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	张明远		院系	计算机科学与技术	
班级	2003104		学号	120L030501	
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	2022.10.17	
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：

(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。（选作内容，加分项目，可以当堂完成或课下完成）
(3) 扩展 HTTP 代理服务器，支持如下功能： （选作内容，加分项目，可以当堂完成或课下完成）

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：

一、浏览器使用代理

为了使电脑访问网址时通过代理服务器，使用快捷键 Windows + I 打开 Windows 10 设置中心、打开网络和 Internet、转到代理、使用代理服务器、输入地址为 127.0.0.1，端口号为 8080，保存。

几点说明

(1) 设置的目的：确保本机的 HTTP 服务请求都是通过设置的代理服务器进行相应的处理的。同时该设置不依赖于具体的浏览器，设置完成后，通过不同的浏览器访问不同的 URL，都会通过该代理服务器的实现；
(2) 设置代理服务器的地址为 127.0.0.1：127.X.X.X 是本地环回地址，用于本地软件环回测试，因此这里设置 127.0.0.1 即可表示本机将所有的请求

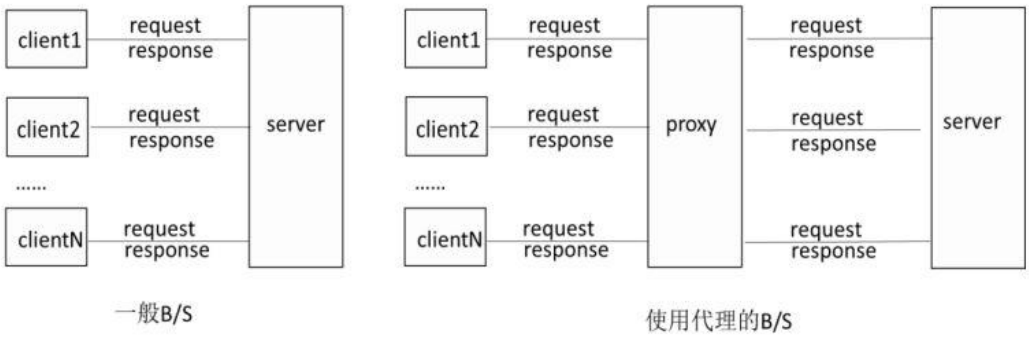
发往 127.0.0.1 代理服务器即本机；

(3) 设置监听端口号为 8080：可以设置为公认端口号之外的任何未被使用的端口号，用于不停监听来自本机的网络请求。

二、代理服务器

1) 代理服务器的概念

代理服务器，俗称“翻墙软件”，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。普通 Web 应用通信方式与采用代理服务器的通信方式的对比如下图所示：



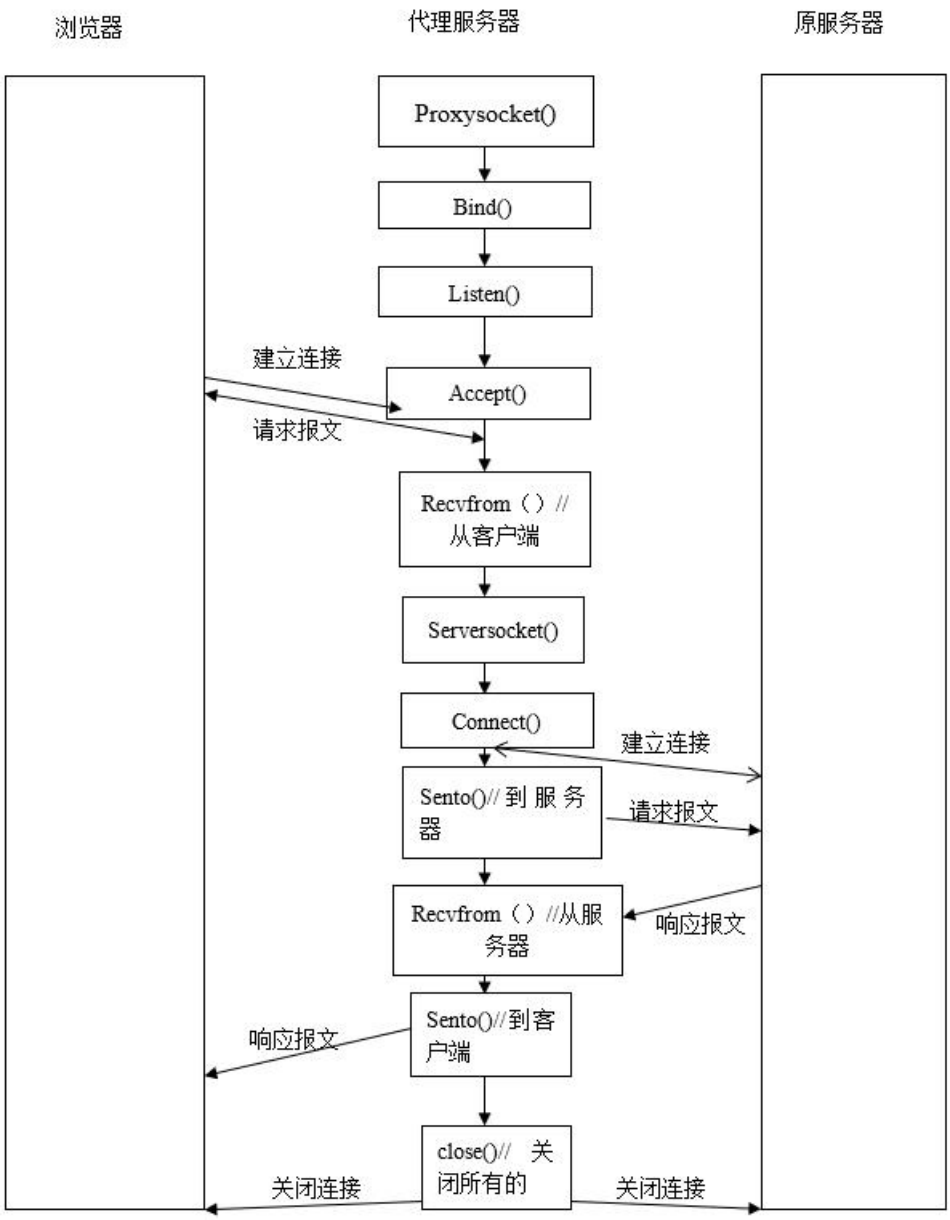
代理服务器可以认为是TCP/IP网络应用的客户端和服务器端的结合。一方面，它是浏览器客户端的服务器端，另一方面，它也是目标服务器的客户端。浏览器将请求报文发送给代理服务器，代理服务器经过一些处理或者不经过处理，将请求报文转发给目标服务器；目标服务器相应请求报文发出响应报文，代理服务器接受到响应报文之后直接将响应报文转发给浏览器客户端。

代理服务器在指定端口（本实验指定8080端口）监听浏览器的访问请求（需要在客户端浏览器进行相应的设置），代理服务器接收到浏览器对远程网站的浏览请求时，首先会查看浏览器来源的IP地址，如果属于被限制的用户，则认为没有接受到访问请求（用户过滤功能）。否则，查看其请求的host主机，如果属于不允许访问的主机，则默认不向目标服务器发送请求（网站过滤功能）；如果属于被引导的网站，则对该网站的请求报文中的host主机地址和url进行更改（网站引导功能）。而对于Cache功能的实现，基本可以概括为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），若找到对象文件，则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。若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进行清理。

