



合肥工业大学

计算机与信息学院

项目设计报告

课 程： 计算机网络

专业班级： 计算机科学与技术 18-2 班

学 号： 2018211958

学生姓名： 孙淼

目录

一、项目实验要求.....	3
1) 项目实验目的	3
2) 项目实验内容	3
3) 项目实验设计	3
二、开发环境与工具	3
三、软件使用说明及测试	3
四、关键问题及其解决方法	10
1) 新建文件路径问题	10
2) 命名规范的重要性	12
3) 根据提示导入库的时候需要注意	13
4) 参数缺失报错	13
五、附录（详细注释的源码）	13
server.java	13
client.java	15
GBN.java	17

一、项目实验要求

1) 项目实验目的

- 1.理解滑动窗口协议的基本原理;
- 2.掌握 GBN 的工作原理;
- 3.掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

2) 项目实验内容

- 1.基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输(服务器到客户的数据传输);
- 2.模拟引入数据包的丢失，验证所设计协议的有效性;
- 3.改进所设计的 GBN 协议，支持双向数据传输。

3) 项目实验设计

- 1.基于 UDP 实现的 GBN 协议，可以不进行差错检测，可以利用 UDP 协议差错检测;
- 2.自行设计数据帧的格式，应至少包含序列号 Seq 和数据两部分;
- 3.自行定义发送端序列号 Seq 比特数 L 以及发送窗口大小 W;
- 4.为了模拟 ACK 丢失，可以利用模 N 运算，每 N 次模拟丢包，或者每 N 次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，如果在发送端，则少发送数据包，在接收端则不发回 ACK;
- 5.当设置服务器端发送窗口的大小为 1 时，GBN 协议就是停-等协议。

二、开发环境与工具

开发工具：Intel® Core(TM)i7-8750H CPU @ 2.20GHz 2.20GHz, windows 10

语言环境：java, eclipse-java-2019-06-R

三、软件使用说明及测试

对于实验内容一，若要只实现单向数据传输，只给服务器端 send, 客户 receive

即可；

对于实验内容二，为了模拟 ACK 丢失，我采用的是模 N 运算，每 N 次模拟丢包，或者每 N 次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，如果在发送端，则少发送数据包，在接收端则不发回 ACK，我选择在接收端实现，也就是每隔 N（我设置 N=100）就不返回 ACK，这部分具体实现代码如下：

```
26 //按照模N运算模拟丢包，N即Lossgap
27 private final int Lossgap = 100;
166 //当模N为0时，人为产生一个丢包
167 if (count % Lossgap != 0) {
168     long seq = recv[0] & 0xFF;
169     if (seq == nowseq) {
170         //accept
171         nowseq++;
172         if (nowseq>=256)nowseq=0;
173         //写在上write to upper
174         result.write(recv, 1, receivePacket.getLength()-1);
175         //发送ACK send ACK
176         recv = new byte[1];
177         recv[0] = new Long(seq).byteValue();
178         receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);
179         datagramSocket.send(receivePacket);
180         System.out.println("receive datagram seq: " + seq);
181         datagramSocket.setSoTimeout(endTime*1000);
182     }else discard
183 }
```

对应实验内容三，此处展示改进后的 GBN 协议，也就是实现了双向数据传输的程序，具体实现在第五部分源码附录以及注释中可以看到，此处简单证明：

```
11 File file1 = new File("./src/computer_net/2.png");//接受文件目录
12 File file2 = new File("./src/computer_net/3.png");//传送文件目录
13 //产生file
14 if (!file1.exists()) {
15     file1.createNewFile();
16 }

12 File file1 = new File("./src/computer_net/1.png");//传送文件目录
13 File file2 = new File("./src/computer_net/4.png");//接受文件目录
14 //其实读取要传送的文件是不需要createNewFile()的
15 /*if(!file1.exists()) {
16     file1.createNewFile();
17 }*/
18 if(!file2.exists()) {
19     file2.createNewFile();
20 }
```

上面分别是服务器端和客户端的文件路径部分，对于服务器端，我选择读取 3.png 用于传向客户端，并且将从客户端接收到的文件 1.png 保存为 2.png（双向传输）；对于客户端，我选择读取 1.png 用于传向服务器端，并且将从服务器端接收到的文件 3.png 保存为 4.png（双向传输）。

对于实验设计二自行设计数据帧的格式，应至少包含序列号 Seq 和数据两部

分，实际上在序列号前面还设计了 base 部分；

```
24 //起始base值
25 private long base = 0;

48 //自行设计数据包最大字节数
49 final int MAX_LENGTH = 1024;

56 //自行设计序列号
57 long sendSeq = base;
```

base 与 sendSeq 序列号初始值相同但是作用不同，base 用于超时重传，sendSeq 单纯作为序列号，用于发送接收的显示。

根据参考资料，GBN 一个分组的发送格式是：

Base(1Byte) + seq(1Byte) + data(max 1024Byte);

解释如下：GBN 协议的传送流程是：从上层应用层获得到一个完整的数据报，将这个数据报进行拆分，因为在以太网中，数据帧的 MTU 为 1500 字节，所以 UDP 数据报的数据部分应小于 1472 字节(除去 IP 头部 20 字节与 UDP 头的 8 字节)

