哈尔滨工业大学

<<计算机网络>>

实验报告

(2019 年度秋季学期)

姓名:	陈鋆
学号:	1170300513
学院:	计算机科学与技术学院
教师:	聂兰顺

实验一 HTTP 代理服务器的设计与实现

一、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。

二、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）。

三、实验过程及结果

Socket 编程的客户端和服务器端主要步骤：

服务器端：

1.创建套接字，绑定套接字的本地 IP 地址和端口号，对端口进行监听。

2. 从连接请求队列中取出一个连接请求，并同意连接。在 TCP 连接过程中进行了三次握手。

3.接收客户端请求

4.对请求进行响应，发送响应数据

5.关闭连接

客户端：

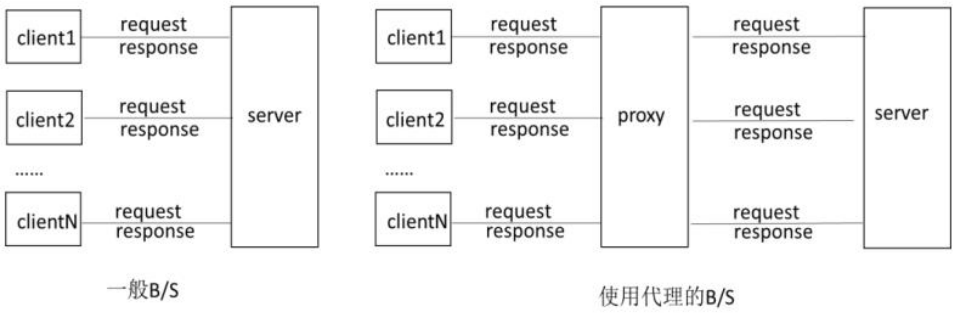
1. 根据目标服务器IP地址与端口号创建套接字，并连接服务器（三次握手）
2. 发送请求报文
3. 接收返回报文
4. 关闭连接

HTTP 代理服务器的基本原理：

功能：

接收来自客户端的 HTTP 请求，并通过这个代理服务器将该请求转发给服务器；同时，服务器也将获得的响应发给代理服务器，然后代理服务器再将该响应发送给客户端。

代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端） 通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。如图所示，为普通 Web 应用通信方式与采用代理服务器的通信方式的对比。