除此之外，本实验要设计的服务器属于多用户代理服务器。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，服务结束后子

线程终止。与此同时，主线程继续接受下一个客户的代理服务。

HTTP代理服务器的流程图如下：



2) Socket 编程 TCP 客户端与软件端的流程

(a) TCP客户端软件流程

1. 根据目标服务器IP地址与端口号创建套接字（socket），
2. 连接服务器（connect）：三次握手
3. 发送请求报文（send）
4. 接收返回报文（recv），返回3或者5
5. 关闭连接（closesocket）

(b) TCP服务器端软件流程

1. 创建套接字（socket），绑定套接字的本地IP地址和端口号（bind），然后转到监听模式并设置连接请求队列大小（listen）。

2. 从连接请求队列中取出一个连接请求，并同意连接（accept）。在TCP连接过程中进行了三次握手。

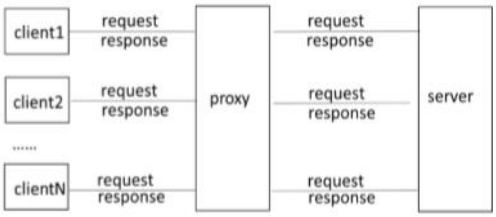
3. 收到请求报文（recv）

4. 发送数据（send）返回3或者5

5. 关闭连接（closesocket）返回2

3) 设计并实现一个基本 HTTP 代理服务器

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。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。



代理服务器作为一个并发的面向连接的服务器的基本流程如下：主线程创建一个主套接字，并绑定熟知端口号，不断监听来自客户端（本机）的请求；接受来自客户端的请求并为其建立一个新的线程，在该线程中，建立一个与之通信的套接字，用于接收客户端的请求以及转发来自服务器端的响应报文到客户端。

主线程1: 创建（主）套接字，并绑定熟知端口号；
主线程2: 设置（主）套接字为被动监听模式，准备用于服务器；
主线程3: 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；
子线程1: 接收一个客户的服务请求（通过新创建的套接字）；
子线程2: 遵循应用层协议与特定客户进行交互；
子线程3: 关闭/释放连接并退出（线程终止）.

代码实现（服务器主线程）：

```
def server_main(ip, port):
    # 服务器初始化
    # 初始化线程池
    pool = ThreadPoolExecutor(max_workers=100)
    # 设置服务器socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((ip, port))
    server_socket.listen(5)
    # 打印提示信息
    print(time.ctime() + ":Server Start")
    print(time.ctime() + ":Server Listening Port {}".format(port))
    print(time.ctime() + ":Waiting for requests...")
    # 主循环
    while True:
        # 接受HTTP请求
        client_socket, address = server_socket.accept()
        # 使用线程池中的线程处理HTTP请求
        pool.submit(sub_thread, client_socket, pool, address)
```

代理服务器作为一个面向连接的客户端的基本流程如下：通过上一步建立的新的线程与服务器端进行通信。代理服务器在这时，作为一个客户机与目标服务器建立连接请求，并转发来自客户端的 HTTP 请求，同时接收来自服务器的响应报文，将其通过上一步建立的与客户端通信的 socket 与客户端进行转发，完成客户端和服务器的一次通信。

代码实现（客户端主线程）：

```
def sub_thread(client_socket, pool, address):
    # 设置时间限制
    time_limit = 60
```

```
            client_socket.settimeout(time_limit)
            # 接受报文
            header = receive_header(client_socket)
            # 无视空报文
            if not header:
                client_socket.close()
                return
            # 解析报文内容
            url, port, method = analyse_header(header)
            # 不允许访问某些网站
            for key in url.split('.'):
                if key in limited_web.keys():
                    return
            # 不支持某些用户访问外部网站
            if address[0] in limited_user.keys():
                return
            # 打印客户端请求详情
            if method != 'CONNECT':
                print(
                    time.ctime() + ':' + threading.current_thread().name + ' ' + method + '
request to ' + url + ':' + str(
                        port))
                print(header)
            # 建立代理服务器和客户端目标服务器连接socket
            transform_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            try:
                # 连接目标服务器
                transform_socket.connect((url, port))
                transform_socket.settimeout(time_limit)
                # 钓鱼
                if url in fished_web.keys():
                    data = b'HTTP/1.1 302 Moved Temporarily\r\nLocation:
'+b'http://today.hit.edu.cn/'+b'\r\n\r\n'
                    client_socket.sendall(data)
                    print('已钓鱼到：http://today.hit.edu.cn/')
                # https代理服务器隧道连接报文
                if method == 'CONNECT':
                    # 完成代理服务器隧道连接
                    data = b"HTTP/1.0 200 Connection Established\r\n\r\n"
                    client_socket.sendall(data)
                    # Web隧道盲转发
                    pool.submit(exchange, client_socket, transform_socket)
                    pool.submit(exchange, transform_socket, client_socket)
                # 其他报文
```

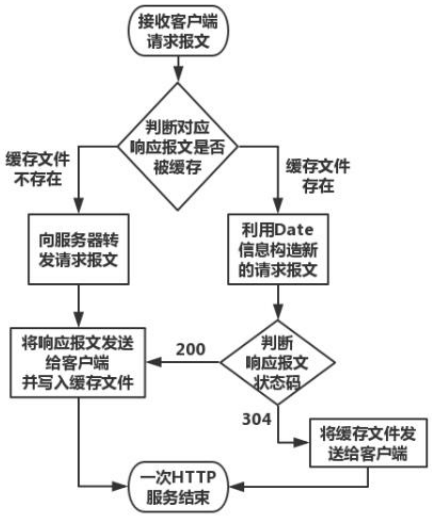
```
            else:
                # 传递报文
                pool.submit(http_solve, transform_socket, client_socket, header, method)
                #http_solve(transform_socket, client_socket, header, method)

        except Exception:
            # 关闭socket
            transform_socket.close()
            client_socket.close()
```

---

4) 实现代理服务器缓存功能

代理服务器进行本地缓存：代理服务器要实现缓存功能，就需要将服务器的响应报文缓存在本地（代理服务器）文件中，并根据客户端的需要进行查询历史文件的缓存信息，并判断是否直接返回缓存文件中的信息作为响应报文。基本判断逻辑如下：代理服务器收到客户端的请求报文，客户端根据请求报文的头部信息，在代理服务器的缓存文件中进行检索；若找不到该文件，说明未被缓存，则直接向服务器递交请求；若找到该文件，则提取该缓存文件中的Date 信息，并重新构造请求报文，在请求报文头部结尾添加行：if-modified-since Date（这里的 Date 指的是提取的响应报文中的 Date 后面的日期）。注意：不可以直接将该行添加在头部第一行，否则会请求报文错误，因为头部行第一行是请求行。发送的修改过的请求报文被服务器接收后，服务器会发送响应报文。若响应报文头部行中含有 304 Not Modified，则说明我们的缓存信息是可用的，未被更新的，所以我们直接将缓存文件中的响应报文发送给客户端；若头部行中含有200 OK，则说明请求信息已被更新，我们需要将服务器发送的响应报文转发给客户端（因为这时候响应报文包含相应的请求响应数据），同时将该响应缓存更新到本地缓存文件中。功能实现逻辑如下图



