哈爾濱工業大學
Harbin Institute of Technology

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	黄栩钧		院系	计算学部软件学院	
班级	2137101		学号	2021111221	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物楼 207		实验时间	2023/10/16	
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					

哈尔滨工业大学计算学部
FACULTY OF COMPUTING, HIT

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，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：

一.Socket 编程的客户端和服务器端主要步骤

客户端：

1. 使用 ServerSocket 创建一个服务器套接字 server，并指定端口(因为该代理服务器只作用于本地 127.0.0.1，所以不用设置别的参数)，用于监听客户端请求

```
ServerSocket server = new ServerSocket(port);
```

2. server 监听到的客户端的请求后，使用 Socket 函数来创建一个唯一的 socket 描述符来建立客户端和代理服务器的连接

```
Socket socket = server.accept();
```

3. 创建线程池以实现多线程接收客户端请求

```
ExecutorService threadPool = Executors.newFixedThreadPool( nThreads: 100);
```

```
threadPool.submit(new ProxyHandler(socket));
```

4. 用 socket 的 getInputStream 方法读取客户端传输的信息，从原本客户端想要向远程服务器传输的 header 中解读出各类信息

```
InputStreamReader r = new InputStreamReader(ClientSocket.getInputStream());
```

```
BufferedReader br = new BufferedReader(r);
String readLine = br.readLine();
```

```
while (readLine != null && !readLine.equals(""))
    header.append(readLine).append("\n");
```

```
Map<String, String> map = parseTelegram(header.toString());
```

5.通过代理服务器获取远程服务器的信息后，用 socket 的 getOutputStream 方法，向客户端传输信息

```
OutputStream ClientSocketOut = ClientSocket.getOutputStream();
```

```
while (( size = inputStream.read(buf)) != -1) {
    ClientSocketOut.write(buf, off: 0,size);    //
}
ClientSocketOut.flush();
```

6.信息传输结束后，通过 socket 的 close 方法关闭套接字

```
ClientSocket.close();
```

服务器端：

1.根据目标主机号和地址，设置套接字

```
Socket ServerSocket = new Socket(host,visitPort);
```

2.建立代理服务器与远程服务器之间的连接

3.通过 getOutStream 方法向远程服务器传输信息

```
OutputStream ServerSocketOut = ServerSocket.getOutputStream();
```

```
ServerSocketOut.write(header.toString().getBytes());
ServerSocketOut.flush();
```

4.通过 getInputStream 方法获取远程服务器返回的信息

```
InputStream ServerSocketIn = ServerSocket.getInputStream();
```

```
BufferedInputStream inputStream=new BufferedInputStream(ServerSocketIn);
```

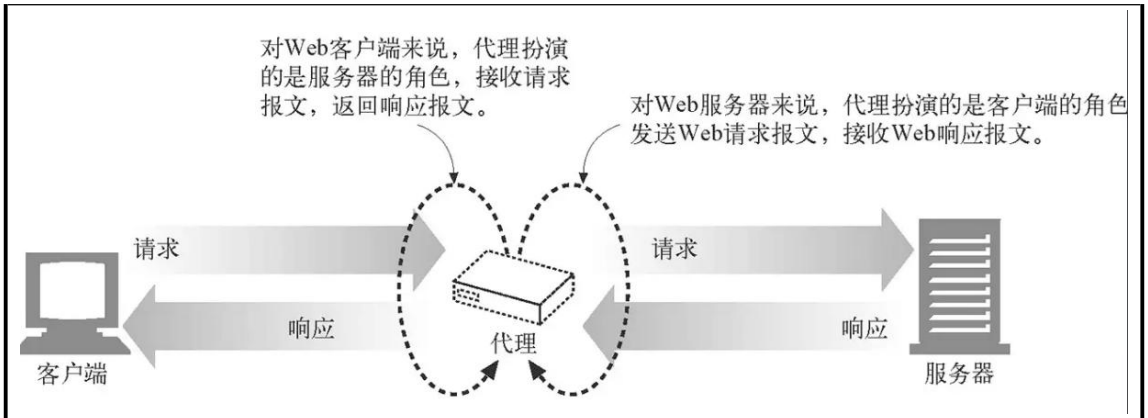
5. 将远程服务器返回的信息通过套接字传输给客户端

```
while (( size = inputStream.read(buf)) != -1) {
    ClientSocketOut.write(buf, off: 0,size);
```

