哈尔滨工业大学

计算机网络

实验报告

(2019 年度秋季学期)

姓名:	史纪元
学号:	1173300919
学院:	计算机科学与技术学院
教师:	刘亚维

实验一 HTTP 代理服务器的设计与实现

一、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

二、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。
(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）。

三、实验过程及结果

1. Socket 编程的客户端和服务器端主要步骤

客户端

1）根据要建立的连接的传输层协议（TCP 或 UDP）创建套接字
2）设置套接字选项参数（可选）
3）调用 connect()函数与服务器端建立连接(UDP 套接字调用 connect()只是指定服务器端地址）
4）调用 send()函数（未连接的 UDP 是 sendto()）向服务器发送请求、调用 recv()函数（未连接的 UDP 是 recvfrom())从服务器接收数据
5）关闭套接字

服务器端

1）创建套接字
2）调用 bind()将套接字绑定到固定 ip+端口地址
3）调用 listen()开启监听（仅用于 TCP 套接字）
4）收到连接请求后，调用 accept()创建新套接字（仅用于 TCP 套接字）

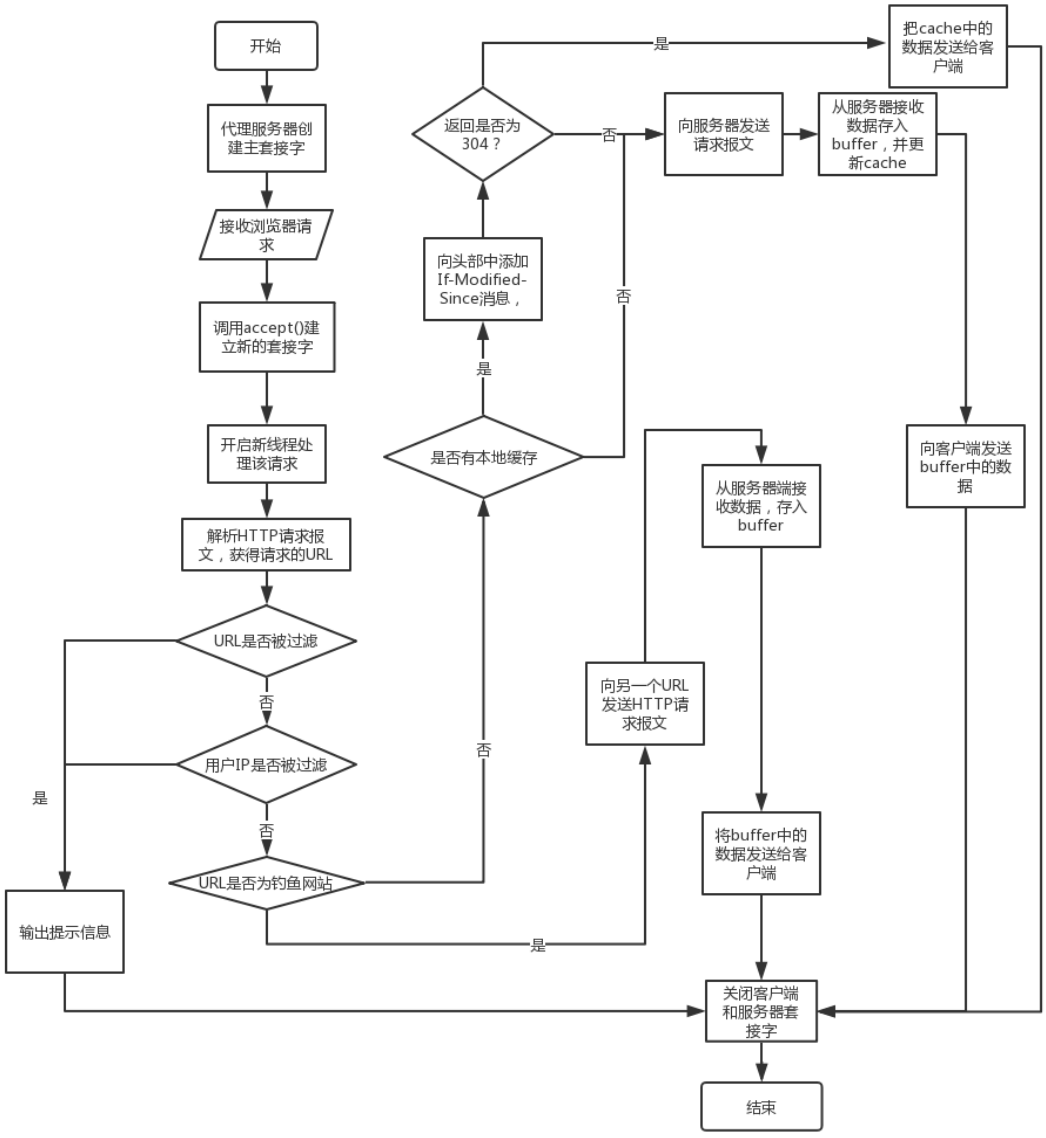
5）调用 send()函数（未连接的 UDP 是 sendto()）向服务器发送请求、调用 recv()函数（未连接的 UDP 是 recvfrom())从服务器接收数据

6）关闭套接字

2. HTTP 代理服务器的基本原理

HTTP 代理服务器在网络中位于客户端和服务器中间，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。HTTP 代理服务器在指定端口监听客户端的访问请求，并对客户端的请求进行解析，再根据代理服务器自身的策略决定如何向服务器端发送请求报文，以此实现缓存、过滤网页、过滤用户等功能。

3. HTTP 代理服务器的程序流程



4. 实现 HTTP 代理服务器的关键技术及解决方案

4.1 解析 HTTP 请求报文

要实现代理服务器的一系列功能，首先需要解析客户端发来的 HTTP 请求报文。我们已经知道 HTTP 请求报文不同部分之间以\r\n 分隔，且请求报文的第一部分为请求行，具体格式为：

