

**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 | 李世轩 | | 院系 | 软件工程 | | |
| 班级 | 2037102 | | 学号 | 120L022109 | | |
| 任课教师 | 李全龙 | | 指导教师 | 李全龙 | | |
| 实验地点 | 格物207 | | 实验时间 | 2022年10月7日 | | |
| 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： |
| 首先分析一下代理服务器：  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。  根据这些分析可以画出程序的主要流程图：  计网实验-代理服务器流程图  接下来开始正式编程，使用代码实现该流程，并增加一些附加的小功能。  首先，设置工作目录，即存放缓存文件的目录，并完成缓存的读入。    这里采用Java的序列化接口来实现对文件的读写。并以此设计了一个工具类。    接下来开始创建socket并监听指定端口。当监听到请求时，启动新的子线程来处理。主线程继续不断地监听。    在子线程中，先读取请求行，并根据其中的信息决定是否要过滤该请求。（注：图中注释的部分是用来用户过滤的，因为在这里将本地ip加入了需要被屏蔽的ip列表中，如果不注释掉，那么所有的请求都会被拦截。）    下一步根据请求行中的数据判断其是否需要被重定向（钓鱼）。若需要，则将请求行中的数据替换。    然后尝试连接目标主机，这里设置了一个重复次数，若在连接多次均未成功的话，放弃本次请求。    在顺利建立连接后，查看该请求是否已经被缓存。若未缓存，则直接将请求发送到目的主机，并记录缓存然后发送数据回浏览器。若已经缓存，则在缓存中查看该条缓存的数据。如果缓存的内容里面该请求是没有Last-Modify属性的，就不用向服务器查询If-Modify了，否则向服务器查询If-Modify。如果服务器给回的响应是304 Not Modified，就将缓存的数据直接发送给浏览器。    服务器返回的不是304 Not Modified的话，就将服务器的响应直接转发到浏览器并记录缓存就好了。 |
| 实验结果： |
| 分点演示：  基础功能，代理请求并正确显示网页。  在浏览器发送请求后，在控制台中可以看到正确的请求报文    以及正确的返回报文    且网页可以正确显示    缓存功能  当第二次请求时，网页将很快显示出来，且可以看到控制台中显示使用缓存数据    用户过滤  将先前注释的代码释放，可以看到所有来自本地ip的请求全部被过滤了      网址过滤  当我们访问被屏蔽的网址时，可以看到其已被过滤。    网站引导  当发送请求到需要被引导的网址时，可以看到请求被重定向    且服务器返回正确信息    页面显示jwts.hit.edu.cn的内容且地址栏显示为today.hit.edu.cn |
| 问题讨论： |
| 1. Socket 编程的客户端和服务器端主要步骤；   服务器端：  建立socket  绑定端口号  监听端口  接受请求  处理请求数据  发送应答报文  继续监听或关闭socket  客户端：  建立socket  连接服务器  发送请求报文  接受应答报文  关闭套接字   1. HTTP 代理服务器的程序流程图；   计网实验-代理服务器流程图   1. 实现 HTTP 代理服务器的关键技术及解决方案；   在用java实现代理服务器中比较关键的难点如下：  对输入流使用readline()时，如果服务器端或客户端发来的报文没有以特殊格式结尾，那么该函数会造成线程阻塞，最终导致socket连接超时，并会造成程序崩溃。解决办法就是设定一个不算短的超时时间，并对socket超时异常进行捕获但不进行任何处理，因为这个超时异常其实并不会影响数据的完整性且可以打断readline造成的阻塞。  另外比较重要的一点就是缓存的实现，个人使用的方法是在主线程中设定的线程同步的list来当作容器，在主线程启动时利用Java的反序列化从文件读入list，并在线程结束时序列化写入文件，这样在一次运行过程中只会出现一次文件的读和写。 |
| 心得体会： |
| 熟悉了HTTP代理服务器的实现原理。  熟悉了Java的socket编程。 |

附录：程序源代码

ProxySever.java

package main;

import constants.ProxyConstants;

import thread.ProxyThread;

import util.SerializationUtil;

import java.io.File;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.List;

import java.util.Scanner;

