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	董琦		院系	数据科学与大数据	
班级	2003501		学号	120L020701	
任课教师	刘亚维		指导教师	刘亚维	
实验地点			实验时间	2022/10/5	
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

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：

概述本次实验的主要内容，包含的实验项等。

(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；

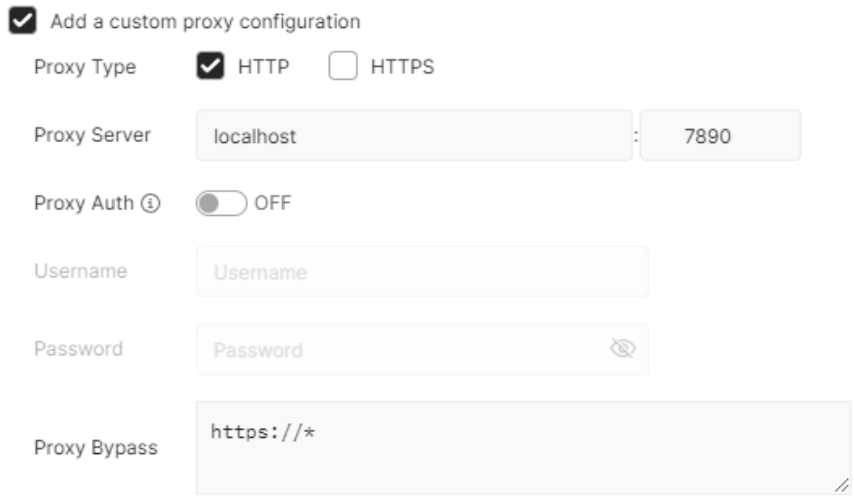
c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。

实验过程：

1.相关操作和简述

（1）设置代理

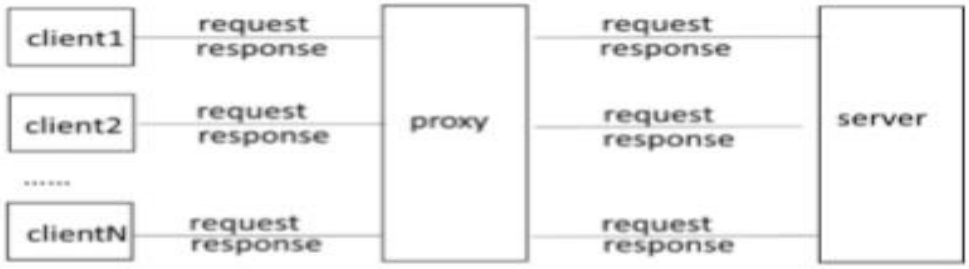
由于没找到浏览器单独设置代理，于是只能在postman上设置一个代理，这里选择的端口是7890端口，并且排除https协议请求，因为我们的代理程序无法处理https协议。