请求方法+空格+URL+空格+协议版本

原始报文为 bytes 类型，需要先解码为字符串，方便进一步操作。再用‘\r\n’划分字符串，得到 list 类型的 HTTP 请求头部信息：

```
    ori_message = client_socket.recv(self.BUFFER_SIZE)
    message = ori_message.decode('utf-8', 'ignore')   # 将 bytes 类型的报文转换为字符串
    header = message.split('\r\n')   # 把报文以\r\n 分割 得到 list
```

得到的 header 格式如下（以今日哈工大首页为例）：

```
header:
['GET http://today.hit.edu.cn/ HTTP/1.1', 'Host: today.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; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/78.0.3904.70 Safari/537.36', 'Accept: text/html,application/xhtml+xml,
 application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3',
 'Accept-Encoding: gzip, deflate', 'Accept-Language: zh-CN,zh;q=0.9', 'Cookie: _ga=GA1.3
 .219149798.1567314271; Drupal.visitor.DRUPAL_UID=16010', '', '']('127.0.0.1', 49492)
```

然后获得头部中的请求行 request_line，即 header[0]，并用空格划分：

```
    request_line = header[0].split()
```

得到请求行：

```
request line:
['GET', 'http://today.hit.edu.cn/', 'HTTP/1.1']
```

可以看到请求行中包含 url 等信息，对请求行做进一步处理。首先提取出 http://之后的部分（请求的对象在服务器主机下的路径）用来当 cache 的文件名（'/'替换为'_'），对于含有参数的 url，要把‘？’之后的部分删去，若 path 的最后一个字符为'/'，还要把'/'删去。

进一步处理 url，获得从'http://'到下一个'/'之间的主机名部分。

```
    tmp = request_line[1][7:]    # 将 http://后的部分取出
    if '?' in tmp:
        # 有些 url 内包含'?',不能作为文件路径,应该去掉'?'之后的部分
        path = tmp[:tmp.index('?')]    # path 为 cache 文件的路径
    else:
        path = tmp
    hostname = path[:path.index('/')]   # 提取主机名
```

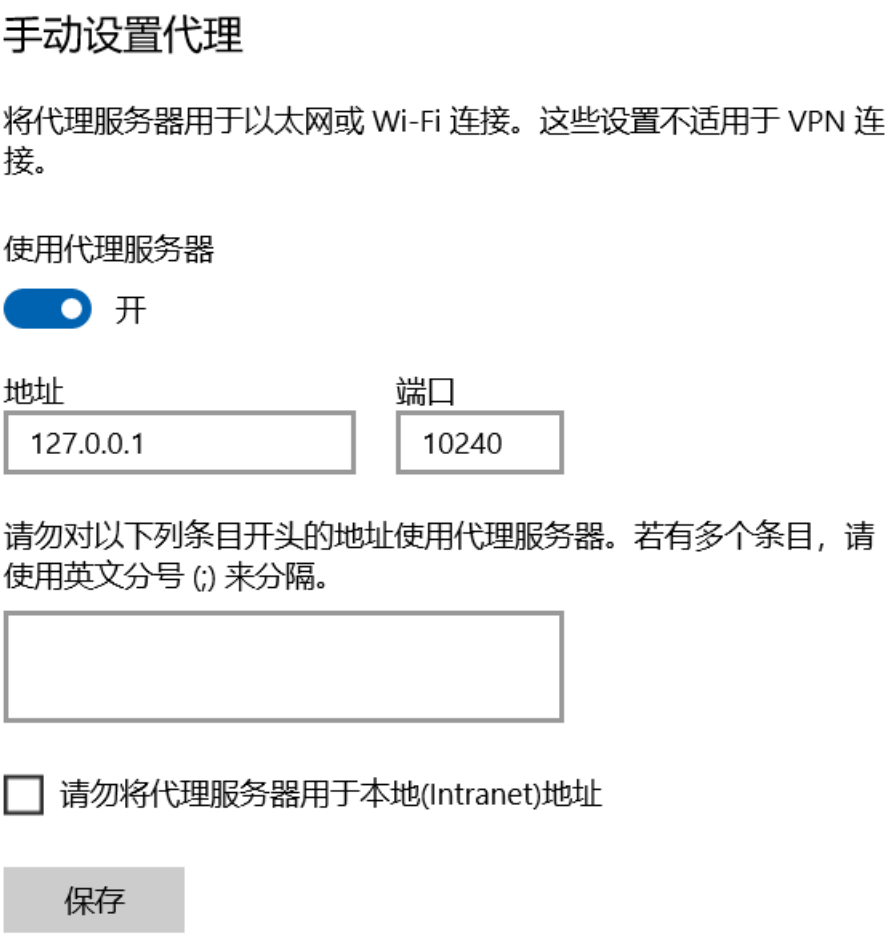
```
    if path[-1] == '/':
        path = path[:-1]      # 若 path 的最后一个字符为'/'则删去
```

4.2 转发请求和响应报文

代理服务器共创建了**三种套接字**，一个用于绑定在固定端口监听浏览器的请求，另外两个一个用于连接客服端，从客户端获得请求报文、发送响应报文；一个用于连接服务器，向服务器转发请求报文、获得响应报文。

主套接字绑定在 10240 端口，监听该端口发出的请求，同时需要把计算机上的网络代理端口设置为 10240，这样浏览器发送的请求就会被代理服务器的主套接字接收并处理。

```
    self.port = 10240  # 绑定的端口号
    # 初始化 socket
    self.proxy_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   #
创建主 socket 用于监听
    self.proxy_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# 设置 socket 参数
    self.MAX_LISTEN = 20  # 最大连接数
    self.proxy_socket.bind(('', self.port))
    self.proxy_socket.listen(self.MAX_LISTEN)
    self.BUFFER_SIZE = 2048
```



代理服务器收到来自客户（即浏览器）的请求后，会调用 accept，创建

一个新的套接字用来与客户端通信，即 client_socket

```
    new_sock, address = proxy.proxy_socket.accept()
    threading.Thread(target=proxy.connect, args=(new_sock, address)).start()
    # 从客户端接收 http 请求
    ori_message = client_socket.recv(self.BUFFER_SIZE)
    # 将接收到的数据转发给客户端
    client_socket.sendall(buff)
```

