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	[redacted]		院系	计算学部	
班级	[redacted]		学号	[redacted]	
任课教师	刘亚维		指导教师		
实验地点	格物 207		实验时间	2[redacted]1	
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					

计算学部

实验目的：

（注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）

本次实验的主要目的。

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：

概述本次实验的主要内容，包含的实验项等。

(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：

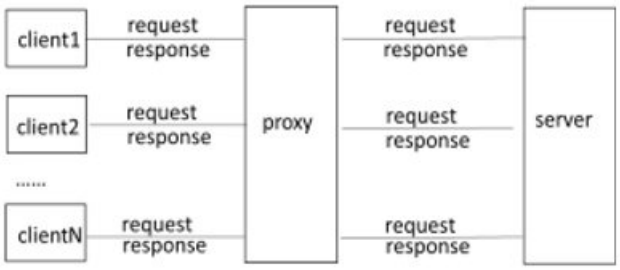
以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

1.浏览器使用代理:

选择使用FireFox浏览器进行实验，在浏览器的连接设置里面配置使用程序配置的代理服务器：127.0.0.1:20201

2. 设计并实现一个HTTP 代理服务器

设计一个多线程并发的代理服务器，主进程创建 HTTP 代理服务的 TCP 主套接字，并通过该主套接字不断监听 ，等待客户端的连接请求。每当收到来自客户端的连接之后，创建一个子线程，由子线程执行代理过程，服务结束之后子线程终止。与此同时，主线程不断接受下 一个客户的代理服务。



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。

```
1.  int port = 20201;
2.          ServerSocket server = new ServerSocket(port);
3.          // server 将一直等待连接的到来
4.          System.out.println("代理服务器启动,监听端口: " + server.getLocalPort());
5.
6.
7.          while (true) {
8.              Socket clientSocket = server.accept();
9.              String UserIP = clientSocket.getInetAddress().getHostAddress();
10.             System.out.println("获取到一个连接! 来自 " + UserIP);
11.             .......
12.         }
```

然后，子进程读取来自套接字的报文，解析报文，与报文中预定的服务器建立远程套接字并传输与接受报文，实现代理，达到代理来自客户端的 HTTP 请求并接收来自服务器的响应报文的目的。

```
13. Socket proxySocket = new Socket(host, 80);
14. System.out.println("代理套接字已建立!:" + proxySocket);
15. BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(proxySocket.
16. getOutputStream()));
17. StringBuffer requestBuffer = new StringBuffer();
18. requestBuffer.append(header);
19. requestBuffer.append("If-Modified-Since: ").append(lastModified).append("\n");
20. writer.write(requestBuffer.append("\n").toString()); // 发送报文
21. writer.flush();
```

接受到响应报文后将其返回来自浏览器的套接字即可

```
OutputStream clientOutput = clientSocket.getOutputStream();
BufferedInputStream remoteInputStream = new
BufferedInputStream(proxySocket.getInputStream());
 int bufferLength = 1;
byte[] buffer = new byte[bufferLength];
                        while (true) {
                            int count = remoteInputStream.read(buffer);
                            if (count == -1) {
                                break;
                            }

                              clientOutput.write(buffer);

                            }
                        }
```