(2) 设计并实现一个基本 HTTP 代理服务器

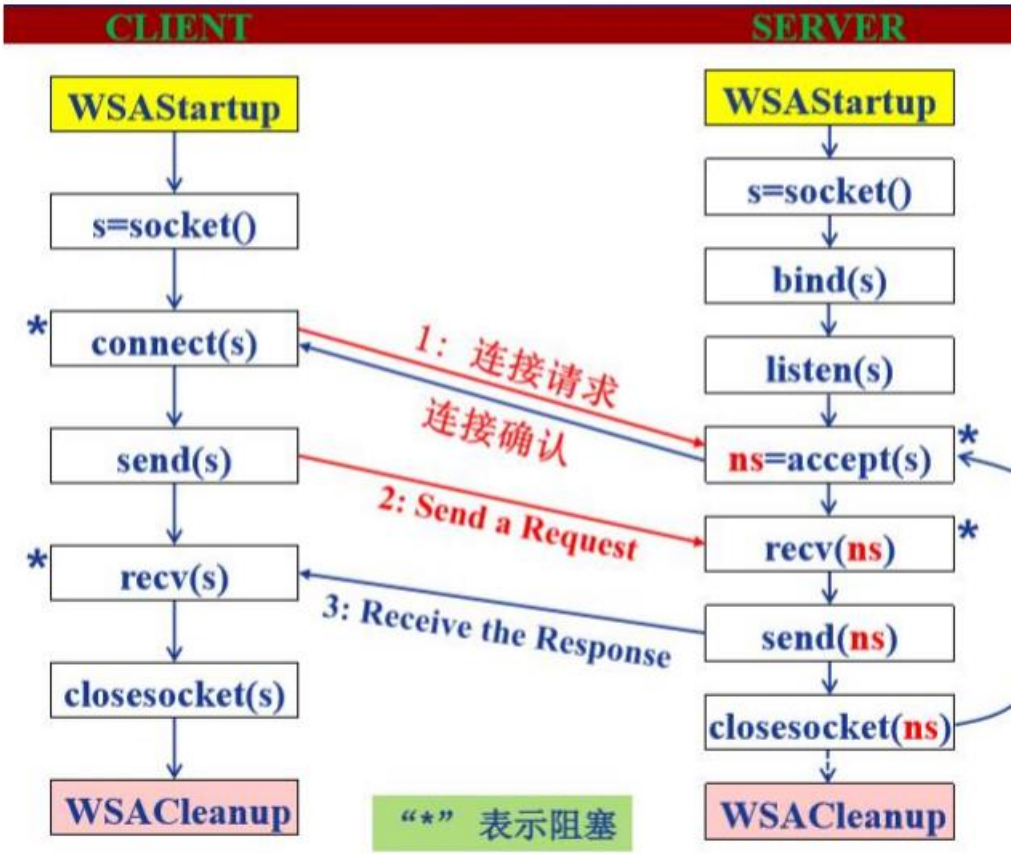
(3) 实现一个多用户代理服务器，即实现为一个多线程的并发服务器。首先代理服务器创建 HTTP 代理服务的 TCP 主套接字，通过该主套接字不断监听 等待客户端的连接请求。当客户端连接之后，创建一个子线程，由子线程执行上述一对一的代理过程，服务结束之后子线程终止。与此同时，主线程不断接受下一个客户的代理服务。

(4) HTTP 代理服务器的基本原理：代理服务器从功能上说就是一个和客户端建立连接的服务器以及和客户请求的服务器建立连接的客户端。对于客户端来说，它的功能是接收来自客户的 HTTP 请求，并对该报文进行相应的处理（解析头部，头部信息修改），并将其转发给相应的服务器端；同时接收来自服务器端的响应报文，并对其进行相应的处理（解析头部，缓存响应），并将其转发给客户端。



使用代理的B/S

5）流程框图如下所示：



2.本次实验过程

1） 新建一个ServerSocket 对象，监听端口server_port为7890，表示该socket 用于不断监听端口 7890 的 HTTP 请求；

```
proxyServer proxyServer=new proxyServer( port: 7890);
```

2） 对代理服务器对象进行设置，配置网页跳转、禁止域名、禁止IP等信息。

```
//设置钓鱼
Map<String,String> redirect=new HashMap<>();
redirect.put("www.hit.edu.cn","jwts.hit.edu.cn");
//设置禁止访问
Set<String> forbiddenHost=new HashSet<>();
forbiddenHost.add("www.baidu.com");
Set<String> forbiddenIP=new HashSet<>();
//forbiddenIP.add("127.0.0.1");

proxyServer.setRedirect(redirect);
proxyServer.setForbiddenHost(forbiddenHost);
proxyServer.setForbiddenIP(forbiddenIP);
```

这里我们使用了一个实体类proxyConfig：

```
@Data
@NoArgsConstructor
public class proxyConfig {

    Map<String,String> redirect;

    Set<String> forbiddenIP;

    Set<String> forbiddenHost;
}
```

3） 启动函数 proxyServer.start()，设置 while 循环条件为 true，让 server_socket 不断监听来自客户端的 HTTP 请求，并为接收的每一个请求新建一个 requestHandler子线程用于处理该请求的连接；

```
while(true){
    Socket in=proxySocket.accept();
    Thread t=new Thread(new requestHandler(in,this.proxyConfig));
    threadPoll.execute(t);
}
```

