

**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 | 冯运 | | 院系 | 计算机学院 | | |
| 班级 | 1603108 | | 学号 | 1160300524 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | 格物楼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）。(4)在HTTP服务器的基础上,设计并实现一个HTTPS代理服务器 |
| 实验过程： |
| 本次试验我使用的编程语言是 java   1. 实现基本的HTTP代理服务器   (1)将浏览器的局域网设置里将代理服务器设置成127.0.0.1:8080 的ip和端口,从而将浏览器的所有请求发送给自己编写的服务器程序。  首先建立一个ServerSocket, 这个ServerSocket是负责监听指定的端口,调用ServerSocket的accept()函数,进入阻塞状态,如果有Sockect连接建立成功,就返回建立成功的Socket对象,命名为Client。  之后通过读取Client 的InputStream输入流,接收到从浏览器发送来的http数据报。将接收的字节流转换成字符串流(即使用BufferReader),然后依据每行进行读写,使用正则表达式Host:\\s\*([\\w.-]\*) 来匹配http报文头部中的host字段,之后建立Server Socket,host为从http协议头部匹配到的host,端口号为http协议服务器固定端口号80。之后将从Client获得的http报文全部转发给Server。  这样代理服务器与客户端和目标服务器的两个Socket都建立好了,之后只需要将从目标服务器发来的数据全部转发给客户端,同时将从客户端发送的数据全部转发给目标服务器,直到客户端和目标服务器有一方关闭了连接,则中断双方的通信,结束转发。  在实现的过程中我对inputStream的运行不太清晰,遇到了读取inputStream无法结束的问题。通过查阅资料,发现Socket的inputStream的数据传输与读取文件的不同,因为读取文件时,当读取到文件的结尾时,inputStream会结束,但是Socket的inputSream只有在连接结束时才会结束,所以要想结束读取,我设置了Socket的 outTime,当超时时会抛出SocketTimeoutException异常,通过捕捉这个异常就可以退出inputStream的读取。  流程框图:    (2)HTTPS代理服务器的实现:  HTTPS代理服务器的实现与HTTP代理服务器的实现大体相同,但是在转发过程中使用的是HTTP协议的隧道技术。在一开始Client建立与代理服务器的连接之后,发送Connect HTTP数据报,告知代理服务器 目标服务器的 host和port,然后代理服务器需要与制定的host和port建立TCP链接,如果链接成功,就返回给Client一个HTTP Response:HTTP/1.1 200 Connection established\r\n\r\n, 告知Client连接成功,之后同HTTP代理服务器,将Client发送给代理服务器的信息全部交付给目标服务器,将目标服务器发送来的信息全部交付给Client。   1. 缓存服务器的实现   在代理服务器收到来自Server的HTTP Response 时,将请求的URL作为HashMap的key,将响应的内容作为value。当客户端再次请求相同的URL时,代理服务器根据URL取出对应的缓存信息,提取出其中的Last-Modified字段的日期,向目标服务器发送一个HTTP请求, 使用GET方法,包含If-Modified-Since: 日期, 如果服务器返回304状态的响应,则代理服务器直接将缓存的副本返回给Client,如果目标服务器返回的是正常的HTTP响应,则代理服务器将新的数据返回给Client,并更新对应的缓存。   1. 拓展代理服务器功能 2. 为了实现屏蔽用户访问某些网站的功能,我建立了一张屏蔽Host的Json格式的文件,然后使用程序读取并解析这个Json文件,从中提取出需要屏蔽的Host,在Client建立链接并发送请求后,解析出请求的Host,判断该Host是否在屏蔽列表中,如果在其中,就立即中断与Client的连接,不再代理转发请求。 3. 为了实现屏蔽某些用户访问代理服务器,我在屏蔽Host的列表中建立了一张屏蔽制定用户的列表,处理方式同上,在Client与代理服务器建立链接后,首先检查Client Socket的Host,如果在屏蔽的列表中,就立即中断与该Client的连接。 4. 钓鱼网站的实现: 在Client与代理服务器建立连接后,并不将请求转发给目标服务器,而是直接返回一个自己构造好的HTTP响应,然后Client就收到了一个钓鱼网站,而不是真正的网站。 |
| 实验结果： |
| 程序运行开始截图:    再浏览器已经设置127.0.0.1:8080的代理后,访问 <http://jwts.hit.edu.cn网址>  得到如下运行截图:    浏览器得到的结果是:    HTTP代理服务器的功能正确实现。  HTTPS代理服务器功能的演示:  在浏览器访问一个HTTPS协议的网站 <https://github.com/Mark-Fritz(我的Github主页)>  得到正常的访问结果。  屏蔽某些网站的访问:  在config.json中,我添加了    Jwts.hit.edu.cn 然后重新访问http://jwts.hit.edu.cn网站  结果是  程序中给出提示:    屏蔽成功  对某些客户端的屏蔽:    然后再次用浏览器访问,得到如下运行结果:    浏览器同样无法访问http://jwts.hit.edu.cn  钓鱼网站的运行结果:  如果开启钓鱼的转发模式,则会得到如下的运行结果:    这个页面可以自己定制。 |
| 问题讨论： |
| 在实现HTTP代理服务器基本功能时,遇到一个无限循环的问题:  即在读取socket的inputStream时,不像对file 的读取那样停止,循环不会正常停止。  通过查阅资料,发现Socket的inputStream的数据传输与读取文件的不同,因为读取文件时,当读取到文件的结尾时,inputStream会结束,但是Socket的inputSream只有在连接结束时才会结束,所以要想结束读取,我设置了Socket的 outTime,当超时时会抛出SocketTimeoutException异常,通过捕捉这个异常就可以退出inputStream的读取。 |
| 心得体会： |
