

COMP2322 Computer Networking

Individual Project Report

Anthony

Summary

The task is to build a multi-threaded Web Server which is capable of processing multiple simultaneous requests in parallel, equipped with similar function in HTTP proxy server. All the requirements in the project have been well fulfilled, and my project consists of 2 JAVA classes, 1. Server.java; 2. HttpRequest.java.

Multi-threading

Compared with processes, the threads, as light-weight processes, require fewer resources and generate less overhead. When multi-threading is applied to the web server, it can enable minimized system resource usage, improved server responsiveness, and improved throughput.

It can be achieved in two ways in JAVA: 1. Extending the Thread class 2. Implementing the Runnable Interface, while noting that run() method needs to be rewritten. In my project, the HttpRequest.java class extends Thread, and every thread starts, i.e., run() method is invoked by start() in the Server.java class. As shown below, a new HttpRequest class is initialized in Server class, and create a new thread for it, and invoke run() by the start() method.

```
// Construct an object to process the HTTP request message
HttpRequest request=new HttpRequest(connection, i);

// request.run()

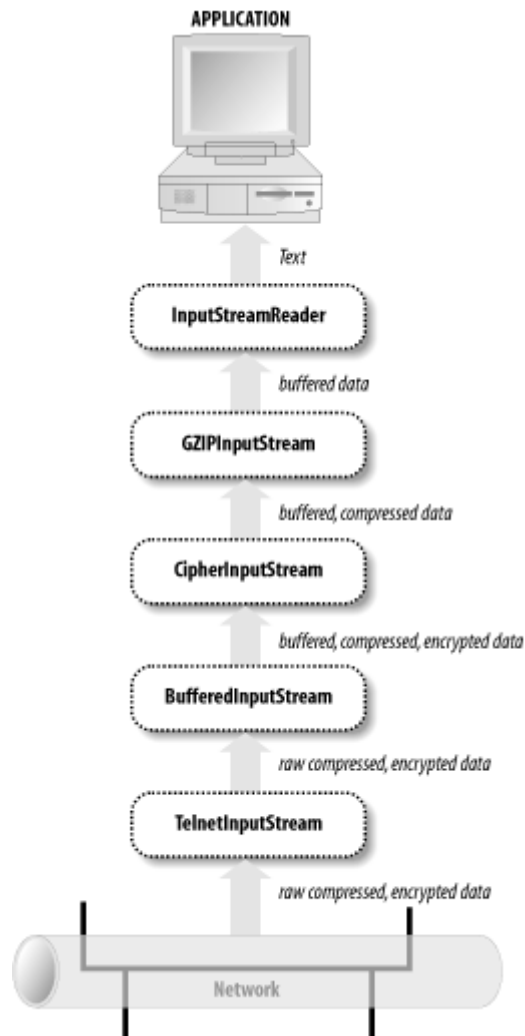
// Create a new thread to process the request
Thread thread=new Thread(request);

// Start the thread
thread.start();
```

It is possible that there may be some unexpected error, thus exception handling in JAVA have been be utilized to handle those unpleasant situations.

Proper request and response message exchanges

The approach to I/O in JAVA is built on the stream, which is a common way for JAVA networking programming.



Some useful variables are used declared in the `HttpRequest` class, as shown below, and they are initialized in the constructor.

```
public class HttpRequest extends Thread{

    // Data stream and output streams for data transfer
    private BufferedReader clientInputStream;
    private DataOutputStream clientOutputStream;

    // Client socket to maintain connection with the client
    private Socket clientSocket;
    private int connectionID;
```

Also, it is of importance to flush the streams before we close them. Failing to flush when you need to can lead to unpredictable, unrepeatable program hangs that are extremely hard to diagnose, as many JAVA programmers argue. Finally, when you're done with a stream, close it by invoking its `close()` method. Given the requirement that Keep-alive needs to be considered, the method is put onto the area when it catches the socket timeout exception.

```
// Copy requested file into the socket's output stream
while((size = fileToClient.read(buffer)) > 0 ) {
    clientOutputStream.write(buffer, off: 0, size);
    clientOutputStream.flush();
}
```

Data can get lost if we don't flush the streams

```
} catch (SocketTimeoutException e){
    System.out.println("Time out");
    clientInputStream.close();
    clientOutputStream.close();
    clientSocket.close();
    System.out.println("Connection Finished");
}
```

Close streams and socket once socket time exception raises.

GET and HEAD commands

For GET request, the program should respond header and body, and for HEAD requests, the program returns header but without body.

