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	马旭		院系	计算科学与技术学院	
班级	1603106		学号	1160300601	
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	2018.10.31	
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					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

目录

1.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。

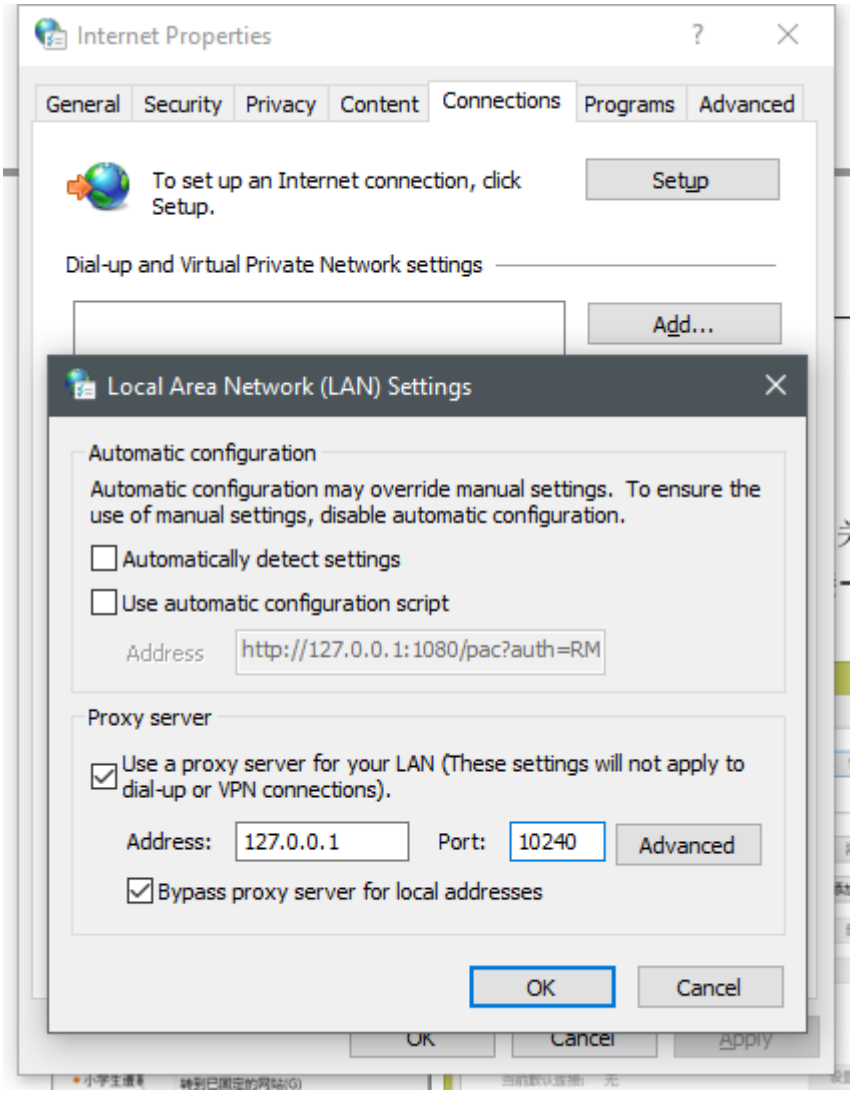
2.实验内容

(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）。

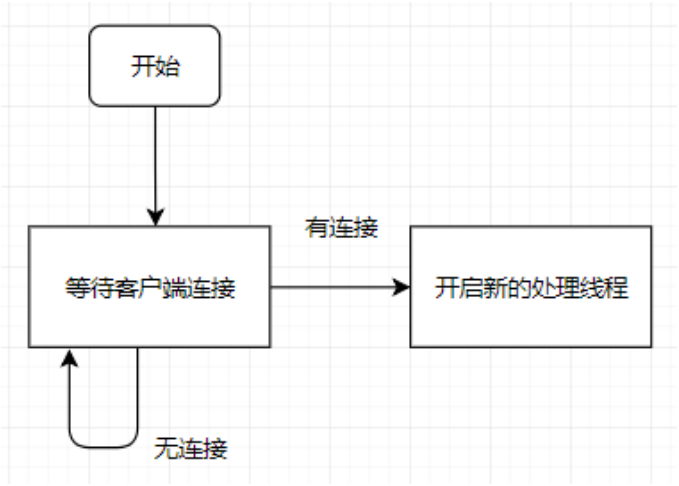
3.实验过程

3.1.浏览器使用代理

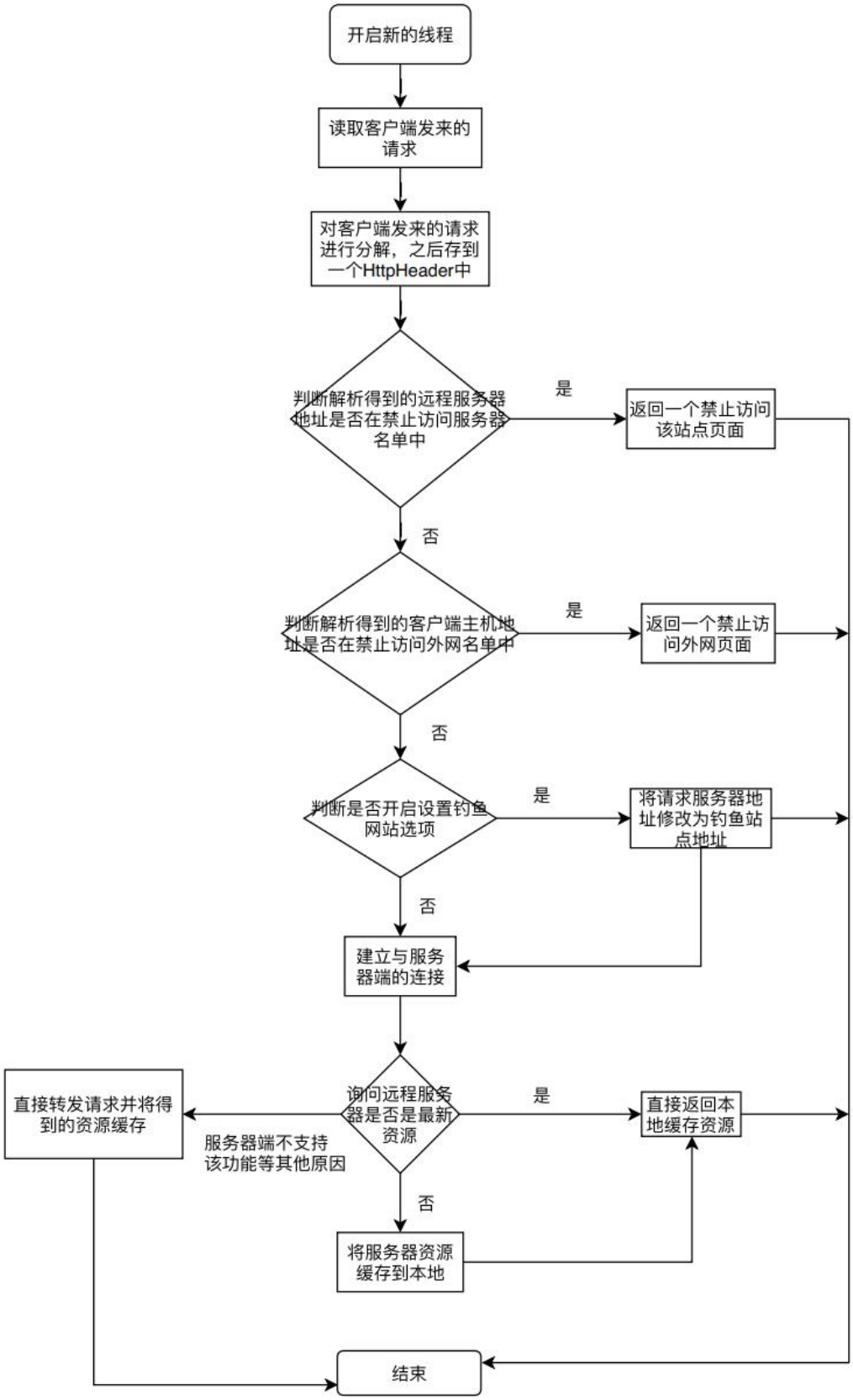


3.2 基本流程图

1. 总体框架流程图



2. 每个处理线程流程图


开启新的线程
读取客户端发来的请求
对客户端发来的请求进行分解，之后存到一个HttpHeader中
判断解析得到的远程服务器地址是否在禁止访问服务器名单中
是
返回一个禁止访问该站点页面
否
判断解析得到的客户端主机地址是否在禁止访问外网名单中
是
返回一个禁止访问外网页面
否
判断是否开启设置钓鱼网站选项
是
将请求服务器地址修改为钓鱼站点地址
否
建立与服务器端的连接
