**《计算机网络》实验报告1**

HTTP代理服务器的设计与实现

**童超宇**

院 （系）：计算机科学与技术学院 专 业：计算机科学与技术

学　　号 ：1152130106 指导教师：刘亚维

**2018年4月**

# 目录

[1 实验目的 3](#_Toc427970343)

[2 实验内容 3](#_Toc1013854166)

[3 实验原理 3](#_Toc2123746975)

[3.1 Socket 编程的客户端和服务器端主要步骤 3](#_Toc1431707869)

[3.2 HTTP代理服务器的基本原理 4](#_Toc1931154328)

[3.3 HTTP代理服务器的程序流程图 4](#_Toc1043857298)

[3.4 HTTP代理服务器的关键技术及解决方案 5](#_Toc159176030)

[4 实验过程与结果 5](#_Toc589116548)

[4.1 屏蔽url 5](#_Toc709108957)

[4.2 钓鱼 6](#_Toc88062360)

[4.3 用户屏蔽 6](#_Toc1928832411)

[4.4 缓存 6](#_Toc262793600)

[5 源代码（有详细注释） 6](#_Toc2039784056)

[6 实验心得 9](#_Toc458287231)

# 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

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。

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。

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）。

# 实验原理

## Socket 编程的客户端和服务器端主要步骤

**服务器端**

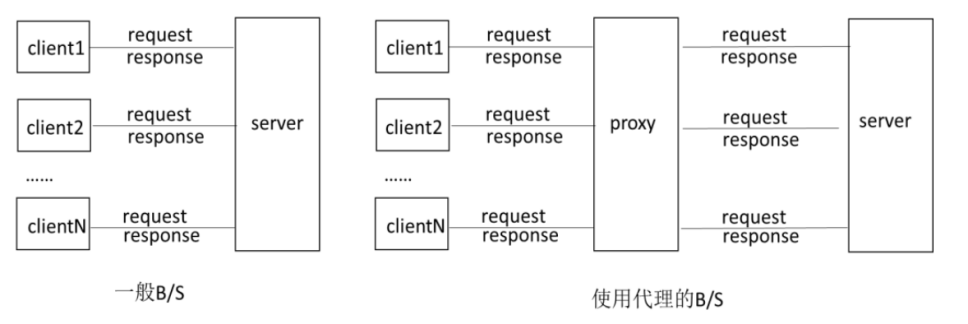
1. 使用socket函数创建一个socket描述符，来唯一标识一个socket。函数原型int socket(int domain, int type, int protocol)
2. 使用bind函数绑定IP地址，端口信息等。函数原型int bind(int sockfd, const struct sockaddr \* addr, socklen\_t addrlen)
3. 使用listen函数进行监听创建的socket。函数原型int listen(int sockfd, int backlog)
4. 使用accept函数接收请求，此时socket连接也就建立好了。函数原型int accept(int sockfd, struct sockaddr \* addr, socklen\_t \* addrlen)
5. 使用read(),write()等函数调用网络I/O进行读写操作，来实现网络中不同进程之间的通信
6. 使用close函数关闭网络连接，即关系相应的socket描述字。函数原型int close(int fd)

**客户端**

1. 创建socket
2. 绑定IP地址，端口信息
3. 设置要连接对方的IP地址和端口属性
4. 使用connect函数连接服务器。函数原型为int connect(int sockfd, const struct sockaddr \* addr, socklen\_t addrlen)
5. 使用read(),write()等函数进行网络I/O的读写
6. 关闭网络连接

