上海大学 2021 ～ 2022 学年 冬 季学期

计算机安全与保密技术 课程成绩评

成绩	

课程名称： 计算机安全与保密技术 课程编号： **08A65002**

论文题目: 基于 socket 设计和实现一个 Http 代理服务器

学生姓名: 王波 学 号: 19122557 所在院系: 计算工程与科学学院

课程论文评阅记录:

课程论文评价要素	项目设计 35%	程序实现 35%	报告内容与格式 30%	报告值得肯定之处	报告存在主要问题
得分（等级或百分制）				[A]技术正确 [B]内容完成 [C]算法新颖 [D]报告清晰 [E]见解独到 [F]格式规范	[A] 技术错误 [B] 技术欠佳 [C] 未达要求 [D] 叙述欠佳 [E] 错字语病 [F] 格式欠佳
其他需要说明情况					

论文成绩: _____ （百分制） 任课教师: _____

（课程论文占期末总评成绩的 60%）

评阅日期: _____

注：后接课程论文，格式参照论文模板或公开发表论文的样式。

基于 Socket 设计和实现一个 Http 代理服务器

王　波

（上海大学 计算机工程与科学学院，上海 200072）

摘要：本文的主要内容是基于 socket 设计和实现一个 Http 代理服务器。在这个实验中，我主要实现了 Post 和 Get 的代理请求，在此之前首先判断请求中是否包含 CONNECT 来实现代理，让服务器代理用户去访问页面。随后在 http 页面进行测试，实验对于 Get 和 Post 的代理请求测试成功，对于 https 的请求也能通过代理进行。

关键词： Socket；Http；Proxy

Design and implement an Http proxy server based on socket

WANG Bo

(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

Abstract: The main content of this article is to design and implement an Http proxy server based on sockets. In this experiment, I mainly implemented post and get proxy requests, and before that, I first determined whether the request contained CONNECT to implement the proxy, and let the server proxy user visit the page. Then test on the http page, the experiment test for Get and Post proxy requests are successful, and https requests can also be performed through the proxy.

Key words: Socket; Http; Proxy

本项目主要通过 java 开发完成，基于 socket 和 Http 协议中 Connect 方法来实现了 Http 代理服务器，使用了 java 线程开发了 Get 和 Post 两种方法实现了对于 HTTP 协议的代理。

1　开发目的

本项目的开发目的在于基于 socket 设计和实现一个 Http 代理服务器。Web 代理（proxy）服务器是网络的中间实体。 代理位于 Web 客户端和 Web 服务器之间，扮演“中间人”的角色。HTTP 的代理服务器即是 Web 服务器又是 Web 客户端。

2　HTTP 协议

2.1　HTTP 协议介绍

协议是指计算机通信网络中两台计算机之间进行通信所必须共同遵守的规定或规则，超文本传输协议(HTTP)是一种通信协议，它允许将超文本标记语言(HTML)文档从 Web 服务器传送到客户端的浏览器。目前我们使用的是 HTTP/1.1 版本。