| 通过这次试验,我对HTTP协议的原理,结构都有了极为深刻的映象,同时对Socket编程有了一次成功的实践,对以后学习和工作中遇到的计算机网络应用场景,可以更好的理解其中的原理以及实现,是非常重要的基础。  在这次试验中,还实现了HTTPS协议的转发,使得我对HTTPS协议也有了一定的了解。  通过实现对屏蔽某些网站的功能和屏蔽某些用户访问代理服务器的功能,我对防火墙的原理有了一定的了解。  通过自己实现一个钓鱼网站,我对如何防护网络钓鱼,如何保证通信的安全有了了解,在实际生产中应该尽量使用更加安全的HTTPS协议,可以有效防护网络钓鱼网站的出现。 |
| 附录: 主要程序代码 |
| Client.java  package Lab1;  import java.io.\*;  import java.net.ConnectException;  import java.net.Socket;  import java.net.SocketTimeoutException;  import java.net.UnknownHostException;  public class Client implements Runnable {  private Socket Client;  private Socket ProxyClient;  private ProtocolHeader header;  private final long aliveTime = 10000;  private FireWall FileWall;  Client(Socket Client, FireWall fireWall) {  this.Client = Client;  this.ProxyClient = null;  this.FileWall = fireWall;  }  @Override  public void run() {  ByteArrayOutputStream ClientCache = new ByteArrayOutputStream(); // used to cache the data from the client  try {  Client.setSoTimeout(200);  CloneStream(ClientCache, Client.getInputStream());  } catch (SocketTimeoutException e) {  } catch (IOException e) {  e.printStackTrace();  }  BufferedReader ObtainReader = new BufferedReader(new InputStreamReader(new ByteArrayInputStream(ClientCache.toByteArray())));  // parse the http or https header  try {  header = new ProtocolHeader(ObtainReader);  // filter some host by fire wall  if (!FileWall.isHostForbidden(header.getHost())) {  this.ProxyClient = new Socket(header.getHost(), header.getPort());  if (header.getPort() == 80) {  System.out.println("HTTP request to Host: " + header.getHost());  ProxyClient.getOutputStream().write(ClientCache.toByteArray());  } else if (header.getPort() == 443) {  System.out.println("HTTPS request to Host: " + header.getHost());  Client.getOutputStream().write("HTTP/1.1 200 Connection established\r\n\r\n".getBytes());  }  } else {  System.err.println("Forbid the destiny host: " + header.getHost());  }  if (ProxyClient != null)  ProxyClient.setSoTimeout(200);  } catch (ConnectException e) {  if (e.getMessage().equals("Connection timed out: connect")) {  System.err.println("Connect to Host: " + header.getHost() + " time out");  } else if (e.getMessage().equals("Connection refused: connect")) {  System.err.println("Connect to Host: " + header.getHost() + " refused");  } else  System.err.println(e.getMessage());  CloseAllConnect();  } catch (UnknownHostException e) {  System.err.println("Unknown host name: " + e.getMessage());  CloseAllConnect();  } catch (IOException e) {  e.printStackTrace();  }  // transport data from server to client  ProxyForward forward;  switch (header.getPort()) {  case 80:  // Phish http proxy forward mode  // forward = new PhishForward();  // normal http proxy forward mode  forward = new HTTPForward();  forward.ProxyForward(Client, ProxyClient, aliveTime);  case 443:  forward = new HTTPSForward();  forward.ProxyForward(Client, ProxyClient, aliveTime);  }  