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	刘璟烁		院系	计算机科学与技术学院		
班级			学号			
任课教师			指导教师			
实验地点			实验时间			
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。

(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）。

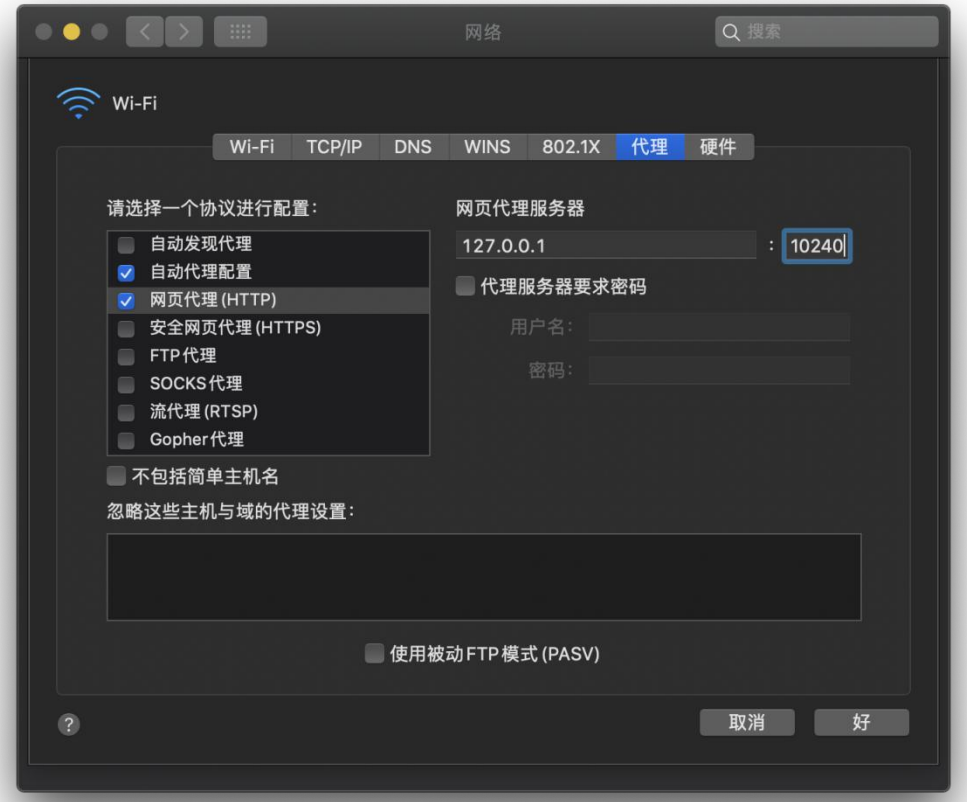
实验过程：

一. 浏览器使用代理（macOS环境下）

在本实验中，我们希望浏览器访问网址时通过代理服务器，因此需要对网络进行相关设置，以 macOS 中对 http 代理进行配置的过程为例，实验具体过程如下：

首先，打开系统偏好设置中的“网络”一栏：

随后选择右下角的“高级”选项，在其中的“代理”部分勾选http代理：

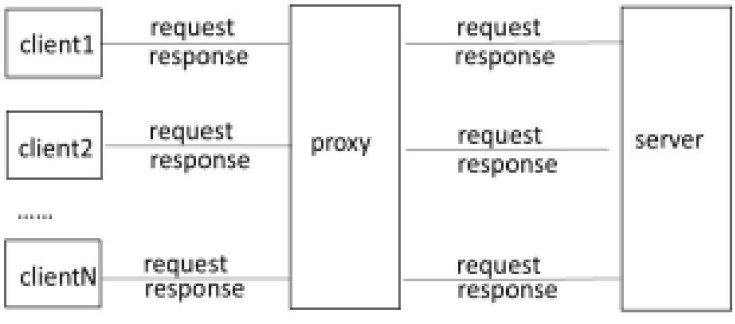


我们可以看到，设置网页代理服务器需要给定代理服务器的ip以及端口号。本次试验中，我们用主机巡回地址“127.0.0.1”作为代理服务器的ip，并随意设置一个公认端口号之外的未被选用的端口号，比如实验指导中选用的“10240”。