询问远程服务器是否是最新资源
是
直接返回本地缓存资源
服务器端不支持该功能等其他原因
直接转发请求并将得到的资源缓存
否
将服务器资源缓存到本地
结束

3.3 实现所有功能

1. 实现用户过滤

在每个处理的线程中首先判断被禁止的客户端 IP 或者 Cookie 是否在禁止名单中，如果在，就不连接外部网络直接向客户端返回一个禁止访问的页面。实现代码如下：

```
// 禁止某些用户访问外部网络
if (forbid.containUser(header.getCookie())
    || forbid.containUser(this.clisocket.getInetAddress().toString())) {
  cliOutStream.write(StaticData.FORBIDUSER_FAIL.getBytes());
  cliOutStream.flush();
  return;
}
```

2. 网站过滤

在每个处理的线程中首先判断远程服务器的 IP 或者 host 是否在被禁止的服务 IP 名单中,如果在，就不连接外部网络直接向客户端返回一个禁止访问页面。实现代码如下：

```
// 禁止对某些url或者主机访问
if (forbid.containUrl(header.getUrl())
    || forbid.containHost(header.getHost())) {
  cliOutStream.write(StaticData.FORBIDURL_FAIL.getBytes());
  cliOutStream.flush();
  return;
}
```

3. 网站引导

在内部实现一个处理钓鱼网站的函数，在处理的线程中首先判断是否用户是否打开网站引导功能，如果打开，就将接下来要连接的远程服务器地址和端口改成内部设置的钓鱼网站的服务器地址和端口。

钓鱼网站的函数如下：

```java
/**
 * 设置钓鱼站点,钓鱼站点默认存储在StaticData的PHISHING_IP中 如果该变量为null或者为空则不进行钓鱼站点的设置
 *
 * @param header
 */
private void setPhishingSite(HttpHeader header) {
  if (StaticData.PHISHING_IP == null || StaticData.PHISHING_IP.isEmpty()) {
    return;
  }
  header.setHost(StaticData.PHISHING_IP);
  header.setPort(StaticData.PHISHING_PORT);
  System.out.println("Set up phishing site successfully!");
}
```

在主函数中进行的判断如下：

```java
// 设置钓鱼网站
if (openPushingSite) {
  this.setPhishingSite(header);
}
```

4.实现 if-modified-since 的缓存功能

在实现该功能时，首先实现一个 askNewFile 询问是否本地是最新文件，在该函数中如果是最新文件就直接返回 true,如果不是最新文件就向远处服务器请求最新文件缓存在本地后返回 true,其他情况返回 false;接着如果返回值为 true 就执行 cacheFile 函数将本地文件传输给远程服务器，如果为 false 就直接与远程服务器连接，获取最新文件，将其缓存在本地并转发给客户端。