CloseAllConnect();  }  /\*\*  \* used to close all socket contain the Client and ProxyClient  \*/  private void CloseAllConnect() {  try {  if (Client != null && !Client.isClosed())  Client.close();  if (ProxyClient != null && !ProxyClient.isClosed())  ProxyClient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  /\*\*  \* clone the input stream to a ByteArrayOutputStream object  \*  \* @param CloneResult the clone result of input stream  \* @param InputStream the input stream to be cloned  \* @throws IOException when read input stream, some exception occur  \*/  static void CloneStream(ByteArrayOutputStream CloneResult, InputStream InputStream) throws IOException {  byte[] buffer = new byte[1024];  int length;  while ((length = InputStream.read(buffer)) != -1) {  CloneResult.write(buffer, 0, length);  }  CloneResult.flush();  }  } |
| Server.java  package Lab1;  import java.io.\*;  import java.net.ServerSocket;  import java.net.Socket;  public class Server {  public static void main(String[] args) throws IOException {  final String FILE = "./src/Lab1/config.json";  FireWall FireWall = new FireWall(FILE);  int PORT = 8080;  System.out.println("Listening to the port: " + PORT);  ServerSocket server = new ServerSocket(PORT);  Socket client;  while (true) {  client = server.accept();  if (!FireWall.isClientForbidden(client.getInetAddress().getHostName()))  new Thread(new Client(client, FireWall)).start();  else {  client.close();  System.err.println("Forbid the client host: " + client.getInetAddress().getHostName());  }  }  }  } |
| HTTPForward.java  package Lab1;  import java.io.ByteArrayOutputStream;  import java.net.Socket;  import java.net.SocketTimeoutException;  public class HTTPForward implements ProxyForward {  @Override  public void ProxyForward(Socket client, Socket server, long MaxWaitTime) {  long LatestDataTransportTime = System.currentTimeMillis();  ByteArrayOutputStream ClientCache; // used to cache the data from the client  ByteArrayOutputStream ServerCache; // used to cache the data from the server  while (server != null && !(client.isClosed() || server.isClosed())) {  try {  ClientCache = new ByteArrayOutputStream();  try {  client.setSoTimeout(200);  Client.CloneStream(ClientCache, client.getInputStream());  } catch (SocketTimeoutException e) {  }  server.getOutputStream().write(ClientCache.toByteArray());  ServerCache = new ByteArrayOutputStream();  try {  Client.CloneStream(ServerCache, server.getInputStream());  } catch (SocketTimeoutException e) {  }  if (ClientCache.size() == 0 && ServerCache.size() == 0) {  // connection out of time, close the socket  if (System.currentTimeMillis() - LatestDataTransportTime > MaxWaitTime)  break;  } else  LatestDataTransportTime = System.currentTimeMillis();  client.getOutputStream().write(ServerCache.toByteArray());  ClientCache.close();  ServerCache.close();  } catch (Exception e) {  break;  }  }  }  } |