代码实现（缓存功能）：

```
    # 检查是否为最新的cache
```

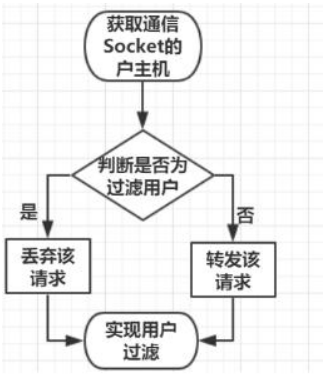
```
def check_cache(header, cache_header, url):
    headers_title = get_title(header)
    last_time = str(cache_header.Last_modified[1:], encoding="utf-8")
    headers_title['If-Modified-Since'] = last_time
    headers_title['Cache-Control'] = ''
    s = requests.get(url, headers=headers_title)
    if s.headers['Date'][:-6] == last_time[:-6]:
        return True
    else:
        return False


def http_solve(transform_socket, client_socket, header, method):
    url = header.split(b'\r\n')[0].split(b' ')[1]
    # 有cache的情况
    cache_header = header_cache.get(url)
    if method == 'GET':
        if cache_header is not None:
            # 检查是否为最新的cache
            if check_cache(header, cache_header, url):
                client_socket.sendall(cache_header.data)
                # 显示缓存数
                print('本地cache发送')
                print("cache大小:{}".format(len(header_cache)))
                return
    # 无cache的情况
    transform_socket.sendall(header)
    cache = b''
    try:
        while 1:
            data = transform_socket.recv(1024)
            cache += data
            if not data:
                break
            client_socket.sendall(data)
    except:
        return
    finally:
        if method == "GET" and cache:
            time_line = get_cache_time_line(cache)
            header_cache[url] = Cached_File(cache, time_line)
            print('cached:\nurl:' + str(cache) + '\ntimeline:' + str(time_line))
        # 显示缓存数
        print("cache大小:{}".format(len(header_cache)))
```

---

5) 实现用户、网页过滤、钓鱼

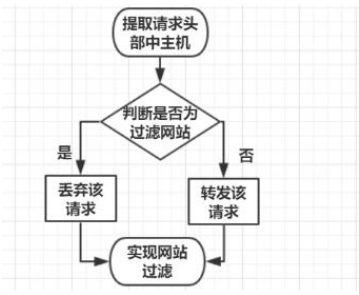
用户过滤：通过配置我的代理服务器的过滤文件，代理服务器根据与客户端通信的套接字获取客户端的主机，若是过滤文件中指明的要被过滤掉的用户，则直接丢弃客户的请求报文，不再向服务器发送请求，以此达到过滤用户目的。处理逻辑流程图如下。



代码实现（用户过滤）：

```
# 不支持某些用户访问外部网站
if address[0] in limited_user.keys():
    return
```

网页过滤：通过配置我的代理服务器过滤文件，代理服务器提取请求报文中的头部行获取目的主机，若是过滤文件中指明的要被过滤掉的主机名，则直接丢弃客户的请求报文，不再向服务器发送请求，以此达到网页过滤的目的。处理逻辑流程图如下。

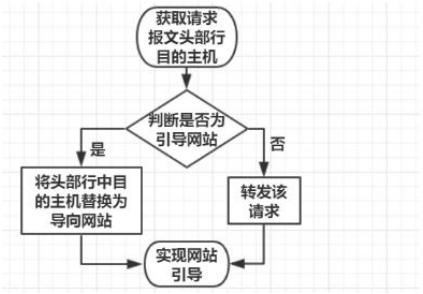


代码实现（网站过滤）：

```
# 不允许访问某些网站
for key in url.split('.'):
```

```
        if key in limited_web.keys():
            return
```

网站引导（钓鱼）：通过配置我的代理服务器的过滤文件，代理服务器提取客户端请求报文中的目的主机，若是过滤文件中指明的要被引导的网站，则将该请求报文中的头部行中的 URL 进行替换，替换为要引导向的 URL 地址，以此达到网页引导的目的。处理逻辑流程图如下



代码实现（客户端主线程）：

```
    # 钓鱼
if url in fished_web.keys():
    data = header.replace(b'jwts.hit.edu.cn',b'today.hit.edu.cn')
    socket1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    socket1.connect(('today.hit.edu.cn', 80))
    http_solve(socket1, client_socket, data, 'GET')
    print('已钓鱼到：http://today.hit.edu.cn/')
```

---

6) 实现 HTTPS 协议支持

HTTPS代理是通过Web隧道（Web tunnel）工作的，Web隧道允许用户通过HTTP连接发送非HTTP流量（例如FTP，Telnet，SMTP），这就使得那些使用非HTTP协议的应用程序可以通过HTTP代理工作了。Web隧道是用HTTP协议的CONNECT方法建立起来的。CONNECT方法不是HTTP/1.1核心规范的一部分，但确是一种得到广泛应用的扩展。客户端通过CONNECT方法请求代理服务器创建一条到达任意目的服务器和端口的TCP链接，代理服务器仅对客户端和服务器之间的后续数据进行盲转发（只是转发，不关心、也不懂发送的内容是什么）。

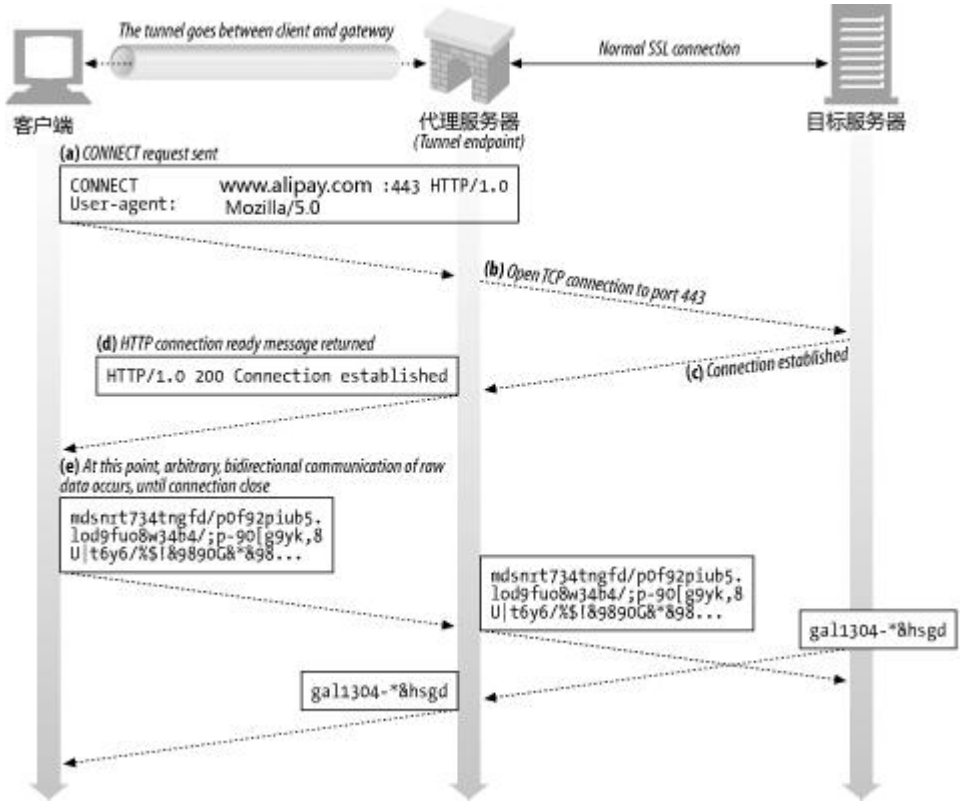
建立Web隧道的详细步骤如下：

1） 客户端通过HTTP协议发送一条CONNECT方法的请求给代理服务器，告知代理服务器需要连接的主机和端口。

2） 代理服务器一旦建立了和目标主机（上例中的www.alipay.com:443）TCP连接，就会回送一条HTTP 200 Connection Established应答给客户端。