其中 askNewFile 函数如下：

```java
/**
 * 询问某文件是否为最新资源，如果已经是最新资源则返回true，如果不是最新资源且从远程服务器获取到最新资源则返回true；
 *  其他情况一律返回false
 *
 * @return
 * @throws Exception
 */
private boolean askNewFile(String strUrl, InputStream inputStream,
    OutputStream outStream) throws Exception {
  Path path = utils.getPathFromURL(strUrl, StaticData.CACHE_FILE_ROOT);
  File file = path.toFile();
  // 如果文件不存在直接返回 false 知道该文件不存在
  // 否则构建一个条件请求头
  if (!file.exists()) {
    return false;
  } else {
    URL url = new URL(strUrl);
    // 获得文件修改的最后时间
    SimpleDateFormat format = new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss",
        Locale.ENGLISH);
    String lastModifyTime = format.format(file.lastModified());

    // 对于主页不进行询问，因为主页不确定是以.jsp,.html,或者其他类型的主页，直接返回false重新加载
    if (url.getPath().endsWith("/")) {
      return false;
    }
```

```java
  // 构建条件请求头
  String request = "GET " + url.getPath() + " HTTP/1.1\r\n" + "Host: "
      + url.getHost() + "\r\n" + "If-modified-since: " + lastModifyTime
      + "\r\n\r\n";

  // 向远程服务器发送条件请求头
  outStream.write(request.getBytes());
  outStream.flush();

  // 从远程服务器接受反馈
  String get = utils.readLineByIs(inputStream);

  // 如果从远程服务器没有收到反馈消息就直接当作本地没有该文件处理
  if (get == null || get.isEmpty()) {
    return false;
  } else if (get.contains("304") || get.contains("200")) { // 如果包含200或者304就说明已经是最新的内容就返回true
    return true;
  } else { // 否则不是最新内容，就从远处获取最新内容
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    fileOutputStream.write(get.getBytes());
    fileOutputStream.flush(); // 先将刚才读取的内容缓存到本地
    utils.fromInputToOutput(inputStream, fileOutputStream,
        StaticData.SERVER_BUFSIZE, null); // 再将流中剩下的内容缓存在本地
    fileOutputStream.close();
    return false;
  }
```

其中 cacheFile 函数如下：

```java
/**
 * 缓存文件函数,判断url资源是否被缓存下来了，如果被缓存下来就将该文件 输出到 cliOutStrem 输出流中
 *
 * @param Strurl
 *          url资源
 * @param outStream
 *          一个输出流
 * @return
 * @throws Exception
 */
private boolean cacheFile(String Strurl, OutputStream outStream)
    throws Exception {

  Path path = utils.getPathFromURL(Strurl, StaticData.CACHE_FILE_ROOT);
  File file = path.toFile();
  if (file.exists()) {
    FileInputStream inputStream = new FileInputStream(file);
    utils.fromInputToOutput(inputStream, outStream, StaticData.SERVER_BUFSIZE,
        null);
    inputStream.close();
    return true;
  } else {
    return false;
  }
}
```