二. http代理服务器程序实现过程

1.基本原理：

(1).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。借用实验中的图示：



使用代理的B/S

为完成这一功能，首先要创建 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。与此同时，子线程对客户端发送的报文进行相应的处理（解析头部，头部信息修改），并根据缓存文件hit与否决定是否将其转发给目标服务器端；向目标服务器发送请求报文后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。

(2).cache基本原理：

代理服务器通过将服务器的曾经发送的响应报文缓存在代理服务器存储文件中，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。

缓存通常是基于键值对来缓存的，键通过hash计算后，存放于内存某个空间，所以键可以理解为索引。而值是存放在内存空间或是磁盘空间上。当用户的用户请求送达至Web服务器，Web服务器会对URL进行hash计算，然后比对缓存（hash表）中的键。如若命中，则根据与之对应的值找到数据存放的位置（这里的值可以理解为指针，指着对应数据存放的位置），从而获取到缓存的结果。

如果缓存命中，代理服务器在对应内存或硬盘空间上找到对应的内容数据，构建成响应报文，直接返回给客户端。具体的，代理服务器向目标服务器发送条件GET报文并等待响应，若响应报文的首部行状态码为304则说明无需更改缓存文件，直接发送给客户端即可；如果状态码为200，则需要进行更改缓存文件并转发报文给客户端。

如果缓存不明中，代理服务器会自行封装成请求报文，把自己当做http的客户端，向上游服务器发起请求。若内容存在，上游服务器会构建成响应报文，返回给代理服务器。当代理服务器收到响应之后，会检查该对象是否可以缓存，如若可以，会对URL进行hash之后生成一个键，存放到对应的hash表中。在相应的内存或磁盘空间上存储对应的内容数据，当操作完成之后，会将数据构建成相应报文，然后响应给客户端。

状态码解析表如下：

状态码范围	状态码类别	典型状态码	原因短语	含义
100-199	信息类状态码	100	continue	收到请求出事部分，请继续发送
200-299	成功状态码	200	OK	请求已经被正常处理
		204	NO Content	请求已经被正常处理，但没有主体数据

		206	Partial Content	对资源部分请求成功
300-399	重定向状态码	301	Moved Permanently	请求的 URL 资源已被更新，响应首部包含新的 URL
		302	Fount	请求的 URL 资源临时更新
		304	Not Modified	请求不符合条件，需要更改条件
400-499	客户端错误状态码	400	Bad Request	客户端发送一个错误请求
		401	Unauthorized	请求需要包含通过 HTTP 认证的信息
		403	Forbidden	访问被拒绝
		404	Not Found	没有找到请求
500-599	服务器错误状态码	500	Internal Server	服务器出现错误，无法提供请求的资源