代理服务器会从客户端接收请求，并把需要转发给客户端的数据通过 client_socket 发送给客户端。

另外，代理服务器还需要创建一个套接字用来与服务器进行通信，即 server_socket

```
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.connect((hostname, 80))
server_socket.sendall(message.encode())
data = server_socket.recv(self.BUFFER_SIZE).decode('utf-8', 'ignore')
```

代理服务器通过 server_socket 向服务器转发客户端的请求报文（经过处理和分析），并从服务器接收响应报文，再通过 client_socket 转发给客户端。

4.3 cache 功能

访问某网站时，先判断./cache 文件夹内是否有该缓存，若无缓存，则将该网页内容缓存到本地。把第一步获得的 path 中的“/”改为“_”，然后写入./cache 文件夹下；若存在缓存，则向请求报文中加入 If-Modified-Since 字段，时间为缓存文件更新的时间。

经过观察，HTTP 协议判断 HTTP 头部是否结束的依据是连续两个\r\n\r\n，所以需要把原报文末尾的两个\r\n 删掉一个，再加入 If-Modified-Since 字段，再在末尾添加两个\r\n

```
        modified_time = os.stat(cache_path).st_mtime  # 缓存最后一次修改的时间
        headers = str('If-Modified-Since:
        '+time.strftime('%a, %d %b %Y %H:%M:%S GMT',
        time.gmtime(modified_time))) # 把 modified-time 按报文要求的格式格式化
        message = message[:-2] + headers + '\r\n\r\n'
```

根据 RFC2616[1]，只要 If-Modified-Since 后跟的时间早于网站更新的时间，就需要重新从服务器获取内容。

接下来向服务器发送该请求，并判断服务器返回代码是否为 304，即网页内容是否未修改。若得到的返回代码是 304，则把本地缓存的内容发送给客户端。否则，把 is_modified 标志设为 True，然后把请求报文发送给服务器端，从服务器接收数据并转发给客户。

```
        server_socket.connect((hostname, 80))
        server_socket.sendall(message.encode())
```

```
        data = server_socket.recv(self.BUFFER_SIZE).decode('utf-8', 'ignore')
        server_socket.close()
        if data[9:12] == '304':
            print("Read from cache")
            with open(cache_path, "rb") as f:
                client_socket.sendall(f.read())
        else:
           is_modified = True
```

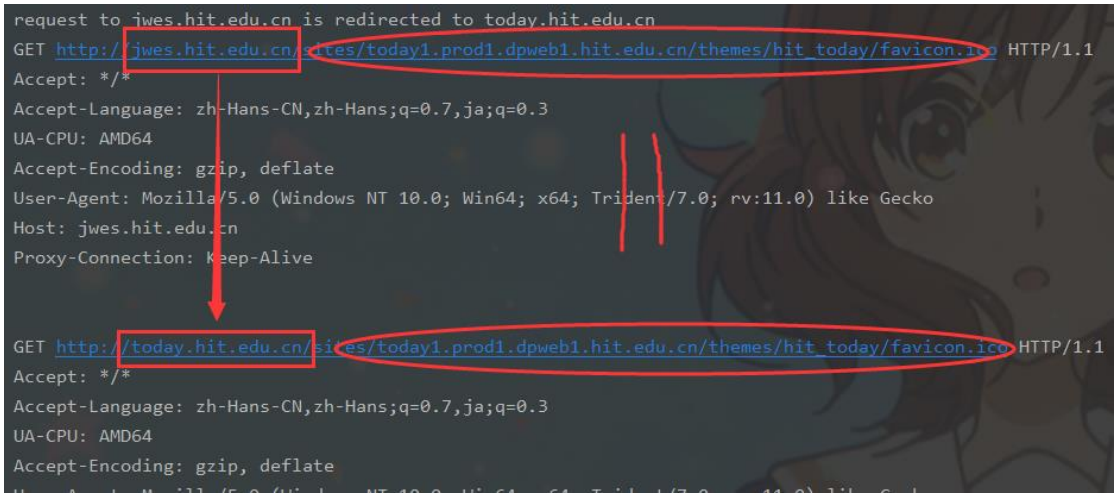
4.4 过滤网站、用户，钓鱼功能

首先新建三个 txt 文件用来保存被过滤的网站、用户以及钓鱼网站。并在与服务器建立连接前先判断网站、用户是否被过滤，用户是否访问钓鱼网站。若网站或用户被过滤，则直接关闭套接字并结束线程，并输出提示信息；

```
# 判断 url 是否被过滤
if self.filter_page(hostname):
    print('request to connect to ' + str(hostname) + ' is rejected\n')
    client_socket.close()
    return

# 判断 ip 地址是否被过滤
if self.filter_user(address[0]):
    print('user '+address[0]+' is forbidden to connect to proxy server')
    client_socket.close()
    return
```

如果用户访问钓鱼网站，则用 server_socket 与被导向网站服务器主机建立连接，并且需要修改请求报文中请求行 url 的主机名部分。因为经过观察，如果仅与导向网站服务器主机建立连接，之后客户端发出的请求的 url 变为“原服务器主机名+导向网站的文件路径”（见下图），而因为原服务器下并没有导向网站服务器主机某路径下的文件，故只能加载出一些文本，而后续的图片、样式等文件不能获取。因此，还需要把后续浏览器发出的请求的主机名部分都改为导向网站的服务器主机名即可实现完整导向网站的访



