

测试文档

16340189 邵梓硕 16340190 沈大伟

写在前面：

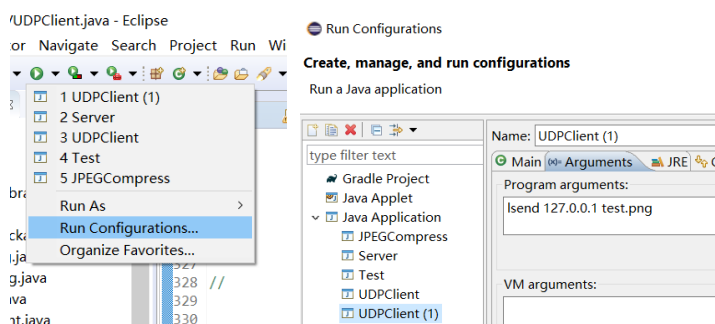
默认客户端的文件存储目录为 `d://fly/java2/`，不更改源码的话要保证这个目录存在，所有要上传的文件都要在这个目录下，下载的文件会自动存储到这个目录。

默认服务端的文件存储目录为 `d://fly/java/`，不更改源码的话要保证这个目录存在，所有客户端上传的文件都会保存在这个目录下，也只有在这个目录下的文件才能供客户端下载。

测试方法：

1. Eclipse 本地测试

Eclipse 可以进行本地测试，在运行客户端的 `UDPClient` 主类时，需要的参数可以在下图中手动输入：



本地测试时的丢包率小，可以用来测试传输与接受、文件的拼接、文件的删除、多用户文件分类等基本功能，并且在本地运行的结果的基础上可以来分析公共网络中丢包带来的影响。

2. Exe 程序测试

利用 `exe4j` 软件可以将 `java` 项目封装为 `exe` 可运行程序，本实验的所有公网测试均通过 `exe` 程序运行。客户端程序为 `lftp.exe`，服务端程序为 `lftp-server.exe`，客户端需要输入 3 个参数，服务端则可以直接运行，下面的大部分截图均是通过 `lftp` 程序运行，具体可以见下面的实际测试。

测试内容：

1. 异常检查(都会显示在客户端)

(1) 客户端欲上传服务端已经存在的文件：

```
D:\eclipse\eclipse\workspace2\runable\lftp>lftp lsend 139.199.83.68 picture.png
Beginning conneting to the Server!
该文件已存在...
```

(2) 客户端欲下载服务端不存在的文件：

```
D:\eclipse\eclipse\workspace2\runable\lftp>lftp lget 139.199.83.68 picture.jpg
Beginning conneting to the Server! 139.199.83.68
error code: -4
服务器中该文件不存在!!!
```

(3) 客户端欲上传的文件不在默认目录下：

```
D:\eclipse\eclipse\workspace2\runable\lftp>lftp lsend 139.199.83.68 picture.jpg
Beginning conneting to the Server!
Begin uploading the file! d://fly/java2/picture.jpg 11
d:\fly\java2\picture.jpg (系统找不到指定的文件。)
```

(4) 客户端欲下载的文件已经在默认目录下：

```
D:\eclipse\eclipse\workspace2\runable\lftp>lftp lget 139.199.83.68 picture.png
Beginning conneting to the Server! 139.199.83.68
Begin downloading the file!
send
waiting
recId: -2
-3
该文件已存在
```

2. 传输过程的具体分析

在传输过程中会在控制台上出现很多输出，其中有一部分是真正输出了传输信息，也有一部分是为了线程抢占而存在的，他们大都是会大量输出的重复信息，比如：

```
A274 276 20 13 false 276 0
A274 276 20 13 false 276 0
A274 276 20 13 false 276 0
A274 276 20 13 false 276 0
A274 276 20 13 false 276 0
A274 276 20 13 false 276 0
```

是为了保证线程的优先级，以免不被执行，虽然这些信息有一定的意义，但意义不大。

在接收文件方的控制台输出中有各种信息：

```

waiting      等待数据包
recId: 8376   接收到的数据包的组号
map.jpg      本次传递的文件的文件名
1024         数据包中数据的长度
[8376] // lastAckId: 8373   接收窗口/返回的组号
send ack 8373      返回 ack 组号