6.信息传输结束后，关闭与远程服务器的连接

二.HTTP 代理服务器的基本原理

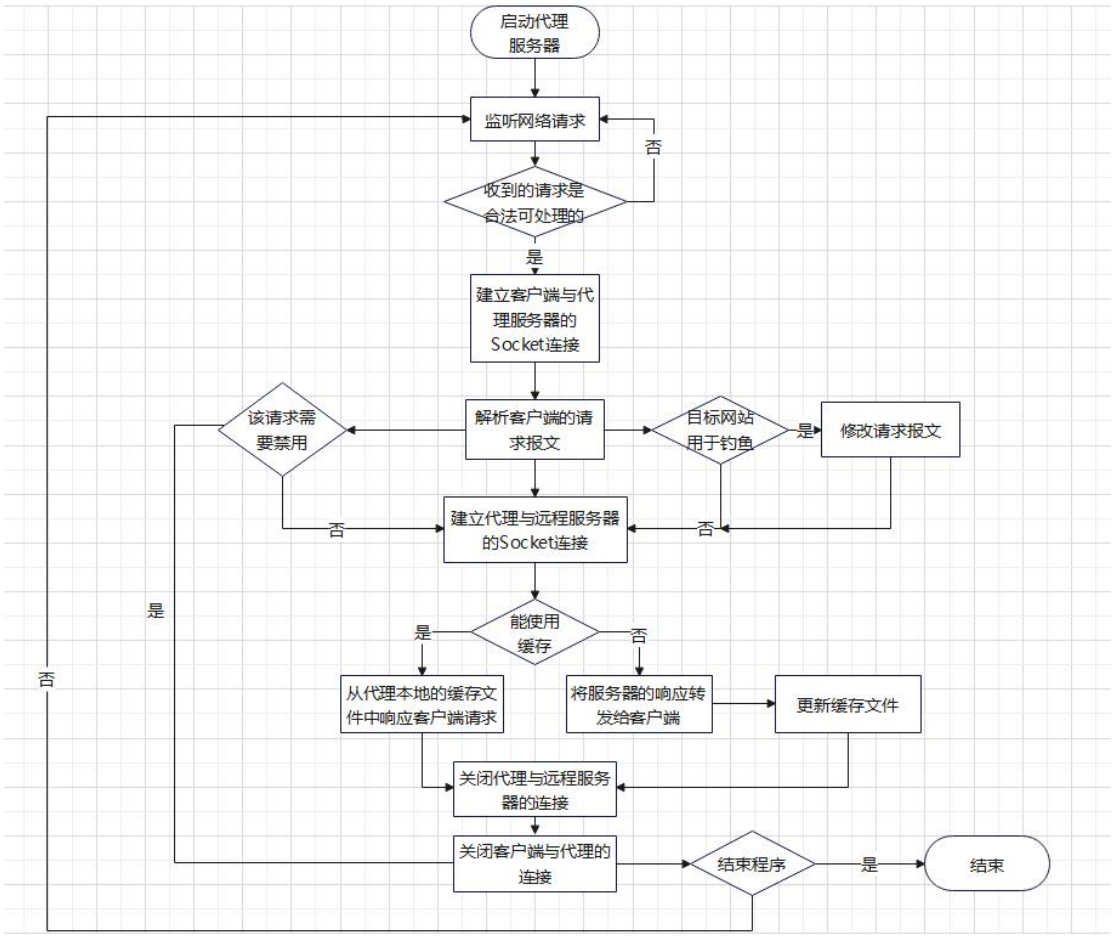
一般的 Web 应用通信方式是由浏览器在访问网站时向远程服务器发送请求，服务器响应后将信息传输给浏览器以显示相应页面。而 HTTP 代理服务器能够为两者实现非直接的连接，即由代理服务器作为中间的桥梁传输数据。



如上图所示，代理服务器会接收客户端的请求，对其进行解析处理后，传输给远程服务器，并在远程服务器响应后，将响应报文传输给客户端。在这个过程中，代理服务器既承担了对客户端时的服务器身份，又承担了对服务器的客户端身份。