所以一个 GBN 数据帧最大传输的数据大小限制为 1024B。需要注意的是：

对于发送方，起始 base 的值，窗口采用链表的数据结构存储每个发出去的数据包(因为后面超时需要重传，所以得按顺序能标示地存下来)进入一个循环，循环结束条件是所有需要传送的数据都已经发送完成,并且窗口中的分组都已经全部确认。在这个循环中,如果窗口内有空余,就开始发送分组,直到窗口被占满,计时器开始计时,之后进入接收 ACK 的状态,收到 ACK 之后，更新滑动窗口的位置,之后如果计时器超时，就将窗口内所有的分组全部重发一次。之后开始下一次循环。(构造数据包的时候只需要构造 1 字节的 seq 值放在头部)

对于接收方，记录一个 base 值,每成功接收一个数据帧,base+1,开始循环顺序接收数据帧,对于 base 不是目标值得数据帧直接丢弃，如果是符合要求的数据帧,就给发送方发送一个 ACK=base 的确认数据帧,直到发送方没有数据传来为止。

对于实验设计三，设计的序列号 sendSeq 为 8 位，对应最大序列号（10 进制）为 0-255，

```
128 //base每到256，就将base和sendSeq重置
129 if (base >= 256) {
130     base = base - 256;
131     sendSeq = sendSeq - 256;
132 }
```

滑动窗口设为 16，也是 256 的整数倍，注意此处若更改窗口大小为 1，则 GBN

```
20 //若此处改窗口大小为1，则为停等协议
21 private int WindowSize = 16;
```

协议变为停等协议

对于实验设计四，为了模拟 ACK 丢失，我采用的是模 N 运算，每 N 次模拟丢包，或者每 N 次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，如果在发送端，则少发送数据包，在接收端则不发回 ACK，我选择在接收端实现，也就是每隔 N（我设置 N=100）就不返回 ACK，这部分具体实现代码如下：

```
26 //按照模N运算模拟丢包，N即Lossgap
27 private final int Lossgap = 100;

166 //当模N为0时，人为产生一个丢包
167 if (count % Lossgap != 0) {
168     long seq = recv[0] & 0xFF;
169     if (seq == nowseq) {
170         //accept
171         nowseq++;
172         if (nowseq >= 256) nowseq = 0;
173         //写在上write to upper
174         result.write(recv, 1, receivePacket.getLength()-1);
175         //发送ACK send ACK
176         recv = new byte[1];
177         recv[0] = new Long(seq).byteValue();
178         receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);
179         datagramSocket.send(receivePacket);
180         System.out.println("receive datagram seq: " + seq);
181         datagramSocket.setSoTimeout(endTime*1000);
182     } else discard
183 }
```

对于实验设计五，更改窗口大小为 1，则 GBN 协议变为停等协议

```
20 //若此处改窗口大小为1，则为停等协议
21 private int WindowSize = 16;
```

实际测试过程如下：

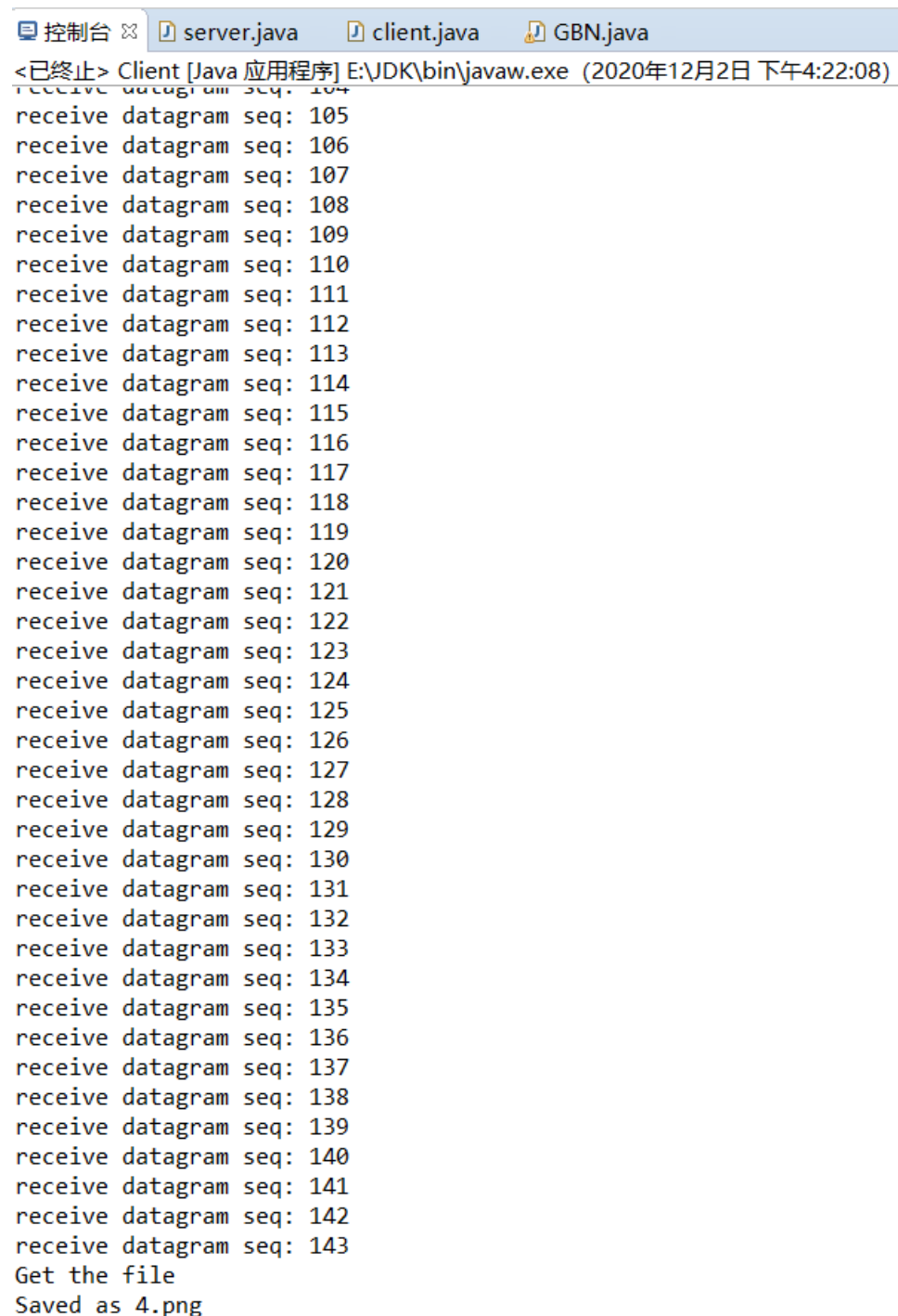
首先运行 server.java，此时服务器已经做好从客户端 8080 接收文件 1.png 的准备，需要运行 client 建立连接。

```
控制台  server.java  client.java  GBN.java
Server [Java 应用程序] E:\JDK\bin\javaw.exe (2020年12月2日 下午4:19:35)
Start to receive file 1.png from localhost 8080
```