3） 此时隧道就建立起来了。客户端通过该HTTP隧道发送的所有数据都会被代理

服务器（通过之前建立起来的与目标主机的TCP连接)原封不动的转发给目标服务器。目标服务器发送的所有数据也会被代理服务器原封不动的转发给客户端。注意：是原封不动的转发，代理服务器并不需要知道内容的含义，也不会尝试去对内容进行解析。



对代理服务器来说只要其支持Web隧道就支持HTTPS协议（和其它非HTTP协议），与证书、加密没有任何直接的关系。从上面的过程可以看出，实现Web隧道并不难，只要代理服务器能够正确处理CONNECT请求，然后对数据进行盲转发即可。从难度上看这比标准的HTTP协议代理还要简单（标准HTTP协议代理需要对客户端和服务端双方的HTTP报文先进行解析，修改后再进行转发）。

代码实现（网站过滤）：

```
if method == 'CONNECT':
    #  完成代理服务器隧道连接
    data = b"HTTP/1.0 200 Connection Established\r\n\r\n"
    client_socket.sendall(data)
    # Web隧道盲转发
    pool.submit(exchange, client_socket, transform_socket)
    pool.submit(exchange, transform_socket, client_socket)
```

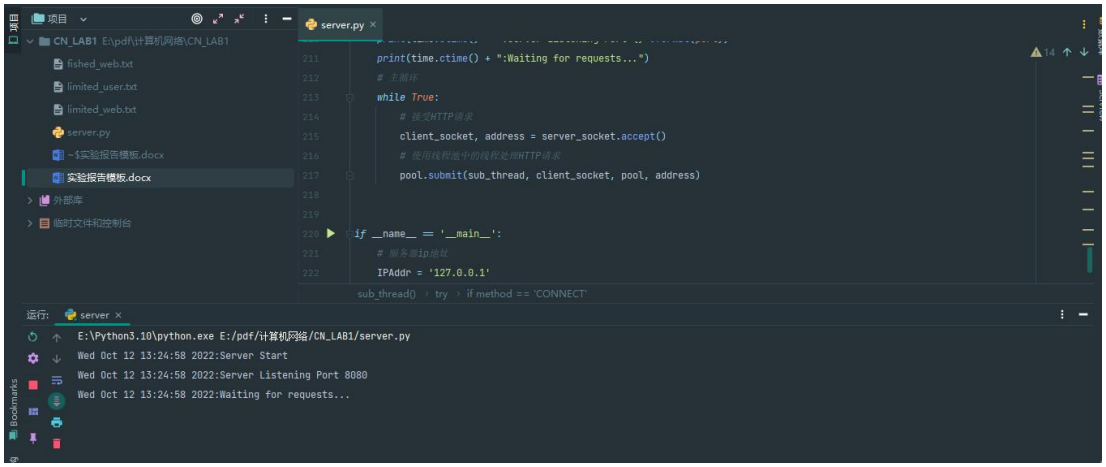
实验结果：

1. 基本 HTTP 代理服务

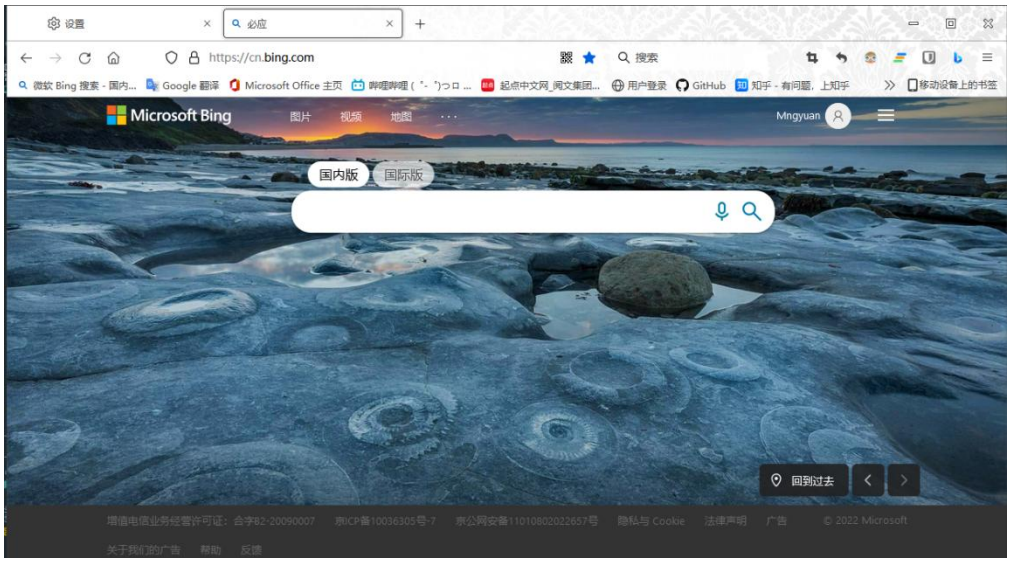
首先，HTTP代理服务器设定于ip地址127.0.0.1的8080端口（暂未打开服务器），将浏览器（使用的浏览器为firefox）的代理服务器设置为同样的地址和端口。

