哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	HTTP 代理服务器的设计与实现				
姓名	宋明烨	院系	计算学部		
班级	2103202	学号	2021112228		
任课教师	詹东阳	指导教师	詹东阳		
实验地点	格物 213	实验时间	2023-4-7		
实验课表现	出勤、表现得分(10)		实验报告得分(40)	实验总分	
	操作结果得分(50)				
教师评语					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

熟悉并掌握 Socket 网络编程的过程与技术；深入理解 HTTP 协议，掌握 HTTP 代理服务器的基本工作原理；掌握 HTTP 代理服务器设计与编程实现的基本技能。

实验内容：

(1) 设计并实现一个基本HTTP 代理服务器。要求在指定端口（例如8080）接收来自客户的HTTP 请求并且根据其中的URL 地址访问该地址所指向的HTTP 服务器（原服务器），接收HTTP 服务器的响应报文，并将响应报文转发给对应的客户进行浏览。

(2) 设计并实现一个支持Cache 功能的HTTP 代理服务器。要求能缓存原服务器响应的对象，并能够通过修改请求报文（添加if-modified-since头行），向原服务器确认缓存对象是否是最新版本。（选作内容，加分项目，可以当堂完成或课下完成）

(3) 扩展HTTP 代理服务器，支持如下功能：（选作内容，加分项目，可以当堂完成或课下完成）

a) 网站过滤：允许/不允许访问某些网站；

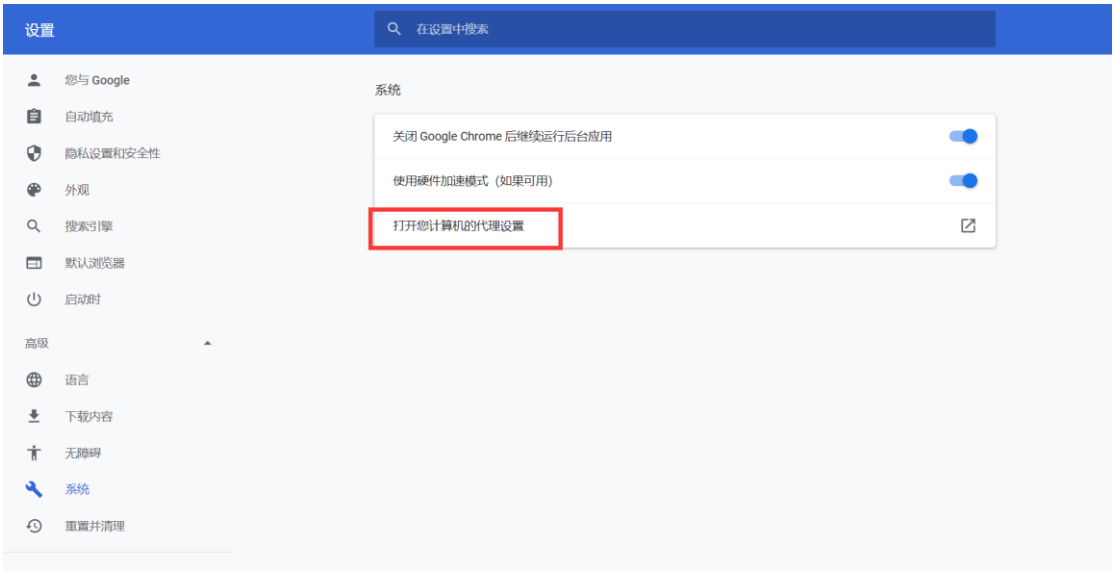
b) 用户过滤：支持/不支持某些用户访问外部网站；

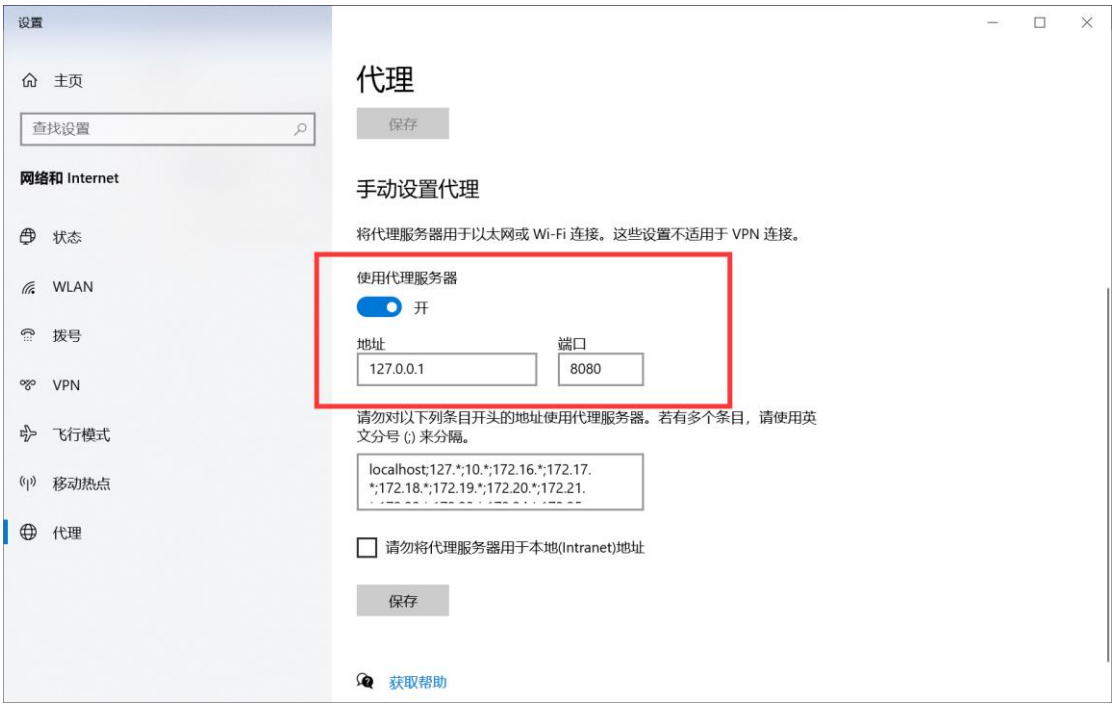
c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。

实验过程：

(1) 浏览器使用代理

在chrome中打开系统设置，打开计算机代理，并在代理中设置代理服务器地址为127.0.0.1:8080。





然后在编写的代理服务器代码中，绑定本机端点地址，端口8080，即可。

(2) 多线程使用

Java的多线程采用ExecutorServive线程池类实现。编写了Thread类实现run方法以实现socket处理子线程。

```
// 使用多线程，需要线程池，防止并发过高时创建过多线程耗尽资源
ExecutorService threadPool = Executors.newFixedThreadPool(100);
```

(3) Socket创建

代理服务器开始运行之后，先创建一个主socket并绑定本机的8080端口，利用这个主socket接受客户机的请求。

```
// 监听指定的端口
int port = 8080;
ServerSocket server = new ServerSocket(port);
// server 将一直等待连接的到来
System.out.println("server 将一直等待连接的到来");
```