发送给远程服务器的报文能够先在代理服务器进行一系统处理，例如根据请求中的主机号、地址、IP 等信息决定是否发送给服务器以实现禁用功能；修改请求中的目标主机号和地址以实现钓鱼功能等。而且作为数据传输的中转站，代理服务器在服务器响应数据缓存在本地，以加快下一次访问。

三.HTTP 代理服务器的程序流程图



四.实现 HTTP 代理服务器的关键技术及解决方案；

1. HTTP 报文解读

在建立起浏览器客户端和代理服务器的连接后，需要接收客户端的请求，并解析其中的报文内容，以获取请求的方式、目标地址、http 版本、主机号、端口号等信息。

在代码实现中，通过定义一个 parseTelegream 方法，将 HTTP 报文头部按其格式逐行提取信息，根据其具体类型从中获取需要的信息，并以 Map 数据结构存储。

```
/**
 * 通过header解析各个参数.
 *
 * @param header HTTP报文头部
 * @return map，包含HTTP版本，method，访问内容，主机，端口
 */
1 个用法
private static Map<String, String> parseTelegram(String header)
```

2. Socket 数据传输

本地代理服务器 socket 接收到客户端请求后，需要建立两个套接字，ClientSocket 和 ServerSocket,分别连接客户端和服务器。值得注意的是，ClientSocket 的输入端用于接收客户端请求，输出端用于传输数据响应客户端. ServerSocket 的输入端用于接收服务器请求，输出端用于向服务器传输来自客户端的请求。

在代码实现中要注意各个输入输出端口的使用，切勿混淆。

```
InputStreamReader r = new InputStreamReader(ClientSocket.getInputStream());  //读取客户端传输的信息
OutputStream ClientSocketOut = ClientSocket.getOutputStream(); //用于从代理服务器向客户端传递信息
```

```
OutputStream ServerSocketOut = ServerSocket.getOutputStream(); //用于将数据写入远程服务器
InputStream ServerSocketIn = ServerSocket.getInputStream(); //用于从远程服务器接收数据
```

一次完整的代理服务器数据传输流程如下

3. Cache 缓存功能

为了加快响应速度，可以在代理服务器中实现缓存功能，初次访问时将服务器返回的数据存储在本地，此后只需判断是否需要更新缓存，无需更新时直接使用代理服务器的缓存文件。其实现逻辑大致如下：

3.1 HTTP 中，只有 method 为 GET 的请求能够实现缓存功能，所以先判断请求的 method 类型

```
if (method.equals("GET")) {    //只有method为GET时才能使用缓存技术
    CacheManage(ServerSocket,header,host,visitAddr,method,ClientSocketOut,ServerSocketOut,ServerSocketIn);
```

3.2 根据此前在 header 中提取的 host 信息，为目标地址创建专门的缓存文件，并查找当前缓存文件夹中是否含有该文件

```
String dir = "cache" + File.separator + host + ".txt";
```

```
for(File file0: Objects.requireNonNull(folder.listFiles()))
{   //查找当前访问网页是否存在对应缓存文件
    BufferedReader fileReader=new BufferedReader(new InputStreamReader(new FileInputStream(file0)));
    if(fileReader.readLine().equals(visitAddr))//缓存命中
```

3.3 若存在该缓存文件，则提取其中的‘最近更新时间’信息，重新编写发给服务器的报文，在其中添加‘If-Modified-Since’字段以及该时间信息，把该报文发送给远程服务器

```
if(line.contains("Date")){
    lastModifiedDate = line.substring( beginIndex: 6);
```

```
.append("If-Modified-Since: ").append(lastModified).append("\n")
```

```
ServerSocketOut.write(remoteRequest.getBytes());
ServerSocketOut.flush();
```

3.4 获取远程服务器的反馈信息，如果返回状态码为 304，则意味着缓存文件没有更新，可以关闭与远程服务器的连接，直接使用当前存在代理服务器中的缓存文件，将其中的数据写入 ClientSocket 的输出流发送给客户端。

```
if(res.contains("304"))
```

```
ServerSocket.shutdownOutput();
```

```
//读取缓存文件
BufferedInputStream inputStream=new BufferedInputStream(new FileInputStream(file));
```

```
while((length=inputStream.read(bytes))!=-1){
    ClientSocketOut.write(bytes, off: 0,length);
```