```

因此我们可以看到各种各样的情景：

(1) 丢包

```

1024
[] // lastAckId: 8373
send ack 8373
waiting
recId: 8376
map.jpg
1024
[8376] // lastAckId: 8373
send ack 8373
waiting
recId: 8377
map.jpg
1024
[8376, 8377] // lastAckId: 8373
send ack 8373

```

上图中服务端一直返回 ack8373，但却收到了 8373 之后的包，说明出现了丢包，在上图这种情况下出现了 8373，8374，8375 的丢包，而接收窗口中缓存了 8376 和 8377 两个包。

(2) 重传

```

recId: 8373
map.jpg
1024
[8376, 8377, 8378, 8379, 8380] // lastAckId: 8374
send ack 8374

```

上图中收到了 8373 的包后，返回的 ackId 变为了 8374，而接收窗口中此时已经积累了 5 个包。这表明了服务端在发送 8380 之后又重传了 8373.。

```

[8376, 8377, 8378, 8379, 8380, 8381] // lastAckId: 8374
send ack 8374
waiting
recId: 8382
map. jpg
1024
[8376, 8377, 8378, 8379, 8380, 8381, 8382] // lastAckId: 8374
send ack 8374
waiting
recId: 8383
map. jpg
1024
[8376, 8377, 8378, 8379, 8380, 8381, 8382, 8383] // lastAckId: 8374
send ack 8374
waiting
recId: 8374
map. jpg
1024
[8376, 8377, 8378, 8379, 8380, 8381, 8382, 8383] // lastAckId: 8375
send ack 8375

```

在上图中客户端连续发送了三次 8374 的 ack，之后服务端就发送了 8374 的包，这是典型的三次重传，但是实际上由于网络环境的复杂，很多情况下会有不止三次的重传，因此在此代码中我们判断的是 $ACKTimes \geq 3$ 而不是恰好等于 3。

(3) 接收窗口的移动

在上面我们看到了如果收到了 ackId 之后的数据包会暂时存储到接收窗口之中，而一旦得到了需要的数据包就会发生接收窗口的移动：

```

waiting
recId: 8374
map. jpg
1024
[8376, 8377, 8378, 8379, 8380, 8381, 8382, 8383] // lastAckId: 8375
send ack 8375
waiting
recId: 8375
map. jpg
1024
[] // lastAckId: 8384
send ack 8384

```

在上图中接收窗口缓存了从 8376 到 8383 的数据包，当收到了 ackId 的 8375 时，接收窗口会一下子释放出缓存的数据包，直接将 ackId 前移到了 8384，此时接收窗口的占用空间为 0。

(4) 等待接收完毕

发送方发送了最后一个数据包后不能直接结束自己的发送线程，因为此时接受端可能

还没有接受全部的数据包，万一出现了数据包的丢失，接收方还需要发送方的重传，因此发送方发送完毕后会一直在发送线程中循环直到接收方 ackId 是最后一个数据包号+1：

```
receiving ack...
continue
continue
continue
continue
continue
continue
continue
continue
continue
continue
continue
continue
continue
continue
continue
the received id is 9214 win: 20 ACKTimes: 0
receiving ack...
the received id is 9215 win: 20 ACKTimes: 0
receiving ack...
the received id is 9216 win: 20 ACKTimes: 0
receiving ack...
continue
break
```





上图中 continue 就代表了发送方在发送完毕之后等待接收方 ack 回最后的组号+1（在这里是 9217），收到后会马上跳出线程，并结束传输。在这之后接收方会进行文件的合并。

3. 接收过程中的多用户分类

由于服务端需要支持多用户，因此再上传文件过程中分类保存他们的文件很重要：

 120.236.174.140	2018/12/4 21:39	文件夹
 120.236.174.150	2018/12/4 21:41	文件夹

在服务端的默认目录下会出现以客户端 ip（这时是经过路由器加密后的 ip）为文件名的文件目录：

 map.jpg	2018/12/4 21:24	文件夹
 map.png	2018/12/4 20:58	文件夹
 test.png	2018/12/4 21:16	文件夹
 user1.jpg	2018/12/4 21:39	文件夹

其中会有以目前正在上传文件的文件名为名的目录，上传的数据包就存储在这些目录下，当传输完成后会自动清除这些文件目录，当用户没有传输文件后，用户的文件目录也会被清楚，所有上传文件会存储在服务端的默认目录下。