2.2　Web 服务器、浏览器、代理服务器的关系

当我们打开浏览器，在地址栏中输入 URL，然后我们就看到了网页。

实际上我们输入 URL 后，我们的浏览器给 Web 服务器发送了一个 Request, Web 服务器接到 Request

后进行处理，生成相应的 Response，然后发送给浏览器， 浏览器解析 Response 中的 HTML,这样我们就看到了网页，过程如下图所示：

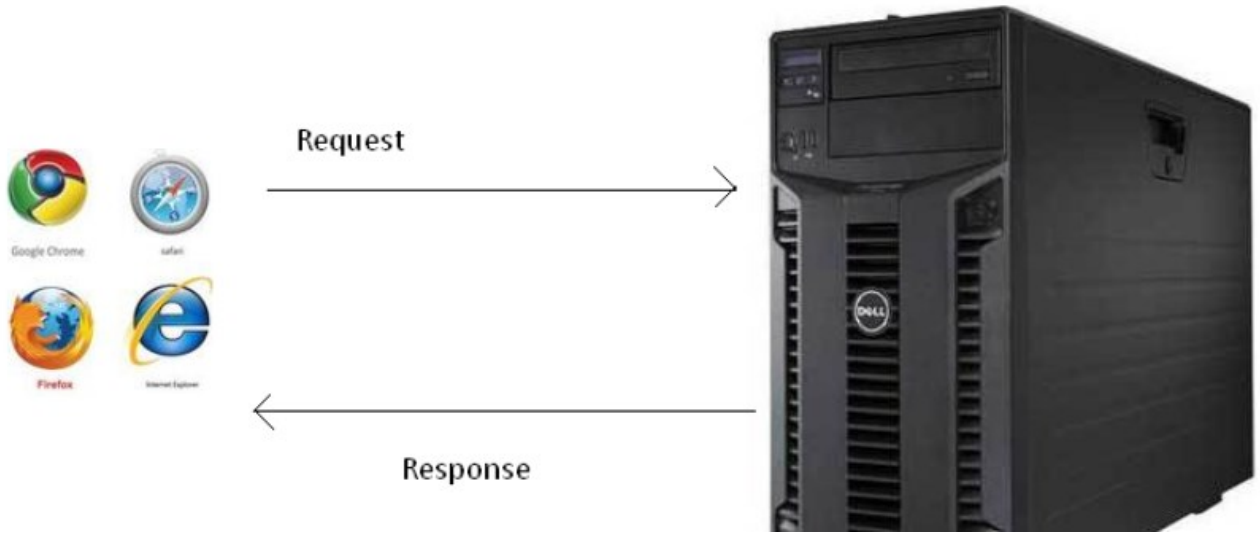

图 1　通过 Url 浏览页面的过程

Fig.1　The process of browsing a page through Url

我们的 Request 有可能是经过了代理服务器，最后才到达 Web 服务器的。过程如下图所示：

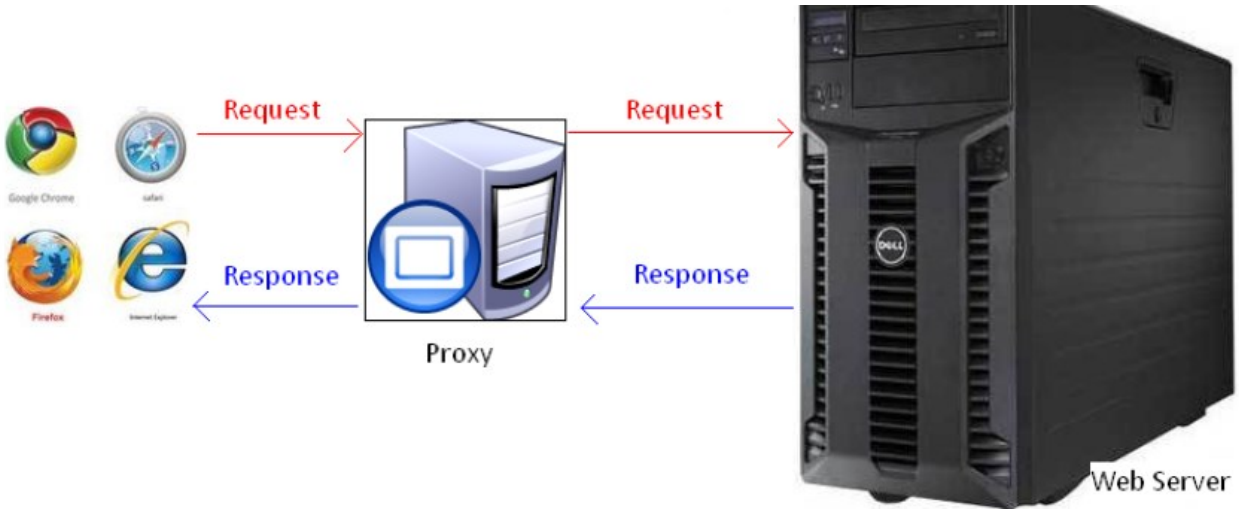

图 2　通过代理服务器来浏览页面的过程

Fig.2　The process of browsing a page through a proxy server

代理服务器就是网络信息的中转站，有什么功能呢？

1. 提高访问速度， 大多数的代理服务器都有缓存功能。
2. 突破限制， 也就是内网穿透了
3. 隐藏身份。

2.3　URL 介绍

URL(Uniform Resource Locator) 地址用于描述一个网络上的资源， 基本格式如下：

```
schema://host[:port#]/path/.../[?query-string][#anchor]
```

scheme	指定低层使用的协议(例如：http, https, ftp)
host	HTTP 服务器的 IP 地址或者域名
port	HTTP 服务器的默认端口是 80，这种情况下端口号可以省略。如果使用了别的端口，必须指明。
path	访问资源的路径
query-string	发送给 http 服务器的数据
anchor-	锚

2.4 Cokkie

http 协议是无状态的，同一个客户端的这次请求和上次请求是没有对应关系，对 http 服务器来说，它并不知道这两个请求来自同一个客户端。为了解决这个问题，Web 程序引入了 Cookie 机制来维护状态.