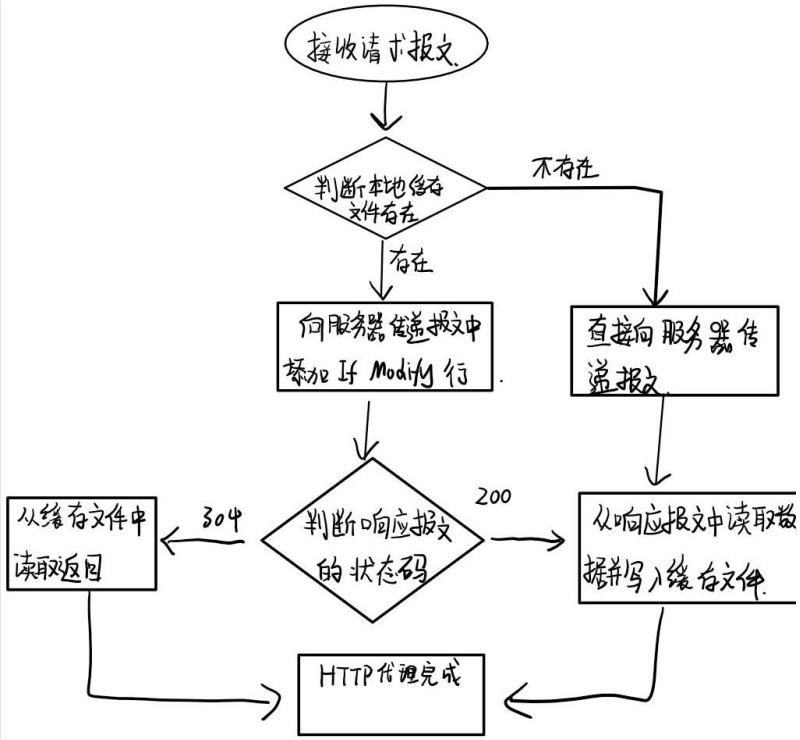
最后，关闭本地代理和与远程服务器建立的套接字，代理完成

```
proxySocket.close();     // 关闭连接远程服务器的 socket
clientSocket.close();// 关闭浏览器与程序的 socket
```

3. 实现代理服务器缓存功能

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

并在本地储存来自网页服务器返回报文的副本，如果访问网页的缓存报文存在的话，就从本地读入缓存并输出到浏览器上，实现代理的缓存功能。



首先判断本地是否存在缓存文件，并获取缓存文件建立的时间

```
File cacheFile = new File("cache/" + (host + path).replace('/', '_').replace(':', '+')
 + ".cache");
                        boolean useCache = false;   // 标记是否用 cache
                        boolean existCache = cacheFile.exists() && cacheFile.length()
 != 0;
                        // 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 (existCache == true) {
                            // 文件存在且大小不为 0，说明访问内容被缓存过
                            System.out.println(visitAddr + "存在本地缓存文件");
                            // 获得修改时间
                            long time = cacheFile.lastModified();
                            SimpleDateFormat formatter = new SimpleDateFormat("EEE, dd
MMM yyyy HH:mm:ss 'GMT'", Locale.ENGLISH);
                            Calendar cal = Calendar.getInstance();
                            cal.setTimeInMillis(time);
                            cal.set(Calendar.HOUR, -7);
                            cal.setTimeZone(TimeZone.getTimeZone("GMT"));
                            lastModified = formatter.format(cal.getTime());
                            System.out.println("缓存建立时间：" + cal.getTime());
                        }
```

然后将缓存文件的时间加入<If-Modified-Since: 对象文件的最新被修改时间>的报文中，将报文传送的服务器端

```
StringBuffer requestBuffer = new StringBuffer();
                        requestBuffer.append(header);
                        requestBuffer.append("If-Modified-
   Since: ").append(lastModified).append("\n");


                        writer.write(requestBuffer.append("\n").toString()); // 发送报文
                        writer.flush();
```