3.5 若远程服务器返回的状态码为 200，则意味缓存文件已更新，需要删除当前的缓存文件，重新建立缓存。此后操作与 3.2 中没有缓存文件时是一致的，故一同在后续给出

```
file.delete();
file.createNewFile();
FileOutputStream outputStream=new FileOutputStream(file);
```

3.6 将从远程服务器读取的数据写入新生成的缓存文件中，同时将其写入代理服务器与客户端的输出流中，建立缓存的同时响应客户端请求

```
FileOutputStream outputStream=new FileOutputStream(file);
```

```
outputStream.write(tempBytes, off: 0,len);   //将之前从远程服务器读取的响应数据写入到缓存文件中
ClientSocketOut.write(tempBytes, off: 0,len);   //同时写入代理服务器与客户端之间的输出流，加快响应速度
```

```
ClientSocketOut.write(buf, off: 0,size);
outputStream.write(buf, off: 0,size);
```

4. 用户禁用，网站禁用和钓鱼

在建立起客户端和代理服务器的连接后，就可以从客户端发送的请求中获取用户信息、预访问网站信息，然后就可以先对这些信息进行分析处理，查看是否有需要屏蔽的用户、网站，或者是否需要对目标地址进行替换以实现钓鱼

要先建立禁用列表，钓鱼替换列表

```
/** 禁止访问的网址。 */
private static final Set<String> forbidSetList = new HashSet<>();
/** 禁止访问的用户。 */
private static final Set<String> forbidUserList = new HashSet<>();
/** 重定向主机map。 */
private static final Map<String, String> redirectHostMap = new HashMap<>();
/** 重定向访问网址map。 */
private static final Map<String, String> redirectAddrMap = new HashMap<>();
```

然后在静态表中存储需要禁用的用户、地址，以及需要替换的源地址和目的地址，通过注释代码与否实现禁用和替换功能的开关

```
static {
    //屏蔽指定网址
    forbidSetList.add("http://jwts.hit.edu.cn/");
    forbidSetList.add("jwts.hit.edu.cn");
      //屏蔽指定用户
    forbidUserList.add("127.0.0.1");
      //设置用于钓鱼的源地址和目的地址
    redirectAddrMap.put("http://jwes.hit.edu.cn/", "http://www.example.com/");
    redirectHostMap.put("jwes.hit.edu.cn", "www.example.com");
}
```

此后只需要在主进程中获取到相应信息时进行判断，如果有需要禁用的请求，就中止进程，如果有需要替换的地址，就调用相应的 redirect 方法替换地址

```
if (forbidUserList.contains(ClientSocket.getInetAddress().getHostAddress())) {
    System.out.println("当前用户已被禁止访问");
```

```
if (visitAddr != null && isForbidden(visitAddr))
```

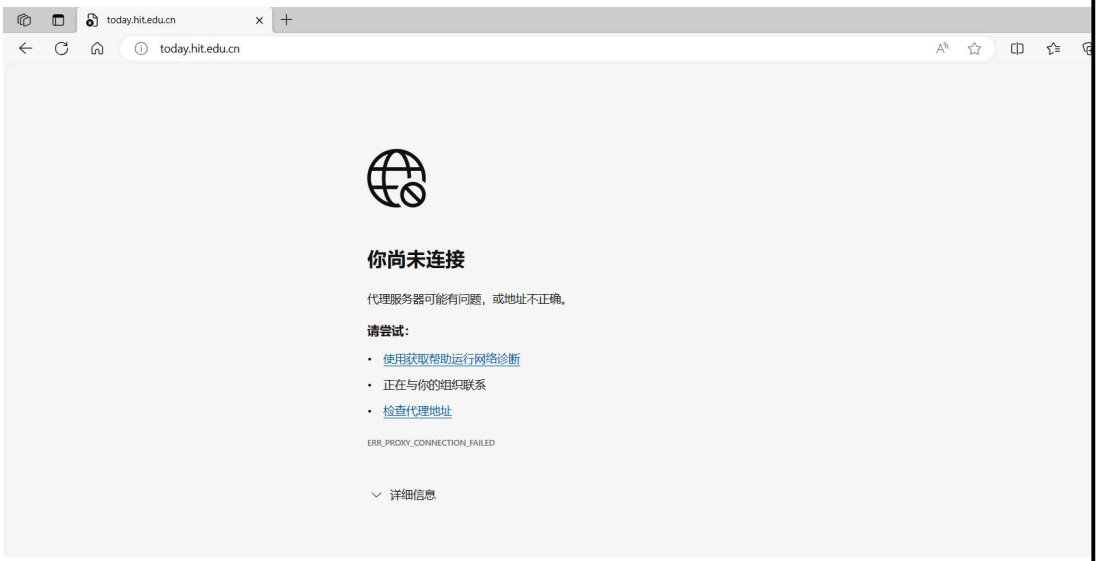
```
String tempRedAddr = redirectAddr(visitAddr);  //判断当前访问地址是否需要被重定向
if (!tempRedAddr.equals(visitAddr)) {
    visitAddr = tempRedAddr;
    host = redirectHost(host);
```

