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	实验 1：HTTP 代理服务器的设计与实现				
姓名	张志路		院系	计算机学院	
班级	1603106		学号	1160300909	
任课教师	聂兰顺		指导教师	聂兰顺	
实验地点	格物 207		实验时间	2018 年 10 月 31 日	
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

实验一：HTTP 代理服务器的设计与实现

1 实验目的

本次实验的主要目的在于熟悉并掌握 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

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）。

3 实验环境

Windows 版本	Windows 10 中文版
系统类型	64 位操作系统，基于 x64 的处理器
Python 版本	Python 3.6.4
编程工具	PyCharm 5.0.3、Anaconda3-5.1.0

4 实验过程

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

4.1 浏览器使用代理

为了使浏览器访问网址时通过代理服务器，必须进行相关设置，以 Chrome 浏览器设置为例：打开浏览器→设置→打开代理设置→局域网设置→代理服务器，将地址设置 127.0.0.1，端口号设置为 10240。具体过程如下图 4-1 所示。

说明：127.0.0.1 是回送地址，指本地机，一般用来测试使用。

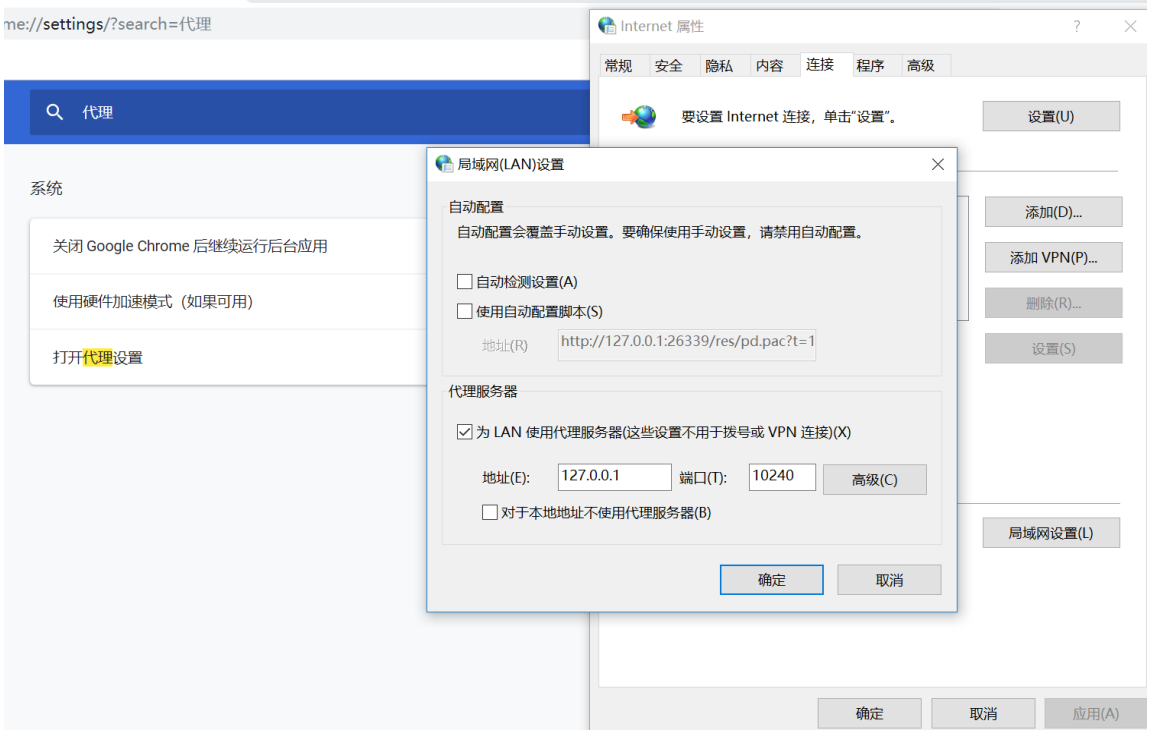


图 4-1：浏览器的代理服务器设置

4.2 Socket 编程的客户端和服务器端主要步骤

(1) 客户端

a. 确定服务器 IP 地址和端口号；

b. 创建套接字；

c. 分配本地端点地址（IP 地址+端口号）；

d. 使用 connect 函数连接服务器（套接字）；

e. 遵循应用层协议进行通信；

f. 关闭网络连接。

(2) 服务器端

a. 创建主套接字，使用 bind 函数绑定 IP 地址，端口信息等；

b. 使用 listen 函数设置主套接字为被动监听模式，准备用于服务器；

c. 反复调用 accept()函数接收下一个连接请求（通过主套接字），并创建一个新的子线程处理该客户响应;