使用socket的accept()函数阻塞接收客户机发来的HTTP请求，每收到一个客户机的HTTP请求，就创建一个客户机socket (clientSocket)，并利用clientSocket创建socket处理子线程。在子线程中处理其HTTP请求消息。

```
   Socket socket = server.accept();
System.out.println("获取到一个连接！来自 " + socket.getInetAddress().getHostAddress());
boolean pass = true;
if (forbidUser.contains(socket.getInetAddress().getHostAddress())) {
    pass = false;
}
boolean finalPass = pass;
new Thread(() -> {
    try {
        System.out.println("建立一个新线程\n");
        // 解析 header
        InputStreamReader r = new InputStreamReader(socket.getInputStream());
        BufferedReader br = new BufferedReader(r);
        String readLine = br.readLine();
        String host;
```

(4) 转发客户机请求

收到客户机的HTTP请求之后，在子线程中，利用BufferedReaderl类接收客户机的请求消息并保存。

```
      StringBuilder header = new StringBuilder();

while (readLine != null && !readLine.equals("")) {
    header.append(readLine).append("\n");
    readLine = br.readLine();
}
```

然后按行对HTTP请求消息进行切分，提取出请求行(request line)，利用主类中定义过得静态parse函数对报文进行切分转换便于转发。

```
      private static Map<String, String> parse(String header) {
    if (header.length() == 0) {
        return new HashMap<>();
```

```
    }
    String[] lines = header.split("\\n");
    String method = null;
    String visitAddr = null;
    String httpVersion = null;
    String hostName = null;
    String portString = null;
    for (String line : lines) {
        if ((line.contains("GET") || line.contains("POST") || line.contains("CONNECT"))
&& method == null) {
            // 这一行包括get xxx httpVersion
            String[] temp = line.split("\\s");  // 按空格分割
            method = temp[0];
            visitAddr = temp[1];
            httpVersion = temp[2];
            // 对addr再获得端口号
            // 端口也在这里
            // 先判断是否包含http://关键字
            if (visitAddr.contains("http://") || visitAddr.contains("https://")) {
                // 包含
                // 再判断是否包含端口号
                String[] temp1 = visitAddr.split(":");
                // 因为有http://带来的冒号，所以如果长度>=3 则有端口号
                // 且temp[1]是host
                if (temp1.length >= 3) {
                    portString = temp1[2];
                }
            } else {
                // 不包含http
                String[] temp1 = visitAddr.split(":");
                // 长度>=2 则有端口号
                if (temp1.length >= 2) {
                    // 有端口号，最后没有斜杠
                    portString = temp1[1];
                }
            }

        } else if (line.contains("Host: ") && hostName == null) {
            String[] temp = line.split("\\s");
            hostName = temp[1];
            int maohaoIndex = hostName.indexOf(':');
            if (maohaoIndex != -1) {
                hostName = hostName.substring(0, maohaoIndex);
            }
        }
```

```
    }

    Map<String, String> map = new HashMap<>();
    // 构造参数map
    map.put("method", method);
    map.put("visitAddr", visitAddr);
    map.put("httpVersion", httpVersion);
    map.put("host", hostName);
    if (portString == null) {
        map.put("port", "80");
    } else {
        map.put("port", portString);
    }
    return map;
}
```

转换后从URL中提取出客户机访问的目标主机名及端口号，如果没有端口号则默认是80。有了目标主机和端口号，就可以对客户机的请求消息进行转发。创建一个代理服务器和目标远程服务器之间的connectRemoteSocket，建立新的socket连接目标主机及端口号，然后使用BufferedWriter.write()方法发出之前收到的客户机发来的HTTP请求消息。这样就顺利完成了请求消息的转发。

```
Socket connectRemoteSocket = new Socket(host, visitPort);

// 这个是连接远程服务器的socket 的stream
BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(connectRemoteSocket.getOutputStream()));
StringBuffer requestBuffer = new StringBuffer();
```

(5) 添加代理服务器缓存

上面的过程是没有添加任何额外功能的HTTP代理服务器。为了实现缓存机制，需要一个缓存文件。针对访问的每个URL创建一个缓存文件。在使用parse函数解析出URL之后，然后到缓存文件夹中寻找对应的文件，如果找到了就证明代理服务器中有缓存；否则没有。

```
      // 文件输出流
FileOutputStream fileOutputStream =
       new FileOutputStream(
              new File(visitAddr.replace('/', 'g') + ".mycache"));
```

如果没有找到缓存文件，则需要创建对应的缓存文件。代理服务器与远程目标主机建立连接之后，发送请求，然后将接收到的数据写入缓存文件。

```
// 不管用不用缓存都要接着读完来自服务器的数据
int bufferLength = 1;
byte[] buffer = new byte[bufferLength];
int count;
System.out.println("Start reading!>.....From > " + visitAddr);
while (true) {
    count = remoteInputStream.read(buffer);
```

```
    if (count == -1) {
        break;
    }
    if (!useCache) {
        // 不用缓存才写这些
        outToBrowser.write(buffer);
        fileOutputStream.write(buffer);
    }
}
fileOutputStream.flush();   // 输出到文件
fileOutputStream.close();   // 关闭文件流
System.out.println("finish");
```

如果找到了缓存文件，则需要向远程目标服务器发送带有If-modified-since字段的请求消息。检查远程服务器的响应码中是否包含304，即判断缓存文件是否需要更新。如果需要更新，则将远程目标服务器的相应写入代理服务器对应的缓存文件，并相应给客户端；如果不需要更新，则直接将代理服务器保存的缓存文件数据返回给客户端。

```
          int len = remoteInputStream.read(tempBytes);
String res = new String(tempBytes, 0, len);
System.out.println(res);
// 判断是否包含304，如果是包含，标记为使用缓存
if (res.contains("304")) {
    // 远程服务器没有更新这个资源，可以直接使用缓存
    System.out.println(visitAddr + " 服务器内容未变更，使用缓存");
    // 刚才的小字节也不要了，后续的报文读完不用，然后直接从文件读
    useCache = true;    // 用缓存
} else {
    System.out.println(visitAddr + " 服务器内容可能变更，不使用缓存");

    // 没有缓存，刚才临时读入的要用上。并且要接着读报文并向浏览器输出
    outToBrowser.write(tempBytes);

    // 临时字节写入缓存文件
    fileOutputStream.write(tempBytes);
}
```

(6) 添加客户主机过滤

在接收到客户机的请求时，检查socket的端点地址，是否存在于禁止访问的客户机列表中，在主函数前定义Set表中记录禁止访问的网络主机。如果客户机被禁止访问则直接返回阻止界面。

```
      /**
 * 禁止访问的用户。
 */
private static Set<String> forbidUser = new HashSet<>();
```

```
// 在输入流结束之后判断
// 判断用户是否被屏蔽
if (!finalPass) {
    System.out.println("From a forbidden user.");
    PrintWriter pw = new PrintWriter(socket.getOutputStream());
    pw.println("You are a forbidden user!");
    pw.close();

    socket.close();
    return;
}
```