/\*\*

\* ProxySever

\*

\* @author: lsxuan

\* @email: 1146887979@qq.com

\* @create: 2022-10-03 20:13

\*/

public class ProxySever {

public static String cachePath;

private static List<CacheUnit> cache;

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

ServerSocket serverSocket;

Socket currsoket = null;

/\*\* users need to set up work space \*/

System.out.println("==============请输入缓存的存储目录，输入 d 则设置为默认目录（程序同一目录下）=================");

Scanner scanner = new Scanner(System.in);

cachePath = scanner.nextLine();

if (cachePath.equals("d")) {

cachePath = ProxyConstants.DEFAULT\_CACHE\_PATH;

}

/\*\* 初始化缓存写对象 \*/

//readCache(cachePath);

File cacheFile = new File(cachePath);

//cacheFile.delete();

if (!cacheFile.exists()) {

cacheFile.createNewFile();

}

cache = ((SerializationUtil.readObjectForList(cacheFile)));

System.out.println("=================================== 工作目录设置完毕====================================");

try {

//设置serversocket，绑定端口8888

serverSocket = new ServerSocket(ProxyConstants.PROXY\_PORT);

int i = 0;

//循环，持续监听从这个端口的所有请求

while (true) {

currsoket = serverSocket.accept();

//启动一个新的线程来处理这个请求

i++;

System.out.println("启动第" + i + "个线程");

new ProxyThread(currsoket).run();

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (currsoket != null) {

currsoket.close();//及时关闭这个socket

}

writeCache();

}

}

synchronized public static void addCacheUnit(CacheUnit cacheUnit) {

cache.add(cacheUnit);

}

synchronized public static boolean ifCacheEmpty() {

return cache.isEmpty();

}

synchronized public static boolean ifCached(String requestLine) {

if (requestLine == null) return false;

for (CacheUnit unit : cache

) {

if (unit.getRequestLine().equals(requestLine)) return true;

}

return false;

}

synchronized public static CacheUnit getCacheUnit(Integer index) {

return cache.get(index).copy();

}

synchronized public static void removeCacheUnit(int cacheUrlIndex) {

cache.remove(cacheUrlIndex);

}

synchronized public static void writeCache(){

SerializationUtil.writeObject(cache, new File(cachePath));

}

synchronized public static String getModifyTime(String requestLine, Boolean ifHasTime, Integer cacheUrlIndex) {

if (requestLine == null) throw new IllegalArgumentException();

String LastModifiTime = null;

for (int i = 0; i < cache.size(); i++) {

CacheUnit unit = cache.get(i);

if (requestLine.equals(unit.getRequestLine())) {

cacheUrlIndex = i;

for (String line : unit.getLines()

) {

if (line.contains("http://"))

break;

if (line.contains("Last-Modified:")) {

LastModifiTime = cachePath.substring(line.indexOf("Last-Modified:"));

ifHasTime=true;

return LastModifiTime;

}

if (line.contains("<html>")) {

ifHasTime = false;

return LastModifiTime;

}

}

}

}

ifHasTime = false;

return LastModifiTime;

}

}

ProxyThread.java

package thread;

import constants.ProxyConstants;

import util.CacheUnit;

import main.ProxySever;

import util.Filter;

import java.io.\*;

import java.net.Socket;

import java.net.SocketException;

import java.net.SocketTimeoutException;

import java.util.StringTokenizer;

/\*\*

\* ProxyThread

\*

\* @author: lsxuan

\* @email: 1146887979@qq.com

\* @create: 2022-10-03 20:16

\*/

public class ProxyThread implements Runnable {

private Socket socket;

public ProxyThread(Socket socket) {

this.socket = socket;

try {

this.socket.setSoTimeout(ProxyConstants.TIMEOUT);

} catch (SocketException e) {

e.printStackTrace();

}

}

@Override

public void run() {

String ip = socket.getLocalAddress().getHostAddress();

String requestLine = null;

BufferedReader clientBufferedReader=null;

try {

// if(Filter.ipList.contains(ip)){

// System.out.println("---------用户已被屏蔽");return;

// }

clientBufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

System.out.println("从浏览器读取第一行....");

requestLine = clientBufferedReader.readLine();

if(requestLine==null){

System.out.println("错误请求");

return;