在线程中处理的逻辑代码如下：

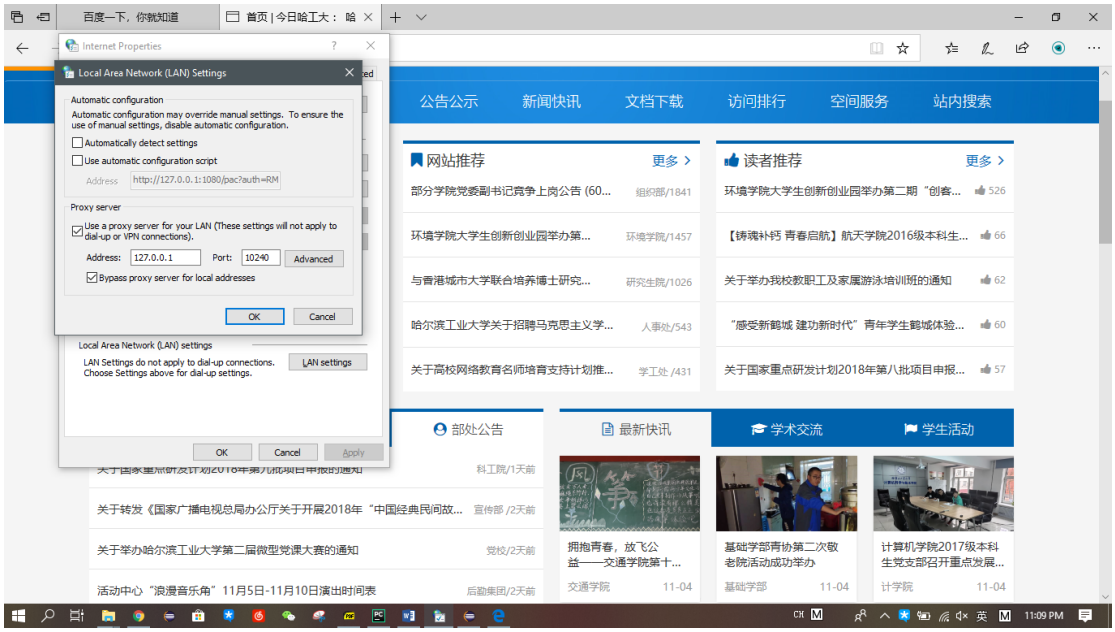
```java
// 询问文件是否是最新版
if (this.askNewFile(header.getUrl(), serInStream, serOutStream)) {
  this.cacheFile(header.getUrl(), cliOutStream);
  return;
}
// 如果需要缓存文件，就得到对应的文件流用来缓存
Path path = utils.getPathFromURL(header.getUrl(),
    StaticData.CACHE_FILE_ROOT);
File file = utils.createFile(path);
fileOutStream = new FileOutputStream(file);
System.out
    .println("Successfully connected remote server: " + header.getHost());

// 处理HTTP请求
ProcessHttp(cliInStream, serInStream, cliOutStream, serOutStream, request,
    fileOutStream);
```