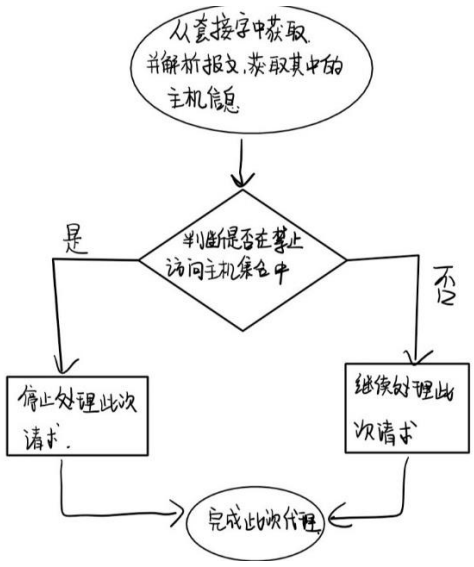
然后在返回的报文中，判断返回状态码，如果出现304 Not Modified且缓存文件长度不为0的话，就从本地的缓存文件中向浏览器返回报文，否则则说明缓存文件不存在，读取服务器返回的报文，并将其写入本地缓存共以供下一次的使用，并向浏览器返回报文。

```
// 从远程服务器获得输入的输入流
                         BufferedInputStream remoteInputStream = new BufferedInputStrea
m(proxySocket.getInputStream());
                        System.out.println("获取来自：" + host + "的输入流");
```

```
72.
73.                         // 先使用一个小字节缓存获得头部
74.                         byte[] tempBytes = new byte[20];
75.                         int len = remoteInputStream.read(tempBytes);
76.                         String res = new String(tempBytes);
77.                         useCache = (res.contains("304") || (res.contains(
78. "200"))) && cacheFile.length() != 0;
79.
80.                         System.out.println("HTTP 状态: " + res + "\n-----------------");
81.
82.                         if (useCache == true) {
83.                             // 用缓存
84.                             // 这是向浏览器输出的输出流
85.                             System.out.println(visitAddr + " 本地缓存尚未过期，使用缓存");
86.                             System.out.println(visitAddr + " 正在使用缓存加载,缓存长度
    " + cacheFile.length());
87.                             // 建立文件读写
88.                             FileInputStream fileInputStream = new Fil
89. eInputStream(cacheFile);
90.                             int bufferLength = 1;
91.                             byte[] buffer = new byte[bufferLength];
92.
93.                             while (true) {
94.                                 int count = fileInputStream.read(buffer);
95.                                 System.out.println(count + "从缓存中加载网
    页..." + visitAddr);
96.                                 if (count == -1) {
97.                                     System.out.println("从缓存中加载完成！");
98.                                     break;
99.                                 }
100.                                 clientOutput.write(buffer);
101.                             }
102.                             clientOutput.flush();
103.                         } else {
104.                             System.out.println(visitAddr + " 本地缓存过期或不可用，不使用缓存
    ");
105.                             clientOutput.write(tempBytes);
106.                         }
107.
108.                         FileOutputStream fileOutputStream =
109.                                 new FileOutputStream(
110.                                         ("cache/" + (host + path).replace('/', '_')
111.                                                 .replace(':', '+') + ".cache"));
112.                         if (!useCache) {
```

```
113.                         fileOutputStream.write(tempBytes);
114.                     }
115.                     int bufferLength = 1;
116.                     byte[] buffer = new byte[bufferLength];
117.                     while (true) {
118.                         int count = remoteInputStream.read(buffer);
119.                         if (count == -1) {
120.                             break;
121.                         }
122.                         if (!useCache) {
123.                             clientOutput.write(buffer);
124.                             fileOutputStream.write(buffer);
125.
126.                         }
127.                     }
128.                     fileOutputStream.flush();    // 输出到文件
129.                     fileOutputStream.close();    // 关闭文件流
130.
131.                     clientOutput.flush();    // 输出到浏览器
132.                     writer.close();
133.
134.                     remoteInputStream.close();
135.                     System.out.println(host + "代理已经完成！");
```

4.网站过滤：允许/不允许访问某些网站：



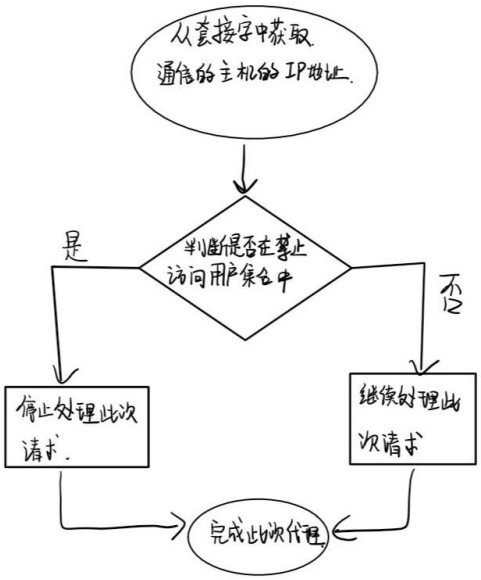
在代码中添加一个forbidHost 集合，并添加禁止访问的主机名，每当收到一个代理请求时，代理服务器提取请求报 文中的头部行获取目的主机，若是forbidHost 集合中指明的禁止访问的主机名，则向浏览器返回一个字符串，提示用户当前网页禁止访问，直接结束线程，关闭套接字，不再向服务器发送请求。

```
136. Map<String, String> map = parse(headerBuilder.toString());
```

```
137. host = map.get("host"); // host
138.                         path = map.get("path");
139.                         fullPath = map.get("fullPath");
140.                         // 端口
141.                         int visitPort = Integer.parseInt(map.getOrDefault("port", "80"));
142.                         // 访问的网站
143.                         String visitAddr = map.get("visitAddr");
144.                         // method
145.                         String method = map.getOrDefault("method", "GET");
146.                         if (forbidHost.contains(host)) {
147.                             System.out.println("访问了禁止访问的网站！");
148.                             PrintWriter pw = new PrintWriter(clientSocket.
149. getOutputStream());
150.                             pw.println("Website: " + visitAddr + " is forbid to v
151. isit because " + host + "  is a forbid host");
152.                             pw.close();
153.                             clientSocket.close();
154.                             return;
155.
156.                         }
```