问。

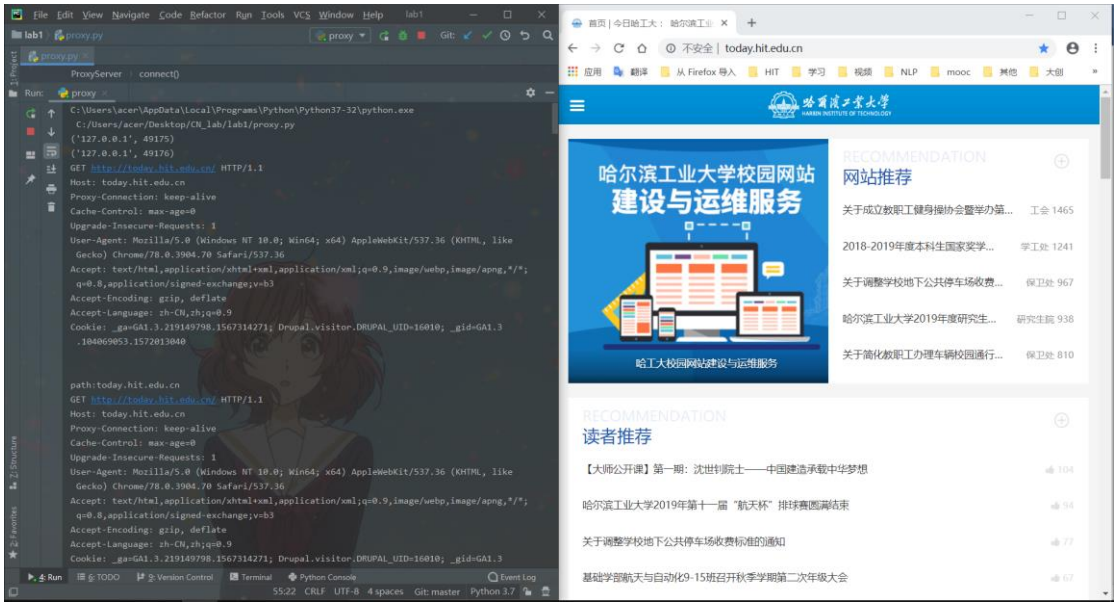
对于出现这种现象的原因，我认为是因为客户端在发送请求时没有考虑可能存在主机名与路径不对应的情况，在与导向服务器建立了连接后，客户端仅知道还需要加载某某路径下的文件，而察觉不到此时主机已经变了，故客户端会发送“原主机名+导向网站路径”这样的错误格式。因此，代理服务器还需要把客户端之后发送的报文请求行 url 的原主机名改为导向网站服务器的主机名，这样便能成功实现钓鱼功能。

```
if self.fish(hostname):
    # 将客户引导到其他网站
    new_hostname = 'today.hit.edu.cn'  # 导向网站的主机名
    print('request to ' + hostname + ' is redirected to ' + new_hostname)
    begin = message.index('h')
    end = message.index('/', 12)   # 找到 url 中主机名的部分
    # 将主机名部分修改为导向网站的主机名
    message = message[:begin+7] + new_hostname + '/' + message[end+1:]
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.connect((new_hostname, 80))   # 与导向网站的服务器建立连接
    server_socket.sendall(message.encode())    # 将请求报文发送给导向网站服务器
    while True:
        # 从服务器接收数据,转发给客户端
        buff = server_socket.recv(self.BUFFER_SIZE)
        if not buff:
            server_socket.close()
            break
        client_socket.sendall(buff)
    client_socket.close()
    server_socket.close()
    return
```

5. HTTP 代理服务器实验验证过程以及实验结果

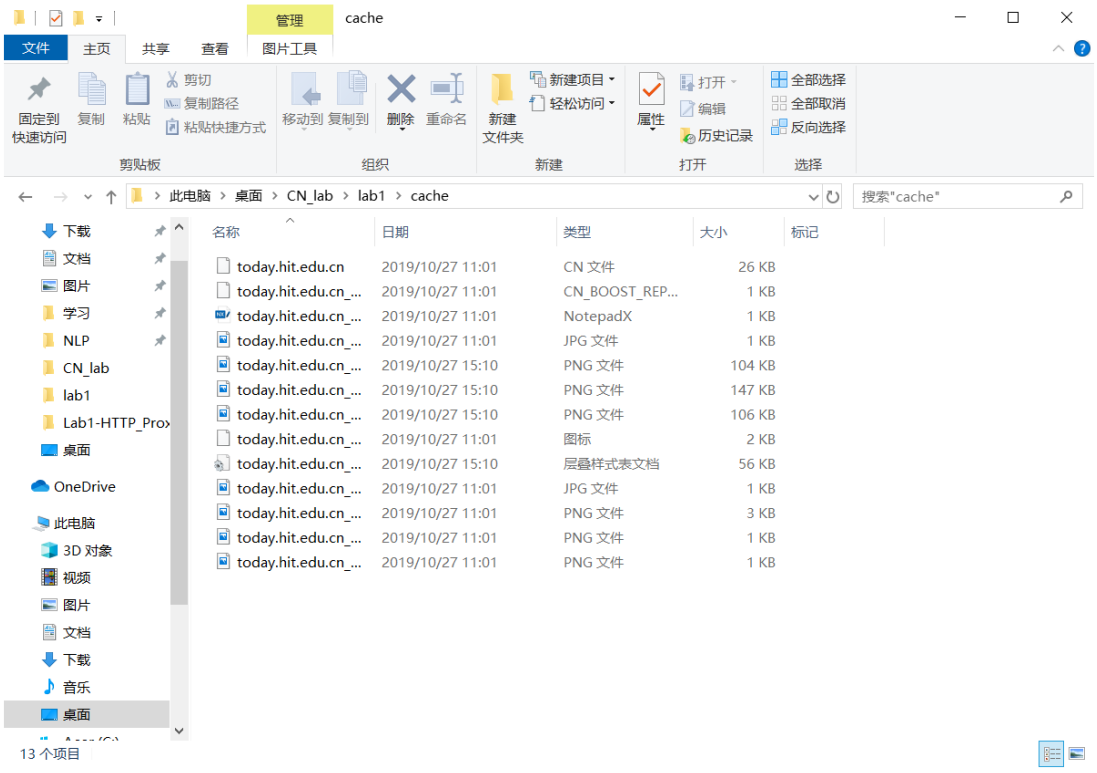
5.1 基本功能

开启代理服务器，运行代理服务器程序，访问 today.hit.edu.cn，代理服务器接收到请求报文，并向服务器转发，再从服务器获得响应报文，转发给客户端，浏览器上显示 today.hit.edu.cn 的页面：