然后我们运行 client.java，连接建立，此时客户端开始向服务器端 7070 传输文件 1.png。

```
Client [Java 应用程序] E:\JDK\bin\javaw.exe (2020年12月2日 下午4:20:34)
Start to send file 1.png to localhost 7070
send the datagram seq: 0
send the datagram seq: 1
send the datagram seq: 2
send the datagram seq: 3
send the datagram seq: 4
send the datagram seq: 5
send the datagram seq: 6
```

待到运行完成，我们可以得到如下结果



```
<已终止> Client [Java 应用程序] E:\JDK\bin\javaw.exe (2020年12月2日 下午4:22:08)
receive datagram seq: 104
receive datagram seq: 105
receive datagram seq: 106
receive datagram seq: 107
receive datagram seq: 108
receive datagram seq: 109
receive datagram seq: 110
receive datagram seq: 111
receive datagram seq: 112
receive datagram seq: 113
receive datagram seq: 114
receive datagram seq: 115
receive datagram seq: 116
receive datagram seq: 117
receive datagram seq: 118
receive datagram seq: 119
receive datagram seq: 120
receive datagram seq: 121
receive datagram seq: 122
receive datagram seq: 123
receive datagram seq: 124
receive datagram seq: 125
receive datagram seq: 126
receive datagram seq: 127
receive datagram seq: 128
receive datagram seq: 129
receive datagram seq: 130
receive datagram seq: 131
receive datagram seq: 132
receive datagram seq: 133
receive datagram seq: 134
receive datagram seq: 135
receive datagram seq: 136
receive datagram seq: 137
receive datagram seq: 138
receive datagram seq: 139
receive datagram seq: 140
receive datagram seq: 141
receive datagram seq: 142
receive datagram seq: 143
Get the file
Saved as 4.png
```

现在我们对其中的细节进行分析：

根据我们的模 100 接受方丢 ACK 包的设计，我们在传输过程中检查，果然在第 99 个序列号出发生了丢包（0-99 刚好是 100 个），我们设计是假设接收方正在等待接收分组 n ，而分组 $n+1$ 却已经到达了，于是，分组 $n+1$ 被直接丢弃，所以发送方并不会出现在连续发送分组 n 、分组 $n+1$ 之后，而分组 $n+1$ 的 ACK 却比分组 n 的 ACK 更早到达发送方的情况。所以在 `sendSeq` 之后的包括第 99 数据包在

内的连续 16 个（滑动窗口大小）的数据包都视作没有收到 ACK 而重传，情况如下：

```
send the datagram seq: 114
resend the datagram's seq: 99
resend the datagram's seq: 100
resend the datagram's seq: 101
resend the datagram's seq: 102
resend the datagram's seq: 103
resend the datagram's seq: 104
resend the datagram's seq: 105
resend the datagram's seq: 106
resend the datagram's seq: 107
resend the datagram's seq: 108
resend the datagram's seq: 109
resend the datagram's seq: 110
resend the datagram's seq: 111
resend the datagram's seq: 112
resend the datagram's seq: 113
resend the datagram's seq: 114
send the datagram seq: 115
```