2.5 HTTP 消息的结构

先看 Request 消息的结构,Request 消息分为 3 部分,第一部分叫 Request line, 第二部分叫 Request header, 第三部分是 body. header 和 body 之间有个空行， 结构如下图：

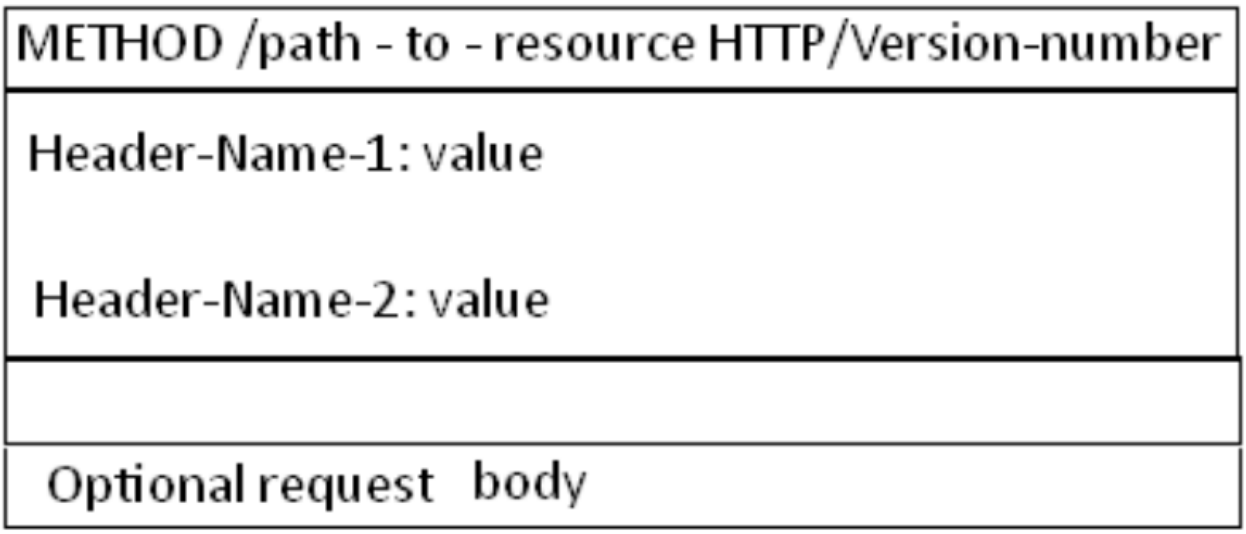

图 3 Http 消息的结构

Fig.3 The structure of the Http message

第一行中的 Method 表示请求方法,比如"POST","GET", Path-to-resoure 表示请求的资源, Http/version-number 表示 HTTP 协议的版本号。当使用的是"GET" 方法的时候， body 是为空的。我们再看 Response 消息的结构, 和 Request 消息的结构基本一样。同样也分为三部分,第一部分叫 Response line, 第二部分叫 Response header，第三部分是 body. header 和 body 之间也有个空行。HTTP/version-number 表示 HTTP 协议的版本号， status-code 和 message 表示的则是状态代码。

2.6 Get 和 Post 方法的区别

Http 协议定义了很多与服务器交互的方法，最基本的有 4 种，分别是 GET,POST,PUT,DELETE. 一个 URL 地址用于描述一个网络上的资源，而 HTTP 中的 GET, POST, PUT, DELETE 就对应着对这个资源的查，改，增，删 4 个操作。 我们最常见的就是 GET 和 POST 了。GET 一般用于获取/查询资源信息，而 POST 一般用于更新资源信息.