			Error	

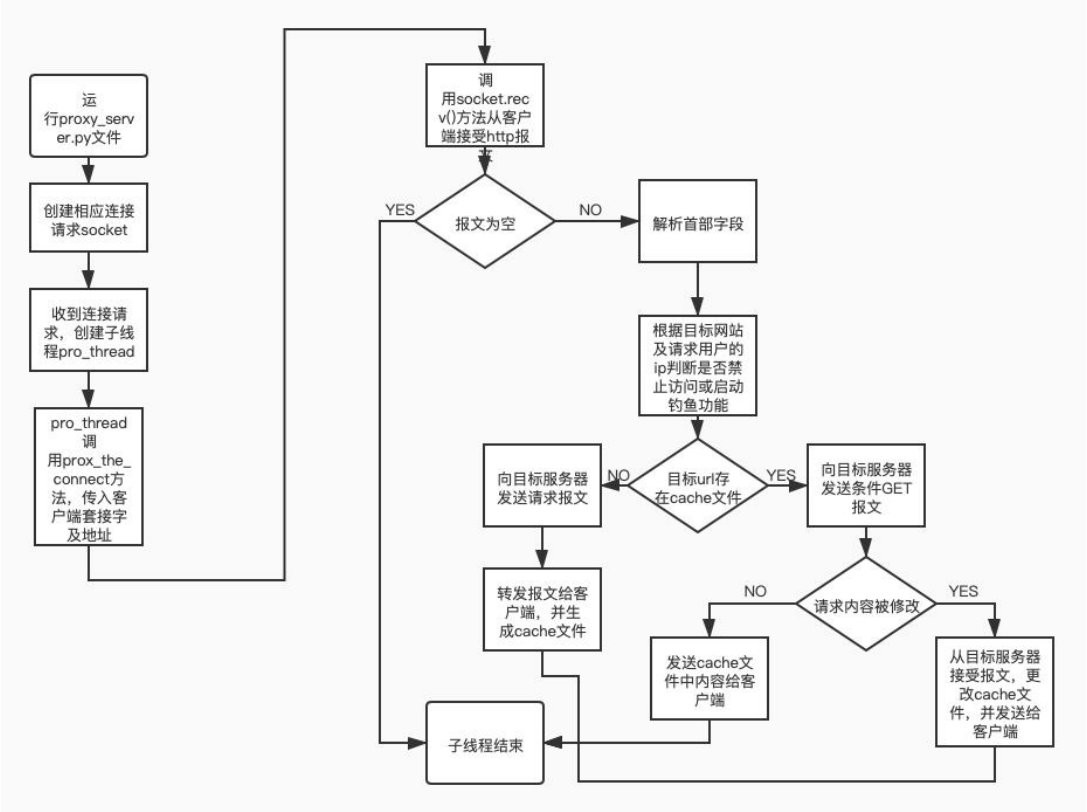
(3).过滤特定网站、用户，以及钓鱼功能的原理

a.过滤特定用户：在源代码中指定需要过滤用户的 ip 地址列表，当新的客户端发起连接请求时，代理服务器根据接受用户连接请求的主套接字获取客户端的地址。若是过滤列表中指明的要被过滤掉的用户，则向用户发送“HTTP/1.1 403 Forbidden\r\n”报文并关闭与该客户端的连接，从而过滤该用户。

b.过滤特定网站：类似前者，首先在源代码中指定需要过滤网站的 hostname 列表，客户端发送请求报文后，代理服务器根据请求报文首部字段中 hostname 获取要访问的主机，若是过滤列表中指明的要被过滤掉的主机，则向用户发送“HTTP/1.1 403 Forbidden\r\n”报文并关闭与该客户端的连接，从而过滤相应网站。

c.指定需要执行钓鱼功能的的 hostname 与所需要引导到的 hostname 构成的字典集合，客户端发送请求报文后，代理服务器根据请求报文首部字段中 hostname 获取要访问的主机，若是字典的键中指明的要被重新引导的主机，则更改用户发送报文段的 url 至相应主机。

2.程序流程图：



3.http代理服务器具体实现过程：

(1).代理服务器端创建主套接字并接受请求

代理服务器为客户端实现代理，首先需要创建一个套接字用于接受客户端的连接请求，如下，创建一个套接字并绑定至指定ip+端口（127.0.0.1:10240），接听客户的连接请求，每当有新的客户连接请求到来时创建一个新的子线程并执行proxy_server()方法执行代理功能：

```
# 生成套接字，绑定至本地指定端口并开始 listen

serv_ip = '127.0.0.1'  # 本地巡回 ip

serv_port = 10240

soc = socket(AF_INET, SOCK_STREAM)

soc.bind((serv_ip, serv_port))

soc.listen(30)  # 最多允许同时接听 30 个请求连接

try:

    while True:

        # 接受一个新的连接请求

        new_conect_soc, addr = soc.accept()

        # 创建一个字线程，并调用 prox_the_connect 函数进行代理

        pro_threa = threading.Thread(target=prox_the_connect,

args=(new_conect_soc, addr))

        pro_threa.start()

finally:

    soc.close()
```

(2)代理服务器接收并解析请求报文

接受客户端报文字节流时，需要设置超时时间（即调用settimeout()方法），否则流的末尾只有到套接字关闭的时候才会出现，这里我们设定2000ms的接收时长，并在循环中不断调用recv()方法接收报文信息，当到达2000ms后产生异常并被except捕获，停止接收；随后对字节形式报文以gbk编码方式进行解码，并解析出请求行，并调用urlparse()方法解析url:

```
new_conect_soc.settimeout(2)  # 设置两秒的

mes = new_conect_soc.recv(4096)

try:
```

```
    while True:
        rec_buf = new_conect_soc.recv(4096)
        if rec_buf:
            mes += rec_buf
        else:
            break
except:
    pass
if not mes:  # 如果报文为空 断开该连接并返回
    new_conect_soc.close()
    return
str_mes = mes.decode('gbk', 'ignore')
http_mes = str_mes.split('\r\n')
req_line = http_mes[0]  # 获取首部字段请求行
target_url = req_line.split()[1]
target_url = urllib.parse.urlparse(target_url)
target_host = target_url.hostname
```