(7) 添加目标网站过滤

在使用parse转换出URL的网站名之后，检查目标主机是否存在于禁止访问的外部网站中，如果目标主机被禁止访问则直接返回禁止访问。

```
    /**
 * 禁止访问的网址。
 */
private static Set<String> forbidSet = new HashSet<>();

if (visitAddr != null && isForbidden(visitAddr)) {
    // 被屏蔽，不允许访问
    System.out.println("Visiting a forbidden site.");
    PrintWriter pw = new PrintWriter(socket.getOutputStream());
    pw.println("You can not visit " + visitAddr + "!");
    pw.close();
}
```

(8) 网站重定向(钓鱼)

在使用parse解析出目标主机名之后，检查hostname或site是否存在于重定向列表中，如果存在，就把客户机的HTTP请求消息中的所有hostname全部更换为重定向目标主机名，serverSocket也去和重定向目标主机建立连接，即可完成重定向。

```
    /**
 * 重定向主机 map。
 */
private static Map<String, String> redirectHostMap = new HashMap<>();

/**
 * 重定向访问网址 map。
 */
private static Map<String, String> redirectAddrMap = new HashMap<>();
```

```
private static String redirectHost(String oriHost) {
    Set<String> keywordSet = redirectHostMap.keySet();
    for (String keyword : keywordSet) {
        if (oriHost.contains(keyword)) {
            System.out.println("originHost: " + oriHost);
            String redHost = redirectHostMap.get(keyword);  // 直接修改方案
            System.out.println("redirectHost: " + redHost);
            return redHost;
        }
    }
    return oriHost;
}


private static String redirectAddr(String oriAddr) {
    Set<String> keywordSet = redirectAddrMap.keySet();
    for (String keyword : keywordSet) {
        if (oriAddr != null && oriAddr.contains(keyword)) {
            System.out.println("originAddr: " + oriAddr);
            String redAddr = redirectAddrMap.get(keyword);  // 直接修改方案
            System.out.println("redirectAddr: " + redAddr);
            return redAddr;
        }
    }
    return oriAddr;
}
```

(9) 响应客户机

无论是否采用缓存、是否采用重定向，在完成请求消息的转发后，收到的消息或是缓存文件都需要返回给客户端。使用clientSocket.send()函数将相应消息发给客户机。

```
fileOutputStream.flush();   // 输出到文件
fileOutputStream.close();   // 关闭文件流
System.out.println("finish");

outToBrowser.flush();   // 输出到浏览器
connectRemoteSocket.close();    // 关闭连接远程服务器的 socket
```

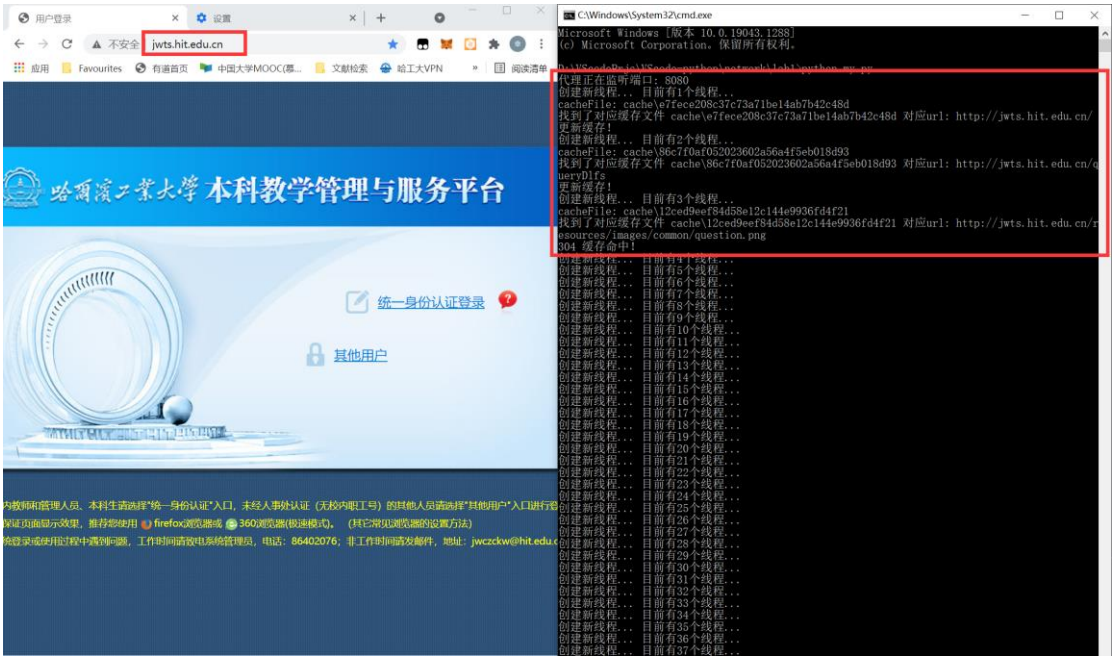
(10)关闭套接字

完成上面的所有工作之后，需要关闭代理服务器和客户机之间的socket、代理服务器和远程目标服务器之间的socket。

```
clientSocket.close()
serverSocket.close()
```

实验结果：

(1) 基础功能



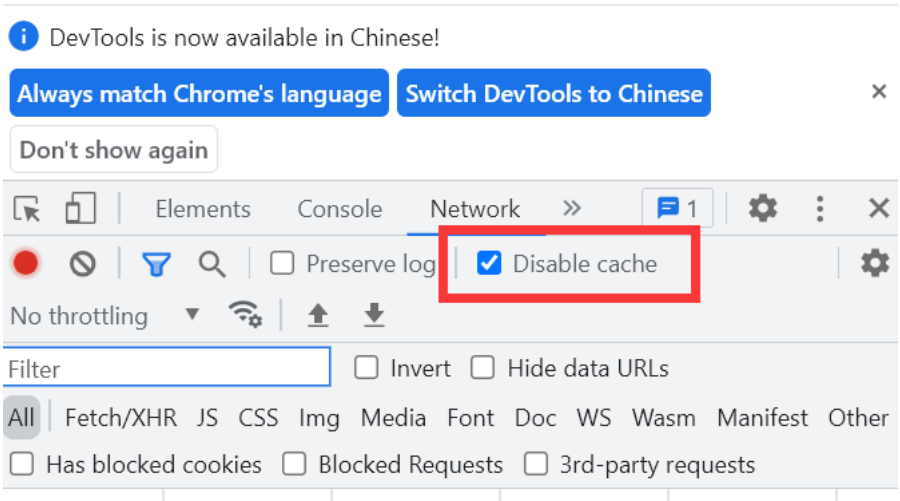
访问jwts.hit.edu.cn，可以正常显示页面，并在控制台打印相关信息。

(2) 缓存功能

缓存功能对于一些动态页面，即使没有发生改变，也不会返回304状态码。只有当访问静态资源，并且在请求消息中添加了If-modified-since字段后，服务器才会返回304状态码。