后面的丢包情况也都一样，情况如下面两张图所示，丢包的数量都是 16

GBN 中如果发送方的滑动窗口中，如果窗口内已经被发送但未收到确认的分组数目未达到窗口长度,就将窗口剩余的分组全部用来发送新构造好的数据,剩余未能发送的数据进行缓存。发送完窗口大小的数据分组后,开始等待接收从接收方发来的确定信息(ACK),GBN 协议采取了累积确认，当发送方收到一个对分组 n 的 ACK 的时候,即表明接收方对于分组 n 以及分组 n 之前的分组全部都收到了。对于已经确认的分组,就将窗口滑动到未确认的分组位置(窗口又有空闲位置,可以发送剩余分组了),对于未确认的分组,如果计时器超时,就需要重新发送,直到收

```
send the datagram seq: 198
resend the datagram's seq: 183
resend the datagram's seq: 184
send the datagram seq: 199
send the datagram seq: 200
resend the datagram's seq: 185
resend the datagram's seq: 186
resend the datagram's seq: 187
resend the datagram's seq: 188
resend the datagram's seq: 189
resend the datagram's seq: 190
resend the datagram's seq: 191
resend the datagram's seq: 192
resend the datagram's seq: 193
resend the datagram's seq: 194
resend the datagram's seq: 195
resend the datagram's seq: 196
resend the datagram's seq: 197
resend the datagram's seq: 198
send the datagram seq: 201
```


到接收方的 ACK 为止。对于超时的触发,GBN 协议会将当前所有已发送但未被确认的分组重传,即如果当前窗口内都是已发送但未被确认的分组,一旦定时器发现窗口内的第一个分组超时,则窗口内所有分组都要被重传。每次当发送方收到一个 ACK 的时候,定时器都会被重置。:

```
send the datagram seq: 26
resend the datagram's seq: 11
resend the datagram's seq: 12
send the datagram seq: 27
send the datagram seq: 28
send the datagram seq: 29
send the datagram seq: 30
send the datagram seq: 31
send the datagram seq: 32
send the datagram seq: 33
send the datagram seq: 34
send the datagram seq: 35
send the datagram seq: 36
send the datagram seq: 37
send the datagram seq: 38
send the datagram seq: 39
send the datagram seq: 40
send the datagram seq: 41
send the datagram seq: 42
send the datagram seq: 43
send the datagram seq: 44
send the datagram seq: 45
send the datagram seq: 46
send the datagram seq: 47
send the datagram seq: 48
```

```
Start to receive file 3.png from localhost 7070
```

```
resend the datagram's seq: 13
resend the datagram's seq: 14
resend the datagram's seq: 15
resend the datagram's seq: 16
resend the datagram's seq: 17
resend the datagram's seq: 18
resend the datagram's seq: 19
resend the datagram's seq: 20
resend the datagram's seq: 21
resend the datagram's seq: 22
resend the datagram's seq: 23
resend the datagram's seq: 24
resend the datagram's seq: 25
resend the datagram's seq: 26
receive datagram seq: 0
```

作为双向传输, client 还完成了从服务器端 7070 接收文件 3.png 的任务, 如下

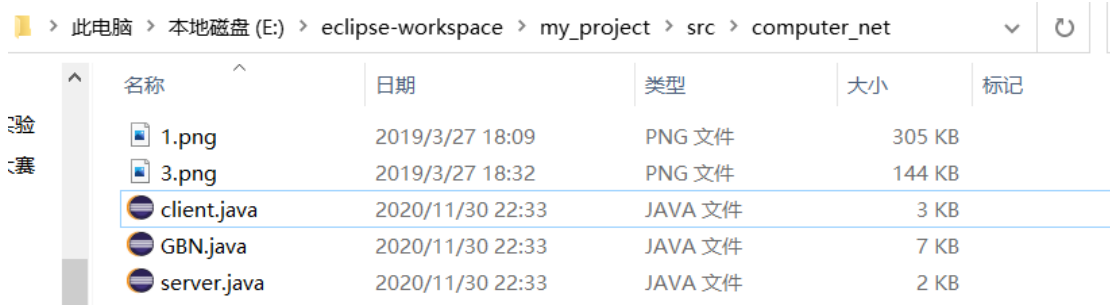
```
Start to receive file 3.png from localhost 7070
```

并且成功将其存储为 4.png, 如下

Get the file
Saved as 4.png

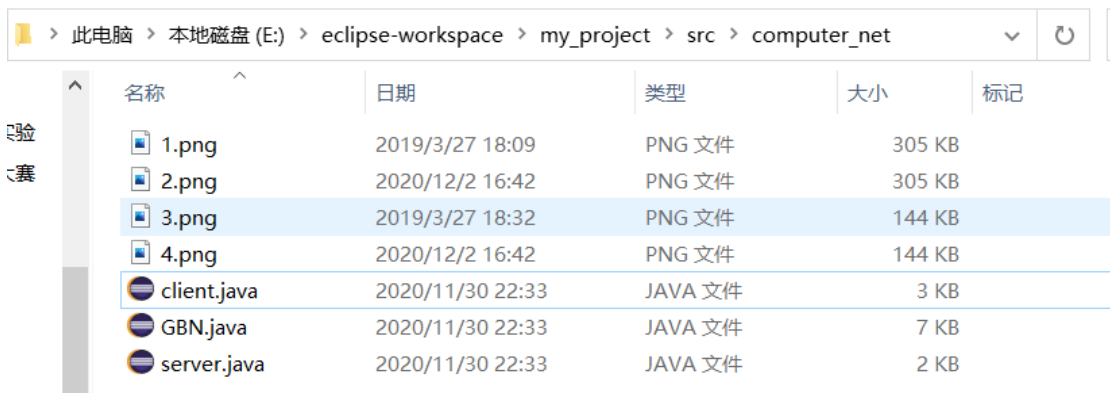
需要注意的是,服务器端 7070 从客户端 8080 接收到的文件 1.png 被存储为 2.png。