4. 传输速率和效果测试

我们用小文件开始，小文件上传计时：

大小: 6.79 KB (6,961 字节)

```
D:\eclipse\eclipse\workspace2\runable\lftp>echo %time%
21:20:37.74

D:\eclipse\eclipse\workspace2\runable\lftp> lftp lsend 139.199.83.68 test.png
all start
文件上传成功!

D:\eclipse\eclipse\workspace2\runable\lftp>
D:\eclipse\eclipse\workspace2\runable\lftp>echo %time%
21:20:42.38
```

开始上传与上传成功的时间差大概（因为是手动输出时间的）是 4s，速率大概为 $6.79 / 4 = 1.7$ (KB/s)。

小文件下载计时：

```
D:\eclipse\eclipse\workspace2\runable\lftp>echo %time%
20:51:44.36

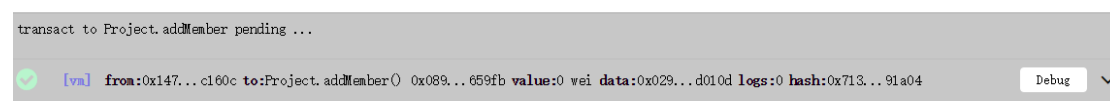
D:\eclipse\eclipse\workspace2\runable\lftp>lftp lget 139.199.83.68 test.png
write end.
文件下载完成!

D:\eclipse\eclipse\workspace2\runable\lftp>echo %time%
20:51:48.78
```

开始下载与下载成功的时间差大概是 4s，速率大概为 1.7KB/s，与上传速率几乎一样。

传输效果：

原图片：



服务端上传的图片：



可以看出是相同的。

大小: 9.00 MB (9,437,238 字节)

大文件上传

大文件上传由于输出很多找不到开始时间的截图，但是当时自己记下了，开始于
23:06:15.10

```
all start  
文件上传成功!  
D:\eclipse\eclipse\workspace2\runable\lftp>echo %time%  
23:08:01.56
```

上传开始和上传成功的时间差大概为 110s，上传速率大概为 $9438 / 110 = 85.8$
(KB/s)

大文件下载由于输出很多找不到开始时间的截图，但是当时自己记下了，开始于
21:04:31.96

```
begin to write  
write end!  
文件下载完成!  
D:\eclipse\eclipse\workspace2\runable\lftp>echo %time%  
21:07:26.85
```

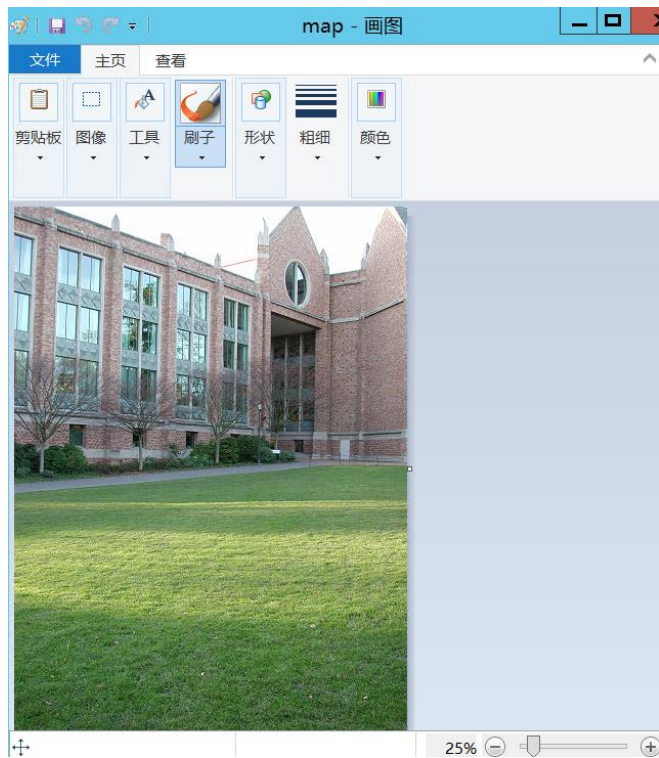
开始下载与下载成功的时间差大概是 180s，下载速率大概为 $9438 / 180 = 52.4$ (KB/s)