实验结果：

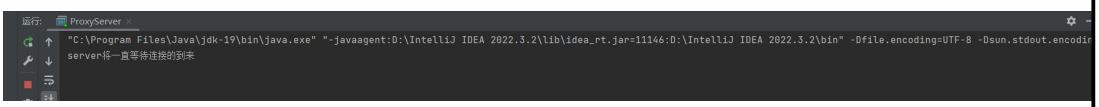
HTTP 代理服务器实验验证过程以及实验结果：

1.开启浏览器代理

2.打开一个使用http协议的网站，发现已经无法访问



3.启动代理服务器

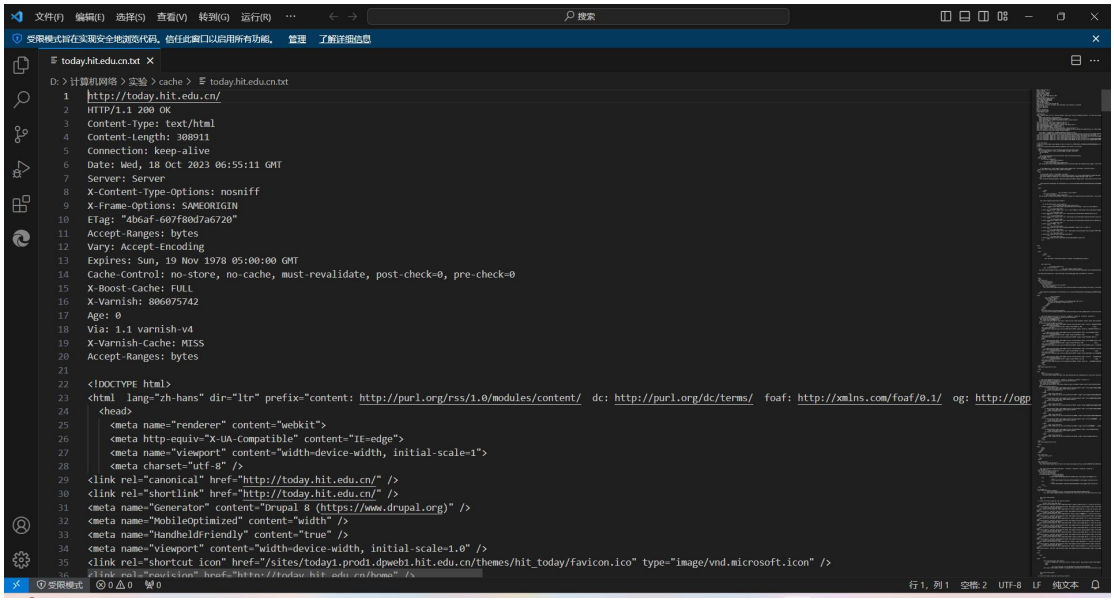


4.刷新刚才的网页，可以发现可以访问了，观察代理服务器的运行窗口，可以看到接收到的网页请求。第一次执行时显示缓存未命中，因为此时还没有缓存文件，再次刷新网页发送请求，显示缓存命中，并且可以在缓存文件中找到存

储下来的响应数据


```
获取到一个来自 127.0.0.1的连接
-------------------
{httpVersion=HTTP/1.1, method=GET, port=80, host=today.hit.edu.cn, visitAddr=http://today.hit.edu.cn/}
today.hit.edu.cn
http://today.hit.edu.cn/     miss!缓存未命中
```