4.实验结果

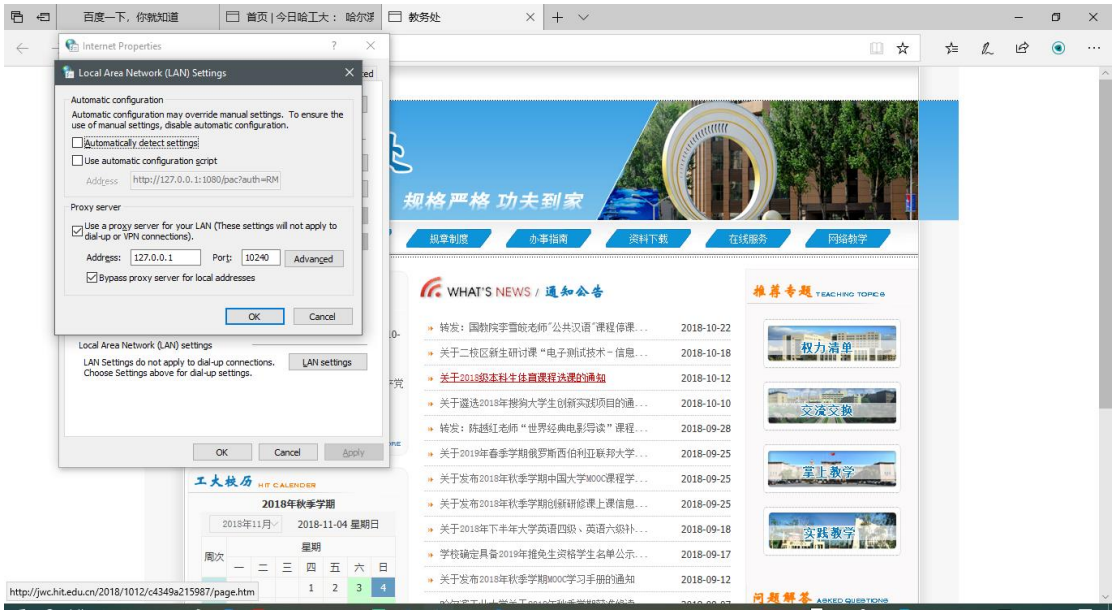
4.1.实现代理

实现今日哈工大代理



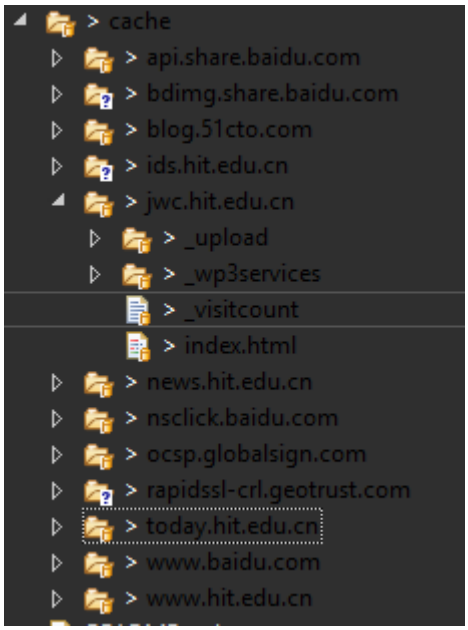

```
Successfully connected remote server: www.baidu.com
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
Successfully connected remote server: today.hit.edu.cn
```

实现工大教务处网页代理

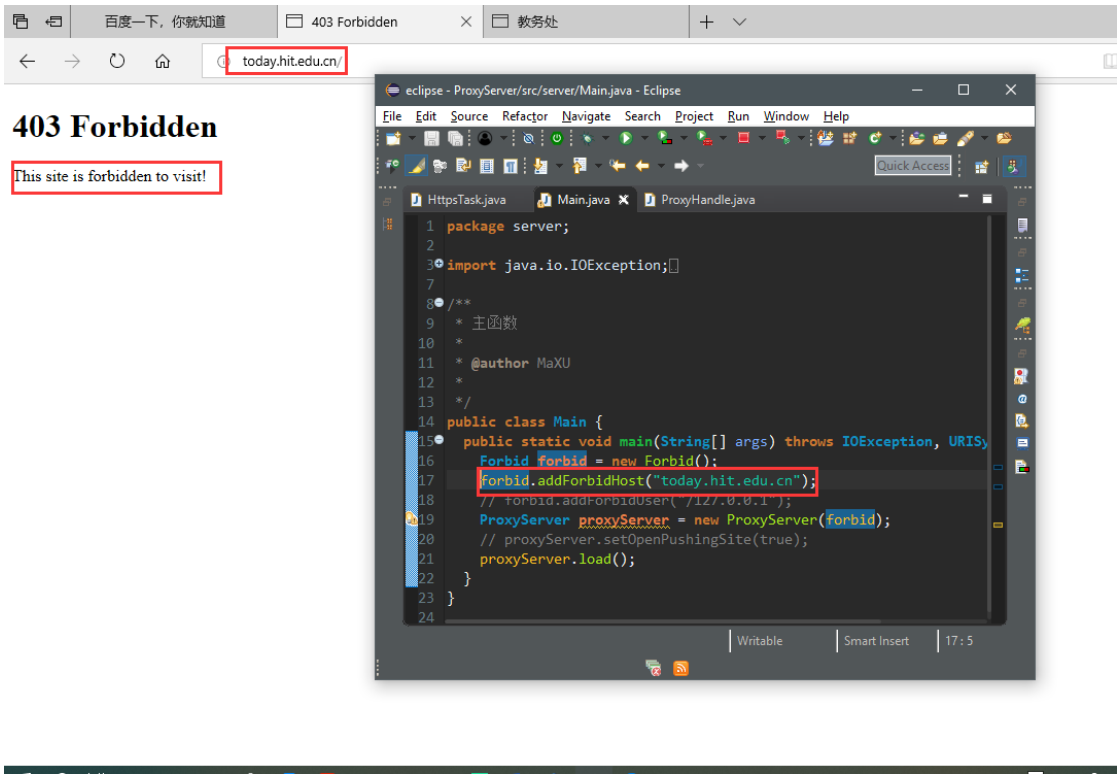



```
Successfully connected remote server: jwc.hit.edu.cn
Successfully connected remote server: jwc.hit.edu.cn
Successfully connected remote server: jwc.hit.edu.cn
Successfully connected remote server: jwc.hit.edu.cn
Successfully connected remote server: jwc.hit.edu.cn
Successfully connected remote server: jwc.hit.edu.cn
```

