

**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 | 黄友勤 | | 院系 | 计算机学院 | | |
| 班级 | 1603108班 | | 学号 | 1160300621 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | 格物207 | | 实验时间 | 2018.11.2 | | |
| 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 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 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。在指定端口能够接收来自客户的 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，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。  3.扩展 HTTP 代理服务器，支持如下功能：  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： |
| **Socket编程的客户端和服务器端主要步骤：**   1. **服务器端：**   (a)创建一个socket，服务器绑定(bind)套接字然后开始监听（listen）  (b)不断接收客户端的服务请求时,每一次请求都创建新的子线程进程处理。  (c)代理结束，关闭socket。  **2. 客户端：**  (a)为了访问网页，创建socket，并连接到服务器特定的端口。  (b)使用socket向服务器发送请求报文  (c)接收服务器返回的应答报文与数据  **HTTP代理服务器流程图：**    **HTTP代理服务器基本原理：**  代理服务器在某一个指定的端口，监听浏览器的访问请求信息，并且根据访问信息。代理服务器得到这一部分访问信息后，可以根据对应的URL判断出有没有缓存。   1. **如果没有缓存**：代理服务器向服务器转发客户端的请求信息，然后服务器再发给代理服务器以网页的对象(一般是字节流)。代理服务器再将这些对象文件转发给客户端，并且缓存对应的URL与对象文件。 2. **如果有缓存**：代理服务器需要向原Web服务器转发客户的请求报文时插入<If-Modified-Since:………>，来判断缓存的时间过后网页对象有没有被修改。   (a)被修改了：重新从服务器获得新的对象转发给客户端，并将新的缓存对象替换原来的缓存对象。  (b)没有修改：直接将缓存对象发送给客户端。  实验主要有三方进行协同工作，分别是:客户端、代理服务器和服务器。  **一、代理服务器接收客户端的套接字**   1. 当用户通过在web端通过浏览器访问网页的时候，它实际上会与服务器建立连接，并且创建套接字。而为了使代理服务器能够正常工作，需要配置代理。这里使用的是火狐浏览器：      1. 代理服务器监听端口10240，使用的是ServerSocket类的函数，指定监听端口port为10240。      1. 每当客户端创建并发送一个套接字时，代理服务器进行监听时会接收套接字。在程序体现如下图：      1. 由于代理服务器采用多线程的方式来提高效率，因此，在while循环中使用以下函数：     每次收到一个客户端的一个套接字，便开启一个新的线程进行处理。  **二、代理服务器处理客户端请求**  1．首先，为了读写方便，将客户端发送的套接字使用InputStream、OutputStream、BufferReader和PrintWriter进行封装。  2. 读取客户端的请求信息，并根据它提取出http请求的几个重要信息，如URL,host，method等等。存储在了HttpHeader类。     1. 创建代理服务器与服务器的套接字(考虑到有连接失败的情况，最多创建五次)。     4. 代理服务器转发客户端的请求信息，并转发服务器的响应报文与数据给客户端。    **三、增加缓存功能**  1.修改代码，使用HashMap存储缓存信息。    在提取出客户端的URL后，判断URL是否被缓存过。   1. 如果没有，则代理服务器转发的时候将时间(Data)与数据存储到创建的HashMap中。 2. 如果被缓存，将请求信息加上<if-Modified-Since>发送给服务器，查看对象是否被修改。     如果没有被修改，直接将缓存信息发给客户端。  否则，说明对象被修改了，那么获取服务器新的对象的字节流并转发给客户端，并在缓存进行更新。    **四、扩展HTTP代理服务器**  **1.网站过滤**  增加过滤网站，这里以jwts.hit.edu.cn为例。    代理服务器得到URL后判断是否与被过滤网站的网站相同，如果是，则不想真正的Web服务器转发客户端的请求信息即可。  **2.用户过滤**  增加要过滤的用户的IP    当代理服务器接收套接字时，判断发过来信息的IP地址是否属于被屏蔽的地址，如果是，则不处理其任何请求。  **3.网站引导(钓鱼)**  增加被屏蔽的网站：    这里以“jwes.hit.edu.cn”为例，如果代理服务器得到的URL是这样的IP地址，那么代理服务器会发送访问被钓鱼的网站的访问请求到真正的Web服务器。  其访问请求如下图：    然后代理服务器转发Web服务器对于这个钓鱼网站的信息给客户端，完成了钓鱼。 |
| 实验结果： |
| 1. **没有Cache情况下，访问今日哈工大(截图红色的No Cache代表没有缓存)：**        1. **有Cache情况下，访问今日哈工大(截图中红色的Cache In Memory代表没有缓存)：**       **发现打开网页的速度明显变快，并且程序中会返回说有Cache in Memory。**    **这时候分为：①缓存的信息在之后被修改了，发回的响应报文会是200(OK)，并发送对应的新的对象的字节流。**    **②缓存的信息没有被修改，返回的是Not Modified。**     1. **网站过滤：**     **当访问jwts.hit.edu.cn的时候，代理服务器拒绝转发请求信息，并发送“被过滤了”的信息给客户端。**   1. **用户过滤：**     **这里是将本机用户屏蔽**   1. **网站引导(钓鱼)：**     **访问的是“jwes.hit.edu.cn”，把它钓鱼到彩票网站。** |
| 问题讨论： |
| 一开始我对实验的原理一知半解，并不了解HTTP客户端、服务器还有代理服务器之间的关系。因此我反复查阅了课本与spoc，再看了很多博客，终于了解了它们之间如何协同作用的。HTTP服务器实质上就是用来转发、处理客户端的请求。  另外，如何将服务器的数据转发给客户端是一个大问题。一开始我发现服务器并不理会我发送的请求。后来经过不断尝试，我才发现，尽管请求信息与响应信息都可以使用string、char、byte等等来发送与接收，但是转发网页的对象文件时，必须用byte，用字节流进行转发。并且，转发完成后，必须使用flush，并在末尾加上”\r\n”才能生效。 |
| 心得体会： |
| 因为使用的是java语言编写，因此一开始的时候对于java如何进行网络编程并不了解。很多时候把原理弄清楚了后，还是有很多语言方面的问题。通过本次实验我对于如何使用java进行网络编程，有了很深刻的了解与体会。  另外，关于HTTP协议，我在本次实验中也对它有了更为深刻的理解。例如304 Not modified，200 OK以及常说的404 Not Found等等，我对于它们背后的逻辑有更为熟悉。  另外，在实现扩展功能的时候，对于“钓鱼”、屏蔽，通过实验我也很好地掌握了它们的原理，在服务器的运转方式方面积累了很多自己的经验。 |