My practice is, if GET command, the body will be sent in the response, which is the information in the file, if not, we choose not to respond the body information.

```
// GET request returns header and body
if(requestType.equals("GET")){
    try{
        sendFile(fileToClient);
    }catch (Exception e){
        e.printStackTrace();
    }
}
} // HEAD request returns header but without body
```

If GET, return the body, i.e., the info in the file.

Four types of responses

1. 400 Bad Request.

```
boolean badRequest = false;
```

badRequest is initialized as false.

A Boolean variable named badRequest is initialized as false, then I choose to check by 4 situations. (1) If we can get File name properly; (2) if there are moreTokens; (3) Check if version pattern mismatches; (4) If more tokens.

```
// Check bad request in four aspects
// 1. If we can get File name properly
if(tokens.hasMoreTokens()){
    fileName = tokens.nextToken();
    fileName="."+fileName;
}else{
    badRequest = true;
}

// 2. Http
if(tokens.hasMoreTokens()){
    version = tokens.nextToken();
}else{
    badRequest = true;
}
```

Two of the four cases

2. 304 Not Modified

The time after If-Modified-Since is compared with the file actual modifying time. If they are the same, then we 304 is returned. Otherwise, return 200, with Last-Modified time.

```

if(lastM.compareTo(ifModified) == 0){
    statusLine = "HTTP/1.1 304 Not Modified" + CRLF;
    contentTypeLine = "Last-Modified: " + LastModifiedLine + CRLF;
    modi = true;
    // System.out.println(lastM);
}

```

If two time are the same, return 304.

3. 404 Not Found.

We firstly initialize a Boolean variable as fileExist.

```

boolean fileExists = true;

```

We get the file name after some processing, and use JAVA IO function FileInputStream to get the file. If there is no such file, FileNotFoundException would occur, then we know we cannot find such file.

```

try{
    fileToClient=new FileInputStream(fileName);

}catch(FileNotFoundException e){
    // If the file does not exist, the FileInputStream() constructor will
    // throw FileNotFoundException
    fileExists=false;
}

```

Use JAVA I/O function to check if there is such exception.

4. 200 OK

OK is the situation except for the other 3 situations: right request format, the file exists, and modified after some period.

```

if(fileExists){
    if(ifModified == null){
        statusLine = "HTTP/1.1 200 OK" + CRLF;
        contentTypeLine="Content Type: " + contentType(fileName) + CRLF
        + "Last-Modified: " + LastModifiedLine + CRLF;
    }
}

```

Handle Last-Modified and If-Modified-Since header fields.

As put before, we firstly get relevant string in the request.

```
if(headerLine.startsWith("If-Modified-Since: ")){
    ifModified = parseDate(headerLine.substring(19));
```

Get the Substring from the request line.

Then, we need to convert the string into Date type to facilitate future comparison.

```
File file = new File(fileName);
SimpleDateFormat sdf = new SimpleDateFormat( pattern: "EEE, dd MMM yyyy HH:mm:ss zzz");
String lastModifiedLine = "";
Date lastM = null;

try{
    lastModifiedLine = sdf.format(file.lastModified());
    lastM=parseDate(lastModifiedLine);
```

Convert String into Date type.

```
if(lastM.compareTo(ifModified) == 0){
    statusLine = "HTTP/1.1 304 Not Modified" + CRLF;
    contentTypeLine = "Last-Modified: " + lastModifiedLine + CRLF;
    modi = true;
    // System.out.println(lastM);
```

If two time are same, then it returns 304 without the file body.

Handle Connection: Keep-Alive header field.

I implement it by utilizing the functions in JAVA. I set the time-out time as 30000. After finishing a data transfer, the timer starts. During 30000 milliseconds, the connection would be kept alive.

After that period, the timeout exception occurs, and then the connection is closed by closing relevant streams and socket, with a string printed out indicating a timeout.





```
try{

    clientSocket.setKeepAlive(true);
    do{
        this.processRequest();
        clientSocket.setSoTimeout(30000);

    } while(clientSocket.getKeepAlive());

} catch (SocketTimeoutException e){
    System.out.println("Time out");
    clientInputStream.close();
    clientOutputStream.close();
    clientSocket.close();
    System.out.println("Connection Finished");
}
```

Demonstration by running program with screenshots.