## HTTP代理服务器的基本原理

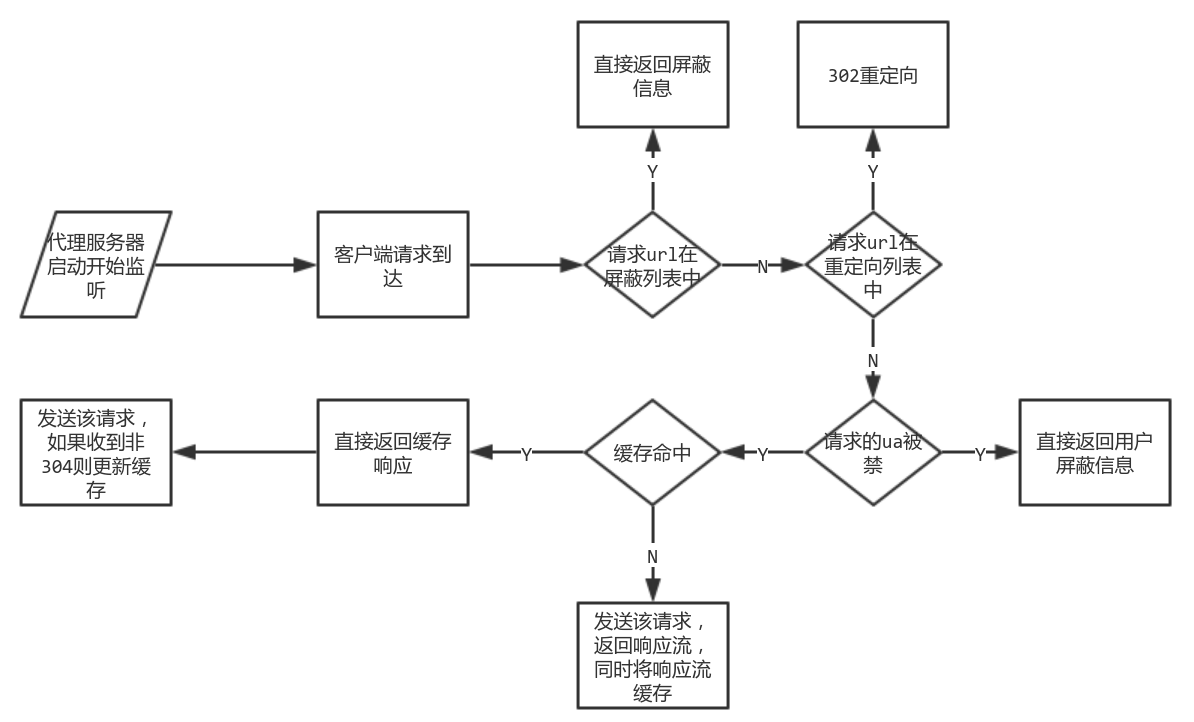
代理服务器，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接



首先，代理服务器创建 HTTP 代理服务的 TCP 主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，读取客户端的 HTTP请求报文，通过请求行中的 URL，解析客户期望访问的原服务器 IP 地址；创建访问原（目标）服务器的 TCP 套接字，将 HTTP 请求报文转发给目标服务器，接收目标服务器的响应报文，当收到响应报文之后，将响应报文转发给客户端，最后关闭套接字，等待下一次连接。

代理服务器还可以加入缓存机制，当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行清理

## HTTP代理服务器的程序流程图



## HTTP代理服务器的关键技术及解决方案

**HTTP**

发送HTTP的请求，接受HTTP响应需要遵循相应的标准格式

**url屏蔽，钓鱼，用户屏蔽**

接受到的请求中包含url，用户信息（cookie，user-agent等），通过这些信息可以做过滤处理，钓鱼通过302重定向实现

**缓存**

缓存可以放在硬盘上，也可以放在内存里（如redis），本程序放在内存中

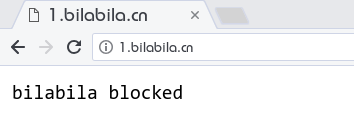
缓存映射中，请求的url+method+headers为键。在nodejs的http.request中，响应是流的形式，将响应流pipe给客户后，该流就结束了，故需将流转成buffer保存，缓存命中时再将保存的buffer转成流。响应流中只包含响应的body，故还需保存头部信息。缓存替换机制是最少命中，故缓存映射的值中还需添加命中次数。综上，映射值由响应流的buffer形式，响应信息（headers，状态码），命中次数三部分构成。