打开一个网页，可以看到，在代理服务器未运行时，无法正常访问网页

将代理服务器于指定端口打开



刷新网页，发现网页又可以正常访问了



这时服务器也输出了接到的相应报文请求

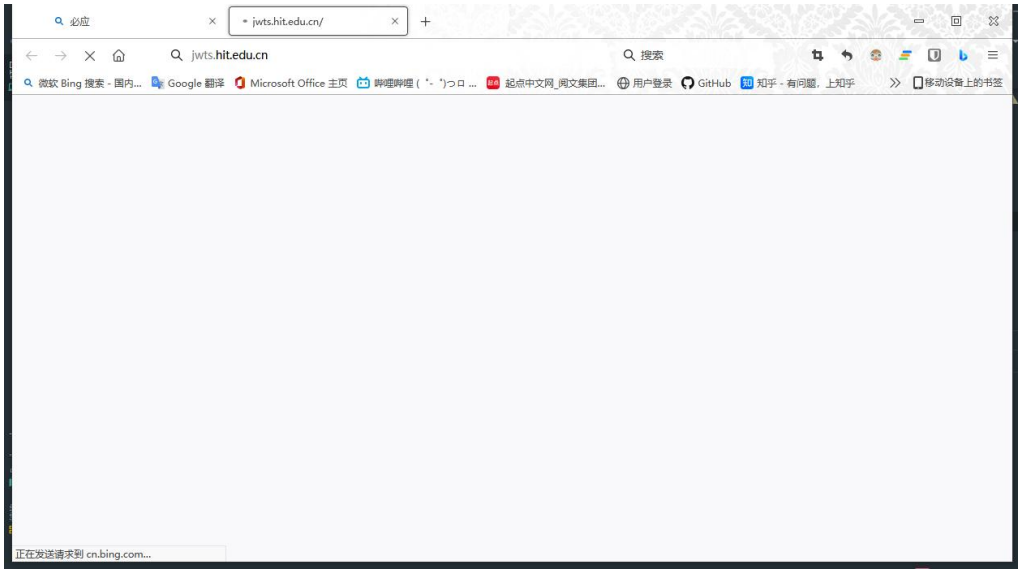
代理服务器运行成功

2. 支持 Cache 功能的 HTTP 代理服务器

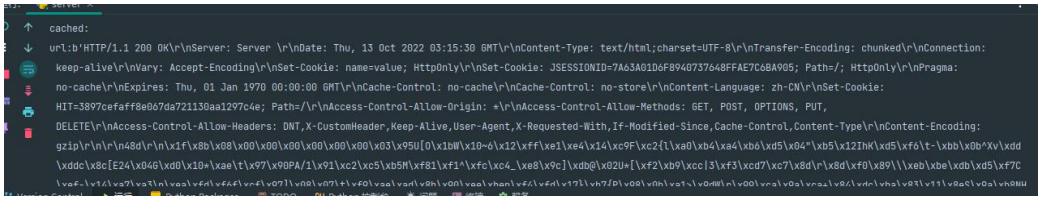
在第一次访问一个大型网站时，由于要传送的数据量大，用户需要等待一段时间，cache 功能能显著减少用户等待时间。