(3).过滤用户、网站、钓鱼功能:
首先过滤特定用户，代理服务器根据接受用户连接请求的主套接字获取客户端的地址。若是过滤列表中指明的要被过滤掉的用户，则向用户发送“HTTP/1.1 403 Forbidden\r\n”报文并关闭与该客户端的连接，从而过滤该用户。
随后过滤特定网站，类似前者，代理服务器根据请求报文首部字段中 hostname 获取要访问的主机，若是过滤列表中指明的要被过滤掉的主机，则向用户发送“HTTP/1.1 403 Forbidden\r\n”报文并关闭与该客户端的连接，从而过滤相应网站。
最后处理钓鱼网站，代理服务器根据请求报文首部字段中 hostname 获取要访问的主机，若是字典的键中指明的要被重新引导的主机，则更改用户发送报文段的 url 至相应主机。

```
if addr[0] in banned_user_list:  # 判断请求用户是否在过滤列表内 若是则拒绝并关闭连接
    new_conect_soc.send(str.encode('HTTP/1.1 403 Forbidden\r\n'))
    new_conect_soc.close()
    return
if target_host in banned_web_list:  # 判断目标主机是否被禁止 处理同上
    new_conect_soc.send(str.encode('HTTP/1.1 403 Forbidden\r\n'))
    new_conect_soc.close()
    return
if target_host in fake_list.keys():  # 判断是否在钓鱼网站列表中
    temp = mes.decode().replace(req_line.split()[1],
"http://"+fake_list[target_host]+"/")  # 更改请求行 url 部分
    temp = temp.replace(target_host, fake_list[target_host])  # 更改 hostname
    http_mes = temp.split('\r\n')
    req_line = http_mes[0]
    target_url = req_line.split()[1]
    target_url = urllib.parse.urlparse(target_url)  # 重新解析 url 用于与目标 server 连接
    mes = str.encode(temp)  # 重新生成字节形式报文
```

(4).cache缓存功能实现：

首先为所有缓存文件创建文件夹，并对请求行调用加密算法进行加密生成对应缓存文件名：

```
cache_size = 1000  # 设定缓存文件最多为 1000 个
cache_dir = os.path.join(os.path.dirname(__file__), 'Cache')  # 指定缓存文件所在文件夹
if not os.path.exists(cache_dir):
    os.mkdir(cache_dir)
```

```
m = hashlib.md5()  # 使用 md5 算法加密 url 生成摘要用于命名缓存文件

m.update(str.encode(target_url.netloc + target_url.path))

filename = os.path.join(cache_dir, m.hexdigest() + '.cache')
```