}

if (!Filter.filter(requestLine)) {

System.out.println("请求" + requestLine + "已被过滤");

return;

}

} catch (IOException e) {

e.printStackTrace();

try {

if(clientBufferedReader!=null)clientBufferedReader.close();

if(socket!=null)socket.close();

} catch (Exception ex) {

ex.printStackTrace();

}

}

System.out.println(requestLine);

String[] hostAndPort = getHostAndPort(requestLine);

String targetHost = hostAndPort[0];

String targetPort = hostAndPort[1];

System.out.println("提取的主机名:" + targetHost + " 提取的端口号: " + targetPort);

String replacement = Filter.map(targetHost);

String oldHost = targetHost;

if (replacement != null) {

requestLine=requestLine.replace(targetHost,replacement);

targetHost = replacement;

System.out.println("请求已被重定向");

}

//尝试连接目标主机

Socket accessSocket = null;

int retry = ProxyConstants.RETRIEVE;

try {

while (retry-- != 0 && (targetHost != null)) {

accessSocket = new Socket(targetHost, Integer.parseInt(targetPort));

if (accessSocket != null) break;

}

Thread.sleep(ProxyConstants.CONNECT\_PAUSE);

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

InputStream webInputStream = null;

BufferedReader webBufferedReader = null;

PrintWriter webPrintWriter = null;

InputStream clientInputStream = null;

OutputStream clientOutputStream = null;

PrintWriter clientOutPrintWriter = null;

if (accessSocket == null) {

System.out.println("目的主机：" + targetHost + ":" + targetPort + "连接失败");

} else {

System.out.println("目的主机：" + targetHost + "连接成功");

System.out.println("请求将发送至:" + targetHost + ":" + targetPort);

try {

accessSocket.setSoTimeout(ProxyConstants.TIMEOUT);

webInputStream = accessSocket.getInputStream();//获取网站返回的响应

webBufferedReader = new BufferedReader(new InputStreamReader(webInputStream));

webPrintWriter = new PrintWriter(accessSocket.getOutputStream());

clientInputStream = socket.getInputStream();//创建从浏览器获取请求的输入流

clientOutputStream = socket.getOutputStream();//创建向浏览器发送响应的流

clientOutPrintWriter = new PrintWriter(clientOutputStream);

boolean ifCached = ProxySever.ifCached(requestLine);

CacheUnit cacheUnit = new CacheUnit(requestLine);

if (!ifCached) {

//将请求直接发往网站，并获取响应，记录响应至缓存

sendRequestToWeb(replacement,oldHost,requestLine, webPrintWriter, clientBufferedReader);

sendResponseToClient(false, -1, webInputStream, clientOutputStream, clientOutPrintWriter, cacheUnit);

} else {//寻找之前缓存过该请求

Boolean ifHasTime = false;

Integer cacheUrlIndex = -1;

String modifyTime = ProxySever.getModifyTime(requestLine, ifHasTime, cacheUrlIndex);

System.out.println("提取到的modifytime：" + modifyTime);

String info = null;

//如果缓存的内容里面该请求是没有Last-Modify属性的，就不用向服务器查询If-Modify了，否则向服务器查询If-Modify

if (ifHasTime) {

StringBuilder msg = new StringBuilder();

msg.append(requestLine).append("\r\n");

System.out.print("向服务器发送确认修改时间请求:\n" + msg);

msg.append("Host: ").append(targetHost).append("\r\n");

msg.append("If-modified-since: ").append(modifyTime).append("\r\n\r\n");

webPrintWriter.write(msg.toString());

webPrintWriter.flush();

info = webBufferedReader.readLine();

System.out.println("服务器发回的信息是：" + info);

}

if (!ifHasTime||info.contains(ProxyConstants.NOT\_MODIFIED) ) {//如果服务器给回的响应是304 Not Modified，就将缓存的数据直接发送给浏览器

System.out.println("使用缓存数据");

StringBuilder sb = new StringBuilder();

//

if (cacheUrlIndex != -1) {

sb.append(ProxySever.getCacheUnit(cacheUrlIndex).getContent()).append("\r\n\r\n");

clientOutputStream.write(sb.toString().getBytes(), 0, sb.toString().length());

clientOutputStream.flush();