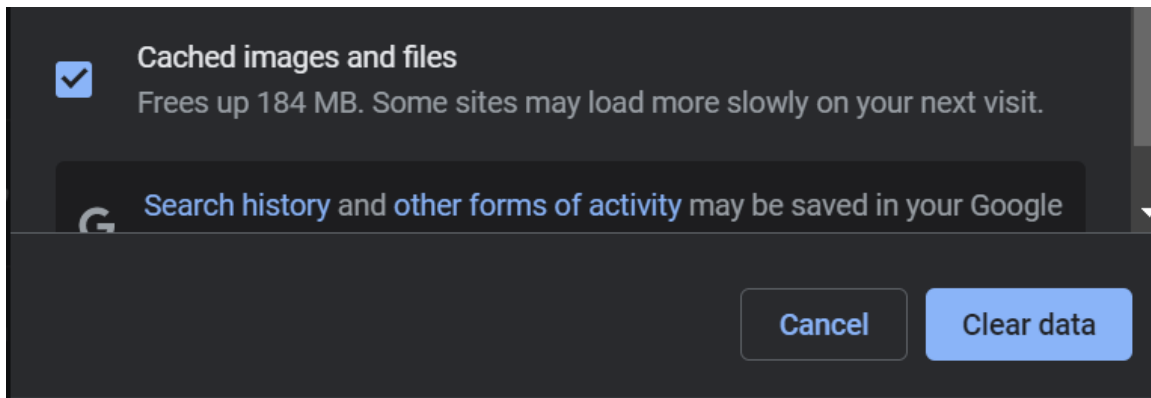
名称	修改日期	类型
 .idea	20/4/2022 1:55 pm	文件夹
 out	17/4/2022 11:53 am	文件夹
 src	20/4/2022 8:31 pm	文件夹
 abc.txt	19/4/2022 7:40 pm	文本文档
 bear.jpg	17/4/2022 5:53 pm	JPG 文件

I put two files in the relative path, one is txt, the other is bear.jpg.

Start running the program.

1. When we haven't input any request in the browser, the waiting sign shows in the terminal, with the port number.

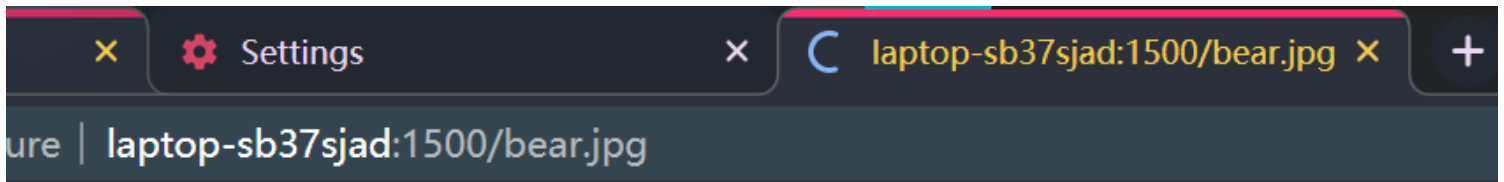
```
D:\Java\bin\java.exe "-javaagent:D:\IntelliJ IDE
Picked up JAVA_TOOL_OPTIONS: -Duser.language=en
Socket created on port: 1500
Waiting for connections
```



Clear browser cache if you run this program for the second or more times.

2. Input in the browser.

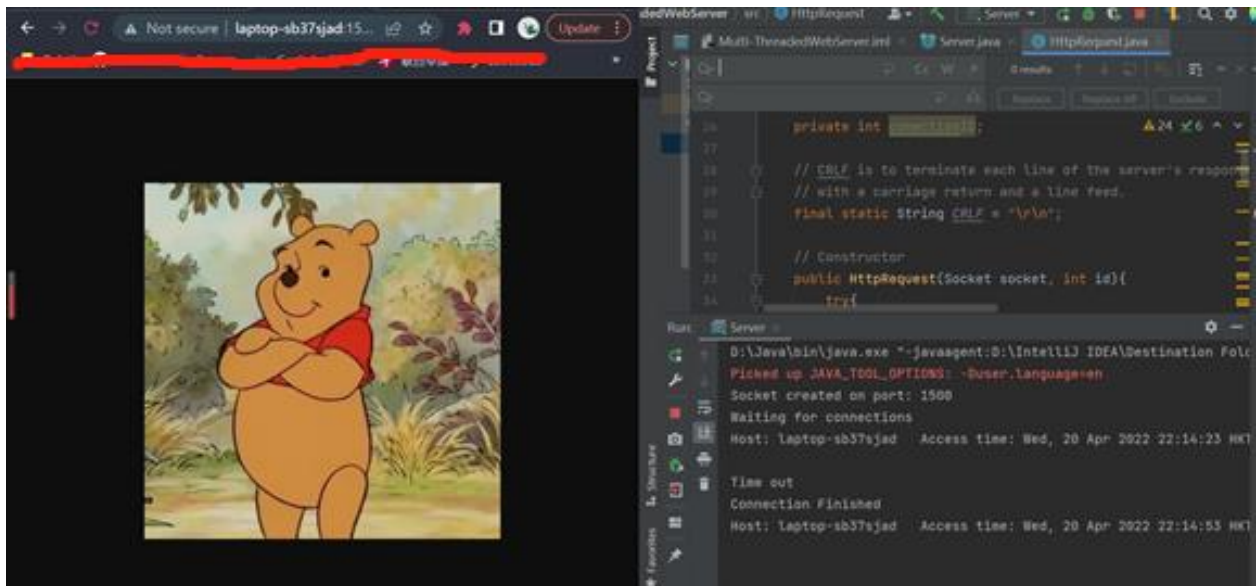
Image file.



