的数据包在后发出去的数据包之后才被接收到，导致数据乱序，最后接受到的文件因为数据顺序不同而md5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。

我完善文件传输功能的方法是优化基于UDP文件传输，大致思路考虑的比较简单，在接收方和发送方添加校验和函数，发送方一开始把校验和加在数据头四个字节中，接收方接受到数据后把校验和提取出来，并自己计算接收到数据正文的校验和，并与发送方的校验和核对，如果相同那么发送ack数据报给发送方，不相同则发送接受错误的数据包，然后阻塞等待发送方重新发送这一组数据。代码如下

校验和函数：

```
private static int checksum(byte[] data,int offset, int length) {
    int sum = 0;

    for (int i = offset; i < length + offset; i++) {
        sum += (int) data[i] & 0xFF;
    }

    return sum;
}
```

UDPFileReceiver：

```
// TODO 实现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);
    int len = dp.getLength();
    if (len == 0)
        break;
    while(true) {
        int checksum = checksum(data,4,len - 4);
        int sum_recv = ByteBuffer.wrap(Arrays.copyOfRange(data, 0, 4)).getInt();
        if(checksum == sum_recv){
            output.write(data, 4, len - 4);
            ack[0] = 1;
            ds.send(sendAck);
            break;
        }
        ack[0] = 0;
        ds.send(sendAck);
        ds.receive(dp);
    }
}
```

UDPFileSender

```
for(;;){
    len = fis.read(bytes);
    if(len==-1) break;
```

```

        packet = new DatagramPacket(addHeader(bytes, checksum(bytes, len)), len + 4,
        InetAddress.getLocalHost(), 9091);
        while(true) {
            socket.send(packet);
            receive = new DatagramPacket(ack, 0, ack.length);
            socket.receive(receive);
            if (ack[0] == 1){
                ack[0] = 0;
                break;
            }
        }
    }
}

```

运行结果:

Receiver

```

D:\Java\jdk1.8.0_40\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:56240', transport: 'socket'
接收来自/169.254.238.114的文件已完成! 对方所使用的 端口号为: 9092
接收文件的md5为: 8b458801249a2ddc48240b1581f1ff1c
Disconnected from the target VM, address: '127.0.0.1:56240', transport: 'socket'

Process finished with exit code 0

```

sender

```

D:\Java\jdk1.8.0_40\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:56245', transport: 'socket'
发送文件生成完毕
发送文件的md5为: 8b458801249a2ddc48240b1581f1ff1c
向LAPTOP-00LM7JDL/169.254.238.114发送文件完毕! 端口号为:9091
Disconnected from the target VM, address: '127.0.0.1:56245', transport: 'socket'

Process finished with exit code 0

```

实验总结

UDP是一种面向数据报的传输协议，它具有简单、高效、轻量级等优点。但是UDP没有提供数据传输的可靠性保障，即数据可能会丢失或丢失顺序，这为文件传输带来了不小的困难。因此，为了优化基于UDP的文件传输，需要在传输过程中进行有序发送、接收端细粒度校验和、发送端数据重传等措施来保证其可靠性。