缓存命中时先返回缓存的响应，再发请求询问是否更新，若状态码不是304则更新缓存

# 实验过程与结果

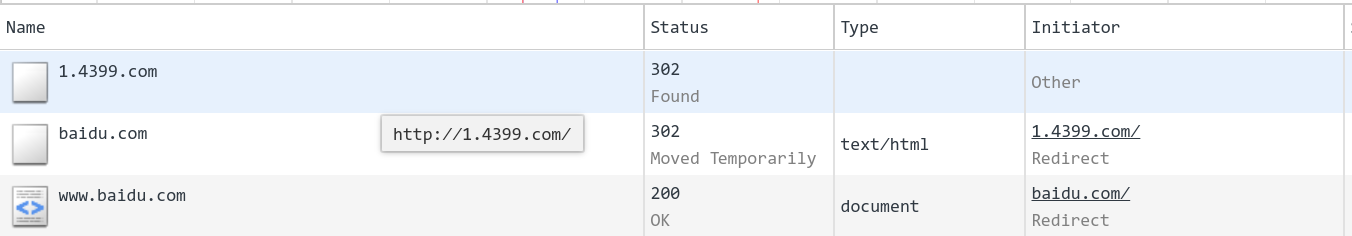
## 屏蔽url

访问1.bilabila.cn 返回对应屏蔽信息



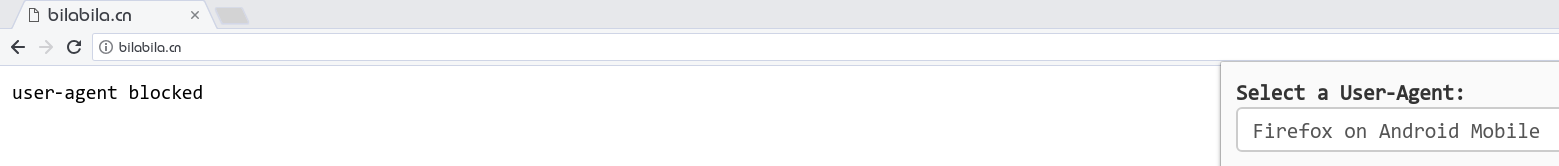
## 钓鱼

访问1.4399.com重定向到https://baidu.com



## 用户屏蔽

user-agent不包含chrome，提示user-agent blocked



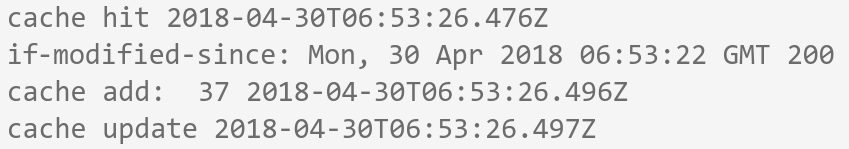
## 缓存

命中后返回缓存响应流，同时发送头部带if-modified-since字段的请求

若响应状态码304，什么都不做



若响应状态码非304，则更新缓存



# 源代码（有详细注释）

1. **const** http = require('http')
2. **const** url = require('url')
3. **const** net = require('net')
4. **const** pass = require('stream').PassThrough
5. **const** fs = require('fs')
6. **const** Duplex = require('stream').Duplex
7. // no external dependency
9. **const** hostname = '127.0.0.1'
10. **const** port = '8888'
11. // block url, block info
12. **const** blockList = **new** Map([
13. // ['www.4399.com', 'blocked'],
14. ['1.bilabila.cn', 'bilabila blocked']
15. ])
16. // redirect url
17. **const** redirectList = **new** Map([
18. ['1.4399.com', 'https://baidu.com'],
19. // ['www.gamersky.com', 'http://bilabila.cn'],
20. ])
21. // user agent allowed
22. **const** uaAllow = ua => ua && ua.toLowerCase().includes('chrome')
23. // cache response
24. **const** cachePool = (size = 20) => {
25. **const** cache = **new** Map()
26. // key=url+method+headers
27. **const** key = req => JSON.stringify({ ...url.parse(req.url),method:req.method,headers:req.headers })
28. **return** {
29. set(req, res) {
30. // value=[buffer,hit,response]
31. **const** v = [undefined, 1, res]
32. **const** k = key(req)
33. cache.set(k, v)
34. // stream to buffer
35. **const** data = []
36. res.on('data', d => { data.push(d) })
37. res.on('end', () => { v[0] = Buffer.concat(data) })
38. console.log('cache add: ', cache.size, (**new** Date).toISOString())
39. // remove the least hit
40. ; (async () => {
41. **if** (cache.size > size) {
42. let min = size + 5, index
43. **for** (let e of cache) {
44. **if** (e[1][1] < min) {
45. min = e[1][1]
46. index = e[0]
47. }
48. }
49. cache.**delete**(index)
50. }
51. })()
52. },
53. get(req) {
54. **const** k = key(req)
55. **const** v = cache.get(k)
56. // if buffer defined
57. **if** (v && v[0]) {
58. console.log('cache hit', (**new** Date).toISOString())