4） 代理服务器设置 client端口 用于与客户端建立连接，首先通过流处理获取客户端 HTTP 请求，对请求头各行调用parser()按行进行解析处理：用以获取请求目标url、hostname、port等信息，用一个ReqInfo实体类储存。

```java
@Data
public class ReqInfo {
    private String method;
    private String url;
    private String version;
    private String hostname;
    1 个用法
    private int hostPort;
    2 个用法
    private Map<String,String> headers;
    private String clientIP;
    private int clientPort;
    1 个用法  DQ
    public ReqInfo(){
        this.headers=new HashMap<>();
        this.hostPort=80;
    }
    1 个用法  DQ
    public void putHeaders(String key,String value) { this.headers.put(key,value); }
}
```

具体的请求处理函数如下：

```java
private void parser(String line){

    if(line.startsWith("GET")||line.startsWith("POST")){
        String[] requestLine=line.split( regex: "\\s+");
        if(requestLine.length==3){
            this.reqInfo.setMethod(requestLine[0]);
            this.reqInfo.setUrl(requestLine[1]);
            this.reqInfo.setVersion(requestLine[2]);
        }
    }
    else if(line.startsWith("Host")){
        String[] requestHeader=line.split( regex: ":");
        this.reqInfo.setHostname(requestHeader[1].trim());
        if(requestHeader.length==3){
            this.reqInfo.setHostPort(Integer.parseInt(requestHeader[2]));
        }
    }
    else {
        String[] requestHeader=line.split( regex: ":");
        this.reqInfo.putHeaders(requestHeader[0],requestHeader[1].trim());
    }
}
```

5） 一个正常的首次请求，代理服务器将请求转发给目的主机：需要建立一个与目的主机通信的套接字，并将请求报文写入到这个套接字的输出流，即可将请求发送到目的主机；

```java
this.client=new Socket();
this.client.connect(new InetSocketAddress(reqInfo.getHostname(), reqInfo.getHostPort()));
```

6） 接着需要代理服务器作为客户端接收来自服务器的响应报文（并将响应报文缓存到对应的文件中），即将与服务器通信的套接字client的输入流 中取到的信息写到与客户端通信的套接字request 的输出流中，即完成代理服务器的转发响应报文的功能。

3. 实现代理服务器缓存功能。

(1) 实现方式

i)把一个 URL 对应的响应报文保存在本地缓存中，文件的路径为"src/cache/" + this.host + "/" + this.URL.hashCode()＋".cache"，其中每个域名为一个文件夹，下面存放的是所有该主机的响应报文缓存内容，文件名为 URL 调用 hashCode 得到的数字+“.cache”；

```
String cacheName="src\\cache\\"+reqInfo.getHostname()+"\\"+reqInfo.getUrl().hashCode()+".cache";
File cacheFile=new File(cacheName);
```

ii) 在构造新的请求报文时，将 if-modified-since Date 语句加在请求报文的倒数第二句（倒数第一句为\r\n即回车换行）。

```
DateFormat df = new SimpleDateFormat( pattern: "EEE, d MMM yyyy HH:mm:ss z", Locale.ENGLISH);
df.setTimeZone(TimeZone.getTimeZone( ID: "GMT"));
msg.append("If-Modified-Since: ").append(df.format(cacheFile.lastModified())).append("\r\n");
msg.append("\r\n");
```

(2) 缓存功能基本逻辑：当首次访问某一域名时，我们先新建对应域名的文件夹，然后来看要访问的URL。如果该URL未缓存，那么直接转发原请求头；若已缓存，则向请求头中插入 if-modified-since Date 语句，若接收到的响应状态码为304，则读取缓存发给客户端，若否则获取最新的内容。

```
if(String.valueOf(response).contains("304 Not Modified")){
    System.out.println("缓存命中");

    while((result=fileReader.read())!=-1)
    {
        serverOut.write(result);
    }
}
else{
    System.out.println("缓存未命中");
    {
        serverOut.write(String.valueOf(response).getBytes());
        while((result = clientIn.read()) != -1){
            serverOut.write(result);
        }
    }
}
```

4. 其他额外功能

其余的三个功能比较简单，我们在配置代理服务器后，代理服务器在处理请求时会把配置内容传递给子线程。

```
Thread t=new Thread(new requestHandler(in,this.proxyConfig));
```