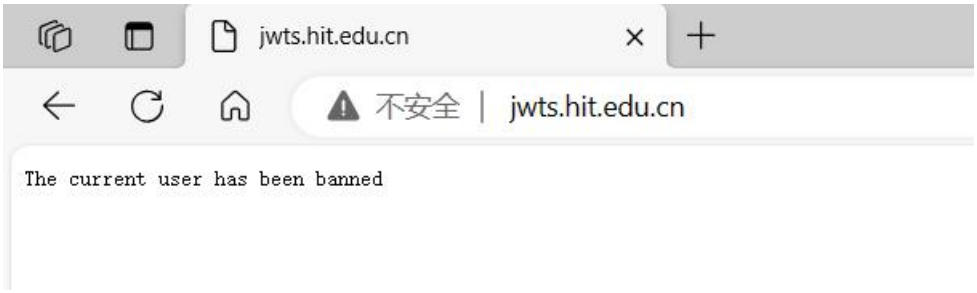
```
获取到一个来自 127.0.0.1的连接
-------------------
{httpVersion=HTTP/1.1, method=GET, port=80, host=today.hit.edu.cn, visitAddr=http://today.hit.edu.cn/}
today.hit.edu.cn
http://today.hit.edu.cn/-->-->缓存命中!
缓存内容未更新，直接使用
```



5.启用用户禁用功能，让代理服务器禁止本机用户访问。这时可以看到此前能打开的网页都显示禁止访问，代理服务器的运行窗口也显示此用户被禁止访问

```
    //屏蔽指定用户
forbidUserList.add("127.0.0.1");
```

```
ProxyServer
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:
server将一直等待连接的到来
当前用户已被禁止访问
```



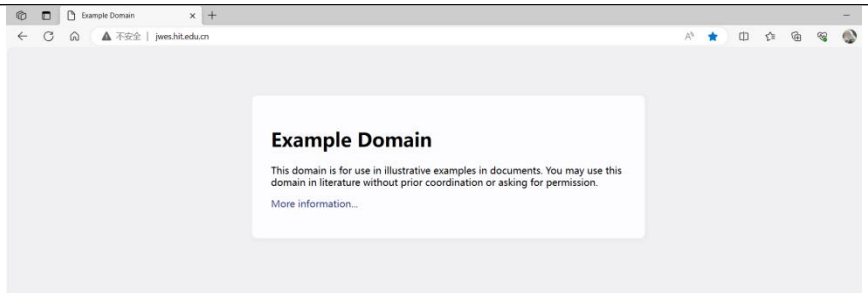
6. 关闭用户禁用后，启用网页禁用功能，可以看到网页已无法访问，运行窗口也页面中也都显示了相应提示

```
//屏蔽指定网址
forbidSetList.add("http://jwts.hit.edu.cn/");
forbidSetList.add("jwts.hit.edu.cn");
```

```
获取到一个来自 127.0.0.1的连接
------------------
{httpVersion=HTTP/1.1, method=GET, port=80, host=jwts.hit.edu.cn, visitAddr=http://jwts.hit.edu.cn/}
该站点已被禁止访问
```

7. 关闭网页禁用，启用网页钓鱼功能，可以看到输入学校的教务系统网址，却跳转到了一个无关的样例网站，运行窗口中的展现了地址的转换。

```
  //设置用于钓鱼的源地址和目的地址
redirectAddrMap.put("http://jwes.hit.edu.cn/", "http://www.example.com/");
redirectHostMap.put("jwes.hit.edu.cn", "www.example.com");
```

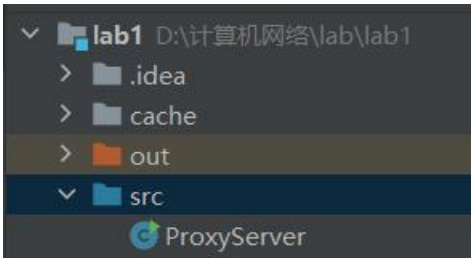



```
获取到一个来自 127.0.0.1的连接
-------------------
{httpVersion=HTTP/1.1, method=GET, port=80, host=jwes.hit.edu.cn, visitAddr=http://jwes.hit.edu.c
originAddr: http://jwes.hit.edu.cn/
redirectAddr: http://www.example.com/
originHost: jwes.hit.edu.cn
redirectHost: www.example.com
www.example.com
http://www.example.com/-->-->缓存命中！
缓存内容未更新，直接使用
```