随后判断该文件名对应的文件是否存在于缓存文件夹中，若存在则向目标服务器发送条件GET报文确认是否需要更新缓存内容，若目标服务器响应报文的条件码为304则不必更新，否则需要接受更新后的内容，重写cache文件并转发给客户端；若不存在cache文件，直接向目标服务器转发请求报文，接受响应报文、写cache、并转发给客户端：

```
if os.path.exists(filename):  # 如果之前已经建立过缓存

    real_ser_sock = socket(AF_INET, SOCK_STREAM)

    real_ser_sock.settimeout(120)

    real_ser_sock.connect((target_url.hostname, target_port))

    temp = req_line + '\r\n'

    t = (time.strptime(time.ctime(os.path.getmtime(filename)),  \

"%a %b %d %H:%M:%S %Y"))

    temp += 'If-Modified-Since: ' + time.strftime( \

  '%a, %d %b %Y %H:%M:%S GMT', t) + '\r\n'  # 获取缓存文件的最后修改时间并指定格

式

    for line in http_mes[1:]:

        temp += line + '\r\n'

    real_ser_sock.sendall(str.encode(temp))

    flag = True

    while True:

        data = real_ser_sock.recv(4096)
```

```
        if flag:
            if data.decode('iso-8859-1').split()[1] == '304':   # 若响应报文条件码为 304
不必更新
                print('Cache hit: {path}'.format(path=target_url.hostname +
target_url.path))
                new_conect_soc.send(open(filename, 'rb').read())
                break
            else:
                cac_file = open(filename, 'wb')   # 原服务器已经更新
                print('Cache updated: {path}'.format(path=target_url.hostname +
target_url.path))
                if len(data) > 0:
                    new_conect_soc.send(data)
                    cac_file.write(data)
                else:
                    break
                flag = False
        else:   # 前面没有 hit 则多次从原服务器读取内容，直到没有内容可读
            cac_file = open(filename, 'ab')
            if len(data) > 0:
                new_conect_soc.send(data)
                cac_file.write(data)
```

```
                else:
                    break
else:  # 否则重新从源服务器获取数据
    real_ser_sock = socket(AF_INET, SOCK_STREAM)  # 为连接至目标服务器创建套接
字

    real_ser_sock.settimeout(120)  # 设置 120s 的超时时延
    real_ser_sock.connect((target_url.hostname, target_port))  # 连接至原服务器
    real_ser_sock.sendall(mes)
    cac_file = open(filename, 'ab')
    while True:
        buf = real_ser_sock.recv(4096) # 可以加入缓存
        if len(buf) > 0:
            new_conect_soc.send(buf)
            cac_file.write(buf)
        else:
            break
```