d. 子线程通过新创建的套接字接收一个客户的服务请求；

e. 遵循应用层协议与特定客户进行交互；

f. 关闭/释放连接并退出（线程终止）。

4.3 HTTP 代理服务器的基本原理

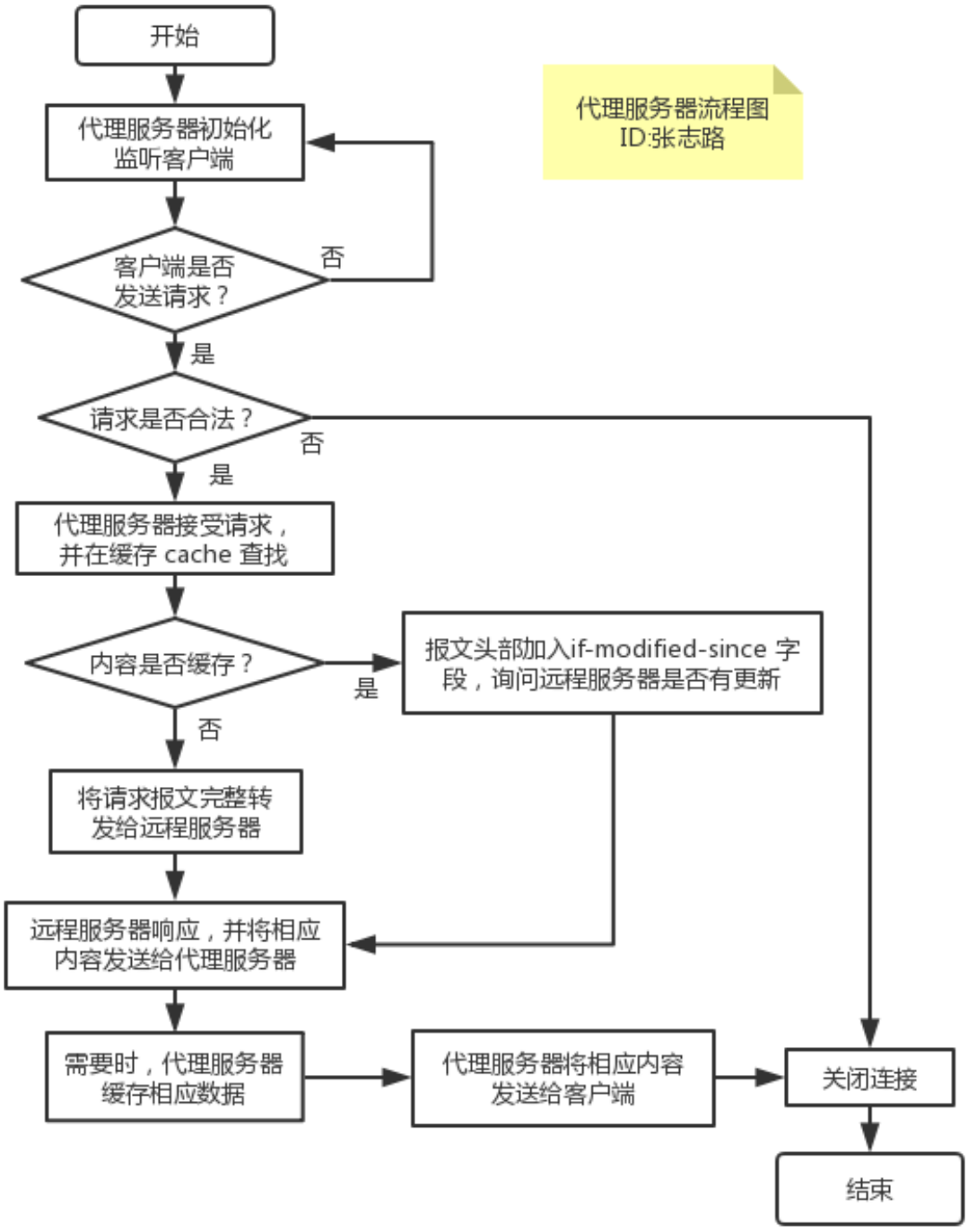
如果不使用代理服务器去访问 HTTP 协议的网站，浏览器会向服务器发送一个请求，服务器会产生响应，客户端从响应中得到相应的数据。

