

**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 | 姜昕阳 | | 院系 | 计算学部 | | |
| 班级 | 2203201 | | 学号 | 2022111742 | | |
| 任课教师 | 詹东阳 | | 指导教师 | 詹东阳 | | |
| 实验地点 | 格物207 | | 实验时间 | 2024.4.3 | | |
| 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。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。  (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： |
| （1）浏览器使用代理  为了使浏览器访问网址时通过代理服务器，必须进行相关设置，以  IE 浏览器设置为例：打开浏览器工具浏览器选项连接局域网设  置代理服务器  使用8080端口  c8b39e03bc22ae27166bb9a66183f26  （2）Socket编程的客户端和服务器端的主要步骤：  服务器端：  1. 启动服务器，并绑定一个本地端口（8080），提供服务。  2. 开始监听客户端的连接请求。  3. 当收到客户端的连接请求时，创建一个新的套接字与客户端通信。  4. 将套接字放入新线程中进行处理，以避免阻塞主进程。  5. 主进程继续监听客户端的新请求。  客户端：  1. 创建套接字，连接到服务器端指定的IP地址和端口号。  2. 向服务器发送请求报文。  3. 等待获取服务器端的响应报文。  4. 处理响应数据，完成通信过程。  5. 关闭套接字，并释放资源。  （3）HTTP代理服务器基本原理  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. HTTP 代理服务器的程序流程图   58b33182a7bff514a6dbb7ece963040  （5）HTTP 代理服务器的关键技术及解决方案  1. 引入必要的包：导入了java.io和java.net的包，以便实现Socket通信和对数据流的操作。  2. 创建ServerSocket：使用ServerSocket类创建一个服务器套接字，并指定端口号为8080。  3. 监听连接：通过调用ServerSocket的accept()方法，服务器开始监听来自客户端的连接请求。  4. 接受连接：一旦有客户端连接，accept()方法将返回一个Socket对象，代表与客户端的通信套接字。  5. 处理连接请求：服务器会为每个连接请求创建一个新的线程，在该线程中处理客户端的请求。具体处理过程如下：  a. 输出连接信息：输出客户端连接的信息，包括主机名和IP地址。  b. 屏蔽用户：根据ifBlockUser变量的值判断是否需要屏蔽用户。如果需要屏蔽用户，会检查客户端的IP地址，如果IP地址符合屏蔽条件，则向客户端返回一个HTTP响应，表示禁止访问网站，并且关闭连接。  c. 解析请求头：通过BufferedReader逐行读取客户端发送的HTTP请求头信息，获取请求类型（GET或CONNECT）、URL以及Host等相关信息。  d. 判断是否钓鱼网站：如果请求的URL符合钓鱼网站的条件，则修改目标地址和请求头，以钓鱼网站的形式返回响应。  e. 处理特定网站访问限制：如果请求的URL符合特定网站的访问限制条件，则向客户端返回一个HTTP响应，表示禁止访问该网站，并且关闭连接。  f. 缓存请求：将请求类型、目标主机、URL、目标端口、请求头等信息写入缓存。  g. 关闭连接：处理完毕后关闭与客户端的连接。  6. 循环监听：使用一个无限循环，使服务器一直保持监听状态，接受并处理客户端的连接请求。  7. 异常处理：使用try-catch语句对可能发生的IOException进行捕获和处理。  用户过滤：faea171f703fd9121dd8225807767a7  钓鱼网站：faea171f703fd9121dd8225807767a7  网站过滤： 94d03bf10e7ef26929097be1a8dc2ed  查找缓存：dff58481d8f18f890ef7b2e0643cb90  更新缓存：提取缓存中的日期，构造报文向服务器发送if-modified-since信息，若返回304，则说明缓存文件已经最新，直接转发给客户端；若返回200，则需要更新缓存，将服务器的响应报文存入缓存，同时转发给客户端。  653ed7f00b3845f218d94cbcf0ce678 |
| 实验结果： |
| HTTP 代理服务器实验验证过程以及实验结果  访问网页，若缓存不存在，则访问该网页并写入缓存中  **9197518d39874f83edc9a6361f11ed5**  若存在，缓存内容未更新则直接使用缓存进入页面，若缓存更新则重新缓存并使用5c592f290ee7bb165995c2be7ac19fa  29b83fe808c8baaae04718785f5c4ae  缓存文件夹如下  9877c5e29a1ac29a817ee7a46fb6ccb  网站过滤：  c6ee30aadbae626e808e2aced66c0d0b80e8bacd4f16b20c0b21ae8587193b  钓鱼网站：  e7c12360a4ca3de6f84aabd2d5b6d2aca2b04eb417f430eaa977f732612758  用户过滤：  bc96aac440bd052b974cb023f35fb6faa03aee2df90eb6b6018ce70ef03691  源码：  import java.io.\*; import java.net.ServerSocket; import java.net.Socket;  public class socketServer {  public static writeCache cacheWriter = new writeCache();  public static void main(String[] args) {  try {  int portID = 8080; //端口号设置为8080  ServerSocket server = new ServerSocket(portID); //对于接受连接的服务器，Java提供了一个ServerSocket类表示服务器Socket  System.out.println("启动服务器");  System.out.println("客户端:" + server.getInetAddress().getLocalHost() + "已连接到服务器");//获取主机名和IP地址  int indexBk = 0;  while (true) {  indexBk ++;  Socket s = server.accept();  System.out.println("第" + indexBk + "条请求:");//输出代理服务器接受的第几条请求  OutputStream output = null;  InputStream input = null;  output = s.getOutputStream();  input = s.getInputStream();    boolean ifBlockUser = true;//此变量为true,则屏蔽上网权限  if (ifBlockUser) {  if (s.getInetAddress().getLocalHost().getHostAddress().equals("172.20.48.121")) {  String body = "<h1>Forbidden visit web!