文件夹初始情况如下所示



名称	日期	类型	大小	标记
1.png	2019/3/27 18:09	PNG 文件	305 KB	
3.png	2019/3/27 18:32	PNG 文件	144 KB	
client.java	2020/11/30 22:33	JAVA 文件	3 KB	
GBN.java	2020/11/30 22:33	JAVA 文件	7 KB	
server.java	2020/11/30 22:33	JAVA 文件	2 KB	

最后运行结束时, 文件夹里情况如下所示



名称	日期	类型	大小	标记
1.png	2019/3/27 18:09	PNG 文件	305 KB	
2.png	2020/12/2 16:42	PNG 文件	305 KB	
3.png	2019/3/27 18:32	PNG 文件	144 KB	
4.png	2020/12/2 16:42	PNG 文件	144 KB	
client.java	2020/11/30 22:33	JAVA 文件	3 KB	
GBN.java	2020/11/30 22:33	JAVA 文件	7 KB	
server.java	2020/11/30 22:33	JAVA 文件	2 KB	

四、关键问题及其解决方法

本次实验总共耗时 20 小时左右, 在实验的过程中, 遇到了大大小小的问题, 现在将部分难解决的问题总结如下:

1) 新建文件路径问题

在实验一开始, 由于上次写 java 实现的 socket 还是在大二的时候, 所以对是否需要建立 file 不太记得, 导致一开始写的包含下面被注释掉的部分, 于是运行的

```
11         File file1 = new File("./src/computer_net/2.png");//接受文件目录
12         File file2 = new File("./src/computer_net/3.png");//传送文件目录
13         //产生file
14         if (!file1.exists()) {
15             file1.createNewFile();
16         }
17         //其实读取要传送的文件是不需要createNewFile()的
18         /*if (!file2.exists()) {
19             file2.createNewFile();
20         }*/
```

时候总是会发生如下的情况

```
控制台  server.java  client.java  GBN.java
Client [Java 应用程序] E:\JDK\bin\javaw.exe (2020年12月2日 下午5:49:01)
Start to send file 1.png to localhost 7070
send the datagram seq: 0
send the datagram seq: 1
send the datagram seq: 2
send the datagram seq: 3
send the datagram seq: 4
send the datagram seq: 5
send the datagram seq: 6
send the datagram seq: 7
send the datagram seq: 8
send the datagram seq: 9
send the datagram seq: 10
send the datagram seq: 11
send the datagram seq: 12
send the datagram seq: 13
send the datagram seq: 14
send the datagram seq: 15
resend the datagram's seq: 0
resend the datagram's seq: 1
resend the datagram's seq: 2
resend the datagram's seq: 3
resend the datagram's seq: 4
resend the datagram's seq: 5
resend the datagram's seq: 6
resend the datagram's seq: 7
resend the datagram's seq: 8
resend the datagram's seq: 9
resend the datagram's seq: 10
resend the datagram's seq: 11
resend the datagram's seq: 12
resend the datagram's seq: 13
resend the datagram's seq: 14
resend the datagram's seq: 15
resend the datagram's seq: 0
resend the datagram's seq: 1
resend the datagram's seq: 2
resend the datagram's seq: 3
resend the datagram's seq: 4
resend the datagram's seq: 5
resend the datagram's seq: 6
resend the datagram's seq: 7
resend the datagram's seq: 8
```

也就是对 16 个初始数据包一直重传，自然接收方得到的文件 2.png 和 4.png 也都是 0kb 如下，原因是读取要传送的文件是不需要 `createNewFile()` 的，因为其已经建立的，但是语新建了路径，最终导致发送方发不出去，接收方文件一直收不到，最后发送方就一直重传。

此电脑 > 本地磁盘 (E:) > eclipse-workspace > my_project > src > computer_net					
名称	日期	类型	大小	标记	
1.png	2019/3/27 18:09	PNG 文件	305 KB		
2.png	2020/12/2 17:48	PNG 文件	0 KB		
3.png	2019/3/27 18:32	PNG 文件	144 KB		
4.png	2020/12/2 17:48	PNG 文件	0 KB		
client.java	2020/11/30 22:33	JAVA 文件	3 KB		
GBN.java	2020/11/30 22:33	JAVA 文件	7 KB		
server.java	2020/11/30 22:33	JAVA 文件	2 KB		

2) 命名规范的重要性

```

55         while ((length = InputStream.read(buffer)) != -1) {
56             CloneResult.write(buffer, 0, length);
57         }

```

一直在上面的地方报错，错误说是不能对 `InputStream` 里面的非静态方法 `read` 进行动态引用，检查了一天，查询了无数资料，最后甚至都尝试将 `read` 方法进行重写，但是发现违反了 `Java` 的规定，于是解决不了。