如果使用代理服务器，当客户在浏览器中设置好代理服务器后，客户端使用浏览器访问所有站点的 HTTP 请求都不会直接发给服务器，而是先发给代理服务器。代理服务器在接受了客户的请求以后，再由它向远程服务器发出请求，并接受远程服务器的数据，然后再由它将客户请求的数据发给客户。

对于支持 Cache 功能的 HTTP 代理服务器，浏览器向代理服务器发送所有的 HTTP 请求，如果所请求对象在缓存中，代理服务器直接返回缓存中的对象；否则，代理服务器向原始服务器发送 HTTP 请求，获取对象，然后返回给客户端并在代理服务器中保存该对象。

使用代理服务器可以实现翻墙，也可以通过缓存技术减少网络流量，提高浏览速度。加了代理服务器之后，相对于客户端来说，代理服务器就是服务器；而相对于真正的服务器来说，代理服务器就是客户端。

4.4 HTTP 代理服务器的程序流程图



4.5 实现 HTTP 代理服务器的关键技术及解决方案

(1) 套接字编程

我们在应用层上传递的所有消息均是使用 socket 传递的，因而我们需要正确地创建 socket，正确地绑定 socket 的地址，正确地设置 socket 的状态，正确地使用 socket 的 API。

(2) 基本代理服务器的正确实现

代理服务器要正确实现上述的客户端套接字编程和服务器端套接字编程，严格遵守上述原理以及流程，此处不再详述。

(3) 多线程

使用多线程组织各部分之间的调度关系，使代理服务器处于一直监听状态，对不同的请求使用使用多线程处理。

(4) 缓存功能

建立 cache 和 modified 字典（python），键为 url，值分别存储缓存的数据以及最后的修改时间。

客户端发送 HTTP 请求给代理服务器，当 cache 的键不包含其 url 时，代理服务器将 HTTP 请求消息转发给服务器，然后接收服务器返回的响应报文，将相应数据添加到 cache 和 modified 字典，并将响应报文返回给浏览器。

当 cache 的键包含其 url 时，在请求报文之中第三行加入“if-modified-since:date”，即在 HTTP 请求消息中声明所持有版本的日期，发送给服务器，然后接收到服务器返回的响应报文，对响应报文进行处理。检查响应报文头部是否为“304notmodified”，如果是，直接将 cache 中的已经缓存的相应数据返回给浏览器；如果不是，首先将该相应数据存入 cache 和 modified 中，即对该 url 对应的值进行更新，然后将响应报文返回给浏览器。

(5) 网站过滤

网站过滤、用户过滤以及网站引导通过设置文件来实现。

文件基本格式如下，其中“host”内容表示禁止的主机，“ip”内容表示禁止的用户，“phishing”内容中后部分为申请前部分网站时所跳转的网站。

```
{
  "host": [
    "www.fudan.edu.cn"
  ],

  "ip": [
    "127.0.0.1"
  ],

  "phishing": {
    "www.tsinghua.edu.cn": "www.xjtu.edu.cn"
  }
}
```

对于网站过滤，每当代理服务器接受来自客户端的请求时，先解析出请求报文头部，将请求报文头部中的 host 与文件禁止的 host 依次进行比较，如果存在相匹配的则禁止该主机访问，直接跳转到结束位置。

(6) 用户过滤

对于用户过滤，每当代理服务器接受来自客户端的请求时，先解析出请求报文头部，得到浏览器的地址信息，也就得到浏览器端的 ip 地址，与被禁的 ip 依次比较，如果存在相匹配的则禁止该用户访问，直接跳转到结束位置。

(7) 网站引导

对于网站引导，每当代理服务器接受来自客户端的请求时，先解析出请求报文头部，将请求报文头部中的 host 与文件“phishing”的前部分进行比较，如果存在相匹配的则进行网站跳转。此时向客户端发送一个携带钓鱼后的网站地址的 302 报文，接收到 302 报文的客户端就会发送一个对钓鱼网站的请求报文,代理服务器会回复钓鱼的网站信息。