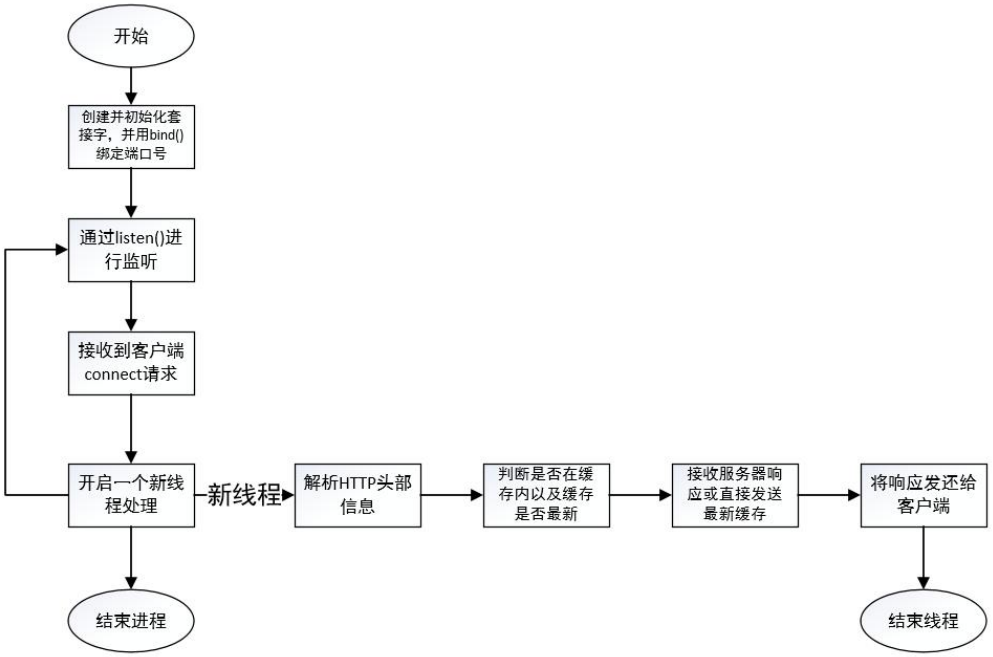


原理：

代理服务器在指定端口（例如 10240）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。

HTTP 代理服务器的程序流程图：



实现 HTTP 代理服务器的关键技术及解决方案：

关键技术：解析 HTTP 头部信息

解决方案：

首先，定义一个 HTTP 信息类：

```
package lab1;
//用来描述HTTP头部信息的类
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;
  }
}
```

接着再通过 ParseHttpHead 函数读取 HTTP 头部信息，将这些头部信息一一比对确认后记录到 HttpHeader 中：

JWLAb/src/lab1/Proxy2.java

```
/**
* 解析http头的信息，获取method，url，host，cookie
 */
public HttpHeader ParseHeader(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";//https里的特殊情况
  }
  //切出Host和Cookie的信息
  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;
}
```

关键技术：建立代理服务器到服务器的连接

解决方案：

使用 ConnectToServer 函数来完成，利用 socket 网络编程接口类，如果连接成功，则返回 socket 类，失败则返回 null，交由上层对连接成功或失败的情况进行处理。

```
/**
* 获取代理服务器与服务器沟通的套接字
 * @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沟通的套接字
 * @throws UnknownHostException
 * @throws IOException
 */
public Socket ConnectToServer(String host, int port, int times) throws UnknownHostException, IOException {
  for (int i = 0; i < times; i++) {
    SerSocket = new Socket(host, port);
    //设置端口连接时间
    SerSocket.setSoTimeout(timeout);
    SeverInputStream = SerSocket.getInputStream();
    SeverBufferReader = new BufferedReader(new InputStreamReader(SeverInputStream));
    SeverOutputStream = SerSocket.getOutputStream();
    SeverPrintWriter = new PrintWriter(SeverOutputStream);

    if (SerSocket != null) {
      return SerSocket;
    }
  }
  return null;
}
```

关键技术：实现基本功能——接收来自客户端的 HTTP 请求，并通过这个代理服务器将该请求转发给服务器；同时，服务器也将获得的响应发给代理服务器，然后代理服务器再将该响应发送给客户端。

解决方案：

主函数（Proxy 类中）监听到客户端的 connect 请求后，开启一个线程，用此线程来完成一次客户端与服务器端之间的数据传输。

线程类内，代理服务器首先接收 HTTP 请求（如果出现错误则关闭套接字），进而用前面所述的解析 HTTP 头部信息的技术提取 HTTP 请求的关键信息。

```
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==========客户端请求==========");
  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 = ParseHeader(header);
  String url = httpHeader.url;
  String host = httpHeader.host;
```

再利用上述的建立代理服务器到服务器的连接的技术，建立起与服务器的连

接（失败则关闭套接字）

```
public Socket ConnectToServer(String host, int port, int times) throws UnknownHostException, IOException {
  for (int i = 0; i < times; i++) {
    SerSocket = new Socket(host, port);
    //设置端口连接时间
    SerSocket.setSoTimeout(timeout);
    SeverInputStream = SerSocket.getInputStream();
    SeverBufferReader = new BufferedReader(new InputStreamReader(SeverInputStream));
    SeverOutputStream = SerSocket.getOutputStream();
    SeverPrintWriter = new PrintWriter(SeverOutputStream);

    if (SerSocket != null) {
      return SerSocket;
    }
  }
  return null;
}
```

```
if (ConnectToServer(host, HttpPort, 5) == null) {
  return;
}
```