昨天 - 2020年12月1日星期二					
<input type="checkbox"/>	下午11:30		不能对类型Users中的非静态方法getUserName()进行静态调用? 怎么解决_百度知道	zhidao.baidu.com	⋮
<input type="checkbox"/>	下午11:29		(1条消息) 不能对非静态字段 / 方法进行静态引用_WenDong1997的博客-CSDN博客_不能对非静态方法进...	blog.csdn.net	⋮
<input type="checkbox"/>	下午11:29		(1条消息) 静态方法访问非静态方法引发的错误: 不能对类型 XXX中的非静态方法 xxx () 进行静态引用...	blog.csdn.net	⋮
<input type="checkbox"/>	下午11:29		不能对类型中的非静态方法进行静态引用_百度搜索	www.baidu.com	⋮
<input type="checkbox"/>	下午11:29		(1条消息) 踩过的坑系列之InputStream.read(byte[])方法_weixin_34411563的博客-CSDN博客	blog.csdn.net	⋮
<input type="checkbox"/>	下午11:29		不能对类型 InputStream 中的非静态方法 read(byte[])进行静态引用_百度搜索	www.baidu.com	⋮
<input type="checkbox"/>	下午11:27		慎用InputStream的read()方法 - Iteye博客	www.iteye.com	⋮
<input type="checkbox"/>	下午11:26		(1条消息) InputStream read () 方法详解_u010276761的博客-CSDN博客	blog.csdn.net	⋮
<input type="checkbox"/>	下午11:26		不能对类型 InputStream 中的非静态方法 read(byte[])进行静态引用_百度搜索	www.baidu.com	⋮
<input type="checkbox"/>	下午11:26		InputStream.read_百度搜索	www.baidu.com	⋮
<input type="checkbox"/>	下午11:25		(1条消息) Java面试之静态方法能否被重写_Machine4869的博客-CSDN博客	blog.csdn.net	⋮
<input type="checkbox"/>	下午11:25		面试题: Java静态/非静态方法重写 - 工程师二号 - 博客园	www.cnblogs.com	⋮
<input type="checkbox"/>	下午11:25		能够将非静态的方法重写为静态方法吗?_百度知道	zhidao.baidu.com	⋮
<input type="checkbox"/>	下午11:24		如何将库里的非静态方法重写为静态方法_百度搜索	www.baidu.com	⋮

最后发现是对一个变量命名时命名了大写，但是自己按照习惯在后面使用的时候使用的是自己习惯的小写，导致最后不统一导致的错误，这种细小的错误往往最难检查到，报错也会五花八门很难找到原因，也最耽误时间，再一次说明了我们在编写代码时需要注意细节。

3) 根据提示导入库的时候需要注意

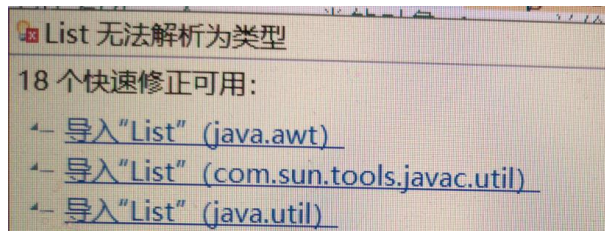
问题如下图，

```
3 //import java.awt.List; //导错包导致38行一直报错:类型 List 不是通用的; 不能使用参数 <ByteArrayOutputStream> 将它参数化
4= import java.util.List; //这才是正确的应该导入的包
```

在代码这部分使用了 List 数据结构

```
48 //滑动窗口缓存, 采用链表的数据结构, 因为数据包需要按序标识并存下, 这样才能实现重传
49 List<ByteArrayOutputStream> datagramBuffer = new LinkedList<>();
```

于是需要导入相应的包, 提示如下



我刚开始没看仔细, 直接选择 import java.awt.List, 结果就是一直报错:

类型 List 不是通用的; 不能使用参数 <String> 将它参数化。

最后检查的时候才发现这个错误是 import 导入错的 List 导致的。

Java.awt.List 是一个界面控件, 是重量级系统列表控件;

Java.util.List 是一种数据容器, 是列表模式的数据容器;

两者区别很大, 不能混用。

4) 参数缺失报错

```
46 //用于UDP类DatagramSocket, 表示接受或发送数据报的套接字, 此处先建立套接字<DatagramSocket>
47 DatagramSocket datagramSocket = new DatagramSocket(ownPort);
```

刚开始没注意, 没有 ownPort, 导致服务器端和客户端都连接不到对方, 一直超时重传错误, 改正加入参数后就行了。

剩下的错误基本上都是对一些现有函数的调用不明确导致的错误, 都没花费太多时间就解决了

五、附录（详细注释的源码）

server.java

```
package computer_net;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
```

```

import java.io.FileOutputStream;
import java.io.IOException;

public class server{
    public static void main(String[] args) throws IOException,
    InterruptedException {
        File file1 = new File("./src/computer_net/2.png");//

```

接受文件目录

```

        File file2 = new File("./src/computer_net/3.png");//

```

传送文件目录

```

        //产生file

```

```

        if (!file1.exists()) {
            file1.createNewFile();
        }

```

//其实读取要传送的文件是不需要createNewFile()的

```

        /*if (!file2.exists()) {
            file2.createNewFile();
        }*/

```

//GBN协议的服务器

```

        GBN server = new GBN("localhost", 8080, 7070);//目标

```

客户端端口为7070，本地服务器端口为8080

```

        System.out.println("Start to receive file 1.png from
        "+"localhost "+ 8080);