5 实验结果

采用演示截图、文字说明等方式，给出本次实验的实验结果。

配置文件内容如下，其中“host”内容表示禁止的主机，“ip”内容表示禁止的用户，“phishing”内容中后部分为申请前部分网站时所跳转的网站。

```
{
  "host": [
    "www.fudan.edu.cn"
  ],

  "ip": [
    "127.0.0.1"
  ],

  "phishing": {
    "www.tsinghua.edu.cn": "www.xjtu.edu.cn"
  }
}
```

(1) 基本代理服务器功能

访问 www.hit.edu.cn, HTTP 头部打印信息和网站页面如下。

```
GET http://www.hit.edu.cn/ HTTP/1.1
Host: www.hit.edu.cn
Proxy-Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: UM_distinctid=16658bde8faa90-06d96d98f71045-4d045769-e1000-1
```

(2) 缓存功能

再次访问 www.hit.edu.cn 时服务器检测到已经缓存该网页内容，且为最新，HTTP 头部信息和提示信息打印如下。

```
http://www.hit.edu.cn/
GET http://www.hit.edu.cn/ HTTP/1.1
Host: www.hit.edu.cn
User-Agent: python-requests/2.18.4
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
If-Modified-Since: Wed, 31 Oct 2018 19:26:20 GMT


Cached Successful!!!
```

(3) 网站过滤功能

访问 www.fudan.edu.cn 时被禁止，HTTP 头部信息、提示信息和访问网站时的页面如下。

```
Proxy Server is running on ip: 127.0.0.1 port: 10240
GET http://www.fudan.edu.cn/ HTTP/1.1
Host: www.fudan.edu.cn
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleW
Accept: text/html,application/xhtml+xml,application/xml
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8


forbidden server host!!!
```

(4) 用户过滤功能

代理服务器检测到用户的 ip 被禁止，打印一连串提示信息。

```
代理服务器正在（ip: 127.0.0.1 ， port: 10240 ）上运行
Forbidden client ip!!!

Forbidden client ip!!!

Forbidden client ip!!!

Forbidden client ip!!!
```

(5) 网站引导功能

访问 www.tsinghua.edu.cn 时被引导到 www.xjtu.edu.cn，提示信息、HTTP 头部信息和访问网站时的页面如下。

```
HTTP/1.1 302 Moved Temporarily
Location: http://www.xjtu.edu.cn
```

```
GET http://www.xjtu.edu.cn/ HTTP/1.1
Host: www.xjtu.edu.cn
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Referer: http://www.tsinghua.edu.cn/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: JSESSIONID=D692D92A2928CFFDE2FE3362B0A5A590
If-None-Match: "ba9a-57983bb621fc0"
```

6 问题讨论

6.1 代理服务器器地址设置 127.0.0.1 的原因

有一特殊的 A 类 IP 地址，网络地址是 127，这类地址称作环回地址，主要用于网络软件测试以及本地机进程间通信。环回地址不离开主机的数据包，也就是说，这些数据包不会通过外部网络接口。使用环回地址，可以帮助我们在同一台主机上实现 client 和 server 的功能。

我们经常使用的是 127.0.0.1 这个地址，且赋给它一个名字：localhost。当 IP 层接收到目的地址为 127.0.0.1 的数据包时，不调用网卡驱动进行二次封装，而是立即转发到本机 IP 层进行处理，不涉及底层操作。

6.2 HTTP 头部简介

HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URI、协议版本、以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。服务器以一个状态行作为响应，相应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。

通常 HTTP 消息包括客户机向服务器的请求消息和服务器向客户机的响应消息。这两种类型的消息由一个起始行，一个或者多个头域，一个只是头域结束的空行和可选的消息体组成。HTTP 的头域包括通用头，请求头，响应头和实体头四个部分。每个头域由一个域名，冒号（:）和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

下面分别介绍通用头，请求头，响应头和实体头。

① 通用头域

通用头域包含请求和响应消息都支持的头域，通用头域包含 Cache-Control、Connection、Date、Pragma、Transfer-Encoding、Upgrade、Via。对通用头域的扩展要求通讯双方都支持此扩展，如果存在不支持的通用头域，一般将会作为实体头域处理。