这里以为例。

在第一次利用代理服务器打开时，网站呈现 3~5 秒的较长时间空白



同时服务器输出已进行 cache，cache 内容为：

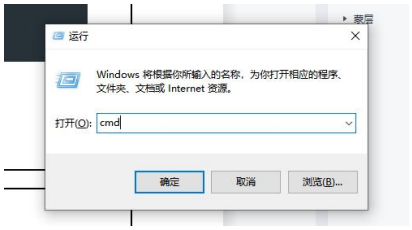


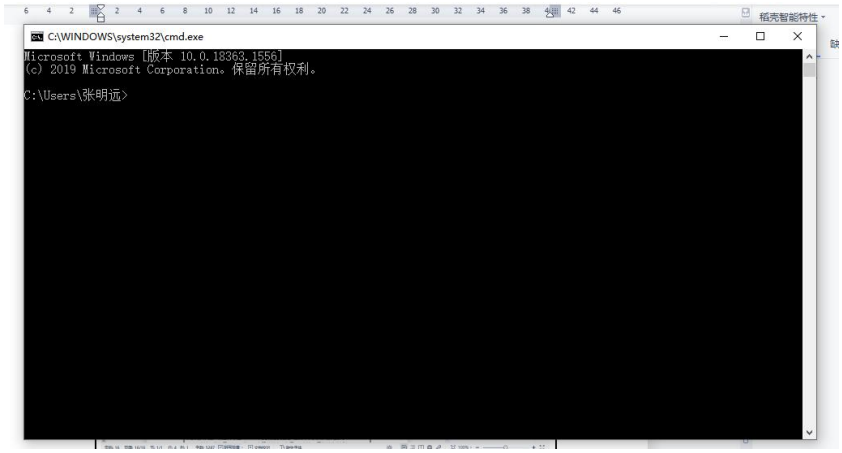
而在第二次打开时就瞬间显示了网页，可见 cache 成功

服务器显示本地 cache 发送

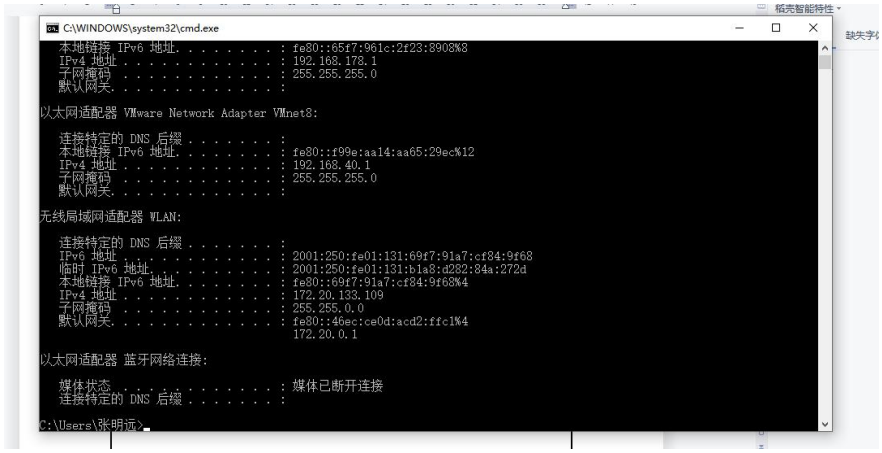
3. 限制用户访问

首先使用 win+R 调出运行窗口，输入 cmd 打开 cmd 窗口



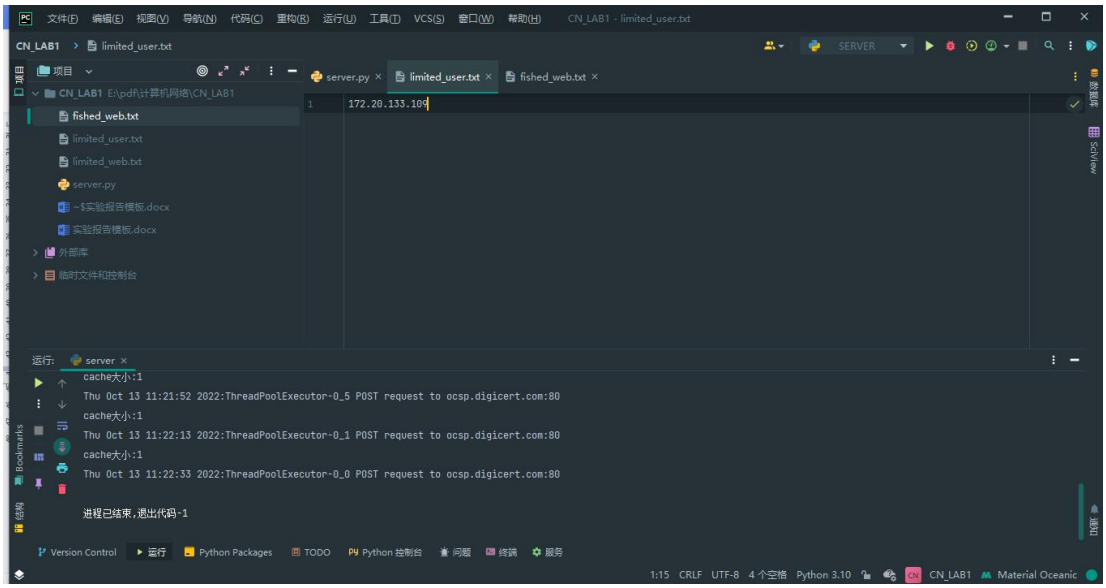


输入 ipconfig 查看自己目前 ip

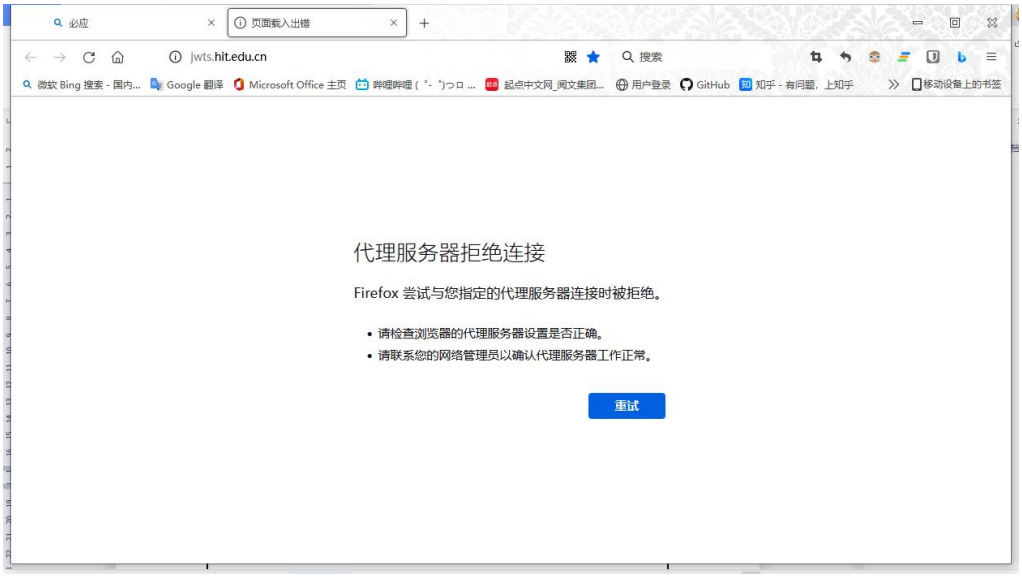


可以看到我目前使用的 wifi 的 ip 为 172.20.133.109

将其加入限制用户名单

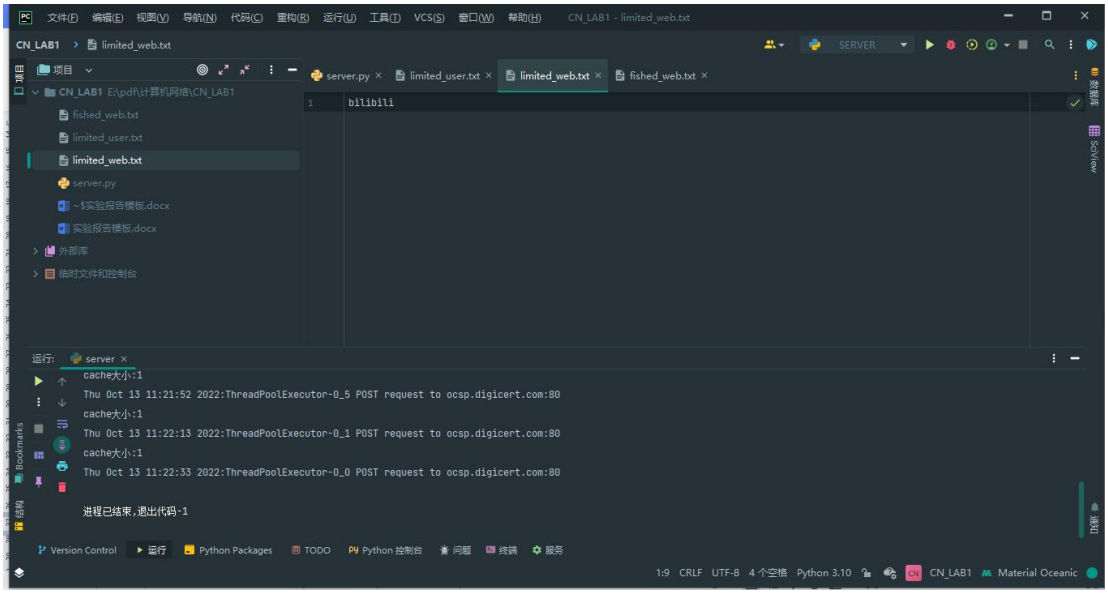


再次尝试访问网站，可以看到访问被阻止

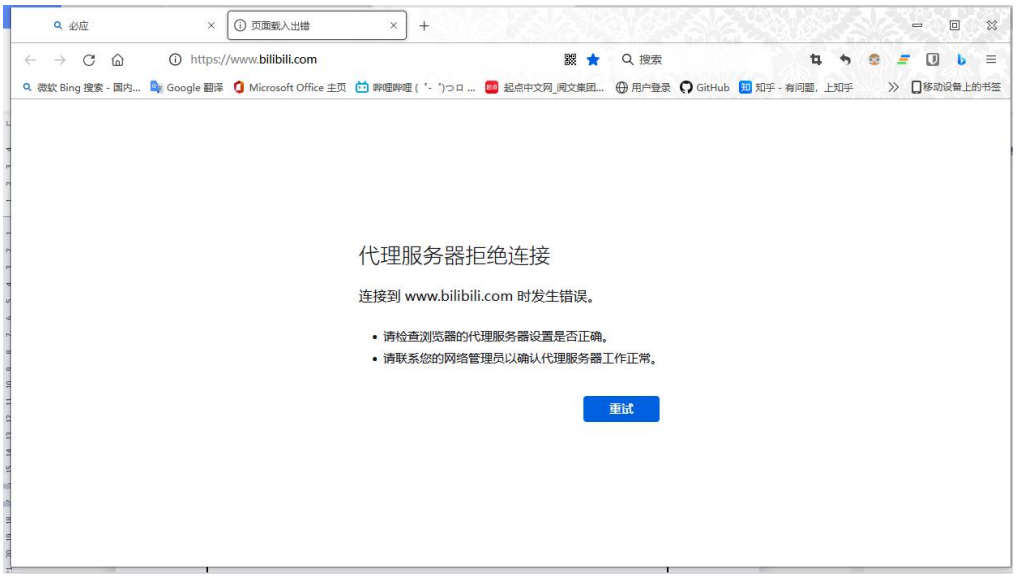


4. 限制网站访问

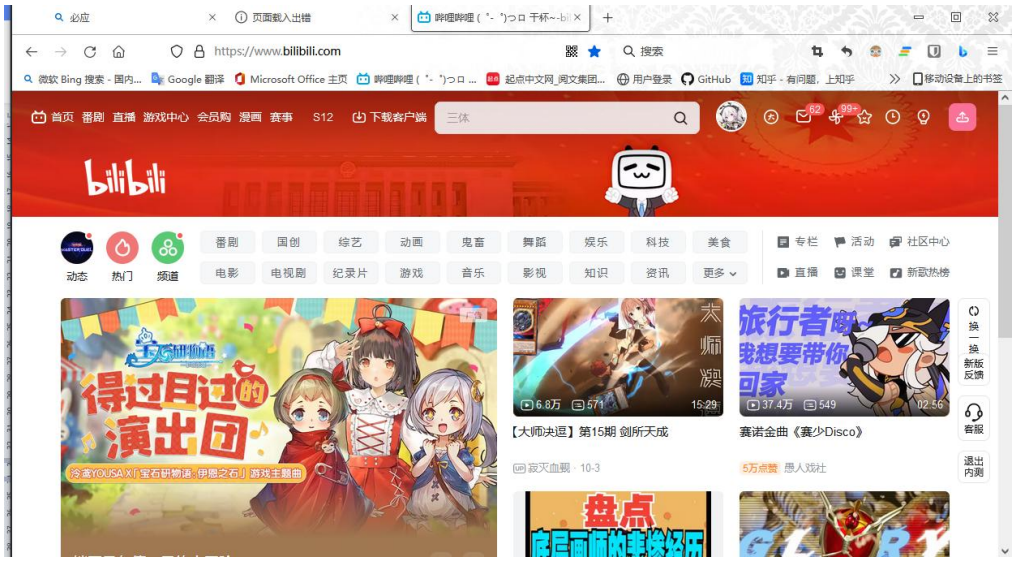
将限制 ip 清除，在限制网站中加入 bilibili



尝试访问，发现无法访问



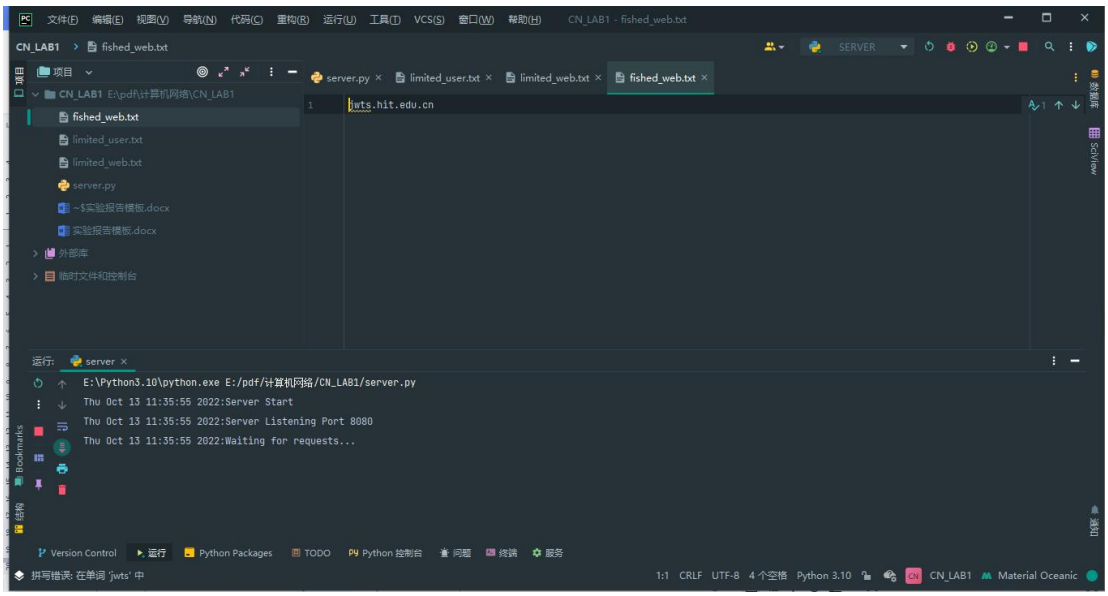
清除后又能重新访问了