5.用户过滤：

在代理服务器通过主套接字获取一个连接的时候，根据套接字中包含的IP地址信息来判断用户是否在禁止的列表中，在代码中添加一个forbidUser集合，并添加禁止访问的主机名，每当收到一个代理请求时，代理服务器提取请求报 文中的头部行获取目的主机，若是forbidUser集合中指明的禁止访问的用户IP，则向浏览器返回一个字符串，提示用户当前网页禁止访问，直接结束线程，关闭套接字，不再向服务器发送请求。

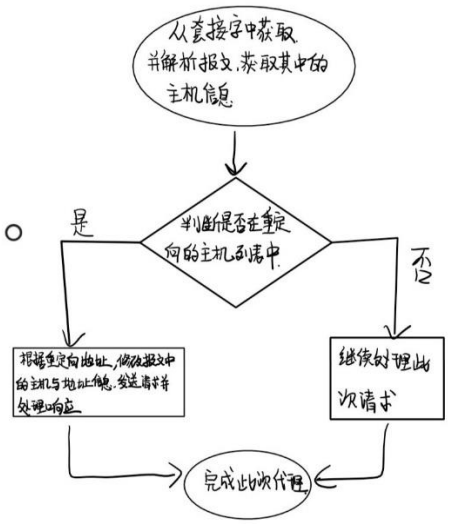



```
157. Socket clientSocket = server.accept();
158.             String UserIP = clientSocket.getInetAddress().getHostAddress();
```

```
159.             System.out.println("获取到一个连接！来自 " + UserIP);
160.             if (forbidUser.contains(UserIP)) {
161.                 System.out.println("用户:" + UserIP + "已被禁止");
162.                 PrintWriter pw = new PrintWriter(clientSocket.getOutputStream());
163.                 pw.println("Forbid User!");
164.                 pw.close();
165.                 clientSocket.close();
166.                 continue;
167.             }
```

6. 网站引导

维护三个映射redirectHostHostMap，redirectHostAddrMap,redirectAddrAddrMap，分布表示主机到主机，主机到地址，地址到地址的重定向，每当遇到一个HTTP访问请求的请求URL和host包含被重新定向的URL和host时，就在HTTP报文中替换对应的内容。