连接成功后，使用 SendToServer 函数向服务器发送 HTTP 报文，并等候服务器响应消息。得到响应消息后（接收错误要关闭套接字）再将其转发给客户端，完成此线程。

```
public void SendToServer(List<String> lst) {
  System.out.println("\n=========代理服务器向服务器发送请求信息=========");
  for (int i = 0; i < lst.size(); i++) {
    String line = lst.get(i);
    SeverPrintWriter.write(line + "\r\n");
    System.out.println(line);
  }
  SeverPrintWriter.write("\r\n");
  SeverPrintWriter.flush();
}
```

```
System.err.println("\n=========无缓存=========");
SendToServer(header);
SendBackToClient(url);
```

关键技术：实现网站屏蔽

解决方案：

检测 HttpHeader 结构体中的 url，若与需要屏蔽的网址一致，则使用 Filter 函数向浏览器发送警告消息以警告用户。

```
public void Filter() {
  for (int i=1;i<419;i++) {
    ClientPrintWriter.write("非法网站！ FBI WARNING!\t\t");
    if (i%11==0) {
      ClientPrintWriter.write("\r\n");
    }
  }
  ClientPrintWriter.write("\r\n");
  ClientPrintWriter.flush();
  ClientPrintWriter.close();
}
```

```
//网站过滤
for (int i=0;i<WebsiteFilter.size();i++) {
  if (host.contains(WebsiteFilter.get(i))) {
    Filter();
    System.err.println("禁止访问："+url);
    return;
  }
}
```

关键技术：用户屏蔽

解决方案：

设置限制访问名单（IP 地址集合），在客户端与代理服务器建立连接阶段检查 IP 地址，如果在名单中则禁止访问。

```
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);
  }
}
```

关键技术：网站引导（钓鱼）

解决方案：

检测请求过来的 HTTP 报文头部，如果发现访问的网址是要被钓鱼的网址，

则调用 fishing 函数将该网址引导到其他网站（钓鱼网址），通过构造一个 HTTP 头部消息来向服务器发送，随后与 run()函数一样，检查缓存。

```
public void fishing(List<String> header){

  header.clear();
  header.add("GET http://www.hit.edu.cn/ HTTP/1.1");
  header.add("Host: www.hit.edu.cn");
  header.add("User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0");
  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=ParseHeader(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=timecache.containsKey(url)&&bytescache.containsKey(url);
  if (!flag) {
    /*
```

关键技术：缓存

解决方案：

接收到 HTTP 报文后，先判断其是否已有缓存（利用 flag 变量），如果有，则发送条件性 GET 方法（通过 TranslateHTTP 函数构造）验证缓存在服务器端是否被更新过；如果没有，则正常发送消息。

```
    if (ConnectToServer(host, HttpPort, 5) == null) {
      return;
    }
    System.out.println("url="+url);
    System.out.println("host="+host);

    boolean flag= timecache.containsKey(url)&&bytescache.containsKey(url);
    if (!flag) {
     // 没有Cache的情况
      System.err.println("\n==========无缓存==========");
      SendToServer(header);
      SendBackToClient(url);
    } else {
     // 有cache的情况
      System.err.println("\n==========有缓存==========");
      SendBackToClientWithCache(header, host,url);
    }

  } catch (SocketException e) {
    //e.printStackTrace();
  } catch (IOException e) {
    //e.printStackTrace();
  }catch (Exception e){
    //e.printStackTrace();
  }
```

```
String modifiTime=timecache.get(url);
// 发送条件性GET方法到服务器
SeverPrintWriter.write(header.get(0) + "\r\n");
SeverPrintWriter.write("Host: "+host + "\r\n");

System.out.println("Modified Time:"+modifiTime);
String str = "If-modified-since: " + modifiTime + "\r\n";
SeverPrintWriter.write(str);
SeverPrintWriter.write("\r\n");
SeverPrintWriter.flush();
```

