

**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 | 郭子阳 | | 院系 | 计算机学院 | | |
| 班级 | 1703101 | | 学号 | 1170300520 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | 格物207 | | 实验时间 | 2019.10.26 | | |
| 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。（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： |
| **HTTP网络应用通信原理**  在HTTP网络应用中，通信的两个进程主要采用客户端/服务器模式（或浏览器/服务器模式），客户端向服务器发送请求，服务器接收到客户端请求后，向客户端提供相应的服务。通信过程如下：    服务器端：   1. 服务器端需要首先启动，并绑定一个本地主机端口，在端口上提供服务 2. 等待客户端请求 3. 接收到客户端请求时，建立起与客户端通信的套接字，开启新线程，将与客户端通信的套接字放入新线程处理 4. 返回第二步，主线程继续等待客户端请求。 5. 关闭服务器   客户端：   1. 根据服务器IP与端口，建立起与服务器通信的socket 2. 向服务器发送请求报文，并等待服务器应答 3. 请求结束后关闭socket   HTTP代理服务器原理  RFC 7230规定，代理在HTTP通信中扮演一个中间人的角色，对于连接来的客户端来说，它扮演一个服务器的角色；对于要连接的远程服务器，它扮演一个客户端的角色。代理服务器就负责在客户端和服务器之间转发报文。如下图所示：  图片包含 屏幕截图  描述已自动生成  代理服务器在指定端口监听浏览器的请求，在接收到浏览器的请求时，首先查看浏览器的IP地址，如果来自被限制的IP地址，就向客户端返回错误信息。否则，从请求头中解析出请求的host主机，如果属于不允许访问的主机，则向客户端返回错误信息，如果属于需要引导的host，则修改请求头内所有的地址字段为被引导的地址。之后，根据URL查找是否缓存中是否有该URL的缓存，如果存在，则从中取出Last-modified头部内容，并构造包含If-modified-since的请求头，向服务器发送确认最新版本的报文，并在返回的请求头第一行里确认是否有“Not Modified”，如果存在该字段，则说明本地缓存未过期，直接将本地缓存内容发送给客户端，否则缓存过期，将服务器的报文直接写回客户端。如果缓存中不存在，就直接将客户端请求转发到服务器，并将服务器返回内容缓存后，再返回给客户端。  代理服务器的拦截用户、拦截主机和钓鱼信息都预先配置在配置文件里，并在程序运行后读入程序中，以按照规则执行。  程序运行流程图如下：    **代码实现：**  主线程读取配置文件，循环接收客户端请求，并将其放入新线程处理：  ConfigUtils.readConfig();  Configuration configuration = Configuration.getInstance();  try {  *// 开启服务端Socket监听*     ServerSocket serverSocket = new ServerSocket(configuration.getServerPort());     System.out.println("监听端口：" + configuration.getServerPort());     while(true) {  *// 客户端连接后创建线程处理*         Socket socket = serverSocket.accept();         new Thread(new HttpHandler(socket)).start();     }  } catch (IOException *e*) {     e.printStackTrace();  }  在子线程中，首先解析客户端请求头，获取url和host等信息：  while ((line = LineReader.readLine(toClientReader)) != null) {  if (line.startsWith("GET") ||  line.startsWith("POST") ||  line.startsWith("CONNECT")) {  url = line.split(" ")[1];  }     if (line.startsWith("Host")) {         hostString = line.split(" ")[1];     }     requestBuilder.append(line).append("\r\n");  }  requestBuilder.append("\r\n");  拦截不允许访问的用户：  if(configuration.getBlockedUsers().contains(  socketToClient.getInetAddress().getHostAddress())) {  if (url.contains("favicon.ico")) {      TeapotUtil.faviconIco(toClientWriter);        return;      }  System.out.println("不允许的用户：" +  socketToClient.getInetAddress().getHostAddress());      TeapotUtil.error418(toClientWriter, "未允许的用户访问");      return;  }  拦截不允许访问的host：  if (configuration.getBlackHostSet().contains(host)) {  if (url.contains("favicon.ico")) {      TeapotUtil.faviconIco(toClientWriter);      return;      }      System.out.println("不允许的网站：" + host);      TeapotUtil.error418(toClientWriter, "未允许的网站访问");      return;  }  判断钓鱼host并更改相关信息：  if (configuration.getGuideMap().containsKey(host)) {  String guideHost = configuration.getGuideMap().get(host);  System.out.println("钓鱼：" + host + "\t引导至：" + guideHost);  requestBuilder = new StringBuilder(  requestBuilder.toString().replace(host, guideHost)  );  url = url.replace(host, guideHost);  host = guideHost;  }  判断并处理https请求：  if(requestBuilder.toString().startsWith("CONNECT")) {  toClientWriter.write("HTTP/1.1 200 Connection Established\r\n\r\n".getBytes());  toClientWriter.flush();  new Thread(new ProxyPipe(toClientReader, toServerWriter)).start();  byte[] buffer = new byte[1024];  int length;  while((length = toServerReader.read(buffer)) >= 0) {  toClientWriter.write(buffer, 0, length);  }  return;  }  缓存存在时，确认缓存是否过期：  String contentStr = new String(ArrayUtils.subarray(content, 0, 1024));  String lastTime = contentStr.substring(contentStr.indexOf("Last-Modified") + 15, contentStr.indexOf("Last-Modified") + 44);  String checkString = "GET " + url + " HTTP/1.1\r\n";  checkString += "Host: " + host + "\r\n";  checkString += "If-modified-since: " + lastTime + "\r\n\r\n";  toServerWriter.write(checkString.getBytes());  toServerWriter.flush();  缓存不存在时，直接向服务器发送请求：  *// 从服务器接收数据，并转发给客户端*  while ((length = toServerReader.read(buffer)) >= 0) {  toClientWriter.write(buffer, 0, length);  byteList.add(buffer.clone());  lengthList.add(length);  }  *// 清空缓冲区（发送）并断开输出流*  toClientWriter.flush();  toClientWriter.close(); |
| 实验结果： |
| 代理服务器端口10240（可在配置文件中更改）  基础转发功能：  请求<http://www.hit.edu.cn>：  图片包含 监视器, 屏幕截图, 室内, 屏幕  描述已自动生成  控制台输出大量请求和缓存保存的信息：  图片包含 文字  描述已自动生成  访问不允许访问的地址（配置文件中为jwts.hit.edu.cn），将会转到418错误页面，并提示未允许的网站访问：  图片包含 屏幕截图  描述已自动生成  当访问钓鱼网站时，会将其导向配置好的页面（配置文件中将today.hit.edu.cn转到news.hit.edu.cn）：  图片包含 屏幕截图  描述已自动生成  当将被拦截的用户访问任何页面时，都会转到418错误页面，并提示未允许的用户访问（配置文件中配置被拦截的用户为127.0.0.1）：  图片包含 屏幕截图, 监视器  描述已自动生成  访问https（百度）：  图片包含 屏幕截图  描述已自动生成 |
| 心得体会： |
| 本次实验实现了一个简易的代理服务器，加深了我们对socket编程的理解，让我们了解到了“神秘”的翻墙软件的实现，受益匪浅。 |