在我们解析完请求头的信息后，便根据对应内容来决定是否允许请求和是否跳转到钓鱼网站，如果不允许，则不做任何行为。

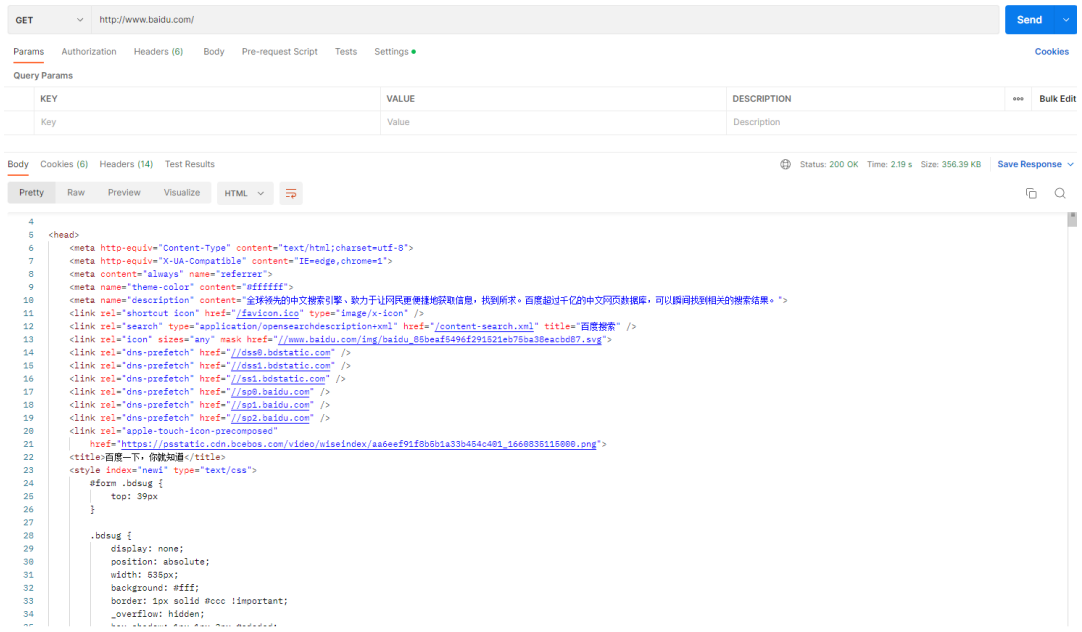
具体逻辑比较简单，下面是方法实现：

```
private void processReq(){
    if(this.proxyConfig.redirect!=null&&this.proxyConfig.redirect.keySet().contains(reqInfo.getHostname()))
    {
        String oldHost=this.reqInfo.getHostname();
        this.msg=new StringBuilder(this.msg.toString().replace(oldHost,this.proxyConfig.redirect.get(oldHost)));
        this.reqInfo.setHostname(this.proxyConfig.getRedirect().get(oldHost));
        this.reqInfo.setUrl(this.reqInfo.getUrl().replace(oldHost,this.proxyConfig.redirect.get(oldHost)));
    }
    if(this.proxyConfig.forbiddenHost.contains(this.reqInfo.getHostname()))
    {
        System.out.println("非法域名，禁止访问");
        legal=false;
    }
    if(this.proxyConfig.forbiddenIP.contains(this.reqInfo.getClientIP()))
    {
        System.out.println("非法IP，禁止访问");
        legal=false;
    }
}
```