接收到服务器的响应消息，有缓存时，判断返回的状态码是否是 304，若是则将缓存的内容发送给客户端；无缓存时，则对该消息进行缓存。且都要将目标服务器返回的数据直接转发给客户端。若非这两者，则说明返回消息有问题，再运行一次 run 线程以获得网页信息。

```
try {
  String ServerMessage = SeverBufferReader.readLine();
  //System.out.println(ServerMessage);
  if (ServerMessage == null) {
    return false;
  }
  System.err.println("\n服务器响应信息首部："+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"))
  {
    System.err.println("\n=======缓存未更新======");
    List<Byte> lst=bytescache.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"))
  {
    System.err.println("\n=======缓存已更新======");
    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 = SeverInputStream.read(bytes)) > 0) {
      ClientOutputStream.write(bytes, 0, len);
    } else if (len < 0) {
      break;
    }
  }
  //缓存无效，去除.
  timecache.remove(url);
  bytescache.remove(url);
```

```
            ClientPrintWriter.write("\r\n");
            ClientPrintWriter.flush();
            ClientOutputStream.close();
        }
        else {  //http消息无效的情况
            bytescache.remove(url);
            timecache.remove(url);
            while (!SeverBufferReader.readLine().equals(""))
            {
                //刷完无效信息
            }
            byte bytes[] = new byte[size];
            int len;
            while (true) {
                if ((len = SeverInputStream.read(bytes)) >= 0) {
                    //ClientOutputStream.write(bytes, 0, len);
                } else if (len < 0) {
                    break;
                }
            }
            run();//重新来一遍
        }
    } catch (IOException e1) {
    }

    return true;
}
```

HTTP 代理服务器实验验证过程以及实验结果：

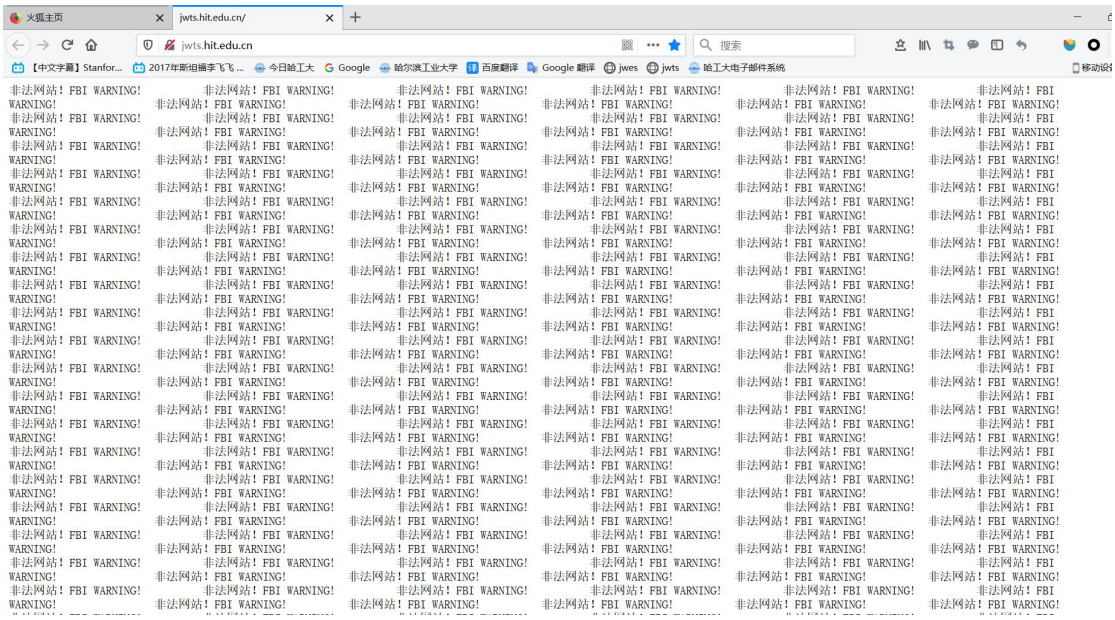
基础功能：

配置：

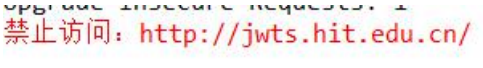
连接 www.hit.edu.cn:

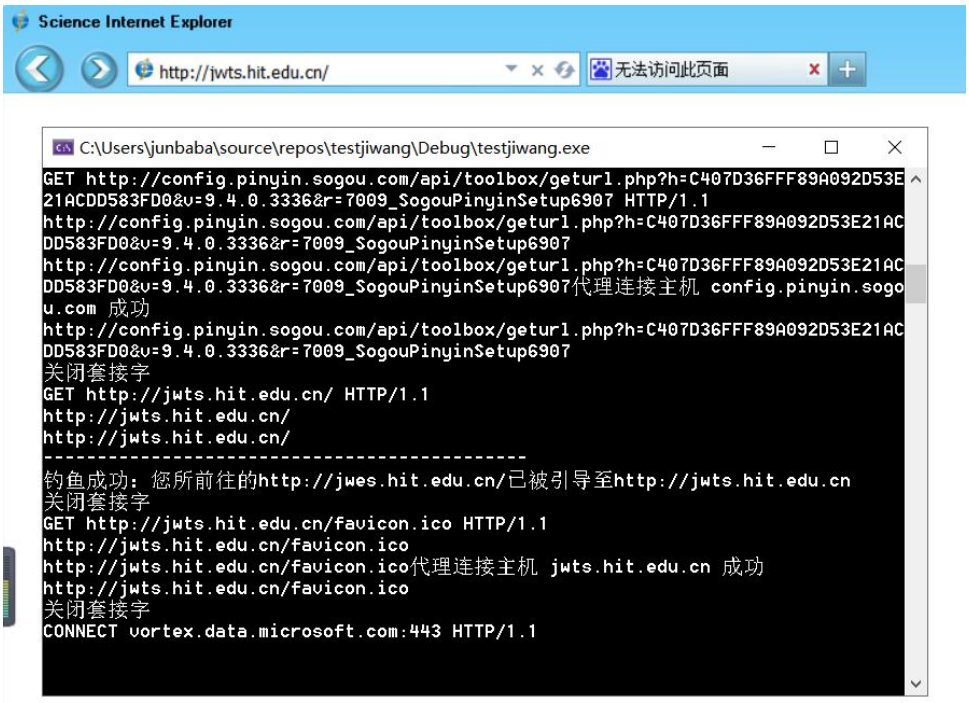

```
=========客户端请求==========
GET http://news.hit.edu.cn/_js/jquery.min.js HTTP/1.1
Host: news.hit.edu.cn
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://news.hit.edu.cn/xxyw/main.htm
Cookie: _ga=GA1.3.1360441355.1510728159; _gid=GA1.3.1825359634.1571400441
url=http://news.hit.edu.cn/_js/jquery.min.js
host=news.hit.edu.cn

=========代理服务器向服务器发送请求信息==========
GET http://news.hit.edu.cn/_js/jquery.min.js HTTP/1.1
Host: news.hit.edu.cn
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: */*
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
Referer: http://news.hit.edu.cn/xxyw/main.htm
Cookie: _ga=GA1.3.1360441355.1510728159; _gid=GA1.3.1825359634.1571400441
```