Here my host name is laptop-sb37sjad, the port number is 1500

The loading process may take several seconds.

You may notice that there would be two requests. One is for the bear.jpg. The other is for favicon.ico, which denotes website icon or page icon. You may put a file name favicon with suffix .ico to see the result, if you are interested.



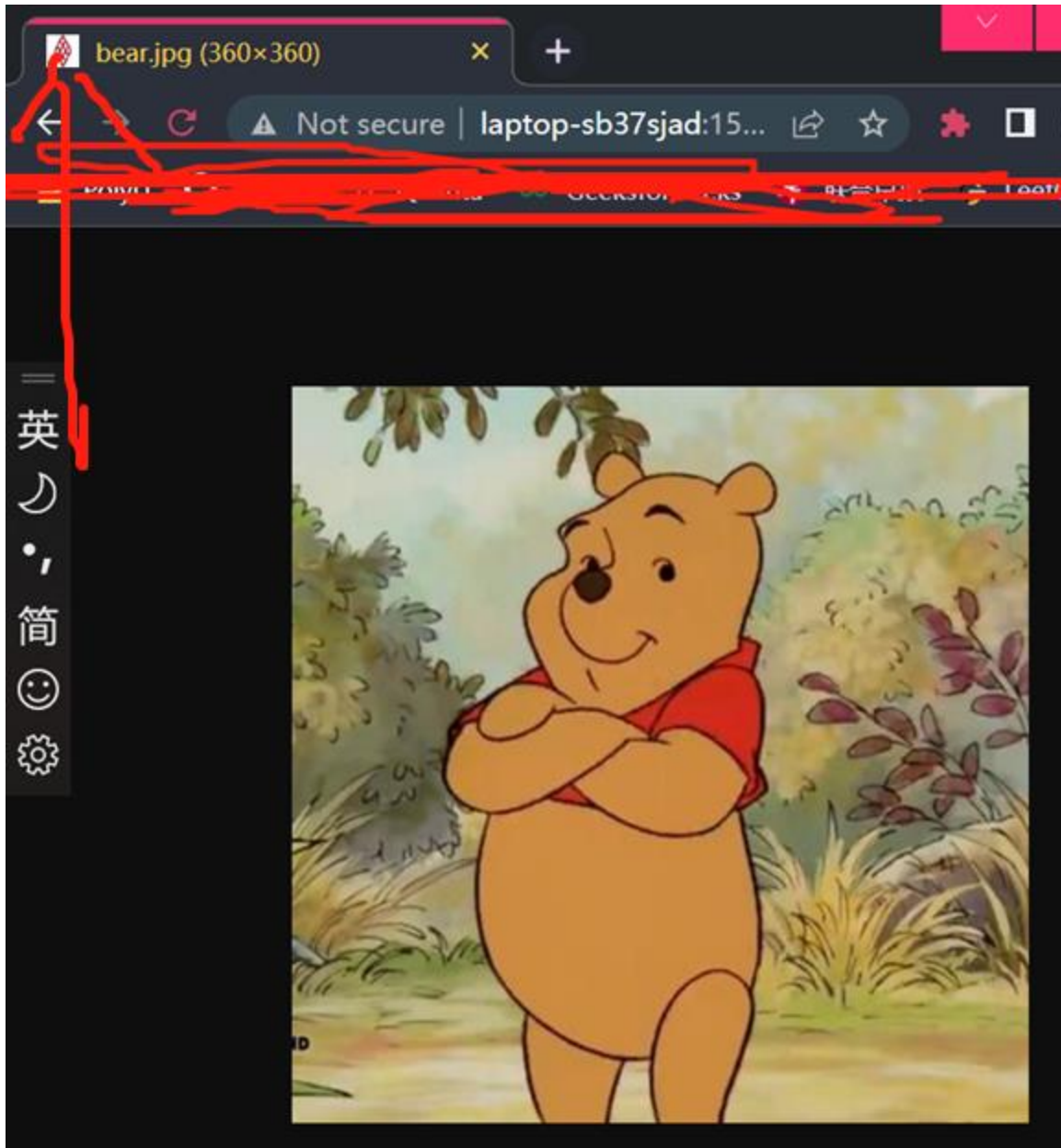
There are two requests in the first time, one of which concerns the icon.

```
Waiting for connections
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:14:23 HKT RequestedFileName: bear.jpg Response type:HTTP/1.1 200 OK

Time out
Connection Finished
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:14:53 HKT RequestedFileName: favicon.ico Response type:HTTP/1.1 404 Not Found
```

If there is no favicon.ico but bear.jpg exists there.