我们看看 GET 和 POST 的区别：

1. GET 提交的数据会放在 URL 之后，以?分割 URL 和传输数据，参数之间以&相连，如 EditPosts.aspx?name=test1&id=123456. POST 方法是把提交的数据放在 HTTP 包的 Body 中.

2. GET 提交的数据大小有限制(因为浏览器对 URL 的长度有限制)，而 POST 方法提交的数据没有限制.

3. GET 方式需要使用 Request.QueryString 来取得变量的值，而 POST 方式通过 Request.Form 来获取变量的值。

4. GET 方式提交数据，会带来安全问题，比如一个登录页面，通过 GET 方式提交数据时，用户名和密码将出现在 URL 上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

2.7 状态码

Response 消息中的第一行叫做状态行，由 HTTP 协议版本号， 状态码， 状态消息 三部分组成。

状态码用来告诉 HTTP 客户端,HTTP 服务器是否产生了预期的 Response.

HTTP/1.1 中定义了 5 类状态码， 状态码由三位数字组成，第一个数字定义了响应的类别

1XX 提示信息 - 表示请求已被成功接收，继续处理

2XX 成功 - 表示请求已被成功接收，理解，接受

3XX 重定向 - 要完成请求必须进行更进一步的处理

4XX 客户端错误 - 请求有语法错误或请求无法实现

5XX 服务器端错误 - 服务器未能实现合法的请求

3 Http 代理服务器

3.1 代理服务器

Web 代理（proxy）服务器是网络的中间实体。 代理位于 Web 客户端和 Web 服务器之间，扮演“中间人”的角色。HTTP 的代理服务器即是 Web 服务器又是 Web 客户端。

Fiddler 就是一个典型的代理服务器，Fiddler 是以代理 web 服务器的形式工作的,它使用代理地址:127.0.0.1, 端口:8888. 当 Fiddler 退出的时候它会自动注销代理，这样就不会影响别的程序。

3.2 代理服务器的作用

1．科学上网

很多人都喜欢用 Facebook， 看 YouTube。但是我们在天朝，天朝有 The Great of Wall(长城防火墙)，屏蔽了这些好网站。通过代理来跳墙，就可以访问了。自己去寻找代理服务器很麻烦， 一般都是用 FQ 软件来自动发现代理服务器的。

2. 匿名访问

经常听新闻，说”xxx”在网络上发布帖子，被跨省追缉了。假如他使用匿名的代理服务器，就不容易暴露自己的身份了。 http 代理服务器的匿名性是指： HTTP 代理服务器通过删除 HTTP 报文中的身份特性（比如客户端的 IP 地址， 或 cookie,或 URI 的会话 ID）， 从而对远端服务器隐藏原始用户的 IP 地址以及其他细节。 同时 HTTP 代理服务器上也不会记录原始用户访问记录的 log(否则也会被查到)。

“暗网”的洋葱路由就利用了这种类似的技术，把你的 Ip 地址通过多个路由器代理，从而达到匿名访问的目的。

3. 内网穿透

这是我最近在玩的一个小项目,我们都知道在校外如果需要访问学校内网就需要通过使用 VPN 来进行，但是学生所使用的 VPN 带宽太小，使得我们访问学校内网速度很慢，而且难以保持稳定。

这也是我做这个项目的出发点，为了更好的 WFH（Work From Home）我从寒假开始前就搭建了这个项目，首先买一台具有公网 Ip 的服务器（这里选用的是腾讯云服务器，每年只要七十多块钱），和学校内的一台电脑（这里选用的是树莓派 4B）在树莓派和腾讯云服务器上分别搭建 Frp 服务，并设置对应好的端口号映射关系，在树莓派上对应端口运行 HTTPPROXY（Http 代理）程序，实现了内网穿透，测试效果良好能很好的访问学校内网站和实现 SSH TELNET 服务。