② 请求头域

请求消息的第一行格式为：“MethodRequest-URIHTTP-Versio”。

Method 表示对于 Request-URI 完成的方法，这个字段是大小写敏感的，包括 OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE。方法 GET 和 HEAD 应该被所有的通用 WEB 服务器支持，其他所有方法的实现是可选的，GET 方法取回由 Request-URI 标识的信息，HEAD 方法也是取回由 Request-URI 标识的信息，只是可以在响应时，不返回消息体；POST 方法可以请求服务器接收包含在请求中的实体信息，可以用于提交表单，向新闻组、BBS、邮件群组和数据库发送消息。Request-URI 表示请求的 URL。Request-URI 遵循 URI 格式，在此字段为星号（*）时，说明请求并不用于某个特定的资源地址，而是用于服务器本身。HTTP-Version 表示支持的 HTTP 版本，例如为 HTTP/1.1。

请求头域允许客户端向服务器传递关于请求或者关于客户机的附加信息。请求头域可能包含下列字段 Accept、Accept-Charset、Accept-Encoding、Accept-Language、Authorization、From、Host、If-Modified-Since、If-Match、If-None-Match、If-Range、If-Range、If-Unmodified-Since、Max-Forwards、Proxy-Authorization、Range、Referer、User-Agent。对请求头域的扩展要求通讯双方都支持，如果存在不支持的请求头域,一般将会作为实体头域处理。

③ 响应头域

响应消息的第一行格式为：“HTTP-VersionStatus-CodeReason-Phrase”。

HTTP-Version 表示支持的 HTTP 版本，例如为 HTTP/1.1。Status-Code 是一个三个数字的结果代码。Reason-Phrase 给 Status-Code 提供一个简单的文本描述。Status-Code 主要用于机器自动识别，Reason-Phrase 主要用于帮助用户理解。Status-Code 的第一个数字定义响应的类别，后两个数字没有分类的作用。第一个数字可能取 5 个不同的值：

1xx:信息响应类，表示接收到请求并且继续处理；

2xx:处理成功响应类，表示动作被成功接收、理解和接受；

3xx:重定向响应类，为了完成指定的动作，必须接受进一步处理；

4xx:客户端错误，客户请求包含语法错误或者是不能正确执行；

5xx:服务端错误，服务器不能正确执行一个正确的请求。

响应头域允许服务器传递不能放在状态行的附加信息，这些域主要描述服务器的信息和 Request-URI 进一步的信息。响应头域包含 Age、Location、Proxy-Authenticate、Public、Retry-After、Server、Vary、Warning、WWW-Authenticate。对响应头域的扩展要求通讯双方都支持，如果存在不支持的响应头域，一般将会作为实体头域处理。

④ 实体头域

请求消息和响应消息都可以包含实体信息，实体信息一般由实体头域和实体组成。

实体头域包含关于实体的原信息，实体头包括 Allow、Content-Base、Content-Encoding、Content-Language、Content-Length、Content-Location、Content-MD5、Content-Range、Content-Type、Etag、Expires、Last-Modified、extension-header。extension-header 允许客户端定义新的实体头，但是这些域可能无法未接受方识别。

6.3 HTTP 与 HTTPS

超文本传输协议 HTTP（HyperText Transfer Protocol）协议被用于在 Web 浏览器和网站服务器之间传递信息，HTTP 协议以明文方式发送内容，不提供任何方式的数据加密，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，因此，HTTP 协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。

为了解决 HTTP 协议的这一缺陷，需要使用另一种协议：安全套接字层超文本传输协议 HTTPS（Hyper Text Transfer Protocol over Secure Socket Layer），为了数据传输的安全，HTTPS 在 HTTP 的基础上加入了 SSL 协议，SSL 依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