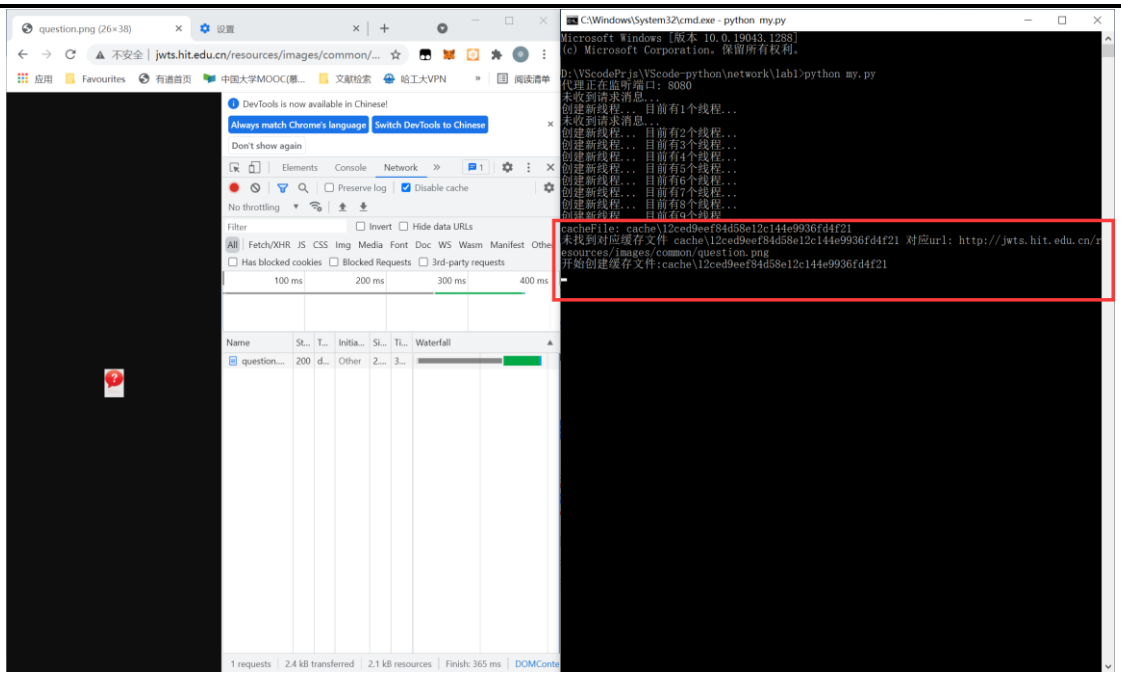
这里我使用一个静态资源http://jwts.hit.edu.cn/resources/images/common/question.png进行测试，这是一张小图片。