```

```

        while (true) {
            ByteArrayOutputStream byteArrayOutputStream =
server.receive();
            if (byteArrayOutputStream.size() != 0) {
                FileOutputStream fileOutputStream = new
FileOutputStream(file1);

```

```

                fileOutputStream.write(byteArrayOutputStream.toByteArray(),
                0, byteArrayOutputStream.size());
                fileOutputStream.close();
                System.out.println("Get the file ");
                System.out.println("Saved as 2.png");
                fileOutputStream.close();
                break;

```

```

        }
        Thread.sleep(50);
    }

    //捕获内存缓冲区的数据，转换成字节数组。

    ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
    client.CloneStream(byteArrayOutputStream, new
FileInputStream(file2));
    System.out.println("\nStart to send file 3.png to"
+ "localhost" + 8080);
    server.send(byteArrayOutputStream.toByteArray());
    }
}

```

client.java

```

package computer_net;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class client {
    public static void main(String[] args) throws IOException,
InterruptedException {
        File file1 = new File("./src/computer_net/1.png");//传送

```

文件目录

```

        File file2 = new File("./src/computer_net/4.png");//接受

```

文件目录

```

    //其实读取要传送的文件是不需要createNewFile()的
    /*if(!file1.exists()) {
        file1.createNewFile();
    }*/
    if(!file2.exists()) {

```

```

        file2.createNewFile();
    }
    ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
    CloneStream(byteArrayOutputStream, new
FileInputStream(file1));

    //协议的选择

    GBN client = new GBN("localhost",7070,8080);//目标服务器

    端口8080, 本地客户端端口7070

    System.out.println("Start to send file 1.png to " +
"localhost " + 7070);
    client.send(byteArrayOutputStream.toByteArray());
    System.out.println("\nStart to receive file 3.png from "
+ "localhost " + 7070);
    while (true) {
        byteArrayOutputStream = client.receive();
        if (byteArrayOutputStream.size() != 0) {
            FileOutputStream fileOutputStream = new
FileOutputStream(file2);

            fileOutputStream.write(byteArrayOutputStream.toByteArray(),
0, byteArrayOutputStream.size());
            fileOutputStream.close();
            System.out.println("Get the file ");
            System.out.println("Saved as 4.png");
            break;
        }

        Thread.sleep(50);
    }
}

/**
 * clone the input stream to a ByteArrayOutputStream
object
 *
 * @param CloneResult the clone result of input stream
 * @param InputStream the input stream to be cloned
 * @throws IOException when read input stream, some
exception occur
 */

```



```

        //将输入流转为字节数组对象

        static void CloneStream(ByteArrayOutputStream
CloneResult, InputStream InputStream) throws IOException {
    //InputStream InputStream
    byte[] buffer = new byte[1024];
    int length;

    while ((length = InputStream.read(buffer)) != -1) {///  

        CloneResult.write(buffer, 0, length);
    }
    CloneResult.flush();
}
}

```

GBN.java

```

package computer_net;

//import java.awt.List; //导错包导致38行一直报错:类型 List 不是通用
//的; 不能使用参数 <ByteArrayOutputStream> 将它参数化

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.*;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketTimeoutException;
import java.net.UnknownHostException;
import java.util.List;
import java.util.LinkedList;

class GBN {
    //获取到本机的InetAddress

    private InetAddress host;

    //目标端口和本机端口

    private int targetPort, ownPort;

    //若此处改窗口大小为1, 则为停等协议

```

```

    private int WindowSize = 16;

    //最大发送时间，最大接收时间，一个数据包的超时时间

    private final int sendMaxTime = 2, receiveMaxTime =
4, endTime=5;

    //起始base值

    private long base = 0;

    //按照模N运算模拟丢包，N即Lossgap

    private final int Lossgap = 100;

    //对GBN中成员变量进行赋值A

    GBN(String host, int targetPort, int ownPort) throws
UnknownHostException {

        //this关键字调用类中成员变量ownPort

        this.ownPort = ownPort;

        //确定主机的IP地址

        this.host = InetAddress.getByName(host);

        //this关键字调用类中成员变量targetPort

        this.targetPort = targetPort;

    }
/**
 *向主机目的地址处传输数据相关功能
 */
//send功能
void send(byte[] content) throws IOException {
    //
    int sendIndex = 0, length;

    //自行设计数据包最大字节数

    final int MAX_LENGTH = 1024;

    //用于UDP类DatagramSocket，表示接受或发送数据报的套接字，此处先
建立套接字<DatagramSocket>

    DatagramSocket datagramSocket = new
DatagramSocket(ownPort);

    //滑动窗口缓存，采用链表的数据结构，因为数据包需要按序标识并存