}

} else {

//服务器返回的不是304 Not Modified的话，就将服务器的响应直接转发到浏览器并记录缓存就好了

System.out.println("有更新，使用新的数据");

clientOutputStream.write(info.getBytes());

sendResponseToClient(true, cacheUrlIndex, webInputStream, clientOutputStream, clientOutPrintWriter, cacheUnit);

}

}

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (webBufferedReader != null) webBufferedReader.close();

if (webPrintWriter != null) webPrintWriter.close();

if (clientBufferedReader != null) clientBufferedReader.close();

if (clientOutPrintWriter != null) clientOutPrintWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

try {

if (socket != null) socket.close();

if (accessSocket != null) accessSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

public void sendRequestToWeb(String replacement,String oldHost,String requestLine, PrintWriter webPrintWriter, BufferedReader clientBufferedReader) throws IOException {

String buffer = requestLine;

System.out.print("发送请求:\n");

try {

while (!buffer.equals("")) {

buffer += "\r\n";

if (replacement!=null&&buffer.contains("Host: ")) {

buffer=buffer.replace(oldHost,replacement);

}

webPrintWriter.write(buffer);

System.out.println(buffer);

buffer = clientBufferedReader.readLine();

}

}catch (SocketTimeoutException ignored){

}

webPrintWriter.write("\r\n");

webPrintWriter.flush();

}

public void sendResponseToClient(boolean ifUpdate, Integer cacheUrlIndex, InputStream webInputStream, OutputStream clientOutputStream, PrintWriter clientOutPrintWriter, CacheUnit cacheUnit) {

byte[] bytes = new byte[2048];

int length = 0;

try {

while (true) {

if ((length = webInputStream.read(bytes)) > 0) {

clientOutputStream.write(bytes, 0, length);

String show\_response = new String(bytes, 0, bytes.length);

System.out.println("服务器发回的消息是:\n---\n" + show\_response + "\n---");

//write cache

cacheUnit.getContent().append(bytes).append("\r\n");

//if(webInputStream.available()<bytes.length)break;

} else break;

}

clientOutPrintWriter.write("\r\n");

clientOutPrintWriter.flush();

} catch (SocketTimeoutException ignored){

}catch (IOException e) {

e.printStackTrace();

}

if (ifUpdate) {

ProxySever.removeCacheUnit(cacheUrlIndex);

}

ProxySever.addCacheUnit(cacheUnit);

}

public static String[] getHostAndPort(String requestLine) {

String host;

String port = null;

String[] result = new String[2];

int index;

int portIndex;

String temp;

StringTokenizer stringTokenizer = new StringTokenizer(requestLine);

stringTokenizer.nextToken();//丢弃第一个字串 这是请求类型 比如GET POST

temp = stringTokenizer.nextToken();//这个字串里面有主机名和端口

int index1 = temp.indexOf("//");

host = temp.substring(index1 == -1 ? 0 : index1 + 2);//比如 http://news.sina.com.cn/gov/2017-12-13/doc-ifypsqiz3904275.shtml -> news.sina.com.cn/gov/2017-12-13/doc-ifypsqiz3904275.shtml

index = host.indexOf("/");

if (index == -1) index = temp.length();

if (index != -1) {

host = host.substring(0, index);//比如 news.sina.com.cn/gov/2017-12-13/doc-ifypsqiz3904275.shtml -> news.sina.com.cn

portIndex = host.indexOf(":");

if (portIndex != -1) {

port = host.substring(portIndex + 1);//比如 www.ghostlwb.com:8080 -> 8080

host = host.substring(0, portIndex);

} else {//没有找到端口号，则加上默认端口号80

port = "80";

}

}

result[0] = host;

result[1] = port;

return result;

}

public static String getURL(String requestLine) {

String[] questLine = requestLine.split(" ");

if (questLine.length != 3) throw new RuntimeException();

return questLine[1];

}

}

ProxyConstants.java

package constants;