此外，为了展示缓存效果，需要在Chrome的F12控制台中“Disable cache”，否则浏览器也会在本地对资源进行缓存，不发送HTTP请求。

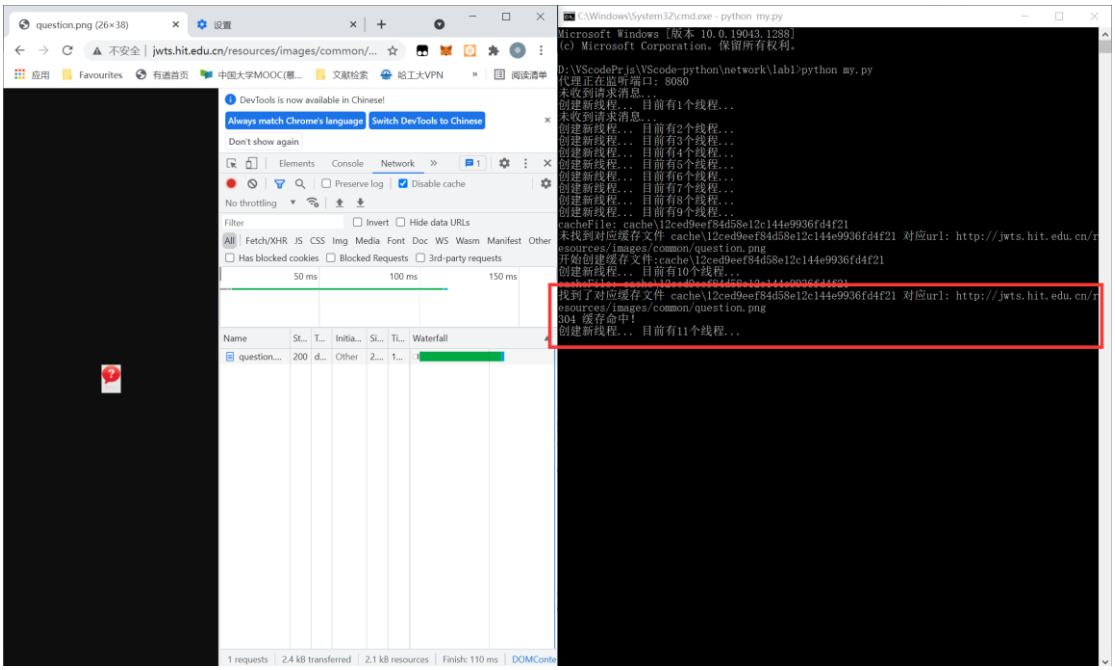


下面是实验效果：

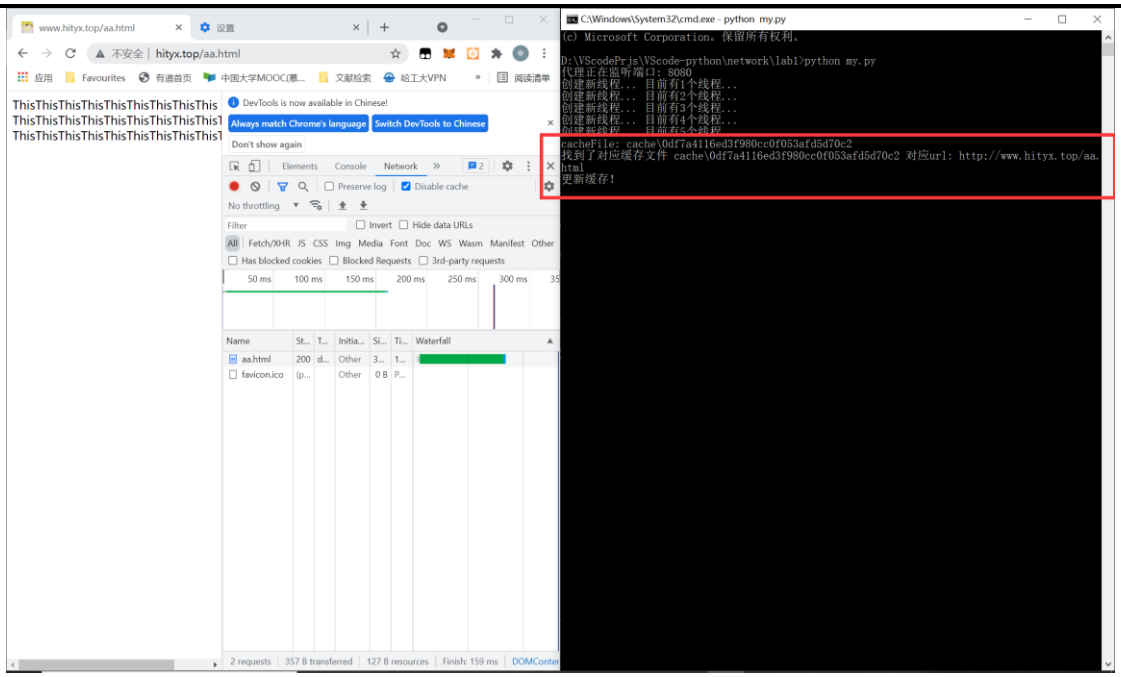
代理服务器未找到缓存文件：



代理服务器找到了缓存文件，并且检查发现该缓存资源无需更新(远程服务器返回304状态码)：



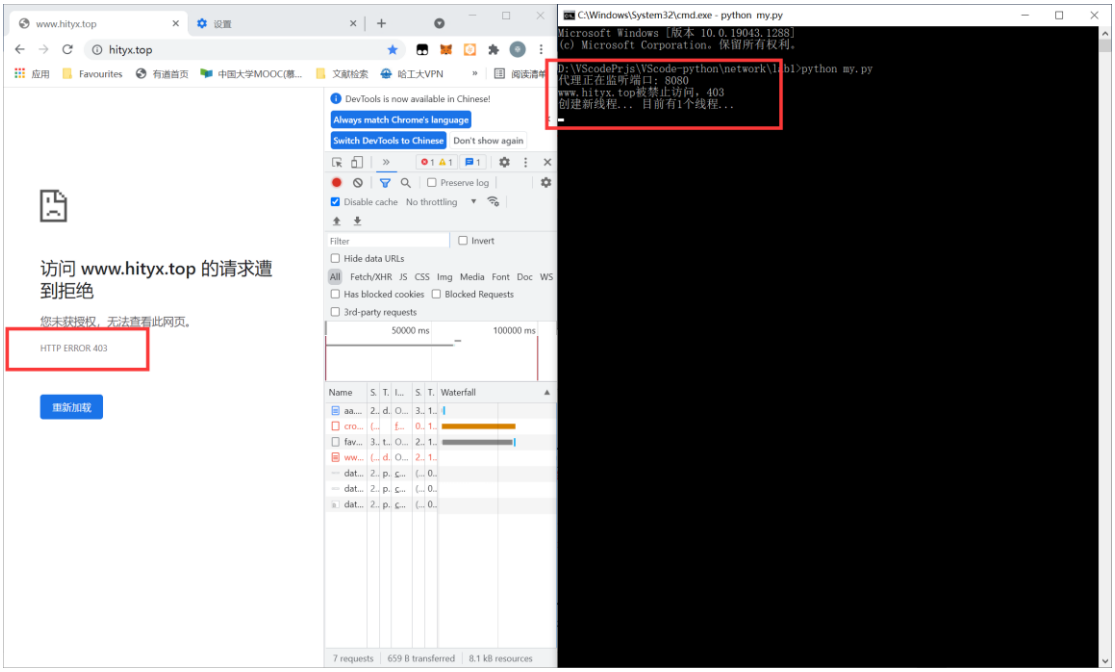
代理服务器找到缓存文件且需要更新 (这一项我采用了另一个动态资源进行测试)：



由此可见该缓存功能可以正常工作。

(3) 网站过滤

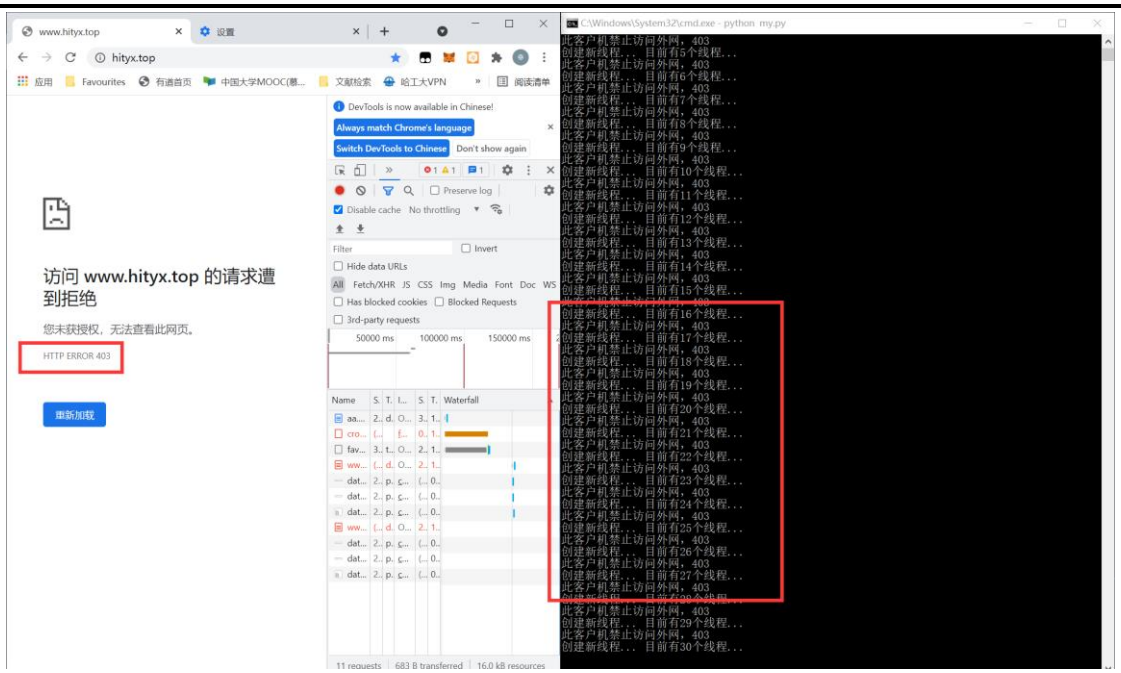
在这里我禁止访问www.hityx.top这个网站。



可以看到返回了403状态码并体现在了浏览器中。

(4) 用户过滤

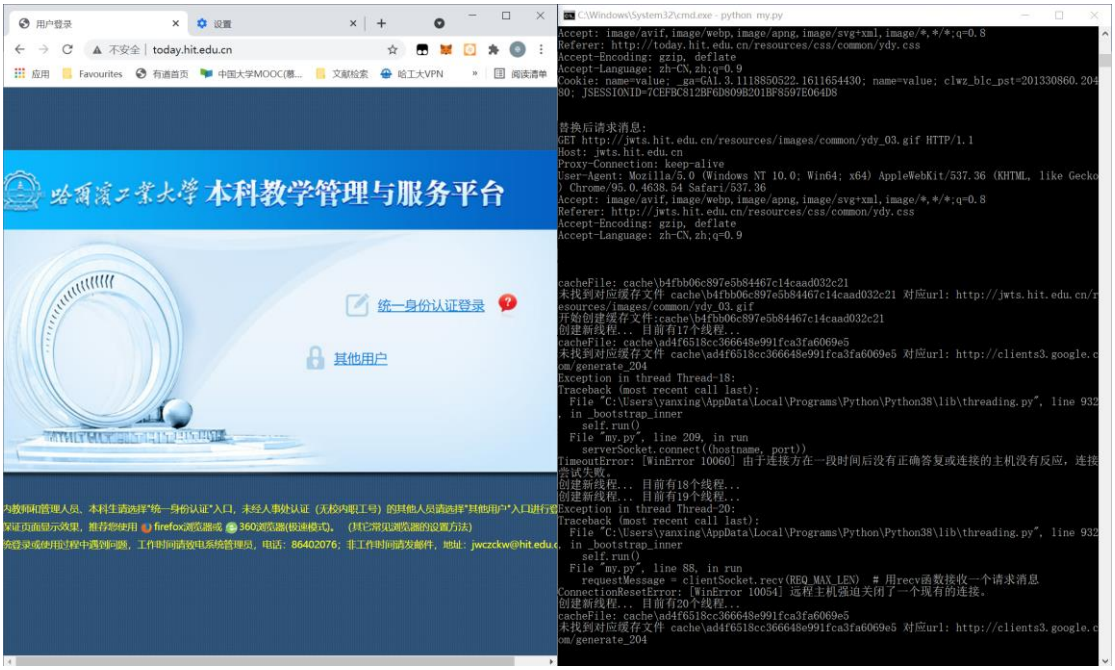
在这里我禁止本机(127.0.0.1)访问外网。



可以看到本机的所有外网访问全球均被拦截。

(5) 网站引导(钓鱼)