传输效果：

原图：

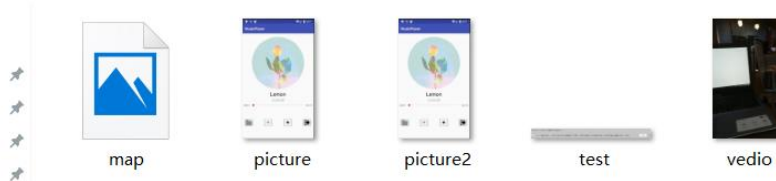


服务端中上传的图：

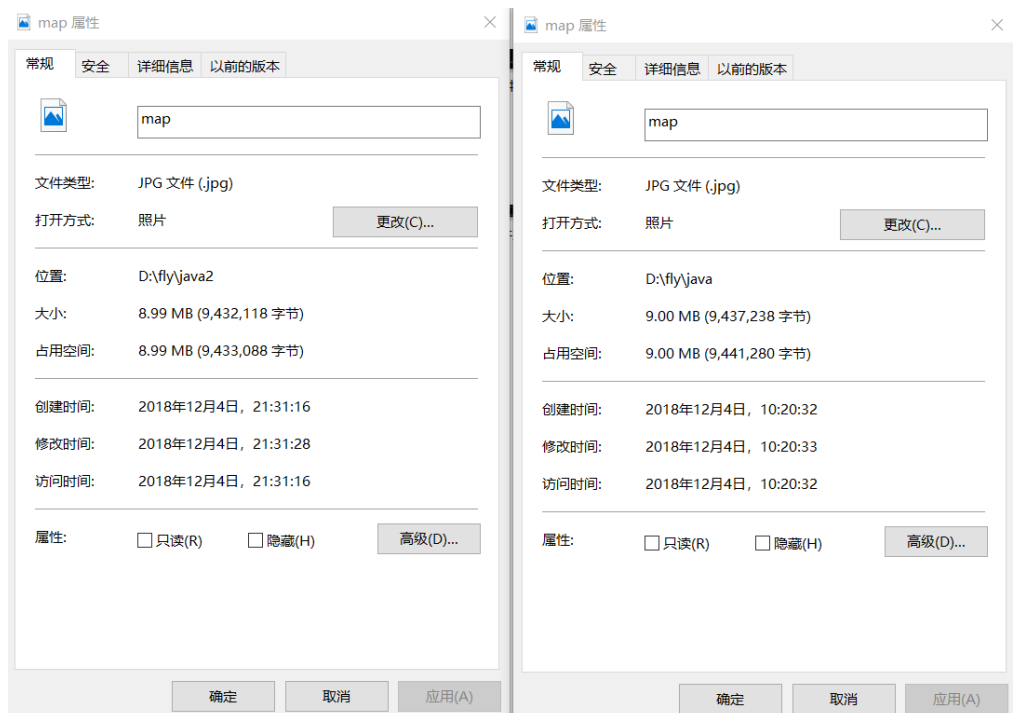


几乎是一样的，那么下载回来的图呢？

此电脑 > DATA (D:) > fly > java2

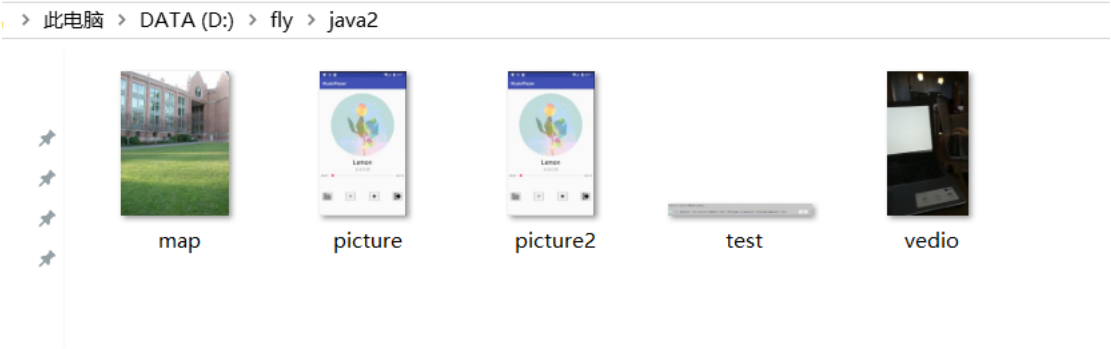


不能显示出 map，看一下下载的属性与原图的属性：



发现数据量小了 1KB 左右，因为传输时按照包的数量进行传输，因此不会出现丢包的情况，因此应该是数据在传输过程发生了丢失，倒是下载的图像数据量变小。但是上传的图片与原图相同，没有发生丢失的情况。