</h1>";//禁止上网  String response = "HTTP/1.1 200 ok\r\n" +  "Content-Length: " + body.getBytes().length + "\r\n" +  "Content-Type: textml; charset-utf-8\r\n" +  "\r\n" +  body + "\r\n"; //返回给浏览器的响应  System.out.println("禁止该用户访问网站");  output.write(response.getBytes());  //按照协议,将返回请求由outputStream写入,会根据指定的decode编码返回某字符串在该编码下的byte数组表示  output.flush();  break;  }  }    String type = null, destHost = null, URL = null, destAddr = null;  int destPort = 80;  int flag = 1;  StringBuilder header = new StringBuilder();  BufferedReader br = new BufferedReader(new InputStreamReader(input));  String readLine = br.readLine();   while (readLine != null && !readLine.equals("")) {  if (flag == 1) { //第一行获取请求类型（connect/get）和URL  type = readLine.split(" ")[0];  URL = readLine.split(" ")[1];  flag = 0;  }  String[] s1 = readLine.split(": ");   for (int i = 0; i < s1.length; i++){  if (s1[i].equals("Host")) {  destAddr = s1[i + 1];  }  }  header.append(readLine).append("\r\n");  readLine = br.readLine();  }//解析首部完成   if (URL == null) {  continue;  }  if (URL.contains("443")) {  System.out.println(URL+" 是 https");  continue;  }   if(URL.contains("www.lib.hit.edu.cn")) {  System.out.println("钓鱼成功!");  destAddr = "hituc.hit.edu.cn";  header = new StringBuilder("GET http://hituc.hit.edu.cn/ HTTP/1.1\n"+  "Host: hituc.hit.edu.cn\n" +  "Proxy-Connection: keep-alive\n" +  "Upgrade-Insecure-Requests: 1\n" +  "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36 Edg/105.0.1343.53\n" +  "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9\n" +  "Accept-Encoding: gzip, deflate\n" +  "Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6\n"+"\n");  }//钓鱼网站   if (destAddr.split(":").length > 1) {//代表包含端口号  destPort = Integer.valueOf(destAddr.split(":")[1]);  }  destHost = destAddr.split(":")[0];  //获取目的地址和端口号  header.append("\r\n");   if (destAddr.contains("yzb.hit.edu.cn")) {//将jwc.hit.edu.cn作为过滤对象  String body = "<h1>You can not visit this site!</h1>";//你不许访问这个网站  String response = "HTTP/1.1 200 ok\r\n" +  "Content-Length: " + body.getBytes().length + "\r\n" +  "Content-Type: textml; charset-utf-8\r\n" +  "\r\n" +  body + "\r\n";  System.out.println("用户访问了yzb.hit.edu.cn，已拒绝");  output.write(response.getBytes());//按照协议，将返回请求由outputStream写入  output.flush();  s.close();  continue;  }   System.out.println(type + "->" + URL);  System.out.println("-----------");   cacheWriter.writeInCache(type, destHost, URL, destPort, indexBk, output, header);  s.close();  }   } catch (IOException e) {  e.printStackTrace();  }  } }  import java.io.\*; import java.net.Socket; import java.util.Scanner;   public class writeCache {  public writeCache(){  }  public static void writeInCache(String type, String destHost, String URL,  int destPort, int indexBk, OutputStream output, StringBuilder header){  try{  if (destHost != null && !destHost.equals("")) {  Socket proxy = new Socket(destHost, destPort);  OutputStream proxyOut = proxy.getOutputStream();  InputStream proxyIn = proxy.getInputStream();  if (type.equals("GET")) {  boolean ifHasFile = false;  System.out.println(destHost);  String dir = "C:\\Users\\27186\\IdeaProjects\\untitled12\\src\\cache\\" + destHost + indexBk + ".txt";  File file = new File(dir);  File folder = new File("C:/Users/27186/IdeaProjects/untitled12/src/cache");  for (File file0 : folder.listFiles()) {  BufferedReader fileReader = new BufferedReader(new InputStreamReader(new FileInputStream(file0)));  if (fileReader.readLine().equals(URL)) {  System.out.println(URL + "--> 存在缓存");  