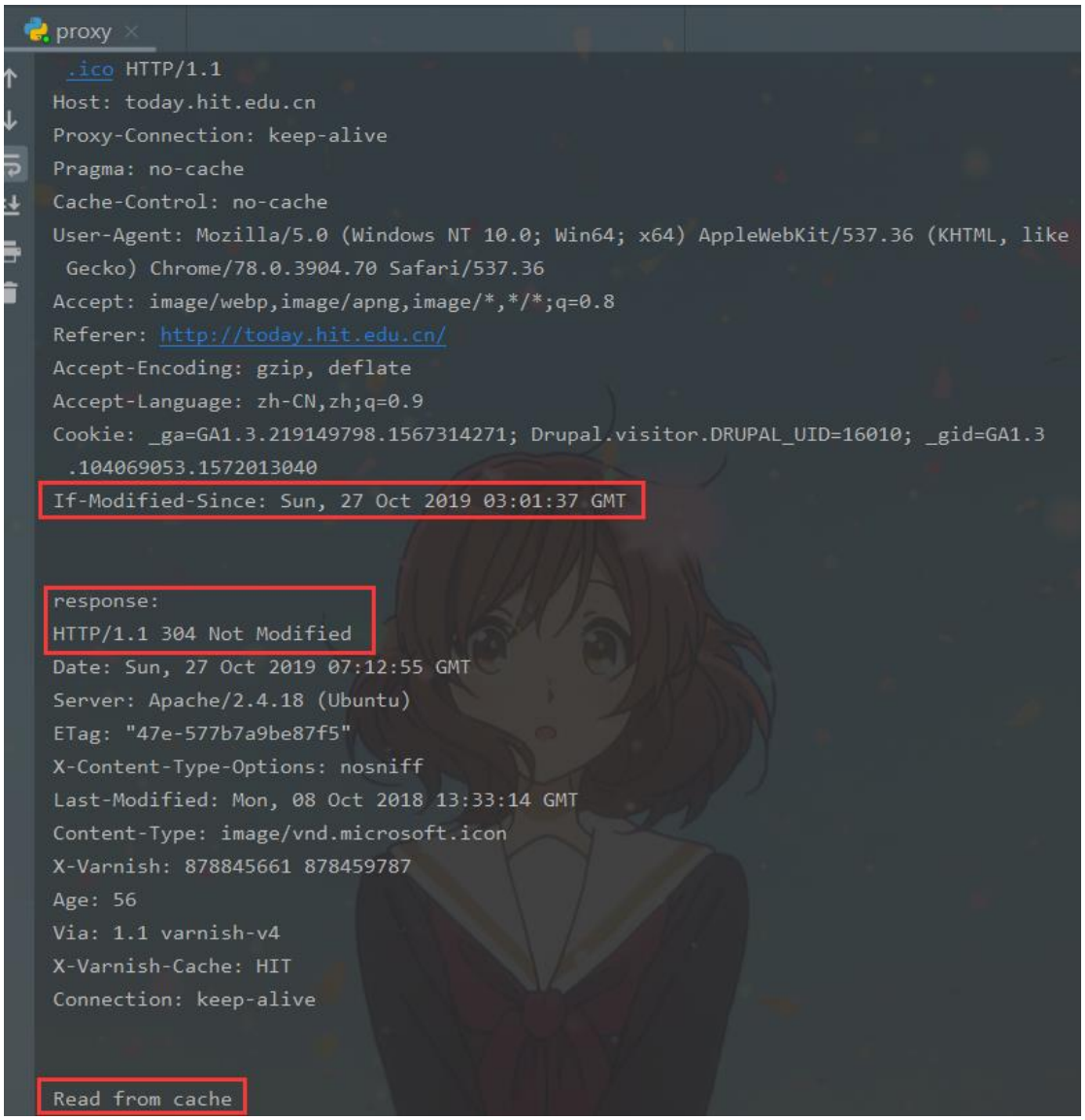


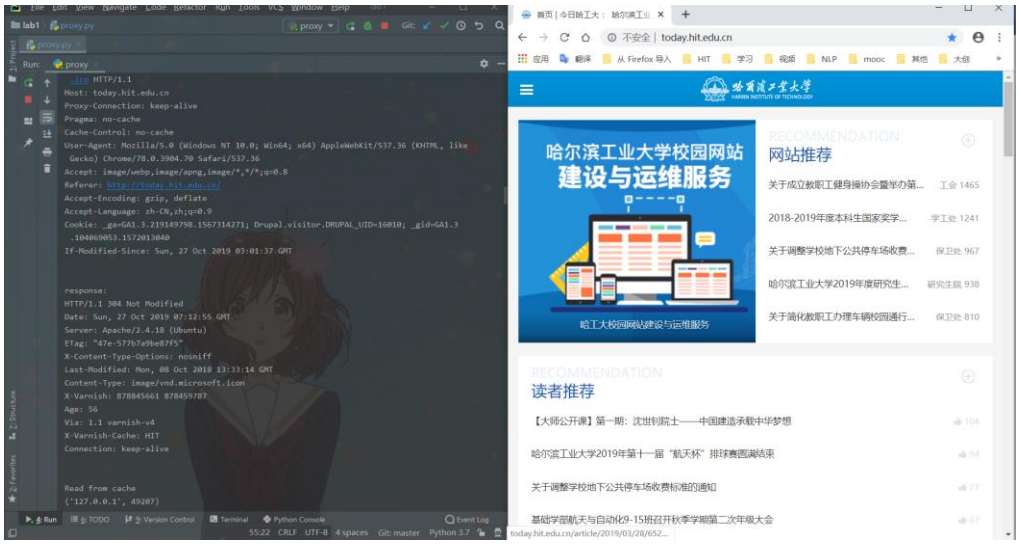
5.2 cache 功能

经过第一次访问 today.hit.edu.cn，本地 cache 文件夹中已经有了该网站的缓存：