/\*\*

\* ProxyConstants

\*

\* @author: lsxuan

\* @email: 1146887979@qq.com

\* @create: 2022-10-03 20:19

\*/

public class ProxyConstants {

public static final String DEFAULT\_CACHE\_PATH = "default\_cache.cah";

public static final int TIMEOUT = 10000;//response time out upper bound

public static final int RETRIEVE = 5;//retry connection 5 times

public static final int CONNECT\_PAUSE = 5000;//waiting for connection

public static final int PROXY\_PORT = 8888;

public static final String ILLEGAL\_REQUEST = "Illegal Request";

public static final String LAST\_MODIFIED = "Last-Modified";

public static final String NOT\_MODIFIED = "Not Modified";

}

CacheUnit.java

package util;

import java.io.Serializable;

/\*\*

\* CacheUnit

\*

\* @author: lsxuan

\* @email: 1146887979@qq.com

\* @create: 2022-10-03 20:51

\*/

public class CacheUnit implements Serializable {

private static final long serialVersionUID = 2333333333333333L;

private String requestLine;

private StringBuilder content;

public CacheUnit(String requestLine) {

this.requestLine = requestLine;

content = new StringBuilder();

}

public CacheUnit(String requestLine, StringBuilder content) {

this.requestLine = requestLine;

this.content = content;

}

public CacheUnit copy() {

return new CacheUnit(this.requestLine, this.content);

}

public String getRequestLine() {

return requestLine;

}

public void setRequestLine(String requestLine) {

this.requestLine = requestLine;

}

public StringBuilder getContent() {

return content;

}

public void setContent(StringBuilder content) {

this.content = content;

}

public String getCacheContent() {

StringBuilder sb = new StringBuilder();

sb.append(requestLine).append("\r\n");

if (!("".equals(content) || content == null)) sb.append(content);

return sb.toString();

}

public String[] getLines() {

return content.toString().split("\r\n");

}

}

Filter.java

package util;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

/\*\*

\* Filter

\*

\* @author: lsxuan

\* @email: 1146887979@qq.com

\* @create: 2022-10-03 21:46

\*/

public class Filter {

private static final Map<String, String> MAP;

static {

MAP = new HashMap<>();

MAP.put("http://www.tsinghua.edu.cn/", "http://www.hit.edu.cn/");

MAP.put("today.hit.edu.cn", "jwts.hit.edu.cn");

}

private static final List<String> filterList;

static {

filterList = new ArrayList<>();

filterList.add("CONNECT");

filterList.add("www.4399.com");

}

/\*\*

\* 过滤某些请求

\* @param requestLine

\* @return

\*/

public static boolean filter(String requestLine) {

if (requestLine == null) return false;

for (String str:filterList

) {

if(requestLine.contains(str))return false;

}

return true;

}

/\*\*

\* 获取钓鱼映射

\* @param requestLine

\* @return

\*/

public static String map(String requestLine) {

return MAP.get(requestLine);

}

/\*\*

\* 需要过滤的用户ip

\*/

public static final List<String> ipList;

static {

ipList = new ArrayList<>();

ipList.add("127.0.0.1");

}

}

SerializationUtil.java

package util;

import java.io.\*;

import java.nio.file.Files;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

/\*\*

\* SerializationUtil

\*

\* @author: lsxuan

\* @email: 1146887979@qq.com

\* @create: 2022-10-04 19:21

\*/

public class SerializationUtil {

/\*\*

\* 序列化,List

\*/

public static <T> boolean writeObject(List<T> list, File file) {

T[] array = (T[]) list.toArray();

ObjectOutputStream out = null;

try {

out = new ObjectOutputStream(Files.newOutputStream(file.toPath()));

out.writeObject(array);

out.flush();

return true;

} catch (IOException e) {

e.printStackTrace();

return false;

} finally {

if (out != null) {

try {

out.close();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

}

/\*\*

\* 反序列化,List

\*/

public static <E> List<E> readObjectForList(File file) {

E[] object;

ObjectInputStream in = null;

try {

in = new ObjectInputStream(Files.newInputStream(file.toPath()));

object = (E[]) in.readObject();

return new ArrayList<E>(Arrays.asList(object));

} catch (EOFException e) {

return new ArrayList<E>();

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

} finally {

if (in != null) {

try {

in.close();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

return null;

}

}