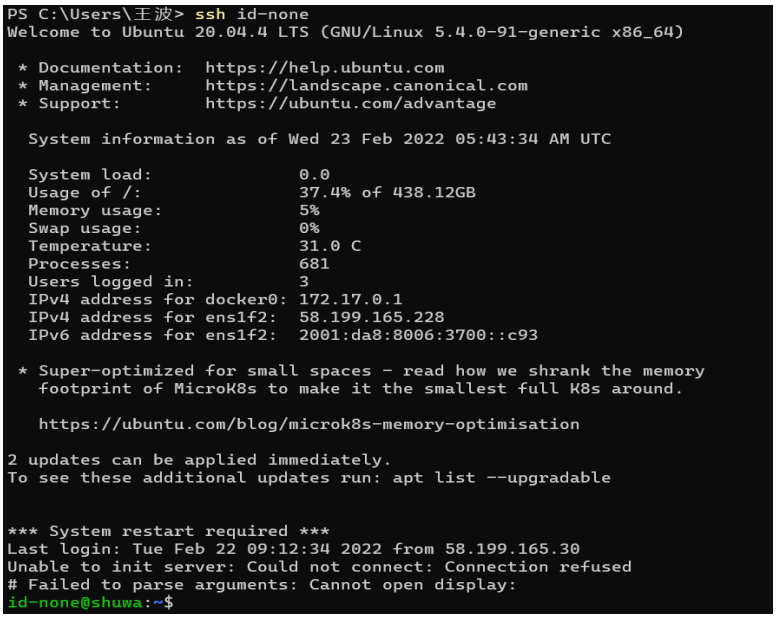

图 4　内网穿透

Fig.4　Intranet penetration

4. 通过代理缓存，加快上网速度

大部分代理服务器都具有缓存的功能，就好像一个大的 cache， 它有很大的存储空间，它不断将新取得数据存储到它本地的存储器上， 如果浏览器所请求的数据在它本机的存储器上已经存在而且是最新的，那么它就不重新从 Web 服务器取数据，而直接将存储器上的数据传给用户的浏览器，这样就能显著提高浏览速度。

4 程序实现和具体细节

在本次使用中，我使用了 java 网络编程来完成了项目内容。主要分为四个文件：HttpConnect.java、HttpExit.java、HttpProxy.java、IOUtils.java。

4.1 HttpConnect.java

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * 新连接处理线程
 */

public class HttpConnect extends Thread {

    private final Socket client;
    private final long createTime = System.currentTimeMillis();
    byte[] clientInputBuffer = new byte[1024 * 1024 * 4];
    private Socket server = null;
    private String host = null;
    private int port = 80;
    private int clientReadLength = 0;
    private DataInputStream clientInputStream = null; //客户端输入流
    private DataInputStream serverInputStream = null; //服务端输入流
    private DataOutputStream clientOutputStream = null; //客户端输出流
    private DataOutputStream serverOutputStream = null;  //服务端输出流
    private String clientInputString = null;
    private List<String> hosts = null;
    public HttpConnect(Socket client) {
        this.client = client;
```

```
    }

    @Override
    public void run() {
        try {
            clientInputStream = new DataInputStream(client.getInputStream());
            clientOutputStream = new DataOutputStream(client.getOutputStream());
            if (clientInputStream != null) {
                clientReadLength = clientInputStream.read(clientInputBuffer, 0,
clientInputBuffer.length); // 从客户端读数据
                if (clientReadLength > 0) { // 读到数据
                    clientInputString = new String(clientInputBuffer, 0,
clientReadLength);
                    // 去掉/n
                    if (clientInputString.contains("\n")) {
                        clientInputString = clientInputString.substring(0,
clientInputString.indexOf("\n"));
                    }
                    if (clientInputString.contains("CONNECT ")) {
                        parseServerHost("CONNECT ([^ ]+) HTTP/");
                    } else if (clientInputString.contains("http://") &&
clientInputString.contains("HTTP/")) {
                        // 从所读数据中取域名和端口号
                        parseServerHost("http://([^/]+)/");
                    }
                    if (host != null && !host.equals("127.0.0.1")) {
                        server = new Socket(host, port);
                        // 根据读到的域名和端口号建立套接字
                        serverInputStream = new
DataInputStream(server.getInputStream());
                        serverOutputStream = new
DataOutputStream(server.getOutputStream());
                        if (serverInputStream != null && server != null) {
                            if (clientInputString.contains("CONNECT ")) {
                                doConnect();
                                return;
                            }
                            doRequest();
                            return;
                        }
                    }
                }
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        IOUtils.close(serverInputStream, serverOutputStream, server,
clientInputStream, clientOutputStream, client);
    }

    /**
     * 解析主机地址
     *
     * @param regExp
     */
    private void parseServerHost(String regExp) {
        Pattern pattern = Pattern.compile(regExp);
        Matcher matcher = pattern.matcher(clientInputString + "/");
        if (matcher.find()) {
            host = matcher.group(1);
            // 判断端口号
            if (host.contains(":")) {
                port = Integer.parseInt(host.substring(host.indexOf(":") + 1));
                host = host.substring(0, host.indexOf(":"));
            }
        }
    }