从主嵌套字的输入流获取客户的请求报文首行信息，提取出目的主机 host，并读取配置文件的每一行，判断HOST数据行是否含有该 Host，若含有该 Host，说明目的主机已被引导， 并提取出该行中 host 后面的信息，为要导向的目的网站。并构造新的请求报文， 即将头部行中的 URL 替换为引导主机对应的 URL，并将其转发给服务器端。

```
168. /**
169.     * 重定向主机
170.     *
171.     * @param oriHost 源主机
172.     * @return 重定向的主机
173.     */
174.    String redirectHost(String oriHost) {
175.        Set<String> keywordSet = redirectHostHostMap.keySet();
176.        for (String keyword : keywordSet) {
177.            if (oriHost.contains(keyword)) {
```

```
178.                 System.out.println("源主机: " + oriHost);
179.                 String redHost = redirectHostHostMap.get(oriHost);
180.                 System.out.println("重定向主机 Host: " + redHost);
181.                 return redHost;
182.             }
183.         }
184.         return oriHost;
185.     }
186.
187.     /**
188.      * 重定向地址
189.      * 根据 redirectAddrMap 中存放的 Host——Address Map 获取重定向地址
190.      *
191.      * @param oriHost 源主机
192.      * @return 重定向后的地址
193.      */
194.     String redirectAddr(String oriHost, String visitAddr) {
195.         Set<String> keywordSet = redirectAddrAddrMap.keySet();
196.         for (String keyword : keywordSet) {
197.             if (oriHost != null && keyword.contains(oriHost)) {
198.                 //直接跳转
199.                 if (visitAddr.equals(keyword)) {
200.                     return redirectAddrAddrMap.get(keyword);
201.                 }
202.                 if (visitAddr.contains(oriHost)) {
203.                     String[] temp = visitAddr.split(oriHost);  // 按空格分割
204.                     String redHost = redirectHostHostMap.get(oriHost);
205.                     String redAddr = temp[0] + redHost + temp[1];
206.                     return redAddr;
207.                 }
208.
209.             }
210.         }
211.         return visitAddr;
212.     }
213. /**
214.      * 重定向主机
215.      *
216.      * @param oriHost 源主机
217.      * @return 重定向的主机
218.      */
219.     String redirectHost(String oriHost) {
220.         Set<String> keywordSet = redirectHostHostMap.keySet();
221.         for (String keyword : keywordSet) {
```

```
            if (oriHost.contains(keyword)) {
                System.out.println("源主机: " + oriHost);
                String redHost = redirectHostHostMap.get(oriHost);
                System.out.println("重定向主机 Host: " + redHost);
                return redHost;
            }
        }
        return oriHost;
    }

    /**
     * 重定向地址
     * 根据 redirectAddrMap 中存放的 Host——Address Map 获取重定向地址
     *
     * @param oriHost 源主机
     * @return 重定向后的地址
     */
    String redirectAddr(String oriHost, String visitAddr) {
        Set<String> keywordSet = redirectAddrAddrMap.keySet();
        for (String keyword : keywordSet) {
            if (oriHost != null && keyword.contains(oriHost)) {
                //直接跳转
                if (visitAddr.equals(keyword)) {
                    return redirectAddrAddrMap.get(keyword);
                }
                if (visitAddr.contains(oriHost)) {
                    String[] temp = visitAddr.split(oriHost);  // 按空格分割
                    String redHost = redirectHostHostMap.get(oriHost);
                    String redAddr = temp[0] + redHost + temp[1];
                    return redAddr;
                }

            }
        }
        return visitAddr;
    }
```

在主进程中判断访问地址是否需要重定向：

```
System.out.println("初始地址: " + visitAddr);
                        String redirectHost = redirectHost(host);
                        if (!host.equals(redirectHost)) {
                            visitAddr = redirectAddr(host, visitAddr);
                            host = redirectHost;
```

```
                String[] temp1 = visitAddr.split("\\?");
                fullPath = temp1[0];
                String[] temp2 = fullPath.split(host);
                if (temp2.length > 1) {
                    path = temp2[1];
                } else {
                    path = "";
                }
                header = RedirectHTTP(header, host, visitAddr);
            }
```