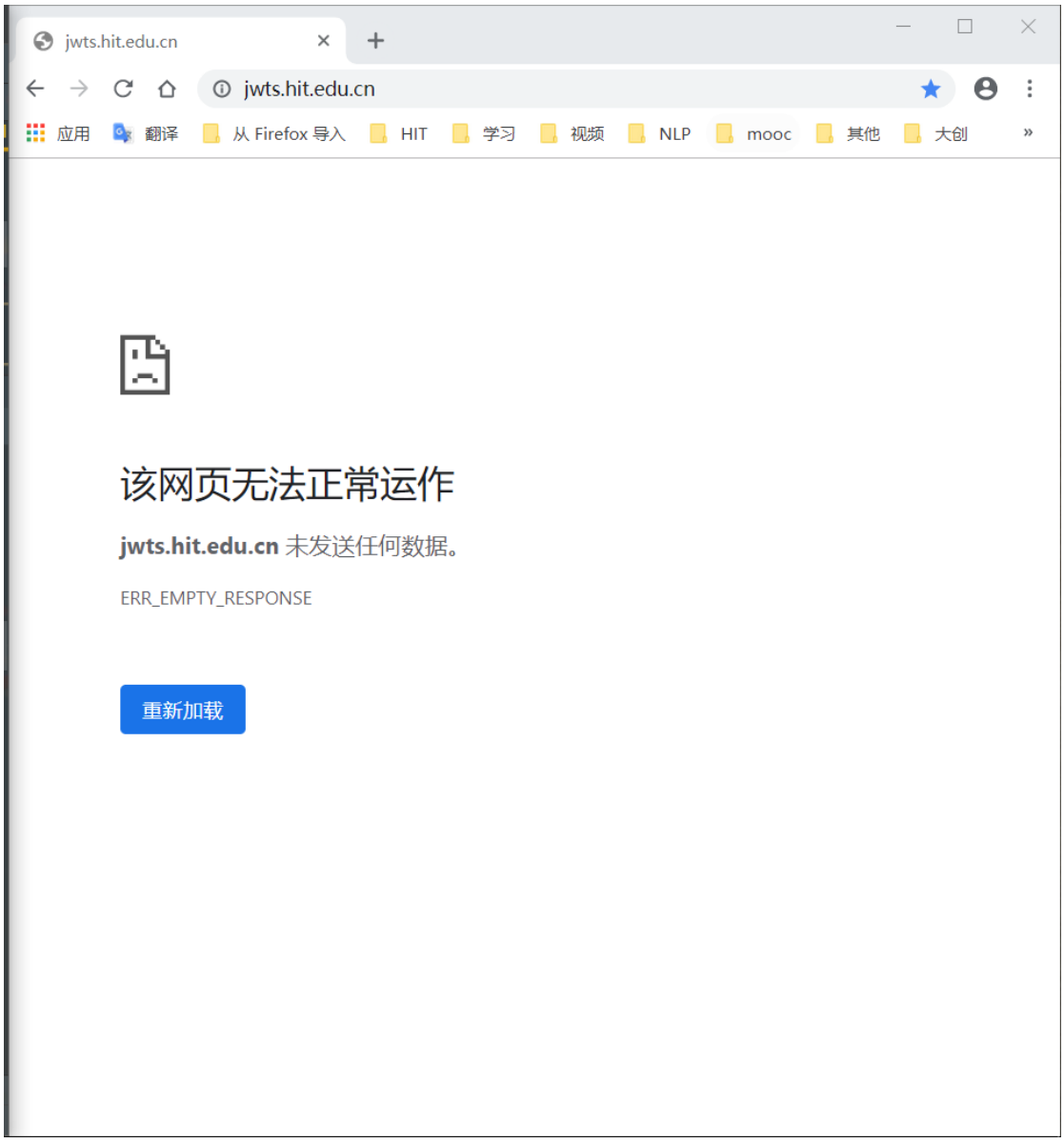
再次访问 today.hit.edu.cn，这时代理服务器在 cache 中找到了对应的文件，故代理服务器向 HTTP 头部中加入 If-Modified-Since 字段，且得到服务器的响应代码为 304，意思是该时间之后没有更新网站，于是代理服务器从本地缓存中读取数据交给客户端：