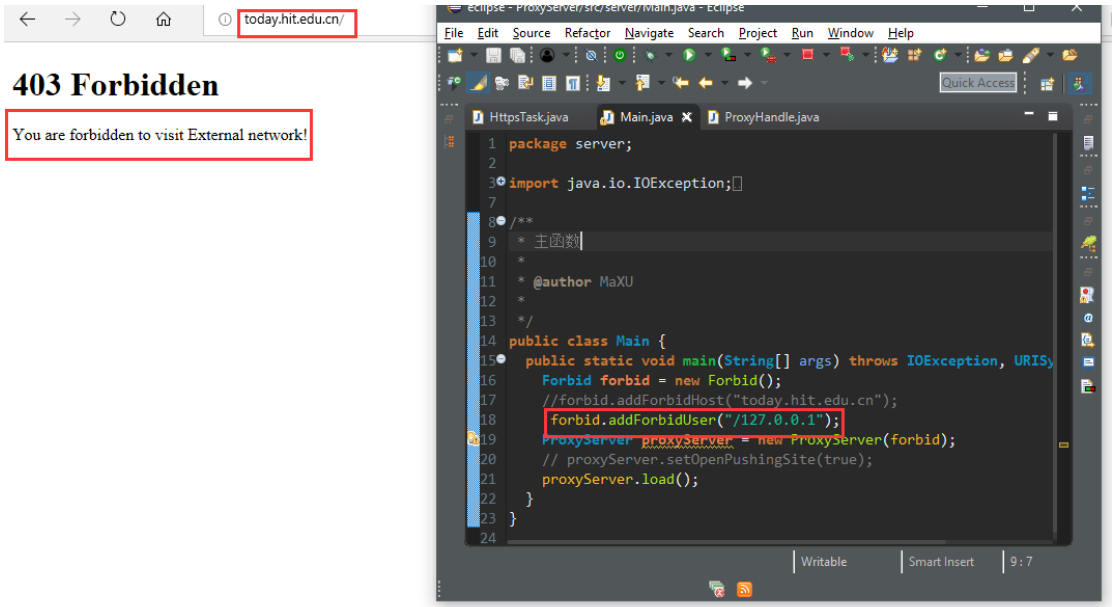
4.2 实现缓存



4.3 实现站点过滤

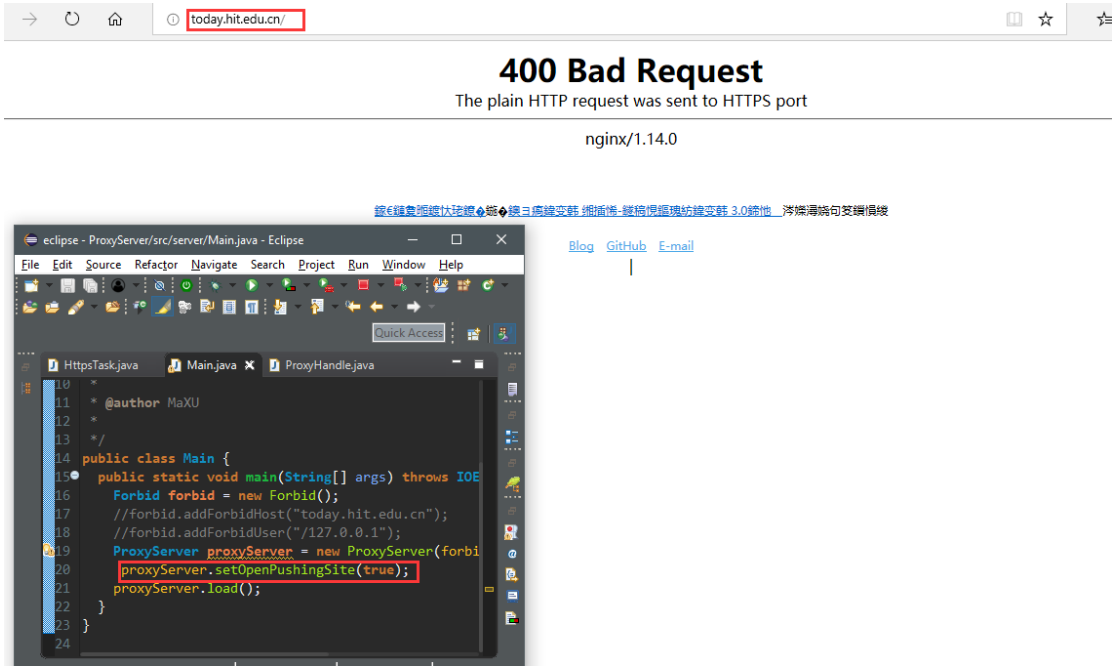


4.4 用户过滤



4.5 实现钓鱼网站

将一个网站转到室友的谷歌镜像站点：45.77.112.221




```
Set up phishing site successfully!
Successfully connected remote server: 45.77.112.221
Set up phishing site successfully!
Successfully connected remote server: 45.77.112.221
Set up phishing site successfully!
Successfully connected remote server: 45.77.112.221
Set up phishing site successfully!
Set up phishing site successfully!
Set up phishing site successfully!
Successfully connected remote server: 45.77.112.221
Set up phishing site successfully!
```

5.问题讨论

1. 在写代理时，开始使用 HTTPS 连接进行的测试，结果发现测试每次都会得到客户端发送的 CONNECT 头，而在处理时一直出错，最后通过上网查找资料发现 HTTP 和 HTTPS 进行代理时是有差别的，在进行 HTTPS 代理时需要客户端与代理服务器通过 CONNECT 头打通隧道，然后再进行正常的数据传输。而 HTTP 则没有这个过程。
2. 在从 Socket 中读取数据时需要使用 Java 中的字节流进行读取而不能通过字符流进行读取，否则可能会有一些莫名的错误，比如缓存图片时，就会报错。
3. 在实现缓存功能时最好的方法是将 HTTP 中的 URL 提取出来，后根据 URL 中的文件路径构建本地文件路径，这样可以分级缓存文件。

4. 直接将一个 HTTP 数据报转发给一个 HTTPS 服务器会得到一个 400 错误，这个在设置钓鱼站点时会遇到，没有解决这个问题。

6.心得体会

1. 通过实验知道了 HTTP 和 HTTPS 代理处理的具体流程。
2. 学会了 Java 中的 socket 编程
3. 对 HTTP 请求头和响应头有了更深入的了解。
4. 巩固了课本上学习到的知识。