实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

1.代理访问：

访问：http://today.hit.edu.cn/ 今日哈工大

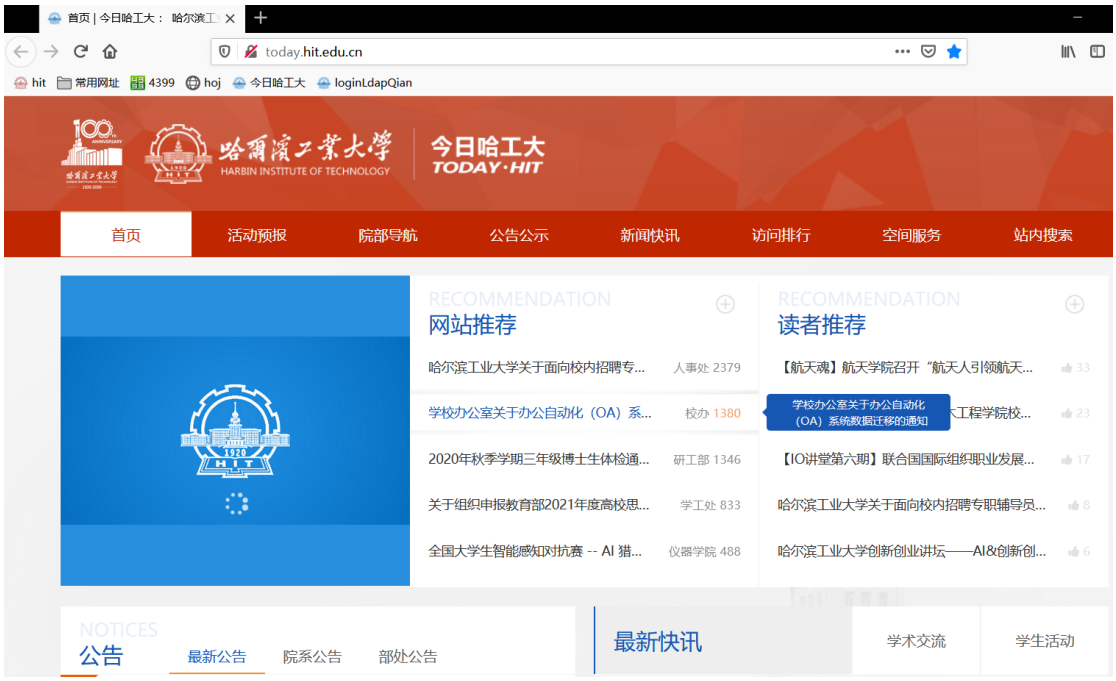
程序输出:

```
----------------
代理服务器获取的 HTTP 头： 长度 395
GET http://today.hit.edu.cn/ HTTP/1.1
Host: today.hit.edu.cn
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
----------------
初始地址：http://today.hit.edu.cn/
代理套接字已建立!:Socket[addr=today.hit.edu.cn/202.118.254.117,port=80,localport=8567]

----------------
代理服务器转发报文：
GET http://today.hit.edu.cn/ HTTP/1.1
Host: today.hit.edu.cn
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

```
25.
26. -----------------
27. 获取来自：today.hit.edu.cn 的输入流
28. HTTP 状态：HTTP/1.1 200 OK
```

浏览器输出：



访问 http://www.4399.com/ 4399小游戏
控制台输出：

```
1. 获取到一个连接！来自 127.0.0.1
2.
3. -----------------
4. 代理服务器获取的 HTTP 头： 长度 556
5. GET http://www.4399.com/ HTTP/1.1
6. Host: www.4399.com
7. User-
   Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
8. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
9. Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
10. Accept-Encoding: gzip, deflate
11. Connection: keep-alive
12. Cookie: home4399=yes; UM_distinctid=1757a13e58726-0131f5ccfb3e5a-4c3f2678-144000-
    1757a13e5886e1; CNZZDATA30039538=cnzz_eid%3D1246378072-1604070422-%26ntime%3D1604070422
13. Upgrade-Insecure-Requests: 1
14. -----------------
15. 初始地址：http://www.4399.com/
```

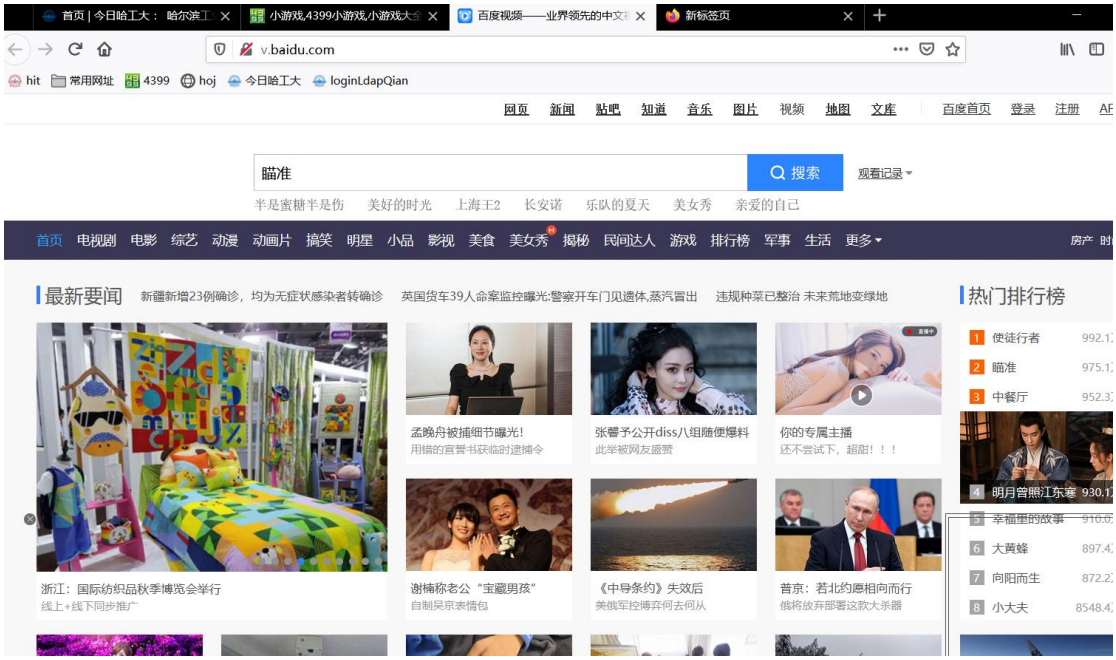
```
http://www.4399.com/存在本地缓存文件
缓存建立时间：Sat Oct 31 13:54:51 CST 2020
代理套接字已建立!:Socket[addr=www.4399.com/202.114.51.75,port=80,localport=8820]