    /**
     * 处理请求
     *
     * @throws IOException
     */
    private void doRequest() throws IOException, InterruptedException {
        serverOutputStream.write(clientInputBuffer, 0, clientReadLength);
        serverOutputStream.flush();
        final CountDownLatch latch;
        if (clientInputString.contains("POST ")) {
            latch = new CountDownLatch(2);
            new HttpExit(clientInputStream, serverOutputStream, latch).start();
        } else {
            latch = new CountDownLatch(2);
        }
        new HttpExit(serverInputStream, clientOutputStream, latch).start();
        latch.await(120, TimeUnit.SECONDS);
        IOUtils.close(serverInputStream, serverOutputStream, server,
clientInputStream, clientOutputStream, client);
        System.out.println("请求地址: " + clientInputString + ", 耗时: " +
```

```
(System.currentTimeMillis() - createTime) + "ms");
    }

    /**
     * 处理连接请求
     *
     * @return
     */
    private void doConnect() throws IOException, InterruptedException {
        String ack = "HTTP/1.0 200 Connection established\r\n";
        ack = ack + "Proxy-agent: proxy\r\n\r\n";
        clientOutputStream.write(ack.getBytes());
        clientOutputStream.flush();
        final CountDownLatch latch = new CountDownLatch(2);
        new HttpExit(serverInputStream, clientOutputStream, latch).start();
        new HttpExit(clientInputStream, serverOutputStream, latch).start();
        latch.await(120, TimeUnit.SECONDS);
        IOUtils.close(serverInputStream, serverOutputStream, server,
clientInputStream, clientOutputStream, client);
    }
}
```

在这个文件中，首先我们对数据流进行符合我预期的处理操作，之后判断是否包含“CONNECT”,如果包含就使用 TCP 链接，之后再通过方法 parseServerHost()方法来对 url 进行处理，读取 post 和 host 来建立 socket 链接，文中还建立了一个列表来反应我们所需要禁止代理的 Ip。之后建立链接后来对 POST 和 GET 进行分别处理，上述说到了 POST 和 Get 的区别。主要是使用 io 流的复制操作，来传递数据包。

4.2 IOUtils.java

```
import java.io.Closeable;
import java.io.IOException;


public class IOUtils {

    private IOUtils() {
    }

    public static void close(Closeable... closeables) {
        isNull(closeables);
    }

    public static void isNull(Closeable[] closeables) {
        if (closeables != null) {
```

```
            for (Closeable closeable : closeables) {
                if (closeable != null) {
                    try {
                        closeable.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
}
```

这也是文章中比较重要的部分，主要处理 io 流的操作和进行 socket close 的异常处理。

4.3 HttpExit.java

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.util.concurrent.CountDownLatch;

/**
 * 流通道处理线程
 */
public final class HttpExit extends Thread {
    private final CountDownLatch countDownLatch;
    private final DataInputStream in;
    private final DataOutputStream out;

    public HttpExit(DataInputStream in, DataOutputStream out, CountDownLatch
countDownLatch) {
        this.in = in;
        this.out = out;
        this.countDownLatch = countDownLatch;

    }

    @Override
    public void run() {
        int len;
        byte[] buf = new byte[10240];
        try {
            while ((len = in.read(buf, 0, buf.length)) != -1) {
                out.write(buf, 0, len);
                out.flush();
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            IOUtils.close(in, out);
            countDownLatch.countDown();
        }
    }
}
```

我们都知道 Http 每次需要发很多内容，如果就使用一个进程的话，那么他的速度就会很慢很慢，影响效率和使用体验，为此我们为该服务建立了线程，通过多线程并行操作，实现了对 Http 的高速代理，为此每个线程的开启和关闭也成为了一个问题，所以我建立了该文件来处理这个问题，使用了上面的 IOUtils 来处理线程 socket 异常关闭情况，防止代理因为异常而关闭。

4.4 HttpProxy.java

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;


public class HttpProxy extends Thread {
    private final ServerSocket server;

    public HttpProxy(int port) throws IOException {
        server = new ServerSocket(port);
        System.out.println("代理端口: " + port);
    }

    public static void main(String[] args) throws IOException {
        int port = 11111;
        if (args != null && args.length > 0 && args[0].matches("\\d+")) {
            port = Integer.parseInt(args[0]);
        }
        new HttpProxy(port).start();
    }