网站过滤：

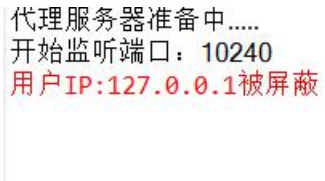

火狐主页
jwts.hit.edu.cn/
jwts.hit.edu.cn
非法网站！FBI WARNING！

网站引导（钓鱼）：


禁止访问：http://jwts.hit.edu.cn/


Science Internet Explorer
http://jwts.hit.edu.cn/
无法访问此页面
C:\Users\junbaba\source\repos\testjiwang\Debug\testjiwang.exe
GET http://config.pinyin.sogou.com/api/toolbox/geturl.php?h=C407D36FFF89A092D53E21ACDD583FD0&v=9.4.0.3336&r=7009_SogouPinyinSetup6907 HTTP/1.1
http://config.pinyin.sogou.com/api/toolbox/geturl.php?h=C407D36FFF89A092D53E21ACDD583FD0&v=9.4.0.3336&r=7009_SogouPinyinSetup6907
http://config.pinyin.sogou.com/api/toolbox/geturl.php?h=C407D36FFF89A092D53E21ACDD583FD0&v=9.4.0.3336&r=7009_SogouPinyinSetup6907代理连接主机 config.pinyin.sogou.com 成功
http://config.pinyin.sogou.com/api/toolbox/geturl.php?h=C407D36FFF89A092D53E21ACDD583FD0&v=9.4.0.3336&r=7009_SogouPinyinSetup6907
关闭套接字
GET http://jwts.hit.edu.cn/ HTTP/1.1
http://jwts.hit.edu.cn/
http://jwts.hit.edu.cn/
钓鱼成功：您所前往的http://jwes.hit.edu.cn/已被引导至http://jwts.hit.edu.cn
关闭套接字
GET http://jwts.hit.edu.cn/favicon.ico HTTP/1.1
http://jwts.hit.edu.cn/favicon.ico
http://jwts.hit.edu.cn/favicon.ico代理连接主机 jwts.hit.edu.cn 成功
http://jwts.hit.edu.cn/favicon.ico
关闭套接字
CONNECT vortex.data.microsoft.com:443 HTTP/1.1

用户屏蔽：


代理服务器准备中.....
开始监听端口：10240
用户IP:127.0.0.1被屏蔽

缓存：

无缓存：

```
========转发响应信息到客户端=========

========无缓存=========
```

有缓存：

```
=========有缓存=========
服务器响应信息首部：HTTP/1.1 200 200

=========有缓存=========
服务器响应信息首部：HTTP/1.1 304 Not Modified
```

HTTP 代理服务器源代码：

```
//用来描述 HTTP 头部信息的类
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;
  }
}
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 {
    //设置端口号
    ProxyServerSocket=new ServerSocket(port);
    //设定连接最长时间
    ProxyServerSocket.setSoTimeout(1000000);
  } catch (IOException e) {
    System.out.println("初始化失败");
    return false;
  }
  return true;

}

static boolean UserFilterInit() {
  UserFilter.add("127.0.0.1");
  UserFilter.add("1.1.1.1");
  UserFilter.add("5.6.7.8");
  UserFilter.add("0.0.0.0");
  return UserFilter.size()>0;
}


public static void main(String[] args) {
  int ProxyPort=10240;
  System.out.println("代理服务器准备中.....");
  if (InitSocket(ProxyPort)) {
    System.out.println("开始监听端口："+ProxyPort);
  }
  else//代理服务器用于监听的套接字创建失败就退出程序。
  {
    System.exit(0);
  }

  //UserFilterInit();
  while (true) {
    try {
      //监听
      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);
```

```
          }
        }

        //创建新线程来代理客户端到服务器端的通信
        new Thread(new ProxyThread(socket)).start();

      } catch (IOException e) {
        System.err.println("连接超时");
      }

    }
  }
}
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;


public class ProxyThread implements Runnable {
  // 定义 http 端口号，默认为 80
  static int HttpPort = 80;
  static int size=100000;

  // 定义超时时间
  static int timeout = 500000;

  //客户端和代理服务器的 socket
  private Socket ClientSocket = null;
  private Socket SerSocket = null;

  //socket 的写入写出方法包装
  /*
  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。