至此已展示完此次HTTP代理服务器的全部功能，包括建立客户端和远程服务器之间的连接，实现缓存功能，用户禁用、网站禁用与网站钓鱼重定向功能。

程序设计结构：

项目结构如下，src中存储了ProxyServer.java用于实现代理服务器功能，cache文件夹用于存储各个缓存文件。



项目代码中有以下几个主要类和方法

```
通过header解析各个参数.

private static Map<String, String> parseTelegram(String header)
```

```
/**
 * 重新编写发送给远程服务器的报文
 * @param host 主机号
 * @param method 请求方法
 * @param visitAddr 访问地址
 * @param lastModified 最近修改时间
 * @return 重构后的报文
 */
private static StringBuilder SetRemoteTemplate(String host, String method, String visitAddr, String lastModified)
```

```
重新编写发送给远程服务器的报文

private static StringBuilder SetRemoteTemplate(String host, String method, String visitAddr, String lastModified)
```

```
/** 实现缓存文件的管理

private static void CacheManage(Socket ServerSocket, StringBuilder header,
                                String host, String visitAddr, String method,
                                OutputStream ClientSocketOut, OutputStream ServerSocketOut, InputStream ServerSocketIn) th
```

```
执行代理相关功能
static class ProxyHandler implements Runnable {
    private final Socket ClientSocket;
    public ProxyHandler(Socket ClientSocket) { this.ClientSocket = ClientSocket; }
    // 创建缓存
    @Override
    public void run() {
```

其主要功能都已在注释中给出，程序运行逻辑大致如下：

1.main函数中启动程序，使用ServerSocket类的对象监听客户端请求，创立线程池，每个线程的请求处理通过类ProxyHandler实现。

2.ProxyHanlder中通过两个套接字进行相应处理，在其中调用parseTelegram解析header，调用SetRemoteTemplate设置报文，调用CacheManage进行缓存管理，以及调用一些简单的方法实现禁用、重定向等功能

问题讨论：

1.在实现代理服务器的缓存功能的过程中，遇到了很多问题，其中包括：缓存文件存储失败、存储的缓存文件都是乱码无法使用、服务器返回的状态码永远都是200、使用缓存时的页面加载速度反倒不如不使用缓存时等等。从修复这些问题的过程中我得出了一些小经验：

- 要注重文件的存储地址和编码格式
- 给请求报文添加if-since-then或者其它字段时，要注重原有的报文格式与信息
- 调用缓存数据时要注重数据流类型

2.使用代理服务器时，有大量图片的网站在没有缓存的情况下往往加载很慢，可能是因为通过代理服务器接收和发送报文请求时没有采用流水形式，以至于经由代理这个中介会产生很多次发送和接收的时间消耗，这一点可作为代理服务器的可提升内容

3.目前实现的代理服务器只能访问使用Http协议的网页，而如今大部分网页都是使用更安全的Https协议。Http都是明文传输数据的，并不安全，而Https则是加密传输的。Https实际上就是架构在SSL\TLS之上的Http协议。可以考虑从修改socket创建方式、丰富报文解析方式、修改数据传输方式等角度拓展升级上前的代理服务器，使其能访问使用Https协议的网页。

心得体会：

通过这次实验，我对Socket网络编程的过程与技术有了更深刻的学习和了解。在实验过程中，通过处理连接客户端和服务器的两个socket,对它们各自的职能和作用，以及accept,connect,close,getInputStream和getOutputStream等socket相关方法的使用也都更加熟练。

在解析客户端请求报文，并重新处理传输给服务器的报文的过程中，我对HTTP协议也有了更深刻的理解，对其传输形式，报文格式都掌握得更加具体了。

在逐步实现Http代理服务器，以及缓存和禁用、钓鱼等功能的过程中，我对其基本工作原理也有了比以往更形象立体的理解，真正做到了对其各流程意义的掌握。在编码实现的过程中，也进一步提升了我的编码水平和逻辑架构思维。