在本地传输相同的文件的结果是什么呢？



可以看到没有发生数据的丢失。因此可以推断是由于网络环境引起的丢失而不是传输算法引起的丢失。

文件大小	上传速率 (KB/s)	下载速率 (KB/s)
小文件	1.7	1.7
大文件	85.5	52.4

传输方式	数据的丢失 (并不是数据包的丢失)
小文件上传	否
小文件下载	否
大文件上传	否
大文件下载	是 (在 1% 的大小内)

总结推论：小文件传输数据量小不易发生丢失，而大文件下载时出由于网络环境引起一定程度是数据的丢失，但是双发在上传时都十分可靠，同时大文件的上传速度明显快于下载速度，这是由于下载速度中包含了文件的拼接于遍历，这一部分花费的时间很长，并且下载过程中需要经过路由表的映射，会有一定的消耗。

5. 多用户操作

现在让两个客户端同时上传图片到服务端：

User1：



```
D:\eclipse\eclipse\workspace2\runable\lftp> lftp lsend 139.199.83.68 user1.jpg
```

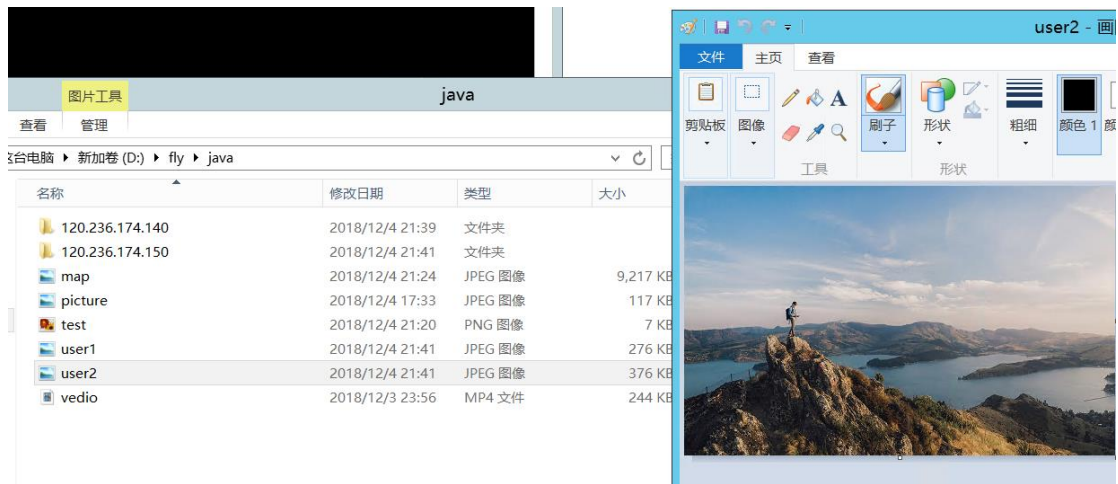
User2 :



```
C:\Users\DELL\Desktop>lftp lsend 139.199.83.68 user2.jpg
```

多用户传输结果（服务端查看）：

名称	修改日期	类型	大小
120.236.174.140	2018/12/4 21:39	文件夹	
120.236.174.150	2018/12/4 21:41	文件夹	
map	2018/12/4 21:24	JPEG 图像	9,217 KB
picture	2018/12/4 17:33	JPEG 图像	117 KB
test	2018/12/4 21:20	PNG 图像	7 KB
user1	2018/12/4 21:41	JPEG 图像	276 KB
user2	2018/12/4 21:41	JPEG 图像	376 KB
vedio	2018/12/3 23:56	MP4 文件	244 KB



可以看到两张图片都与原图相同地上传到了服务端之中，说明服务端可以支持多用户的操作。