7 心得体会

结合实验过程和结果给出实验的体会和收获。

通过本次实验，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。

纸上得来终觉浅，绝知此事要躬行。通过实验也使得我对该部分的理论内容加深了理解，一些原来理解有偏差的点得到纠正，真正起到了巩固与提升的作用，真正做到了学以致用。

附录

```
import sys
import json
import socket
import select
import _thread
from urllib.parse import urlparse
import time
import requests

'''''代理服务器核心功能'''
class Proxy(object):

    def __init__(self, web_proxy_socket, modified_time, cached_data):
        """
        初始化
        :param web_proxy_socket: 用于侦听客户端的代理服务器套接字实例
        :param modified_time: 修改内容的最新时间(set)
        :param cached_data: 访问数据集(set)
        """
        self.webclient_proxy_socket, (self.webclient_ip, self.webclient_port) = web_proxy_socket.accept()
        # 接受来自 WebClient 的请求并返回一个新套接字,通过这个新套接字（webclient_proxy_socket）发送和修改消息
        self.BUF_SIZE = 66500  # recv Maximum receive
        self.HTTP_METHOD = ['GET', 'POST'] # HTTP 方法
        self.request = ''
        self.method = ''
        self.port = 80 # 远程服务器端口
        self.host = ''
        self.url = ''
        self.modified_time = modified_time
        self.cached_data = cached_data

    def run(self):
        self.request = self.webclient_proxy_socket.recv(self.BUF_SIZE)
        # 从客户端中得到原始数据，接收 TCP 数据，数据以字符串形式返回，bufsize 指定要接收的最大数据量。
        if not self.request: # 为空时返回
            return
        #print(self.request.decode('utf8','ignore'))
        #print('\n')
```

```
39.
40.         # 分析 http 消息
41.         lines = self.request.split(b'\r\n')
42.         firstline = lines[0].split()
43.         self.method = firstline[0] # get 或 post
44.         self.url = firstline[1]
45.         parse_url = urlparse(self.url)
46.         # 将 url 分解成部件的 6 元组：
    <schema>://<net_loc>/<path>;<params>?<query>#<fragment>
47.         # 例如
    (scheme='http',netloc='www.hit.edu.cn',path='224/list.psp', params='', query
    ='', fragment='')
48.         self.host = parse_url.netloc
49.
50.         '''''主模块'''
51.         if not self.website_filtering(): # 如果非禁止网站
52.             f_data = self.is_Phishing()
53.             if f_data: # 如果是钓鱼网站
54.                 self.webclient_proxy_socket.send(bytes(f_data, encoding="utf
    8")) # 发送 TCP 数据，将 string 中的数据发送到连接的套接字。
55.                 self.webclient_proxy_socket.close()
56.                 return
57.             try:
58.                 sock_info = socket.getaddrinfo(self.host, self.port)[0] # 返
    回五元组(family,socktype,proto,canonname,sockaddr)
59.             except BaseException as e:
60.                 sys.exit(1)
61.             else:
62.                 # 为远程目标服务器构建套接字
63.                 try:
64.                     self.proxy_trgserver_socket = socket.socket(sock_info[0]
    , sock_info[1])
65.                     self.proxy_trgserver_socket.connect((self.host, self.por
    t)) # 客户端套接字主动初始化 TCP 服务器连接
66.                     self.proxy_trgserver_socket.send(self.request) # 发送 TCP
    数据，将 string 中的数据发送到连接的套接字
67.                 except:
68.                     sys.exit(1)
69.                 input = [self.proxy_trgserver_socket, self.webclient_proxy_s
    ocket]
70.
71.                 self.cached_modified()
72.
73.                 while True:
```

```
74.                     readable, writable, exceptional = select.select(input, [
    ], input, 3) # (inputs,outputs,inputs,timeout)
75.                     # 开始 select 监听,对 input 中的服务端 server 进行监听
76.                     # select 函数阻塞进程，直到 inputs 中的套接字被触发,readable
    返回被触发的套接字（服务器套接字）
77.                     if exceptional:
78.                         break
79.
80.                     # discover triked socket to exchange data
81.                     # 循环判断是否有客户端连接进来,当有客户端连接进来时 select 将
    触发
82.                     for sock in readable:
83.                         try:
84.                             data = sock.recv(self.BUF_SIZE) # 接收 TCP 数据，
    数据以字符串形式返回，bufsize 指定要接收的最大数据量。
85.                             if data:
86.                                 if sock is self.proxy_trgserver_socket: # 无
    钓鱼
87.                                     self.webclient_proxy_socket.send(data) #
     发送 TCP 数据，将 string 中的数据发送到连接的套接字。
88.                                 if sock is self.webclient_proxy_socket: # 钓
    鱼
89.                                     self.proxy_trgserver_socket.send(data) #
     发送 TCP 数据，将 string 中的数据发送到连接的套接字。
90.                             else:
91.                                 break
92.                         except:
93.                             break