```

下，这样才能实现重传

```
List<ByteArrayOutputStream> datagramBuffer = new
LinkedList<>();

//实例化一个Integer类的对象timer，并给该对象赋值为0（计时器）
Integer timer = 0;

//自行设计序列号
long sendSeq = base;
do {
    //直到窗口开始滑动
    while (sendIndex < content.length && sendSeq < 256 &&
datagramBuffer.size()<WindowSize) {
        //向滑窗缓存中加入byte型数据
        datagramBuffer.add(new ByteArrayOutputStream());
        //若数据部分字节数未超过限定的最大值，则不变；若超过，就取
限定的最大值
        length = content.length - sendIndex < MAX_LENGTH ?
content.length - sendIndex : MAX_LENGTH;
        //构造数据帧
        ByteArrayOutputStream oneSend = new
ByteArrayOutputStream();
        //初始化byte数组
        byte[] temp = new byte[1];
        //
        temp[0] = new Long(sendSeq).byteValue();
        //从temp中向oneSend写入从0开始处一字节的数据
        oneSend.write(temp, 0, 1);
        //从content中向oneSend写入从sendIndex开始处length字节的
数据
        oneSend.write(content, sendIndex, length);
        //DatagramPacket表示存放数据的数据报，此句表示向host地址
targetPort端口处发送oneSend.size()长度的oneSend.toByteArray()
```

```

        DatagramPacket datagramPacket = new
DatagramPacket(oneSend.toByteArray(), oneSend.size(), host,
targetPort);

        //从datagramSocket发送datagramPacket
        datagramSocket.send(datagramPacket);

        //向滑动缓存中sendSeq-base的位置写入content中从
sendIndex开始处length字节的数据

        datagramBuffer.get((int) (sendSeq -
base)).write(content, sendIndex, length);

        //后移length，循环用
        sendIndex += length;

        //“输出的数据包序号为sendSeq”
        System.out.println("send the datagram seq: "
+sendSeq);

        //序号+1，循环用
        sendSeq++;
    }

    //倒计时函数实现1秒倒计时
    datagramSocket.setSoTimeout(1000);

    //创建一个接受ACK的数据包
    DatagramPacket receivePacket;

    try{//从base开始接收ACK receive ACKs for base
        while (true) {
            //1500字节的recv字节数组
            byte[] recv = new byte[1500];

            //存放recv.length长的recv的数据报
            receivePacket = new DatagramPacket(recv,
recv.length);

            //套接字开始接收数据包，当接收到时，receivePacket填满
为接收到的数据

            datagramSocket.receive(receivePacket);

```

```

        //与11111111按位与，结果化为int型赋予ack，这一步是为
        了实现接收方按序接收数组

        int ack = (int) ((recv[0] & 0xFF));
        //如果接收到了第一个，就继续接收第二个，并顺便清空缓存

        if (ack == base) {
            base++;
            datagramBuffer.remove(0);
            break;
        }
    }

    //超时功能
} catch (SocketTimeoutException e) {
    timer++;
}

//如果超出设定的超时时间
if (timer > this.sendMaxTime) {
    // 重发所有没有收到ACK（超时）的数据报

    for (int i = 0; i < datagramBuffer.size(); i++) {
        ByteArrayOutputStream resender = new
ByteArrayOutputStream();
        byte[] temp = new byte[1];
        temp[0] = new Long(i + base).byteValue();
        resender.write(temp, 0, 1);

        resender.write(datagramBuffer.get(i).toByteArray(), 0,
datagramBuffer.get(i).size());
        DatagramPacket datagramPacket = new
DatagramPacket(resender.toByteArray(), resender.size(), host,
targetPort);
        datagramSocket.send(datagramPacket);
        System.err.println("resend the datagram's seq:
"+ (i + base));
    }

    //重置timer

    timer = 0;

    //base每到256，就将base和sendSeq重置

```

```

        if (base >= 256) {
            base = base - 256;
            sendSeq = sendSeq - 256;
        }
    }

    //直到数据全部传输完成

    }while (sendIndex < content.length ||
datagramBuffer.size() !=0) ;

    //关闭套接字

    datagramSocket.close();
}

/**
 * 接收主机目的地址处的数据相关功能
 */
//receive
ByteArrayOutputStream receive() throws IOException {
    //用于 used to simulate datagram loss

    int count = 1,time=0;
    long nowseq = 0;

    //存储接收的 store the received content

    ByteArrayOutputStream result = new
ByteArrayOutputStream();

    //接收数据报和发送ACK的UDP套接字  UDP socket to receive
datagram and send ACKs
    DatagramSocket datagramSocket = new
DatagramSocket(ownPort);

    //一个暂时的数据报 one temp datagram packet

    DatagramPacket receivePacket;
    //
    datagramSocket.setSoTimeout(endTime*1000);
    while (true) {

        //接收一个数据报并且返回ACK

        try {
            byte[] recv = new byte[1500];
            receivePacket = new DatagramPacket(recv,

```

```

recv.length, host, targetPort);
    datagramSocket.receive(receivePacket);

    //当模N为0时, 人为产生一个丢包

    if (count % Lossgap != 0) {
        long seq = recv[0] & 0xFF;
        if (seq == nowseq) {
            //accept
            nowseq++;
            if (nowseq>=256)nowseq=0;

            //写在上write to upper

            result.write(recv, 1,
recv.length, host, targetPort);
            datagramSocket.send(receivePacket);
            System.out.println("receive datagram seq: "
+ seq);

            datagramSocket.setSoTimeout(endTime*1000);
        }//else discard
        }
    }catch (SocketTimeoutException e) {
        break;
    }
    //datagramSocket.setSoTimeout(endTime);
    count++;
}
datagramSocket.close();
return result;
}
}

```