在这里我将所有对今日哈工大(today.hit.edu.cn)的访问全部引导至哈工大教务处(jwts.hit.edu.cn)。



可以看到在浏览器的地址栏中显示的是today.hit.edu.cn，但是网页的内容显示的并不是今日哈工大而是教务处网页。

问题讨论：

(1) Socket 编程的客户端和服务器端主要步骤；

客户端：

1. 初始化套接字库

C语言中需要通过WSAStartup函数来加载套接字库，在python环境下，只需要通过import socket就可以完成加载套接字库；

2. 创建socket

利用socket(AF_INET,SOCK_STREAM)方法创建套接字，第一个参数代表协议族，AF_INET表示是Internet通信;第二个参数代表套接字类型，SOCK_STREAM表示是面向TCP连接的流式套接字；有时后面还会有第三个参

数，代表协议号，默认设置为0；

3. 向服务器发出连接请求

采用connect()方法与服务器端连接，一般函数参数为一个元组形式(hostname,port),若连接出错则会返回错误；

4. 连接建立后，向服务器请求数据，并置于等待状态，等待接收服务器返回的数据

利用套接字的send()方法向服务器端发送请求消息， send()函数是发送一次数据，返回值为成功发送的字节数，该值可能会小于需要发送的字节数。

发送完请求消息后，开始处于等待状态，当服务器端返回数据到达时，利用recv()函数接受数据，返回的类型为字符串形式，其中还可以规定接受的最大字节数；

5. 关闭连接

调用close()函数关闭socket连接；

6. 关闭套接字库

C语言中需要调用WSACleanup函数释放所使用的Windows Sockets Dll，但是在python中无需显式关闭套接字库；

服务器端：

1. 初始化套接字库

本实验在python环境下，只需要通过import socket就可以完成加载套接字库；

2. 创建套接字

利用socket(AF_INET,SOCK_STREAM)方法创建套接字，第一个参数代表协议族，AF_INET表示是Internet通信;第二个参数代表套接字类型，SOCK_STREAM表示是面向TCP连接的流式套接字；有时后面还会有第三个参数，代表协议号，默认设置为0；

3. 绑定套接字

通过bind(address)方法将套接字绑定到指定主机和端口号中，其中参数为元组形式(host,port)，分别为主机IP地址、端口号；

4. 监听端口

使用listen(backlog)函数进行监听，其中参数backlog代表最多允许多少个客户连接服务器，值至少为1，一般设置为5，当有多个连接请求时，需要进行排队，队列满了时则拒绝请求；

5. 接收连接请求，返回新的套接字

调用accept()方法，表示接受连接请求，并返回新的套接字对象和客户端地址，以元组形式返回；

6. 接收客户端请求消息，返回请求数据，与其通信

调用recv()函数接受客户端请求消息；

返回请求数据时用send()函数；

7. 关闭套接字

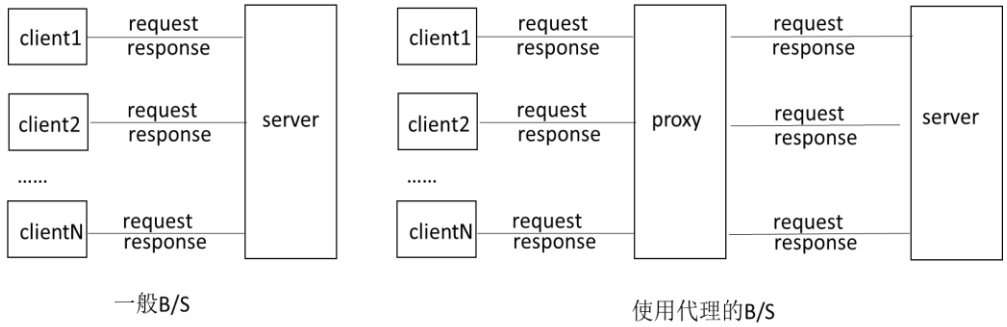
调用close()函数关闭连接；

8. 关闭套接字库

C语言中需要调用WSACleanup函数释放所使用的Windows Sockets Dll，但是在python中无需显式关闭套接字库；

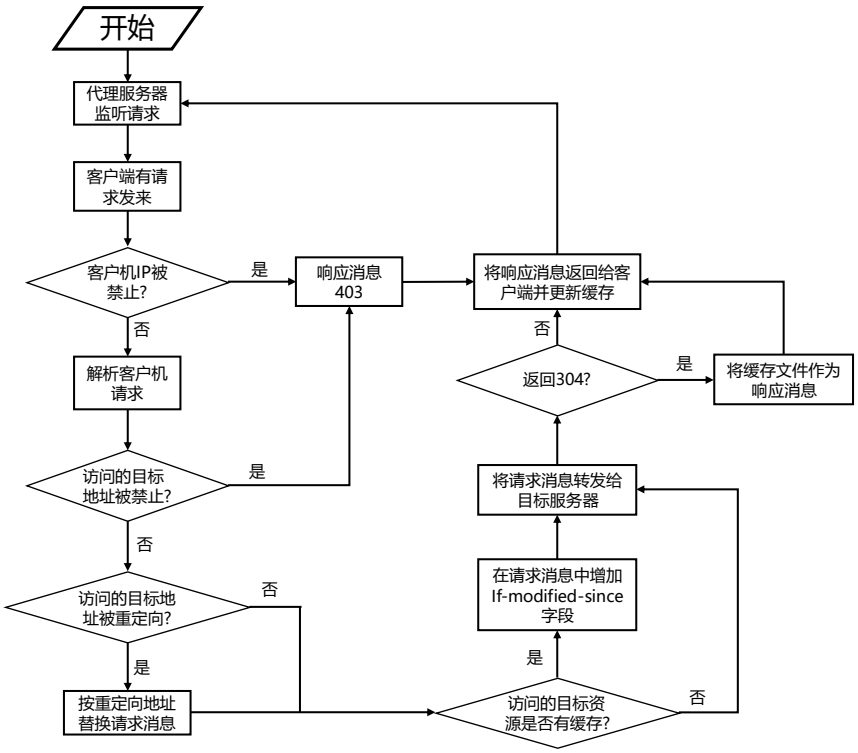
(2) HTTP 代理服务器的基本原理；

代理服务器，俗称“翻墙软件”，允许一个网络终端(一般为客户端)通过这个服务与另一个网络终端(一般为服务器)进行非直接的连接。如下图所示，为普通Web应用通信方式与采用代理服务器的通信方式的对比。