Say I download PolyU badge as the favicon.ico. then try request bear.jpg once again. Then

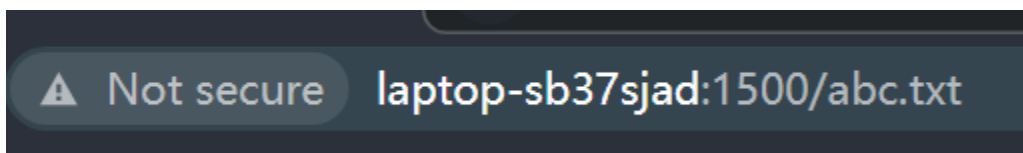


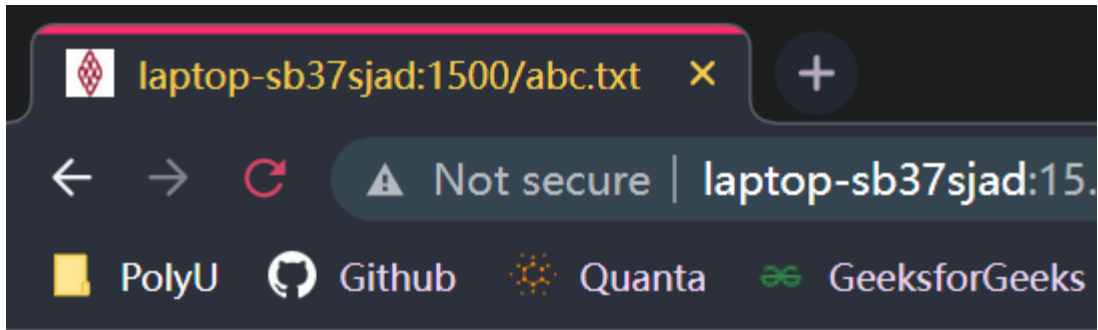
PolyU badge as website icon

```
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:22:06 HKT RequestedFileName: favicon.ico Response type:HTTP/1.1 200 OK
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:22:07 HKT RequestedFileName: bear.jpg Response type:HTTP/1.1 304 Not Modified
```

304 Not Modified for bear.jpg and 200 OK for the website icon file.

Let's request abc.txt file.





Dr. LOU is very patient.

Text file content, with the same web icon.

```
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:28:37 HKT RequestedFileName: abc.txt Response type:HTTP/1.1 200 OK
Time out
Connection Finished
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:30:33 HKT RequestedFileName: abc.txt Response type:HTTP/1.1 304 Not Modified
```

Request for twice, the first would be 200 OK, and the second would be 304 not modified.

You may notice that it's based on persistent HTTP connection. It is kept alive for 30000 milliseconds. After that, timeout occurs, and this connection is closed, as bonus part requests. Then another new socket is initialized for listening.

Let's also try some 404 Not Found situation. Say we request aaaaa.txt file, which does not exist in our relative path.

```
Waiting for connections
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:36:58 HKT RequestedFileName: aaaaa.txt Response type:HTTP/1.1 404 Not Found
Host: laptop-sb37sjad Access time: Wed, 20 Apr 2022 22:37:00 HKT RequestedFileName: aaaaa.txt Response type:HTTP/1.1 404 Not Found
```

404 Not Found is sent, given there is no such file

Room for improvement

1. It should be noted that, the HTML content in the entityBody 404 Not Found response may fail to be sent properly sometimes, after implementing keep-alive.

Reference:

1. <https://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html>
2. <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

http message format: response

status line
(protocol
status code
status phrase)

header
lines

```
HTTP/1.0 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html
```

data, e.g.,
requested
html file

```
data data data data data ...
```

http message format: request

ASCII (human-readable format;
try telnet to www server, port 80)

request line
(GET, POST,
HEAD commands)

header
lines

Carriage return,
line feed
indicates end
of message

```
GET /somedir/page.html HTTP/1.0
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```

(extra carriage return + line feed)