| 源代码： |
| **①HttpHeader.java**  package cnExperiment1;  **public** **class** HttpHeader {  **protected** String method;  **protected** String url;  **protected** String host;  **protected** String cookie;  **public** HttpHeader(String method, String url, String host, String cookie) {  **super**();  **this**.method = method;  **this**.url = url;  **this**.host = host;  **this**.cookie = cookie;  }      }  **②Proxy.java**  package cnExperiment1;  import **java.io.IOException**;  import **java.net.ServerSocket**;  import **java.net.Socket**;  import **java.util.ArrayList**;  import **java.util.List**;  **public** **class** Proxy {  **private** **static** ServerSocket ProxyServerSocket;  **private** **static** List<String> UserFilter=**new** ArrayList<>();  **static** **boolean** InitSocket(**int** port) {  **try** {  ProxyServerSocket=**new** ServerSocket(port);  ProxyServerSocket.setSoTimeout(1000000);  } **catch** (IOException e) {  System.out.println("初始化ProxyServerSocket失败");  **return** **false**;  }  **return** **true**;    }  **static** **boolean** UserFilterAdd() {  UserFilter.add("127.0.0.1");  UserFilter.add("1.1.1.1");  UserFilter.add("2.2.2.2");  **return** UserFilter.size()>0;  }      **public** **static** **void** main(String[] args) {  **int** ProxyPort=10240;  System.out.println("---------正在准备代理服务器---------");  **if** (InitSocket(ProxyPort)) {  System.out.println("开始监听端口"+ProxyPort);  }      *//UserFilterAdd();*    **while** (**true**) {  **try** {  Socket socket=ProxyServerSocket.accept();    String address=socket.getInetAddress().getHostAddress();  **for** (**int** i=0;i<UserFilter.size();i++) {  **if** (address.equals((UserFilter.get(i)))) {  System.err.println("用户IP:"+address+"被屏蔽");  System.exit(0);  }  }    **new** Thread(**new** ProxyProcess(socket)).start();    } **catch** (IOException e) {  System.out.println("连接超时");  }    }  }  }  **③ProxyProcess.java**  package cnExperiment1;  import **java.io.BufferedReader**;  import **java.io.DataOutputStream**;  import **java.io.IOException**;  import **java.io.InputStream**;  import **java.io.InputStreamReader**;  import **java.io.OutputStream**;  import **java.io.PrintWriter**;  import **java.net.Socket**;  import **java.net.SocketException**;  import **java.net.UnknownHostException**;  import **java.util.ArrayList**;  import **java.util.HashMap**;  import **java.util.List**;  import **java.util.Map**;  */\**  *客户端上的使用*  *1.getInputStream方法可以得到一个输入流，客户端的Socket对象上的getInputStream方法得到输入流其实就是从服务器端发回的数据。*  *2.getOutputStream方法得到的是一个输出流，客户端的Socket对象上的getOutputStream方法得到的输出流其实就是发送给服务器端的数据。*  *服务器端上的使用*  *1.getInputStream方法得到的是一个输入流，服务端的Socket对象上的getInputStream方法得到的输入流其实就是从客户端发送给服务器端的数据流。*  *2.getOutputStream方法得到的是一个输出流，服务端的Socket对象上的getOutputStream方法得到的输出流其实就是发送给客户端的数据。*  *\*/*  **public** **class** ProxyProcess **implements** Runnable {  *// http端口，默认为80*  **static** **int** HttpPort = 80;  **static** **int** size=100000;  *// 超时时间*  **static** **int** timeout = 500000;  *//客户端和代理服务器的socket*  **private** Socket ClientSocket = **null**;  **private** Socket ProxyClientSocket = **null**;  *//socket的写入写出方法包装*  **private** InputStream ClientInputStream = **null**;  **private** InputStream ProxyInputStream = **null**;  **private** BufferedReader ClientBufferReader = **null**;  **private** BufferedReader ProxyBufferReader = **null**;  **private** OutputStream ClientOutputStream = **null**;  **private** OutputStream ProxyOutputStream = **null**;  **private** PrintWriter ClientPrintWriter = **null**;  **private** PrintWriter ProxyPrintWriter = **null**;  *// 对象被缓存的具体时间*  **static** Map<String, String> cacheTime = **new** HashMap<>();  *// 对象被缓存的具体数据*  **static** Map<String, List<Byte>> cacheBytes = **new** HashMap<>();  **static** List<String> WebsiteFilter=**new** ArrayList<>();  **static** Map<String,String> guide=**new** HashMap<>();      **public** ProxyProcess(Socket clientsocket) **throws** IOException {  **super**();  **this**.ClientSocket = clientsocket;  ClientInputStream = clientsocket.getInputStream();  ClientBufferReader = **new** BufferedReader(**new** InputStreamReader(ClientInputStream));  ClientOutputStream = clientsocket.getOutputStream();  ClientPrintWriter = **new** PrintWriter(ClientOutputStream);    WebsiteFilter.add("jwts.hit.edu.cn");  guide.put("jwes.hit.edu.cn", "http://today.hit.edu.cn/");    }  */\*\**  *\* 解析http头的信息，获取method，url，host，cookie*  *\*/*  **public** HttpHeader parse(List<String> header) {  String firstLine = header.get(0);  String method = **null**;  String url = **null**;  String host = **null**;  String cookie = **null**;  **if** (firstLine.charAt(0) == 'G') {  method = "GET";  url = firstLine.substring(4, firstLine.length() - 9);  } **else** **if** (firstLine.charAt(0) == 'P') {  method = "POST";  url = firstLine.substring(5, firstLine.length() - 9);  } **else** {  method = "CONNECT";  }  **for** (**int** i = 0; i < header.size(); i++) {  **if** (header.get(i).startsWith("Host")) {  host = header.get(i).substring(6, header.get(i).length());  } **else** **if** (header.get(i).startsWith("Cookie")) {  cookie = header.get(i).substring(8, header.get(i).length());  }  }  HttpHeader httpHeader = **new** HttpHeader(method, url, host, cookie);  **return** httpHeader;  }  */\*\**  *\* 获取代理服务器和服务器套接字*  *\* @param host 主机名*  *\* @param port 端口*  *\* @param times 连接次数*  *\* @return 代理服务器与服务器的Sockek*  *\* @throws UnknownHostException*  *\* @throws IOException*  *\*/*  **public** Socket ConnectToServer(String host, **int** port, **int** times) **throws** UnknownHostException, IOException {  **for** (**int** i = 0; i < times; i++) {  ProxyClientSocket = **new** Socket(host, port);  ProxyClientSocket.setSoTimeout(timeout);  ProxyInputStream = ProxyClientSocket.getInputStream();  ProxyBufferReader = **new** BufferedReader(**new** InputStreamReader(ProxyInputStream));  ProxyOutputStream = ProxyClientSocket.getOutputStream();  ProxyPrintWriter = **new** PrintWriter(ProxyOutputStream);  **if** (ProxyClientSocket != **null**) {  **return** ProxyClientSocket;  }  }  **return** **null**;  }  */\*\**  *\* 代理服务器向服务器发送请求信息*  *\* @param lst 请求信息*  *\*/*  **public** **void** SendToServer(List<String> lst) {  System.out.println("\n---------Send Request to Server---------");  **for** (**int** i = 0; i < lst.size(); i++) {  String line = lst.get(i);  ProxyPrintWriter.write(line + "\r\n");  System.out.println(line);  }  ProxyPrintWriter.write("\r\n");  ProxyPrintWriter.flush();  }    */\*\**  *\* 没有缓存的情况下，代理服务器从服务器转发响应信息到客户端*  *\* @param url*  *\* @return*  *\*/*  **public** **boolean** SendBackToClient(String url) {  */\**  *\* 必须采用bytes数组，否则由于ASCII码与unicode编码的差异，无法识别*  *\*/*    System.out.println("\n---------Retransmission From Server To Client---------");    List<Byte> lst=**new** ArrayList<>();    **try** {  *// String time = null;*  **byte** bytes[] = **new** **byte**[size];  **int** len;  **while** (**true**) {  **if** ((len = ProxyInputStream.read(bytes)) >= 0) {  ClientOutputStream.write(bytes, 0, len);  **for** (**int** i=0;i<len;i++) {  lst.add(bytes[i]);  }  } **else** **if** (len < 0) {  **break**;  }  }    **byte** b[]=**new** **byte**[lst.size()];  **for** (**int** i=0;i<lst.size();i++) {  b[i]=lst.get(i);  }    String s=**new** String(b);  String time=findTime(s);  cacheTime.put(url, time);  cacheBytes.put(url, lst);    ClientPrintWriter.write("\r\n");  ClientPrintWriter.flush();  ClientOutputStream.close();  } **catch** (IOException e) {  } **catch** (Exception e) {  }  **return** **true**;  }    */\*\**  *\* 根据字符串获取其中的Date时间*  *\* @param s*  *\* @return*  *\*/*  **public** String findTime(String s) {  **int** begin=s.indexOf("Date");  **int** end=s.indexOf("GMT");  *//System.out.println(s.substring(begin+6, end+3));*  **return** s.substring(begin+6, end+3);  }  */\*\**  *\* 有缓存的情况下，给客户端需要的信息*  *\* 1.在缓存时间后服务器没有修改对象，则将缓存直接发送给客户端*  *\* 2.在缓存时间后服务器修改对象了，则将Sever的新对象发送给客户端*  *\* @param header*  *\* @param host*  *\* @param url*  *\* @return*  *\*/*  **public** **boolean** SendBackToClientWithCache(List<String> header,String host, String url) {  String modifiTime=cacheTime.get(url);  *// 发送确认是否修改的报文到服务器*  ProxyPrintWriter.write(header.get(0) + "\r\n");  ProxyPrintWriter.write("Host: "+host + "\r\n");    System.out.println("Modified Time:"+modifiTime);  String str = "If-modified-since: " + modifiTime + "\r\n";  ProxyPrintWriter.write(str);  ProxyPrintWriter.write("\r\n");  ProxyPrintWriter.flush();  **try** {  String ServerMessage = ProxyBufferReader.readLine();  *//System.out.println(ServerMessage);*  **if** (ServerMessage == **null**) {  **return** **false**;  }  System.out.println("Server Message First Line:"+ServerMessage);  *// 如果服务器在缓存时间后未修改对象，直接转发给客户端缓存*  **if** (ServerMessage.contains("Not Modified")) {  List<Byte> lst=cacheBytes.get(url);  **byte** bytes[]=**new** **byte**[lst.size()];  **for** (**int** i=0;i<lst.size();i++) {  bytes[i]=lst.get(i);  }  ClientOutputStream.write(bytes);  ClientPrintWriter.write("\r\n");  ClientPrintWriter.flush();  ClientPrintWriter.close();  }  *//如果修改过对象，则将新的对象按字节发给客户端*  **else** if (ServerMessage.contains("OK")) {  DataOutputStream d=**new** DataOutputStream(ClientOutputStream);  **byte**[] b=(ServerMessage+"\r\n").getBytes();  d.write(b);  *//ClientPrintWriter.write("\r\n");*  **byte** bytes[] = **new** **byte**[size];  **int** len;  **while** (**true**) {  **if** ((len = ProxyInputStream.read(bytes)) > 0) {  ClientOutputStream.write(bytes, 0, len);  } **else** **if** (len < 0) {  **break**;  }  }    cacheTime.remove(url);  cacheBytes.remove(url);  *//cacheTime.put(url, time);*  *//cacheBytes.put(url, lst);*    ClientPrintWriter.write("\r\n");  ClientPrintWriter.flush();  ClientOutputStream.close();  }  **else** {  cacheBytes.remove(url);  cacheTime.remove(url);  **while** (!ProxyBufferReader.readLine().equals("")) {  ;  }  **byte** bytes[] = **new** **byte**[size];  **int** len;  **while** (**true**) {  **if** ((len = ProxyInputStream.read(bytes)) >= 0) {  *//ClientOutputStream.write(bytes, 0, len);*  } **else** **if** (len < 0) {  **break**;  }  }  run();  }  } **catch** (IOException e1) {  }  **return** **true**;  }    **void** Filter() {  **for** (**int** i=1;i<419;i++) {  ClientPrintWriter.write("被过滤了\t");  **if** (i%11==0) {  