```
 */
  private InputStream ClientInputStream = null;//客户端输入流
  private InputStream SeverInputStream = null;//目的服务器输入流
  private BufferedReader ClientBufferReader = null;//客户端输入字节流
  private BufferedReader SeverBufferReader = null;//目的主机输入字节流
  private OutputStream ClientOutputStream = null;//客户端输出流
  private OutputStream SeverOutputStream = null;//目的服务器输出流
  private PrintWriter ClientPrintWriter = null;//客户端输出字节流
  private PrintWriter SeverPrintWriter = null;//目的主机输出字节流

  // 利用键值对进行缓存
  // 对象被缓存的具体时间
  static Map<String, String> timecache = new HashMap<>();

  // 对象被缓存的具体数据
  static Map<String, List<Byte>> bytescache = new HashMap<>();

  //用于网站过滤的列表
  static List<String> WebsiteFilter=new ArrayList<>();

  //用于钓鱼
  static Map<String,String> fishguide=new HashMap<>();


  public ProxyThread(Socket clientsocket) throws IOException {
    super();
    this.ClientSocket = clientsocket;
    this.ClientInputStream = clientsocket.getInputStream();
    this.ClientBufferReader = new BufferedReader(new
InputStreamReader(ClientInputStream));
    this.ClientOutputStream = clientsocket.getOutputStream();
    this.ClientPrintWriter = new PrintWriter(ClientOutputStream);

    WebsiteFilter.add("jwts.hit.edu.cn");
    fishguide.put("jwes.hit.edu.cn", "http://www.hit.edu.cn/");
```

```
}

/**
 * 解析 http 头的信息，获取 method，url，host，cookie
 */
public HttpHeader ParseHeader(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";//https 里的特殊情况
  }
  //切出 Host 和 Cookie 的信息
  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;
}


/**
 * 获取代理服务器与服务器沟通的套接字
 * @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沟通的套接字
 * @throws UnknownHostException
 * @throws IOException
 */
public Socket ConnectToServer(String host, int port, int times) throws
```

```
UnknownHostException, IOException {
    for (int i = 0; i < times; i++) {
      SerSocket = new Socket(host, port);
      //设置端口连接时间
      SerSocket.setSoTimeout(timeout);
      SeverInputStream = SerSocket.getInputStream();
      SeverBufferReader = new BufferedReader(new
InputStreamReader(SeverInputStream));
      SeverOutputStream = SerSocket.getOutputStream();
      SeverPrintWriter = new PrintWriter(SeverOutputStream);

      if (SerSocket != null) {
        return SerSocket;
      }
    }
    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=========代理服务器向服务器发送请求信息=========");
    for (int i = 0; i < lst.size(); i++) {
      String line = lst.get(i);
      SeverPrintWriter.write(line + "\r\n");
      System.out.println(line);
    }
    SeverPrintWriter.write("\r\n");
    SeverPrintWriter.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=========转发响应信息到客户端=========\n");
```

```
  List<Byte> lst=new ArrayList<>();

  try {
    // String time = null;
    byte bytes[] = new byte[size];
    int len;
    while (true) {
      if ((len = SeverInputStream.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);
    }
    //将二进制数组转为字符串
    String s=new String(b);
    String time=findTime(s);
    timecache.put(url, time);
    bytescache.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=timecache.get(url);
    // 发送条件性 GET 方法到服务器
    SeverPrintWriter.write(header.get(0) + "\r\n");
    SeverPrintWriter.write("Host: "+host + "\r\n");

    System.out.println("Modified Time:"+modifiTime);
    String str = "If-modified-since: " + modifiTime + "\r\n";
    SeverPrintWriter.write(str);
    SeverPrintWriter.write("\r\n");
    SeverPrintWriter.flush();

    try {
      String ServerMessage = SeverBufferReader.readLine();
      //System.out.println(ServerMessage);
      if (ServerMessage == null) {
        return false;
      }
      System.err.println("\n 服务器响应信息首部: "+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"))
      {
        System.err.println("\n=======缓存未更新======");
        List<Byte> lst=bytescache.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"))
{
  System.err.println("\n=======缓存已更新======");
  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 = SeverInputStream.read(bytes)) > 0) {
      ClientOutputStream.write(bytes, 0, len);
    } else if (len < 0) {
      break;
    }
  }
  //缓存无效，去除.
  timecache.remove(url);
  bytescache.remove(url);