ifHasFile = true;  file = file0;  break;  }  }   if (ifHasFile) {  Scanner sc = new Scanner (new FileReader(file));  String line, date = null;  while(sc.hasNextLine()) {  line = sc.nextLine();  if (line.contains("Date")) {  date = line.substring(6);  }  }  StringBuffer ifGetReqBuffer = new StringBuffer();  ifGetReqBuffer.append("GET " + URL + " HTTP/1.1\r\n");  ifGetReqBuffer.append("Host: " + destHost + ":" + destPort + "\r\n");  ifGetReqBuffer.append("If-modified-since: " + date + "\r\n");  ifGetReqBuffer.append("\r\n");  String ifGetReq = ifGetReqBuffer.toString();  proxyOut.write(ifGetReq.getBytes());  proxyOut.flush();  byte[] tempBytes = new byte[30];  int len = proxyIn.read(tempBytes);  String res = new String(tempBytes, 0, len);  if (res.contains("304")) {  System.out.println("缓存内容未更新，直接使用");  BufferedInputStream inputStream = new BufferedInputStream(new FileInputStream(file));  String pad = URL + "\r\n";  inputStream.read(pad.getBytes());  int length = -1;  byte[] bytes = new byte[1024];  proxy.shutdownOutput();  while ((length=inputStream.read(bytes)) != -1){  output.write(bytes, 0, length);  }  output.flush();  } else {  System.out.println("缓存内容更新，重新缓存并使用");  file.delete();  file.createNewFile();  FileOutputStream outputStream = new FileOutputStream(file);  String pad = URL + "\r\n";  outputStream.write(pad.getBytes("utf-8"));//将URL作为缓存文件的第一行  outputStream.write(tempBytes, 0, len);  output.write(tempBytes, 0, len);  proxy.shutdownOutput();  BufferedInputStream inputStream = new BufferedInputStream(proxyIn);  byte[] buf = new byte[1024];  int size = 0;  while (( size = inputStream.read(buf)) != -1) {  output.write(buf,0,size);  outputStream.write(buf,0,size);  }  output.flush();  outputStream.flush();  }  }  else {  proxyOut.write(header.toString().getBytes("utf-8"));  proxyOut.flush();  proxy.shutdownOutput();  //如果缓存不存在，则将服务器传来的数据传给客户端并且缓存  System.out.println(URL + "--> 缓存不存在");  file.createNewFile();  FileOutputStream outputStream = new FileOutputStream(file);  String pad = URL + "\r\n";  outputStream.write(pad.getBytes("utf-8"));//将URL作为缓存文件的第一行  BufferedInputStream inputStream = new BufferedInputStream(proxyIn);  byte[] buf = new byte[1];  int size = 0;  while (( size = inputStream.read(buf)) != -1) {  output.write(buf, 0, size);  outputStream.write(buf, 0, size);  }   output.flush();  outputStream.flush();  }  }  } else {  Socket proxy = new Socket(destHost, destPort);  OutputStream proxyOut = proxy.getOutputStream();  InputStream proxyIn = proxy.getInputStream();  proxyOut.write(header.toString().getBytes());  proxyOut.flush();  proxy.shutdownOutput();  BufferedInputStream inputStream = new BufferedInputStream(proxyIn);  byte[] buf = new byte[1];  int size = 0;  while (( size = inputStream.read(buf)) != -1) {  output.write(buf, 0, size);  }  output.flush();  }  } catch (IOException ioe){  ioe.printStackTrace();  System.exit(-1);  }  } } |
| 问题讨论： |
| 如何支持HTTPS协议？是否可以使用SSL/TLS来加密通信？  要支持HTTPS协议，服务器端需要进行SSL/TLS配置，并在代码中使用相应的库来处理加密通信。这样可以确保客户端和服务器之间的通信是安全的，数据不会被窃取或篡改。 |
| 心得体会： |
| 1. 理解了Socket编程的基本原理：通过这个实验，我深入了解了Socket编程的基本原理和工作机制。了解了如何创建服务器套接字、监听连接请求、接受客户端连接，并通过输入输出流实现与客户端的数据交互。  2. 深入了解了HTTP协议：在处理客户端请求的过程中，我需要解析HTTP请求头部信息，包括请求类型、URL、Host等字段。这让我更加深入地了解了HTTP协议的结构和常用字段，对Web通信有了更深入的理解。  3. 掌握了简单的请求处理逻辑：通过实验中的条件判断和请求处理过程，我学会了如何根据特定条件对客户端的请求进行过滤、拦截或转发。这对于实际应用中的网络安全、访问控制等方面具有重要意义。  总的来说，这个实验不仅帮助我巩固了网络编程和HTTP协议的相关知识，还提高了我的编程能力和问题解决能力。通过实践，我深刻体会到了理论知识的实际应用，为我今后的学习和工作打下了坚实的基础。 |