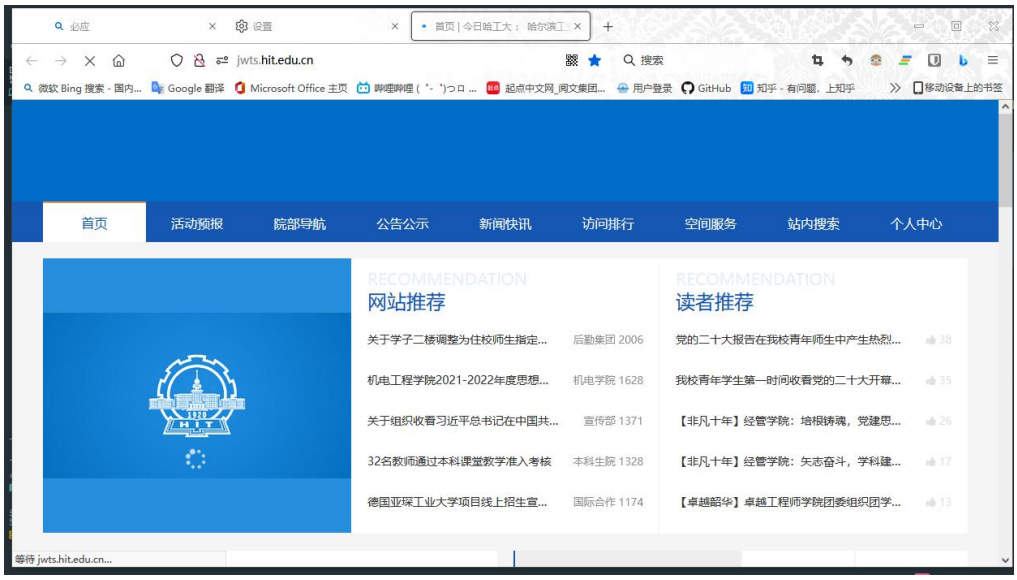
5. 钓鱼

在钓鱼未启动时，访问 jwts.hit.edu.cn，能够正常访问

在钓鱼功能中加入 jwts.hit.edu.cn



尝试访问网站，发现被引流到 http://today.hit.edu.cn/



钓鱼功能成功

问题讨论：

对于使用代理服务器访问一些网站的时候存在图片无法加载或者加载的很慢的情况，猜测是因为对于代理服务器来说，HTTP发送报文的时候是不采用流水的，因此中间需要多次发送和接受过程，而且还经过了代理服务器这一中介，可能使得速度更慢，因此在现在的HTTP报文传送中猜测大多使用了流水线形式或其他形式加速发送报文。

心得体会：掌握了 Socket 编程有关的知识结构，较为深入的理解了应用层的 HTTP 协议的相关内容。熟悉并掌握 Socket 网络编程的过程与技术；同时深入的理解了

HTTP 协议，掌握了 HTTP 代理服务器的基本工作原理，并实现了简单的的缓存功能以及过滤和引导功能，同时掌握了 HTTP 代理服务器设计与编程实现的基本技能。对这一章节的知识有了一个系统全面的了解和体会。

结合实验过程和结果给出实验的体会和收获。

(1)通过编程训练，掌握了 MOOC 中学习不到的知识，比如：代理服务器的工作原理及其实现；

(2) 同时也掌握了重点知识的细节内容，比如：HTTP 请求报文的详细格式内容和携带的对应的内容以及响应报文的详细格式和携带的对应的内容；

(3) 通过python编程，对python的多线程和线程池机制印象深刻，我在实现代理服务器的功能时，用了多次多线程和线程池机制，加深了我对python代码的认识。