  ClientPrintWriter.write("\r\n");
  ClientPrintWriter.flush();
  ClientOutputStream.close();
}
else {  //http 消息无效的情况
  bytescache.remove(url);
  timecache.remove(url);
  while (!SeverBufferReader.readLine().equals(""))
  {
    //刷完无效信息
  }
  byte bytes[] = new byte[size];
  int len;
  while (true) {
    if ((len = SeverInputStream.read(bytes)) >= 0) {
      //ClientOutputStream.write(bytes, 0, len);
    } else if (len < 0) {
      break;
```

```
          }
        }
        run();//重新来一遍
      }
    } catch (IOException e1) {
    }

    return true;
  }

  public void Filter() {
    for (int i=1;i<419;i++) {
      ClientPrintWriter.write("非法网站! FBI WARNING!\t\t");
      if (i%11==0) {
        ClientPrintWriter.write("\r\n");
      }
    }
    ClientPrintWriter.write("\r\n");
    ClientPrintWriter.flush();
    ClientPrintWriter.close();
  }

  public void fishing(List<String> header){

    header.clear();
    header.add("GET http://www.hit.edu.cn/ HTTP/1.1");
    header.add("Host: www.hit.edu.cn");
    header.add("User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0)
Gecko/20100101 Firefox/70.0");
    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=ParseHeader(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=timecache.containsKey(url)&&bytescache.containsKey(url);
    if (!flag) {
      /*
       * 没有 Cache 的情况
       */
      System.err.println("\n=========无缓存=========");
      SendToServer(header);
      SendBackToClient(url);
    } else {
      /*
       * 有 Cache 的情况
       */
      System.err.println("\n=========有缓存=========");
      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=========客户端请求=========");
      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 = ParseHeader(header);
String url = httpHeader.url;
String host = httpHeader.host;

//网站引导（钓鱼）
for (String h:fishguide.keySet()) {
  if (host.contains(h)) {
    fishing(header);
    return;
  }
}
//不处理 https 里的 CONNECT 方法
if (httpHeader.method.equals("CONNECT")) {
  return;
}

//网站过滤
for (int i=0;i<WebsiteFilter.size();i++) {
  if (host.contains(WebsiteFilter.get(i))) {
    Filter();
    System.err.println("禁止访问："+url);
    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= timecache.containsKey(url)&&bytescache.containsKey(url);
if (!flag) {
 // 没有 Cache 的情况
  System.err.println("\n=========无缓存=========");
```

```
        SendToServer(header);
        SendBackToClient(url);
      } else {
       // 有 cache 的情况
        System.err.println("\n==========有缓存==========");
        SendBackToClientWithCache(header, host,url);
      }

    } catch (SocketException e) {
      //e.printStackTrace();
    } catch (IOException e) {
      //e.printStackTrace();
    }catch (Exception e){
      //e.printStackTrace();
    }

  }
}
```

四、实验心得

纸上得来终觉浅，绝知此事要躬行。本来我自以为通过 MOOC 上的课程已经较好地掌握了 socket 接口编程知识以及代理服务器的知识。但真正做起实验来，我才发现在 MOOC 中我只是大概了解了一下 socket 编程的函数操作，并且大致了解了代理服务器的功能，而对其具体的实现，并没有相应的概念。这次实验让我加深了对 socket 编程和代理服务器的理解，并且从完成这部分内容中收获了相当大的乐趣。

这也和辩证的历史唯物主义哲学中认知与实践的关系相一致，认知与实践不可分割，认知指导实践，而在实践中我们又得到新的认知。因此，在今后的学习过程中，我不仅仅要了解一个知识点表面的内容，更要敢于动手实践，多做多写多练，才能发现自己的不足。