-----------------
代理服务器转发报文：
GET http://www.4399.com/ HTTP/1.1
Host: www.4399.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: home4399=yes; UM_distinctid=1757a13e58726-0131f5ccfb3e5a-4c3f2678-144000-
1757a13e5886e1; CNZZDATA30039538=cnzz_eid%3D1246378072-1604070422-%26ntime%3D1604070422
Upgrade-Insecure-Requests: 1
If-Modified-Since: Sat, 31 Oct 2020 13:54:51 GMT

-----------------
获取来自：www.4399.com 的输入流
HTTP 状态：HTTP/1.1 304 Not Mod
```

浏览器输出：

访问http://v.baidu.com/ 百度视频



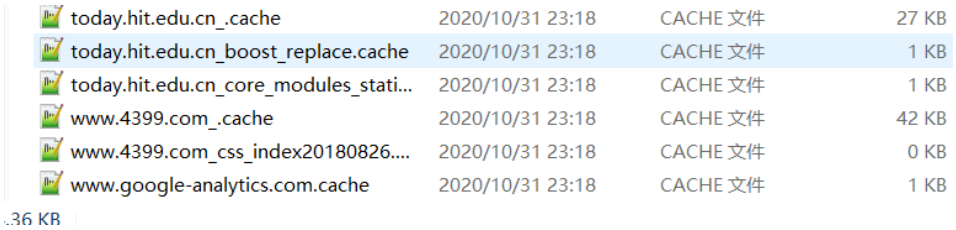
2. 实现代理服务器缓存功能

首先清空缓存文件夹

然后访问4399小游戏

因为本地没有缓存未见，则不使用缓存

然后观察本地文件，发现多出了缓存文件



关闭网页，再次访问，在终端中发现状态码 304，not mod

```
-----------------
获取来自：www.4399.com的输入流
HTTP 状态：HTTP/1.1 304 Not Mod
-----------------
http://www.4399.com/ 本地缓存尚未过期，使用缓存
http://www.4399.com/ 正在使用缓存加载,缓存长度311
从缓存中加载完成!
```

并用缓存加载完成

网页的显示也正常

3. 实现禁止访问网站

在代码中添加禁止访问主机：www.4399.com

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

    Proxy proxy = new Proxy();
    //proxy.forbidUser.add("127.0.0.1");
    proxy.forbidHost.add("www.4399.com");
//      proxy.forbidHost.add("api2.firefoxchina.cn");
```

然后打开代理服务器，用浏览器访问http://www.4399.com/，因为host：www.4399.com被我们禁止，所以网页提示此网页已被禁止访问

4. 实现禁止用户访问

（因为没有局域网环境，只能用127.0.0.1 本机进行测试）

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


    Proxy proxy = new Proxy();
    proxy.forbidUser.add("127.0.0.1");
    //proxy.forbidHost.add("www.4399.com");
```

在浏览器任意访问一个网页，都会提示禁止用访问

5. 实现钓鱼

在代码中添加重定向关系

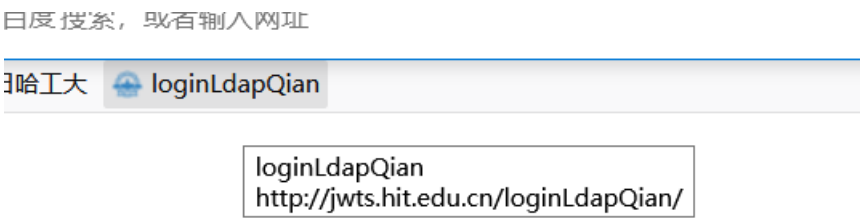
```
public static void main(String[] args) throws IOException {