ClientPrintWriter.write("\r\n");  }  }  *//ClientPrintWriter.write("被过滤了\t被过滤了 被过滤了 被过滤了 被过滤了 被过滤了\r\n");*  ClientPrintWriter.write("\r\n");  ClientPrintWriter.flush();  ClientPrintWriter.close();  }    **private** **void** phishing(List<String> header){  */\**  *\* 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:63.0) Gecko/20100101 Firefox/63.0*  *\* Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*//\*;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*  *\*/*  header.clear();  header.add("GET http://www.lottery.gov.cn/ HTTP/1.1");  header.add("Host: www.lottery.gov.cn");  header.add("User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:63.0) Gecko/20100101 Firefox/63.0");  header.add("Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8");  header.add("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");  header.add("Accept-Encoding: gzip, deflate");  header.add("Connection: keep-alive");  header.add("Upgrade-Insecure-Requests: 1");  HttpHeader httpHeader=parse(header);    String host=httpHeader.host;  String url=httpHeader.url;  */\**  *\* 获取代理服务器与服务器的Socket*  *\*/*  **try** {  **if** (ConnectToServer(host, HttpPort, 5) == **null**) {  **return**;  }  } **catch** (UnknownHostException e) {  } **catch** (IOException e) {  }  System.out.println("url="+url);  System.out.println("host="+host);    **boolean** flag=cacheTime.containsKey(url)&&cacheBytes.containsKey(url);  **if** (!flag) {  */\**  *\* 没有Cache的情况*  *\*/*  System.err.println("\n---------No Cache---------");  SendToServer(header);  SendBackToClient(url);  } **else** {  */\**  *\* 有Cache的情况*  *\*/*  System.err.println("\n---------Cache in Memory---------");  SendBackToClientWithCache(header, host,url);  }    }  @Override  **public** **void** run() {  **try** {  ClientSocket.setSoTimeout(timeout);  String line = **null**;  List<String> header = **new** ArrayList<>();  */\**  *\* 获取从客户端发送的请求信息*  *\*/*  line = ClientBufferReader.readLine();  **if** (line == **null**) {  **return**;  }  header.add(line);  System.out.println("\n---------Request From Client---------");  System.out.println(line);  **while** (!(line = ClientBufferReader.readLine()).equals("")) {  header.add(line);  System.out.println(line);  }    *//解析报文信息获取http信息*  HttpHeader httpHeader = parse(header);  String url = httpHeader.url;  String host = httpHeader.host;  */\**  *\* 钓鱼*  *\*/*  **for** (String h:guide.keySet()) {  **if** (host.contains(h)) {  phishing(header);  **return**;  }  }    **if** (httpHeader.method.equals("CONNECT")) {  **return**;  }  */\**  *\* 过滤网站*  *\*/*  **for** (**int** i=0;i<WebsiteFilter.size();i++) {  **if** (host.contains(WebsiteFilter.get(i))) {  Filter();  System.err.println(url+"has been filterred");  **return**;  }  }      */\**  *\* 获取代理服务器与服务器的Socket*  *\*/*  **if** (ConnectToServer(host, HttpPort, 5) == **null**) {  **return**;  }  System.out.println("url="+url);  System.out.println("host="+host);    **boolean** flag=cacheTime.containsKey(url)&&cacheBytes.containsKey(url);  **if** (!flag) {  */\**  *\* 没有Cache的情况*  *\*/*  System.err.println("\n---------No Cache---------");  SendToServer(header);  SendBackToClient(url);  } **else** {  */\**  *\* 有Cache的情况*  *\*/*  System.err.println("\n---------Cache in Memory---------");  SendBackToClientWithCache(header, host,url);  }  } **catch** (SocketException e) {  } **catch** (IOException e) {  }**catch** (Exception e){    }  }  } |