代理服务器在指定端口（例如8080）监听浏览器的访问请求（需要在客户端浏览器进行相应的设置），接收到浏览器对远程网站的浏览请求时，代理服务器开始在代理服务器的缓存中检索URL对应的对象（网页、图像等对象），找到对象文件后，提取该对象文件的最新被修改时间；代理服务器程序在客户的请求报文首部插入<If-Modified-Since: 对象文件的最新被修改时间>，并向原Web服务器转发修改后的请求报文。如果代理服务器没有该对象的缓存，则会直接向原服务器转发请求报文，并将原服务器返回的响应直接转发给客户端，同时将对象缓存到代理服务器中。代理服务器程序会根据缓存的时间、大小和提取记录等对缓存进行清理。

(3) HTTP 代理服务器的程序流程图；



(4) 实现HTTP 代理服务器的关键技术及解决方案；

a）单用户代理服务器

单用户的简单代理服务器可以设计为一个非并发的循环服务器。首先，代理服务器创建 HTTP 代理服务的 TCP 主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，读取客户端的 HTTP 请求报文，通过请求行中的 URL，解析客户期望访问的原服务器 IP 地址；创建访问原（目标）服务器的 TCP 套接字，将 HTTP 请求报文转发给目标服务器，接收目标服务器的响应报文，当收到响应报文之后，将响应报文转发给客户端，最后关闭套接字，等待下一次连接。

b）多用户代理服务器

多用户的简单代理服务器可以实现为一个多线程并发服务器。首先，代理服务器创建 HTTP 代理服务的 TCP

主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，创建一个子线程，由子线程执行上述一对一的代理过程，服务结束之后子线程终止。与此同时，主线程继续接受下一个客户的代理服务。

(5) HTTP 代理服务器实验验证过程以及实验结果；

见此报告前“实验过程”和“实验结果”部分。

(6) HTTP Java代理服务器源代码（带有详细注释）。

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;


public class Server {

    /**
     * 重定向主机 map。
     */
    private static Map<String, String> redirectHostMap = new HashMap<>();

    /**
     * 重定向访问网址 map。
     */
    private static Map<String, String> redirectAddrMap = new HashMap<>();

    /**
     * 禁止访问的网址。
     */
    private static Set<String> forbidSet = new HashSet<>();

    /**
     * 禁止访问的用户。
     */
    private static Set<String> forbidUser = new HashSet<>();

    static {
        // 更改这些内容达到屏蔽访问或钓鱼的目的

//        redirectAddrMap.put("http://jwts.hit.edu.cn", "http://today.hit.edu.cn/");
//        redirectAddrMap.put("http://jwts.hit.edu.cn/loginLdapQian",
"http://today.hit.edu.cn/");
//        redirectHostMap.put("jwts.hit.edu.cn", "today.hit.edu.cn");
```

```
//        forbidSet.add("http://jwts.hit.edu.cn/");
//        forbidSet.add("http://jwts.hit.edu.cn");
        //forbidSet.add("http://jwts.hit.edu.cn/");
        //forbidSet.add("http://jwts.hit.edu.cn/");

//        forbidUser.add("127.0.0.1");

    }


    private static boolean isForbidden(String site) {
        return forbidSet.contains(site);
    }


    private static String redirectHost(String oriHost) {
        Set<String> keywordSet = redirectHostMap.keySet();
        for (String keyword : keywordSet) {
            if (oriHost.contains(keyword)) {
                System.out.println("originHost: " + oriHost);
                String redHost = redirectHostMap.get(keyword);  // 直接修改方案
                System.out.println("redirectHost: " + redHost);
                return redHost;
            }
        }
        return oriHost;
    }


    private static String redirectAddr(String oriAddr) {
        Set<String> keywordSet = redirectAddrMap.keySet();
        for (String keyword : keywordSet) {
            if (oriAddr != null && oriAddr.contains(keyword)) {
                System.out.println("originAddr: " + oriAddr);
                String redAddr = redirectAddrMap.get(keyword);  // 直接修改方案
                System.out.println("redirectAddr: " + redAddr);
                return redAddr;
            }
        }
        return oriAddr;
    }


    private static Map<String, String> parse(String header) {
```

```
        if (header.length() == 0) {
            return new HashMap<>();
        }
        String[] lines = header.split("\\n");
        String method = null;
        String visitAddr = null;
        String httpVersion = null;
        String hostName = null;
        String portString = null;
        for (String line : lines) {
            if ((line.contains("GET") || line.contains("POST") || line.contains("CONNECT")) &&
method == null) {
                // 这一行包括 get xxx httpVersion
                String[] temp = line.split("\\s");  // 按空格分割
                method = temp[0];
                visitAddr = temp[1];
                httpVersion = temp[2];
                // 对 addr 再获得端口号
                // 端口也在这里
                // 先判断是否包含 http://关键字
                if (visitAddr.contains("http://") || visitAddr.contains("https://")) {
                    // 包含
                    // 再判断是否包含端口号
                    String[] temp1 = visitAddr.split(":");
                    // 因为有 http://带来的冒号，所以如果长度>=3 则有端口号
                    // 且 temp[1]是 host
                    if (temp1.length >= 3) {
                        portString = temp1[2];
                    }
                } else {
                    // 不包含 http
                    String[] temp1 = visitAddr.split(":");
                    // 长度>=2 则有端口号
                    if (temp1.length >= 2) {
                        // 有端口号，最后没有斜杠
                        portString = temp1[1];
                    }
                }

            } else if (line.contains("Host: ") && hostName == null) {
                String[] temp = line.split("\\s");
                hostName = temp[1];
                int maohaoIndex = hostName.indexOf(':');
                if (maohaoIndex != -1) {
                    hostName = hostName.substring(0, maohaoIndex);
```

```
            }
        }
    }

    Map<String, String> map = new HashMap<>();
    // 构造参数 map
    map.put("method", method);
    map.put("visitAddr", visitAddr);
    map.put("httpVersion", httpVersion);
    map.put("host", hostName);
    if (portString == null) {
        map.put("port", "80");
    } else {
        map.put("port", portString);
    }
    return map;
}


public static void main(String[] args) throws IOException {
    // 监听指定的端口
    int port = 8080;
    ServerSocket server = new ServerSocket(port);
    // server 将一直等待连接的到来
    System.out.println("server 将一直等待连接的到来");

    // 使用多线程，需要线程池，防止并发过高时创建过多线程耗尽资源
    ExecutorService threadPool = Executors.newFixedThreadPool(100);

    while (true) {
        //阻塞等待连接
        Socket socket = server.accept();
        System.out.println("获取到一个连接！来自 " +
socket.getInetAddress().getHostAddress());
        boolean pass = true;
        if (forbidUser.contains(socket.getInetAddress().getHostAddress())) {
            pass = false;
        }
        boolean finalPass = pass;
        new Thread(() -> {
            try {
                System.out.println("建立一个新线程\n");
                // 解析 header
                InputStreamReader r = new InputStreamReader(socket.getInputStream());
                BufferedReader br = new BufferedReader(r);
```