59. // hit +1
60. v[1]++
61. // buffer to stream
62. **const** stream = **new** Duplex()
63. stream.push(v[0])
64. stream.push(**null**)
65. // stream, cache response
66. **return** [stream, v[2]]
67. }
68. **return** []
69. }
70. }
71. }
72. // cache 500 response
73. **const** cache = cachePool(500)
74. //request mode for http
75. **const** request = (req, res) => {
76. **const** ua = req.headers['user-agent']
77. , u = url.parse(req.url)
78. , block = blockList.get(u.hostname)
79. , redirect = redirectList.get(u.hostname)
80. , allow = uaAllow(ua)
81. , [cacheStream, cacheRes] = cache.get(req)
82. , opt = {
83. hostname: u.hostname,
84. port: u.port,
85. path: u.path,
86. method: req.method,
87. headers: req.headers
88. }
89. **if** (block) {
90. res.end(block)
91. } **else** **if** (redirect) {
92. res.writeHead(302, { 'Location': redirect })
93. res.end()
94. } **else** **if** (!allow) {
95. res.end('user-agent blocked')
96. } **else** **if** (cacheRes) { // cache hit
97. // request with if-modified-since
98. **const** lastModified = req.headers['if-modified-since'] || cacheRes.headers['last-modified'] || ''
99. opt.headers['if-modified-since'] = lastModified
100. http.request(opt, res => {
101. console.log('if-modified-since:', lastModified, res.statusCode)
102. **if** (res.statusCode !== 304) {
103. cache.set(req, res)
104. console.log('cache update', (**new** Date).toISOString())
105. }
106. }).end()
107. **if** (!cacheRes.headers['last-modified'])
108. cacheRes.headers['last-modified'] = (**new** Date).toUTCString()
109. res.writeHead(cacheRes.statusCode, cacheRes.headers)
110. cacheStream.pipe(res, { end: **true** })
111. } **else** { // cache not hit
112. http.request(opt, pRes => {
113. **if** (!pRes.headers['last-modified'])
114. pRes.headers['last-modified'] = (**new** Date).toUTCString()
115. res.writeHead(pRes.statusCode, pRes.headers);
116. cache.set(req, pRes)
117. pRes.pipe(res, { end: **true** })
118. }).end()
119. }
120. }
122. http.createServer()
123. .on('request', request)
124. .listen(port, hostname, () => {
125. console.log(`proxy run: ${hostname}:${port}`)
126. })

# 实验心得

对代理服务器的原理有了更深入的理解

对nodejs的net，http模块有了实践

更清晰地认识到http的安全性问题