实验结果：（这里我使用了 postman 软件来访问，因为有时候浏览器会自动变成 Https 协议

1. 简单代理和缓存的实现：

访问www.baidu.com



可以看到是加载出了百度的首页。

同时也缓存了对应的cache文件。

下面是缓存部分内容：

```
HTTP/1.1 200 OK
Bdpagetype: 1
Bdqid: 0xdc902dde00012c4f
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Date: Fri, 14 Oct 2022 16:48:47 GMT
Server: BWS/1.1
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=1; path=/
Set-Cookie: H_PS_PSSID=36553_37490_37300_34813_37404_36789_37539_37499_37582_26350_37365; path=/; domain=.baidu.com
Traceid: 166576612703495616101589325316496546895
Vary: Accept-Encoding
X-Frame-Options: sameorigin
X-Ua-Compatible: IE=Edge,chrome=1
Transfer-Encoding: chunked

ded
<!DOCTYPE html><!--STATUS OK--><html><head><meta http-equiv="Content-Type" content="text/html;charset=utf-8"><meta http-equiv="X-UA-Compatible" content="I
8000
-color:#f0f0f0}.bdsug .bdsug-pcDirect-s .bdsug-pc-direct-tip,.bdsug .bdsug-pcDirect-is-s .bdsug-pc-direct-tip-is{background-position:0 -15px}.bdsug .bdsug
html{color:#000;overflow-y:scroll;overflow:-moz-scrollbars}
body,button,input,select,textarea{font-size:12px;font-family:Arial,sans-serif}
h1,h2,h3,h4,h5,h6{font-size:100%}
em{font-style:normal}
small{font-size:12px}
ol,ul{list-style:none}
a{text-decoration:none}
a:hover{text-decoration:underline}
legend{color:#000}
```

2. 缓存的使用，检测是否更新：

   连续俩次访问同一网址，控制台输出如下：

```
Exception in thread "pool-1-thread-8" java.lang.NullPointerException Create breakpoint
    at com.dq.web.lab1.requestHandler.run(proxyServer.java:86) <4 个内部行>
创建文件夹成功:src\cache\www.qqkk.com
http://www.qqkk.com/未缓存
任务结束
Exception in thread "pool-1-thread-9" java.lang.NullPointerException Create breakpoint
    at com.dq.web.lab1.requestHandler.run(proxyServer.java:86) <4 个内部行>
http://www.qqkk.com/已缓存
缓存命中
任务结束
```

   可以看到缓存被使用了。

3. 禁止访问URL

   比如我们将www.baidu.com设为禁止访问网站：

```
Set<String> forbiddenHost=new HashSet<>();
forbiddenHost.add("www.baidu.com");
```

   然后使用postman发送请求，会发现是无法接收到响应的：

Could not get response

Error: socket hang up | View in Console

Learn more about troubleshooting API requests

4. 禁止IP访问：

比如我们设置本机地址为禁用IP：

```
Set<String> forbiddenIP=new HashSet<>();
forbiddenIP.add("127.0.0.1");
```

那么再访问时,则看到我们服务器终端拦截了请求：

```
代理服务器启动,监听7890端口。
非法IP，禁止访问
任务结束
非法IP，禁止访问
任务结束
非法IP，禁止访问
任务结束
非法IP，禁止访问
任务结束
非法IP，禁止访问
任务结束
非法IP，禁止访问
任务结束
非法IP，禁止访问
任务结束
```

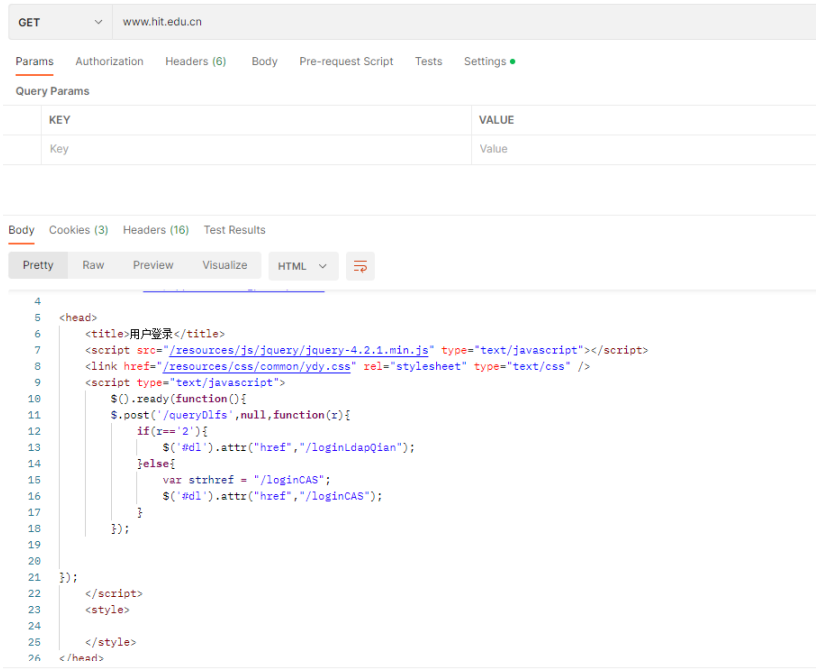
5. 钓鱼网站：

我们设置如下：

```
Map<String,String> redirect=new HashMap<>();
redirect.put("www.hit.edu.cn","jwts.hit.edu.cn");
```