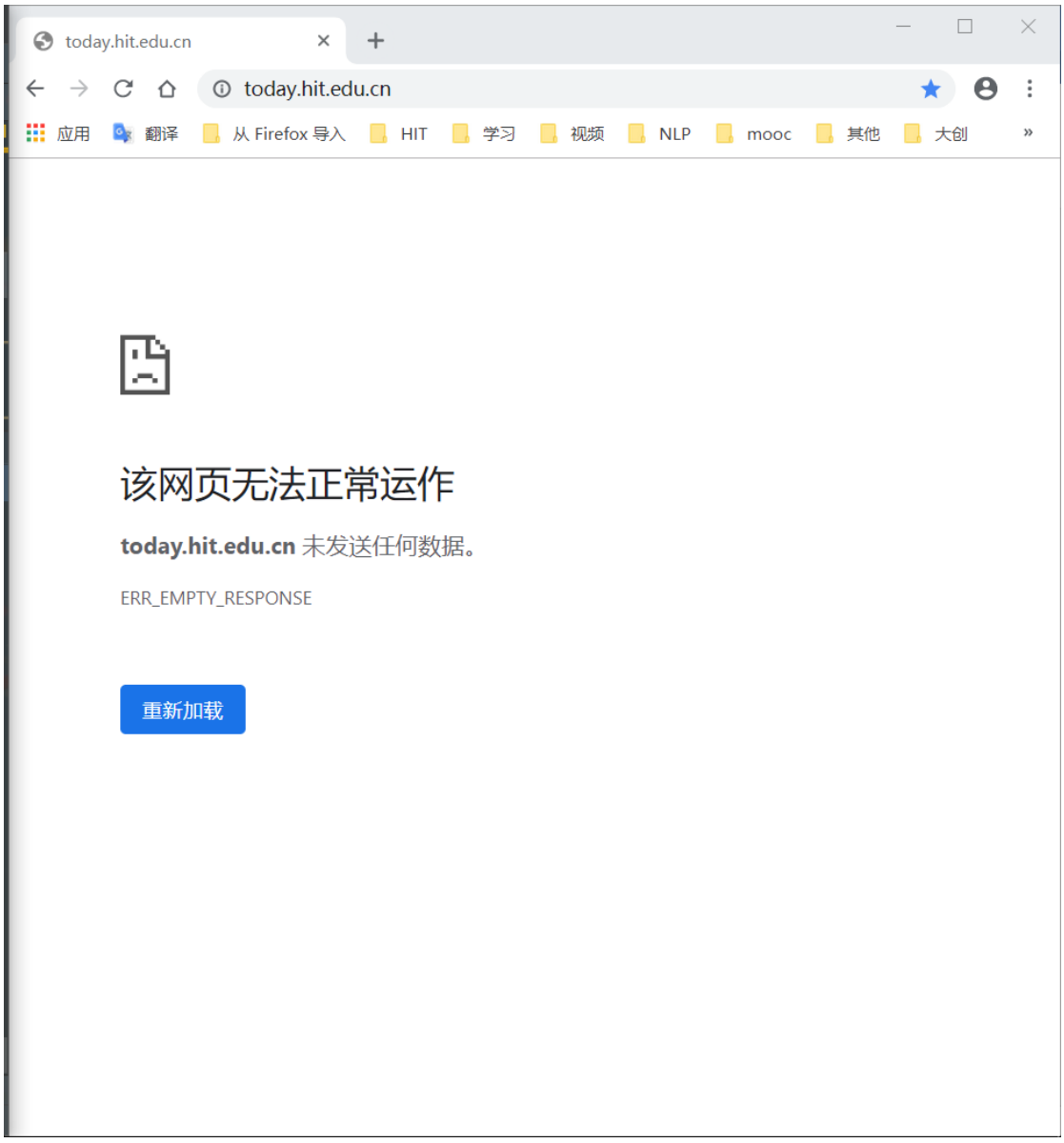
5.3 过滤网站、用户、网站引导

访问被过滤的 url：jwts.hit.edu.cn，请求被拒绝，并输出提示信息

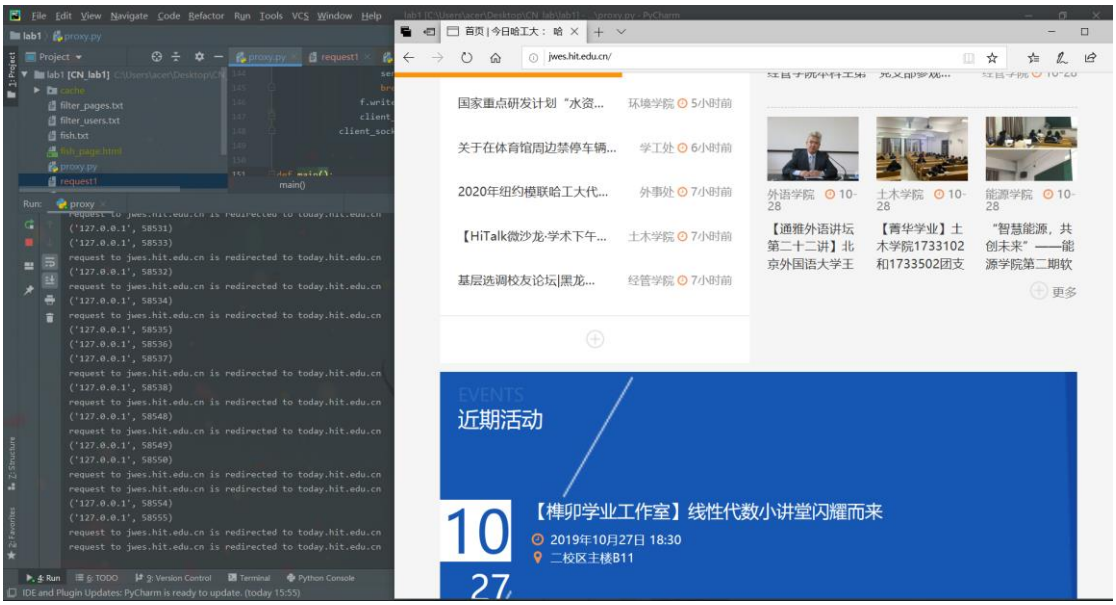


将过滤 IP 设置为 127.0.0.1（本地环回地址），则禁止用户的连接请求，关闭套接字，并输出提示信息：


GET http://today.hit.edu.cn/ HTTP/1.1
Host: today.hit.edu.cn
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.70 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: _ga=GA1.3.219149798.1567314271; Drupal.visitor.DRUPAL_UID=16010; _gid=GA1.3.104069053.1572013040
user 127.0.0.1 is forbidden to connect to proxy server


today.hit.edu.cn
today.hit.edu.cn
应用
翻译
从 Firefox 导入
HIT
学习
视频
NLP
mooc
其他
大创
该网页无法正常运作
today.hit.edu.cn 未发送任何数据。
ERR_EMPTY_RESPONSE
重新加载