附源代码：

```
import random
import socket
import threading
from concurrent.futures import ThreadPoolExecutor
import time

import requests



# cache项定义
class Cached_File:
    def __init__(self, data, Last_modified):
        self.data = data
        self.Last_modified = Last_modified



# cache
header_cache = {}
# 网站限制
limited_web = {}
# 用户限制
limited_user = {}
# 钓鱼网站
fished_web = {}
# 钓鱼导向的网站
fish_to = b'today.hit.edu.cn/'
# 读取信息
with open('limited_web.txt', 'r', encoding='utf-8') as limited_file:
    for line in limited_file:
        limited_web[line.rstrip()] = 1
with open('limited_user.txt', 'r', encoding='utf-8') as limited_file:
    for line in limited_file:
```

```
            limited_user[line.rstrip()] = 1
with open('fished_web.txt', 'r', encoding='utf-8') as limited_file:
    for line in limited_file:
        fished_web[line.rstrip()] = 1


# 接收数据
def receive_header(client_socket):
    header = b''
    try:
        while True:
            msg = client_socket.recv(4096)
            # 拼接报文
            header = b"%s%s" % (header, msg)
            if header.endswith(b'\r\n\r\n') or not msg:
                return header
    except Exception:
        return header


# 分析报文，提取出url，port和method
def analyse_header(header):
    global url
    header_split = header.split(b'\r\n')
    head_line = header_split[0].decode('utf8')
    head_line_split = head_line.split(' ')
    method = head_line_split[0]
    # 代理服务器连接请求的情况
    if method == "CONNECT":
        url = head_line_split[1]
        if ':' in url:
            url, port = url.split(':')
            port = int(port)
        else:
            # 默认端口443
            port = 443
    # 其他请求的情况
    else:
        if header == b'':
            return
        # 查找Host行
        for line in header_split:
            if line.startswith(b"Host:"):
                url = line.split(b" ")
```

```
                    if len(url) < 2:
                        continue
                    url = url[1].decode('utf8')
                    break
            if ':' in url:
                url, port = url.split(':')
                port = int(port)
            else:
                port = 80
        return url, port, method


# https的Web隧道
def exchange(server, client):
        try:
            while 1:
                data = server.recv(4096)
                if not data:
                    return
                client.sendall(data)
        except:
            pass


#  获取Last-Modified信息
def get_cache_time_line(cache):
        for line in cache.split(b'\r\n'):
            if line.startswith(b'Date:'):
                return line[5:]
        return b''


def get_title(cache_header):
        title = {}
        header = cache_header.split(b'\r\n')
        for i in range(len(header)):
            if i == 0:
                continue
            if header[i] == b'':
                continue
            [t, item] = header[i].split(b': ')
            t = str(t, encoding='utf-8')
            item = str(item, encoding='utf-8')
            title[t] = item
```

```
    return title


# 检查是否为最新的cache
def check_cache(header, cache_header, url):
    headers_title = get_title(header)
    last_time = str(cache_header.Last_modified[1:], encoding="utf-8")
    headers_title['If-Modified-Since'] = last_time
    headers_title['Cache-Control'] = ''
    s = requests.get(url, headers=headers_title)
    if s.headers['Date'][:-6] == last_time[:-6]:
        return True
    else:
        return False


def http_solve(transform_socket, client_socket, header, method):
    url = header.split(b'\r\n')[0].split(b' ')[1]
    # 有cache的情况
    cache_header = header_cache.get(url)
    if method == 'GET':
        if cache_header is not None:
            # 检查是否为最新的cache
            if check_cache(header, cache_header, url):
                client_socket.sendall(cache_header.data)
                # 显示缓存数
                print('本地cache发送')
                print("cache大小:{}".format(len(header_cache)))
                return
    # 无cache的情况
    transform_socket.sendall(header)
    cache = b''
    try:
        while 1:
            data = transform_socket.recv(1024)
            cache += data
            if not data:
                break
            client_socket.sendall(data)
    except:
        return
    finally:
        if method == "GET" and cache:
            time_line = get_cache_time_line(cache)
```

```
                header_cache[url] = Cached_File(cache, time_line)
                # print('cached:\nurl:' + str(cache) + '\ntimeline:' + str(time_line))
            # 显示缓存数
            print("cache大小:{}".format(len(header_cache)))


def sub_thread(client_socket, pool, address):
    # 设置时间限制
    time_limit = 10
    client_socket.settimeout(time_limit)
    # 接受报文
    header = receive_header(client_socket)
    # 无视空报文
    if not header:
        client_socket.close()
        return
    # 解析报文内容
    url, port, method = analyse_header(header)
    # 不允许访问某些网站
    for key in url.split('.'):
        if key in limited_web.keys():
            return
    # 不支持某些用户访问外部网站
    if address[0] in limited_user.keys():
        return
    # 打印客户端请求详情
    if method != 'CONNECT':
        print(
            time.ctime() + ':' + threading.current_thread().name + ' ' + method + ' request to ' +
url + ':' + str(
                port))
        # print(header)
    # 建立代理服务器和客户端目标服务器连接socket
    transform_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        # 连接目标服务器
        transform_socket.connect((url, port))
        transform_socket.settimeout(time_limit)
        # 钓鱼
        if url in fished_web.keys():
            data = header.replace(b'jwts.hit.edu.cn',b'today.hit.edu.cn')
            socket1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            socket1.connect(('today.hit.edu.cn', 80))
            http_solve(socket1, client_socket, data, 'GET')
```

```
                print('已钓鱼到：http://today.hit.edu.cn/')
            # https代理服务器隧道连接报文
            if method == 'CONNECT':
                # 完成代理服务器隧道连接
                data = b"HTTP/1.0 200 Connection Established\r\n\r\n"
                client_socket.sendall(data)
                # Web隧道盲转发
                pool.submit(exchange, client_socket, transform_socket)
                pool.submit(exchange, transform_socket, client_socket)
            # 其他报文
            else:
                # 传递报文
                pool.submit(http_solve, transform_socket, client_socket, header, method)
                # http_solve(transform_socket, client_socket, header, method)

        except Exception:
            # 关闭socket
            transform_socket.close()
            client_socket.close()


# 服务器主程序
def server_main(ip, port):
    # 服务器初始化
    # 初始化线程池
    pool = ThreadPoolExecutor(max_workers=100)
    # 设置服务器socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((ip, port))
    server_socket.listen(10)
    # 打印提示信息
    print(time.ctime() + ":Server Start")
    print(time.ctime() + ":Server Listening Port {}".format(port))
    print(time.ctime() + ":Waiting for requests...")
    # 主循环
    while True:
        # 接受HTTP请求
        client_socket, address = server_socket.accept()
        # 使用线程池中的线程处理HTTP请求
        # sub_thread(client_socket, pool, address)
        pool.submit(sub_thread, client_socket, pool, address)
```

```
if __name__ == '__main__':
    # 服务器ip地址
    IPAddr = '127.0.0.1'
    # 服务器端口
    Port = 8080
    # 服务器开始运行
    server_main(IPAddr, Port)
```