```
String readLine = br.readLine();
String host;

StringBuilder header = new StringBuilder();

while (readLine != null && !readLine.equals("")) {
    header.append(readLine).append("\n");
    readLine = br.readLine();
}

// 在输入流结束之后判断
// 判断用户是否被屏蔽
if (!finalPass) {
    System.out.println("From a forbidden user.");
    PrintWriter pw = new PrintWriter(socket.getOutputStream());
    pw.println("You are a forbidden user!");
    pw.close();

    socket.close();
    return;
}

// 打印参数表
Map<String, String> map = parse(header.toString());

System.out.println("--------------------");
System.out.println(map);
System.out.println("--------------------");

host = map.get("host"); // host
// 端口
String portString = map.getOrDefault("port", "80");
// 端口
int visitPort = Integer.parseInt(portString);
// 访问的网站
String visitAddr = map.get("visitAddr");
// method
String method = map.getOrDefault("method", "GET");

// 判断是否屏蔽掉这个网站
if (visitAddr != null && isForbidden(visitAddr)) {
    // 被屏蔽，不允许访问
    System.out.println("Visiting a forbidden site.");
    PrintWriter pw = new PrintWriter(socket.getOutputStream());
    pw.println("You can not visit " + visitAddr + "!");
```

```
                    pw.close();
                } else {
                    // 获得跳转主机和资源
                    String tempRedAddr = redirectAddr(visitAddr);
                    if (tempRedAddr!=null && !tempRedAddr.equals(visitAddr)) {
                        visitAddr = tempRedAddr;
                        host = redirectHost(host);
                    }

                    // 看看在不在缓存中
                    // 获得一下文件
                    File cacheFile = new File(visitAddr.replace('/', 'g') + ".mycache");
                    boolean useCache = false;   // 标记是否用 cache

                    // 默认的最后修改时间，用于文件不存在的时候
                    String lastModified = "Thu, 01 Jul 1970 20:00:00 GMT";

                    if (cacheFile.exists() && cacheFile.length() != 0) {
                        System.out.println("使用缓存\n");
                        // 文件存在且大小不为 0，说明访问内容被缓存过
                        System.out.println(visitAddr + " 有缓存");
                        // 获得修改时间
                        Calendar cal = Calendar.getInstance();
                        long time = cacheFile.lastModified();
                        SimpleDateFormat formatter = new SimpleDateFormat("EEE, dd MMM yyyy
HH:mm:ss 'GMT'", Locale.ENGLISH);
                        cal.setTimeInMillis(time);
                        cal.set(Calendar.HOUR, -7);
                        cal.setTimeZone(TimeZone.getTimeZone("GMT"));
                        lastModified = formatter.format(cal.getTime());
                        System.out.println(cal.getTime());
                    }


                    // 创建新的 socket 连接远程服务器
                    Socket connectRemoteSocket = new Socket(host, visitPort);

                    // 这个是连接远程服务器的 socket 的 stream
                    BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(connectRemoteSocket.getOutputStream()));

                    StringBuffer requestBuffer = new StringBuffer();
                    requestBuffer.append(method).append(" ").append(visitAddr)
                            .append(" HTTP/1.1").append("\r\n")
                            .append("HOST: ").append(host).append("\n")
```

```
                            .append("Accept:text/html,application/xhtml+xml,application/xml
;q=0.9,image/webp,*/*;q=0.8\n")
                            .append("Accept-Encoding:gzip, deflate, sdch\n")
                            .append("Accept-Language:zh-CN,zh;q=0.8\n")
                            .append("If-Modified-Since:
").append(lastModified).append("\n")
                            .append("User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36 Edg/111.0.1661.62\n")
                            .append("Encoding:UTF-8\n")
                            .append("Connection:keep-alive" + "\n")
                            .append("\n");
                    writer.write(requestBuffer.toString()); // 发送报文
                    // 打印看看
                    System.out.println(requestBuffer.toString());
                    // 发送报文
                    writer.flush();

                    // 这是向浏览器输出的输出流
                    OutputStream outToBrowser = socket.getOutputStream();

                    // 文件输出流
                    FileOutputStream fileOutputStream =
                            new FileOutputStream(
                                    new File(visitAddr.replace('/', 'g') + ".mycache"));

                    // 从远程服务器获得输入的输入流
                    BufferedInputStream remoteInputStream =
                            new BufferedInputStream(connectRemoteSocket.getInputStream());

                    // 先使用一个小字节缓存获得头部，包含第一行就够了
                    byte[] tempBytes = new byte[20];
                    int len = remoteInputStream.read(tempBytes);
                    String res = new String(tempBytes, 0, len);
                    System.out.println(res);
                    // 判断是否包含 304，如果是包含，标记为使用缓存
                    if (res.contains("304")) {
                        // 远程服务器没有更新这个资源，可以直接使用缓存
                        System.out.println(visitAddr + " 服务器内容未变更，使用缓存");
                        // 刚才的小字节也不要了，后续的报文读完不用，然后直接从文件读
                        useCache = true;    // 用缓存
                    } else {
                        System.out.println(visitAddr + " 服务器内容可能变更，不使用缓存");

                        // 没有缓存，刚才临时读入的要用上。并且要接着读报文并向浏览器输出
                        outToBrowser.write(tempBytes);
```

```
                // 临时字节写入缓存文件
                fileOutputStream.write(tempBytes);
            }
            if (useCache) {
                // 用缓存
                // 这是向浏览器输出的输出流
                System.out.println(visitAddr + " 正在使用缓存加载");
                // 建立文件读写
                FileInputStream fileInputStream = new FileInputStream(cacheFile);
                int bufferLength = 1;
                byte[] buffer = new byte[bufferLength];
                int count;

                while (true) {
                    count = fileInputStream.read(buffer);
                    System.out.println("Reading>.... From file>..." + count);
                    if (count == -1) {
                        break;
                    }
                    outToBrowser.write(buffer);
                }
                outToBrowser.flush();
            }
            // 不管用不用缓存都要接着读完来自服务器的数据
            int bufferLength = 1;
            byte[] buffer = new byte[bufferLength];
            int count;
            System.out.println("Start reading!>.....From > " + visitAddr);
            while (true) {
                count = remoteInputStream.read(buffer);
                if (count == -1) {
                    break;
                }
                if (!useCache) {
                    // 不用缓存才写这些
                    outToBrowser.write(buffer);
                    fileOutputStream.write(buffer);
                }
            }
            fileOutputStream.flush();   // 输出到文件
            fileOutputStream.close();   // 关闭文件流
            System.out.println("finish");

            outToBrowser.flush();   // 输出到浏览器
```

```
                    connectRemoteSocket.close();    // 关闭连接远程服务器的 socket

                }
                socket.close();// 关闭浏览器与程序的 socket
            } catch (IOException e) {

            }
        }).start();
    }
}


}
```

心得体会：

本次对 HTTP 代理服务器的实现，更加理解了代理服务器的原理和执行过程；
同时感受到了多线程执行的优化；
熟悉了 JAVA socket 编程；