客户访问位于 fish.txt 中的网址 jwes.hit.edu.cn，被引导到 today.hit.edu.cn，并加载出全部内容。（网站引导）



6. HTTP 代理服务器源代码

```
import socket
import threading
import os
import time


class ProxyServer:
    def __init__(self):
        self.port = 10240  # 绑定的端口号
        # 初始化 socket
        self.proxy_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)   # 创建主 socket 用于监听
        self.proxy_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)   # 设置 socket 参数
        self.MAX_LISTEN = 20  # 最大连接数
        self.proxy_socket.bind(('', self.port))
        self.proxy_socket.listen(self.MAX_LISTEN)
        self.BUFFER_SIZE = 2048
        self.cache_dir = './cache/'  # cache 文件夹
        if not os.path.exists(self.cache_dir):
            os.mkdir(self.cache_dir)     # 创建 cache 目录

    @staticmethod
    def filter_page(url):
```

```
        # 判断 url 是否被过滤
        with open('filter_pages.txt', 'r') as f:
            filter_pages_list = f.readlines()
        if url in filter_pages_list:
            return True
        return False

    @staticmethod
    def filter_user(ip):
        # 判断用户 ip 是否被过滤
        with open('filter_users.txt', 'r') as f:
            filter_users_list = f.readlines()
        if ip in filter_users_list:
            return True
        return False

    @staticmethod
    def fish(url):
        # 判断用户访问的是否是钓鱼网站
        with open('fish.txt', 'r') as f:
            fish_pages = f.readlines()
        if url in fish_pages:
            return True
        return False

    def connect(self, client_socket, address):
        # 从客户端接收 http 请求
        ori_message = client_socket.recv(self.BUFFER_SIZE)
        message = ori_message.decode('utf-8', 'ignore')   # 将 bytes 类
型的报文转换为字符串
        header = message.split('\r\n')   # 把报文以\r\n 分割 得到 list
        # header 的第一行为请求行
        # 将 Request Line 的 method URL 和 version 3 个部分分开 strip 去除
首部空格
        request_line = header[0].split()
        # print(message)
        if len(request_line) > 1:
            tmp = request_line[1][7:]    # 将 http://后的部分取出
            if '?' in tmp:
                # 有些 url 内包含'?',不能作为文件路径,应该去掉'?'之后的部分
                path = tmp[:tmp.index('?')]    # path 为 cache 文件的路径
            else:
                path = tmp
            hostname = path[:path.index('/')]   # 提取主机名
```

```
            if path[-1] == '/':   # 若 path 的最后一个字符为'/'则删去
                path = path[:-1]
        else:
            # url 为空时关闭线程
            print("url is null")
            client_socket.close()
            return

        # 判断 url 是否被过滤
        if self.filter_page(hostname):
            print('request to connect to ' + str(hostname) + ' is
rejected\n')
            client_socket.close()
            return

        # 判断 ip 地址是否被过滤
        if self.filter_user(address[0]):
            print('user '+address[0]+' is forbidden to connect to proxy
server')
            client_socket.close()
            return

        # 判断是否访问钓鱼网站
        if self.fish(hostname):
            # 将客户引导到钓鱼网站
            new_hostname = 'today.hit.edu.cn'  # 钓鱼网站的主机名
            print('request to ' + hostname + ' is redirected to ' +
new_hostname)
            begin = message.index('h')
            end = message.index('/', 12)   # 找到 url 中主机名的部分
            message = message[:begin+7] + new_hostname + '/' +
message[end+1:]   # 将主机名部分修改为新的主机名
            server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            server_socket.connect((new_hostname, 80))   # 与钓鱼网站的
服务器建立连接
            server_socket.sendall(message.encode())    # 将请求报文发送
给钓鱼网站服务器
            while True:
                # 从服务器接收数据,转发给客户端
                buff = server_socket.recv(self.BUFFER_SIZE)
                if not buff:
                    server_socket.close()
                    break
```

```
                client_socket.sendall(buff)
            client_socket.close()
            server_socket.close()
            return

        '''实现缓存功能'''
        cache_path = self.cache_dir + path.replace('/', '_')  # 将路径中的'/'换成'_', 因为 windows 文件名不能带'/'
        is_modified = False   # 将是否修改过的 flag 设为 false
        if os.path.exists(cache_path):
            # 若缓存文件存在 则判断服务器端是否修改过这个网页
            modified_time = os.stat(cache_path).st_mtime  # 获得缓存最后一次修改的时间
            headers = str('If-Modified-Since: '+time.strftime('%a, %d %b %Y %H:%M:%S GMT', time.gmtime(modified_time)))
            # 把 modified-time 按报文要求的格式格式化
            message = message[:-2] + headers + '\r\n\r\n'  # 把 If-Modified-Since 字段加入到请求报文中, 注意 http 用\r\n\r\n 判断请求头部结束

            # 向服务器发送该请求
            server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            server_socket.connect((hostname, 80))
            server_socket.sendall(message.encode())
            data = server_socket.recv(self.BUFFER_SIZE).decode('utf-8', 'ignore')
            server_socket.close()

            if data[9:12] == '304':
                # 若收到的响应代码为 304,则说明服务器未修改该网页,故从缓存读取数据交给客户端
                print("Read from cache")
                with open(cache_path, "rb") as f:
                    client_socket.sendall(f.read())
            else:
                # 否则说明网站该过该网页,将 is_modified 设为 True
                is_modified = True

        if not os.path.exists(cache_path) or is_modified:
            # 若不存在该缓存文件或服务器修改过该网页,则需要向服务器请求访问
            # 连接服务器端
            server_socket = socket.socket(socket.AF_INET,
```

```
socket.SOCK_STREAM)
            server_socket.connect((hostname, 80))
            server_socket.sendall(message.encode())
            f = open(cache_path, 'wb')
            while True:
                buff = server_socket.recv(self.BUFFER_SIZE)
                if not buff:
                    f.close()
                    server_socket.close()
                    break
                f.write(buff)  # 将接收到的数据写入缓存
                client_socket.sendall(buff)  # 将接收到的数据转发给客户
端
            client_socket.close()


def main():
    proxy = ProxyServer()  # 初始化代理服务器
    while True:
        # 进入循环，监听 10240 端口，收到客户端的请求则调用 accept 并创建一
个线程，调用 proxy.connect()对请求进行处理
        new_sock, address = proxy.proxy_socket.accept()
        print(address)
        threading.Thread(target=proxy.connect, args=(new_sock,
address)).start()


if __name__ == '__main__':
    main()
```

四、实验心得

通过本次实验，我熟悉了 socket 网络编程的流程、各个 socket 函数的使用方法和作用以及 HTTP 代理服务器的工作原理以及主要功能，并且通过实际操作对 HTTP 请求报文的格式和内容有了更清楚的了解。特别是在给头部添加 If-Modified-Since 信息时，需要观察 HTTP 头部格部分如何划分、以及了解 HTTP 协议如何判断头部的结束。

在实现网站引导功能时，我的直觉是把请求的 url 全都换成导向网站的 url 就可以，但是发现这样只能加载出来文字。后来通过比较引导前后的请求报文，以将 jwes.hit.edu.cn 引导到 today.hit.edu.cn 为例，我发现引导前一些请求出现了 jwes.hit.edu.cn/...today1、hit 等字样，而引导后全都是 http://today.hit.edu.cn/。因此我尝试找到 url 中的主机名部分，仅把主机名修改为导向网站的主机名，而不能把请求的整个 url 全修改。这样后续的图片、样式等内容果然成功加载了出来。

五、参考资料

[1]. https://tools.ietf.org/html/rfc2616