即将www.hit.edu.cn跳转到jwts.hit.edu.cn

那么我们使用postman向www.hit.edu.cn发送请求，会发现显示内容为jwts.hit.edu.cn：



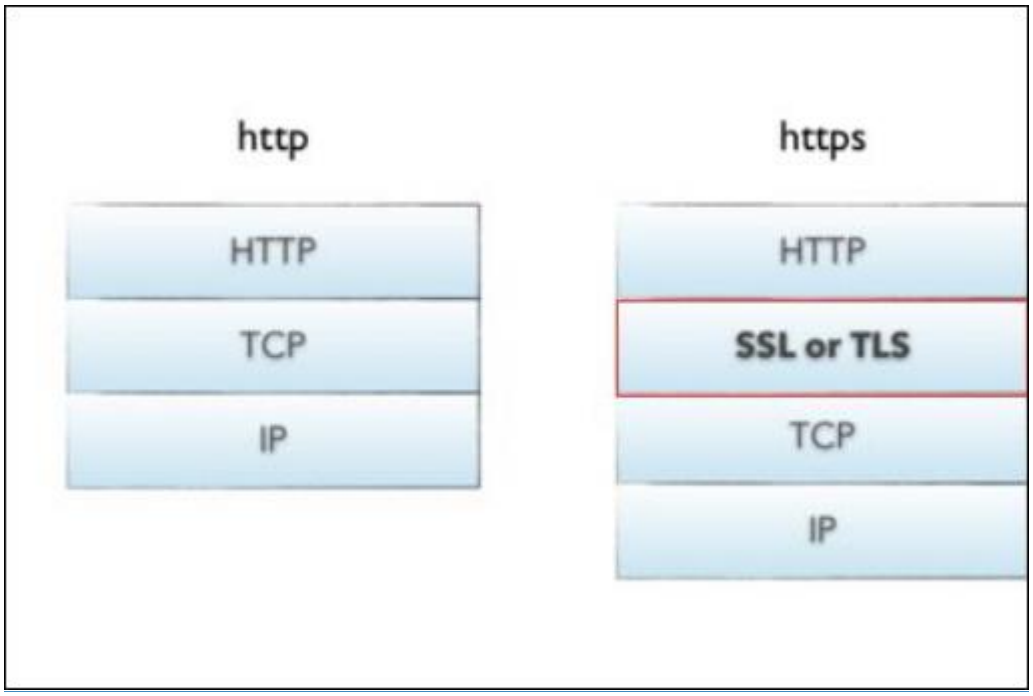
问题讨论：

1. 为什么不能访问https的网站，只能访问http的网站？

http协议和https协议的区别：传输信息安全性不同、连接方式不同、端口不同、证书申请方式不同

a)传输信息安全性不同

- http协议：是超文本传输协议，信息是明文传输。如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息。
- https协议：是具有安全性的ssl加密传输协议，为浏览器和服务器之间的通信加密，确保数据传输的安全。



b)连接方式不同

- http协议：http的连接很简单，是无状态的。
- https协议：是由SSL＋HTTP协议构建的可进行加密传输、身份认证的网络协议。

c)端口不同

- http协议：80。
- https协议：443.

d)证书申请方式不同

- http协议：免费申请。
- https协议：需要到ca申请证书，一般免费证书很少，需要交费。

心得体会：

1. 深入了解明白了网络应用的Socket API(TCP)调用基本流程,
2. 详细掌握了HTTP请求报文，响应报文的格式以及具体细节
3. 掌握了客户端和服务器端通过套接字交互的流程。
4. 深入了解代理服务器的作用，代理服务器可以实现基本服务器的功能，也可以屏蔽，可以访问一些正常服务器无法访问的网站，还可以进行缓存，“钓鱼”，其次代理服务器实际上既是客户端又是服务器端。