    Proxy proxy = new Proxy();
    //proxy.forbidUser.add("127.0.0.1");
    //proxy.forbidHost.add("www.4399.com");
//      proxy.forbidHost.add("api2.firefoxchina.cn");
//
    //proxy.forbidHost.add("acm.hit.edu.cn");
    proxy.redirectHostHostMap.put( k: "jwts.hit.edu.cn",  v: "today.hit.edu.cn");
    proxy.redirectHostAddrMap.put( k: "jwts.hit.edu.cn",  v: "http://today.hit.edu.cn/");
    proxy.redirectAddrAddrMap.put( k: "http://jwts.hit.edu.cn/loginLdapQian/",  v: "http://today.hit.edu.cn/");

    proxy.execute();
}
```

进入浏览器测试

尝试访问：http://jwts.hit.edu.cn/loginLdapQian/



浏览器会跳转到今日哈工大网页

终端输出内容，展示了报文发生的内容改变

```
1. 获取到一个连接！来自 127.0.0.1
2.
3. -----------------
4. 代理服务器获取的 HTTP 头： 长度 556
5. GET http://www.4399.com/ HTTP/1.1
6. Host: www.4399.com
7. User-
   Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
8. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
9. Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
10. Accept-Encoding: gzip, deflate
11. Connection: keep-alive
12. Cookie: home4399=yes; UM_distinctid=1757a13e58726-0131f5ccfb3e5a-4c3f2678-144000-
    1757a13e5886e1; CNZZDATA30039538=cnzz_eid%3D1246378072-1604070422-%26ntime%3D1604070422
13. Upgrade-Insecure-Requests: 1
14. -----------------
15. 初始地址：http://www.4399.com/
```

```
http://www.4399.com/存在本地缓存文件
缓存建立时间：Sat Oct 31 13:54:51 CST 2020
代理套接字已建立!:Socket[addr=www.4399.com/202.114.51.75,port=80,localport=8820]

-----------------
代理服务器转发报文：
GET http://www.4399.com/ HTTP/1.1
Host: www.4399.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: home4399=yes; UM_distinctid=1757a13e58726-0131f5ccfb3e5a-4c3f2678-144000-
1757a13e5886e1; CNZZDATA30039538=cnzz_eid%3D1246378072-1604070422-%26ntime%3D1604070422
Upgrade-Insecure-Requests: 1
If-Modified-Since: Sat, 31 Oct 2020 13:54:51 GMT

-----------------
获取来自：www.4399.com 的输入流
HTTP 状态：HTTP/1.1 304 Not Mod
```

问题讨论：

对实验过程中的思考问题进行讨论或回答。

遇到的问题：

Java中流出现了阻塞问题，导致程序运行阻塞

这是因为缓冲区设置的太小了，有可能装不下输入流的内容，增加缓冲区大小或循环读入即可

构建HTTP报文后发送，无法直接收到响应

在转发报文的时候，少添加了一个回车，导致报文出错无法收到响应

心得体会：

结合实验过程和结果给出实验的体会和收获。

掌握了 Socket 编程有关的知识结构，较为深入的理解了应用层的 HTTP 协 议的相关内容。熟悉并掌握 Socket 网络编程的过程与技术；同时深入的理解了 HTTP 协议，掌握了 HTTP 代理服务器的基本工作原理，并实现了简单的的缓存 功能以及过滤和引导功能，同时掌握了 HTTP 代理服务器设计与编程实现的基本 技能。对这一章节的知识有了一个系统全面的了解和体会。

通过编程训练，具体实现了一个代理服务器，对Java中的流，套接字等内容有了更深入的了解，掌握了HTTP 请求报文的详细格式与内容，每项内容所实现的功能