    @Override
    public void run() {
        while (true) {
            try {
                Socket client = server.accept();
                System.out.println(client);
                //使用线程处理收到的请求
                new HttpConnect(client).start();
            } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
    }
}
```

这是文件的主方法，在这里可以调用直接所写的类来完成 Http 代理操作，而且可以在编译后的.calss 文件在命令行中指定代理的端口号。

5 实验截图

首先，在系统代理内设置代理，然后开启服务，访问页面。

图 5　代理设置

Fig.5　Proxy settings

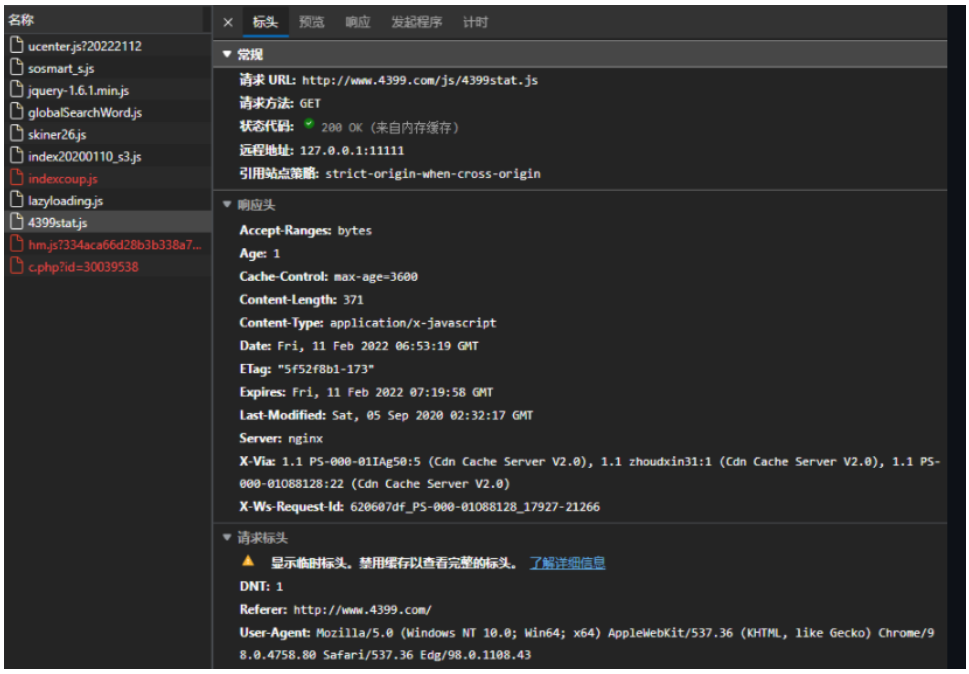

图 6　访问页面

Fig.6　Visit the page

图 7　过滤 Ip

Fig.7　Filter IP

6　总结

通过本次实验，我了解了 HTTP 协议的工作方式和相关网络编程知识，更加实际的了解到了 HTTP 的工作过程。通过本次代理实验，不仅让我知道了知识，我还从中发现了乐趣，这次实验比之前的实验都好玩，没有什么按部就班的操作，没有什么指导和想法，完全通过自己的摸索，了解，来对充实自己的知识储备和能力，此外通过这次实验，我更多的了解到了自己的不足和缺点，也提出了以后的项目改进的地方。

1. 我虽然使用了 java 线程开发，但是我 http 代理的速度依然很慢，无法共多人使用，高并发就是一个考虑的点。
2. 对于 http 包我选择了直接复制转发的方式，并没有对包进行过多的解析，有些问题还是不太了解。
3. 之前使用 frp 做内网穿透的时候，http 代理，速度较快，但是不稳定，有时候会产生中断现象。

这些问题，也许就是我之后改进的方向。

致谢： 感谢戴佳筑老师对本工作的大力支持，在此表示感谢！其次也感谢一起做 Http 代理的同学们，虽然我是第一个做完这个项目的，但是我的功能还不完善，和他们讨论了很多问题，最终才有了这个项目的结果。

参考文献：

[1] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, H. Frystyk, Larry Masinter, Paul J. Leach, & Tim Berners-Lee (1997). Hypertext Transfer Protocol -- HTTP/1.1 ACM Conference on Hypertext.

[2] Keira Hopkins (2019). HTTP proxy servlet is deployed to a servlet container on a machine with its own queue manager

[3] https://github.com/arloor/HttpProxy

[4] https://blog.csdn.net/Michaelwubo/article/details/81985038