94.             self.webclient_proxy_socket.close()
95.             self.proxy_trgserver_socket.close()
96.
97.
98.     '''''缓存修改后的内容'''
99.     def cached_modified(self):
100.         #print(self.cached_data)
101.         if self.url in self.cached_data:
102.             cached_time = self.modified_time[self.url]
103.             # 将修改时间转换为格林尼治平时间
104.             head = { 'If-Modified-
    Since': time.strftime('%a, %d %b %Y %H:%M:%S GMT', time.gmtime(cached_time))
     }
105.             try:
106.                 r = requests.get(str(self.url, encoding="utf-
    8"), headers=head)
```

```
107.                 if r.status_code == 304: # 已经缓存且为最新
108.                     print('Cached Successful!!!\n')
109.                     self.webclient_proxy_socket.send(self.cached_data[self.
    url])
110.                 else:  # 非最新缓存
111.                     data = self.proxy_trgserver_socket.recv(self.BUF_SIZE)

112.                     self.cached_data[self.url] = data
113.                     self.modified_time[self.url] = time.time()
114.                     self.webclient_proxy_socket.send(data)
115.             except:
116.                 pass
117.         else:  # 未缓存
118.             try:
119.                 data = self.proxy_trgserver_socket.recv(self.BUF_SIZE)
120.             except BaseException as e:
121.                 pass
122.             else:
123.                 self.cached_data[self.url] = data
124.                 self.modified_time[self.url] =  time.time()
125.                 self.webclient_proxy_socket.send(data)
126.         fw1 = open('catch_modified_time.txt', 'w')
127.         fw1.write(str(self.modified_time))
128.         fw1.close()
129.         fw2 = open('catch_cached_data.txt', 'w')
130.         fw2.write(str(self.cached_data))
131.         fw2.close()
132.
133.     '''''网站过滤与用户过滤，返回 true or false'''
134.     def website_filtering(self):
135.         with open('webrules.json', 'r') as f:
136.             webrule_json = json.load(f)
137.             if self.webclient_ip in webrule_json['ip']: # 非法客户端 ip
138.                 print('Forbidden client ip!!!\n')
139.                 return True
140.             for h in webrule_json['host']: # 非法目标服务器
141.                 host_str = str(self.host, encoding="utf-8")
142.                 if host_str.endswith(h):
143.                     print('Forbidden server host!!!\n')
144.                     return True
145.         return False
146.
147.
148.     '''''网站引导（钓鱼），返回文本 or false'''
```

```
149.     def is_Phishing(self):
150.         with open('webrules.json', 'r') as f:
151.             webrule_json = json.load(f)
152.             for phi in webrule_json['phishing']:
153.                 host_str = str(self.host, encoding="utf-8")
154.                 if host_str == phi:
155.                     print(str('HTTP/1.1 302 Moved Temporarily\r\n'
156.                         'Location: http://' +webrule_json['phishing'][host_
    str] + '\r\n\r\n'))
157.                     return str('HTTP/1.1 302 Moved Temporarily\r\n'
158.                         'Location: http://' +webrule_json['phishing'][host_
    str] + '\r\n\r\n')
159.         return False
160.
161.
162. '''''代理服务器'''
163. class Proxy_Server(object):
164.     def __init__(self, ip, port, maxnum):
165.         '''''
166.         代理服务器初始化
167.         :param ip: 代理服务器 IP 地址
168.         :param port: 代理服务器端口号
169.         :param Cnum: 代理服务器允许的最多的客户端连接数
170.         '''
171.
172.         self.ip = ip
173.         self.port = port
174.         self.web_proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_S
    TREAM) # 创建套接字连接客户端 (IPV4, TCP)
175.         self.web_proxy_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSE
    ADDR, 1) # reset socket
176.         self.web_proxy_socket.bind((ip, port)) # 使用 bind 函数绑定代理服务器
    IP 地址，端口号
177.         self.web_proxy_socket.listen(maxnum)  # 使用 listen 函数进行监听创建的
    socket。
178.         self.modified_time = dict() # 存储修改数据的最后时间
179.         self.cache_data = dict() # 存储修改过的数据
180.
181.     def run(self):
182.         print ("代理服务器正在（ip: %s ，port: %s ）上运行
    "%(str(self.ip), str(self.port)))
183.         while True:
184.             # 多线程
```

```
185.             _thread.start_new_thread(Proxy(self.web_proxy_socket, self.modi
    fied_time, self.cache_data).run, ())
186.
187.
188. if __name__ == '__main__':
189.     Proxy_Server('127.0.0.1', 10240, 5).run()
```