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。

1.基本访问功能

访问哈工大主页：http://hit.edu.cn/

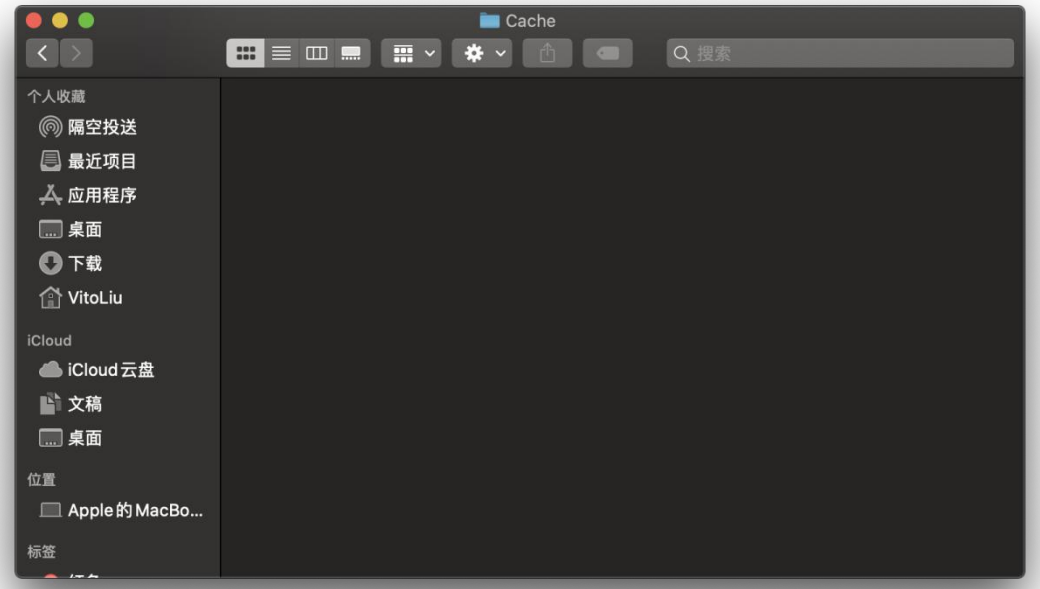
访问计算学部主页：

均可以访问成功。

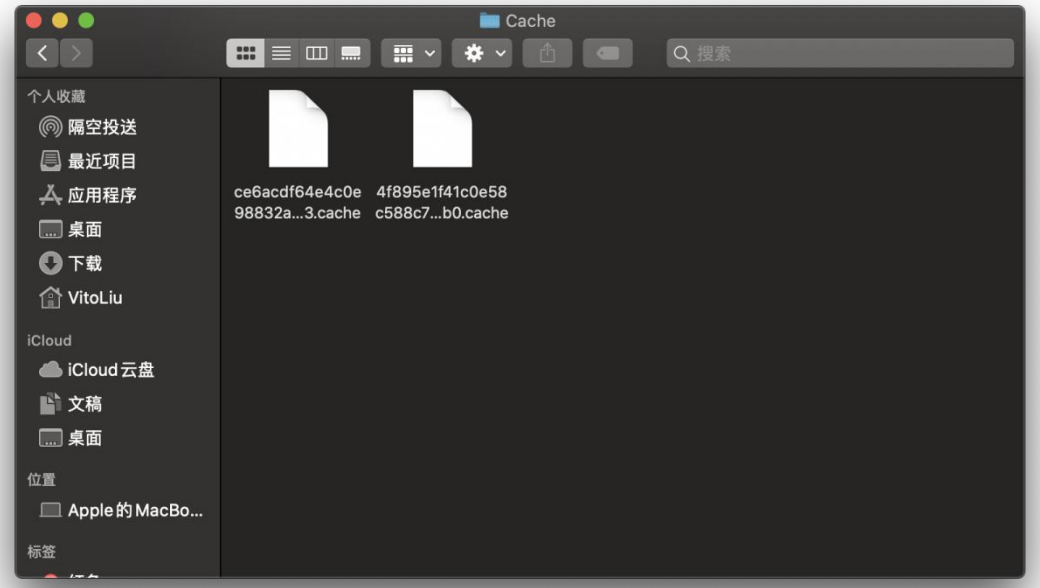
2.缓存功能

访问特定网站后，应该在Cache文件夹中创建出特定的缓存文件：

发送请求报文前，Cache文件夹为空：



访问hit.edu,cn后，生成md5加密摘要作为文件名的的缓存文件：



3.过滤用户、特定网站，钓鱼功能

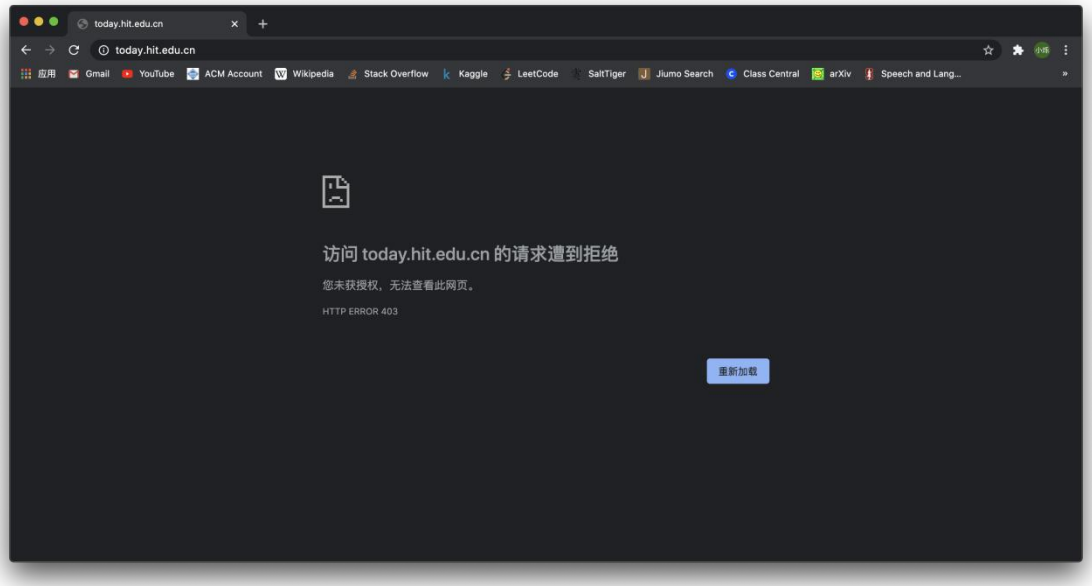
本次实验，过滤网站、用户，以及钓鱼网站如下：

```
banned_web_list = ['today.hit.edu.cn']  # 禁止网站列表

banned_user_list = []  # '127.0.0.1'  # 禁止用户列表
```

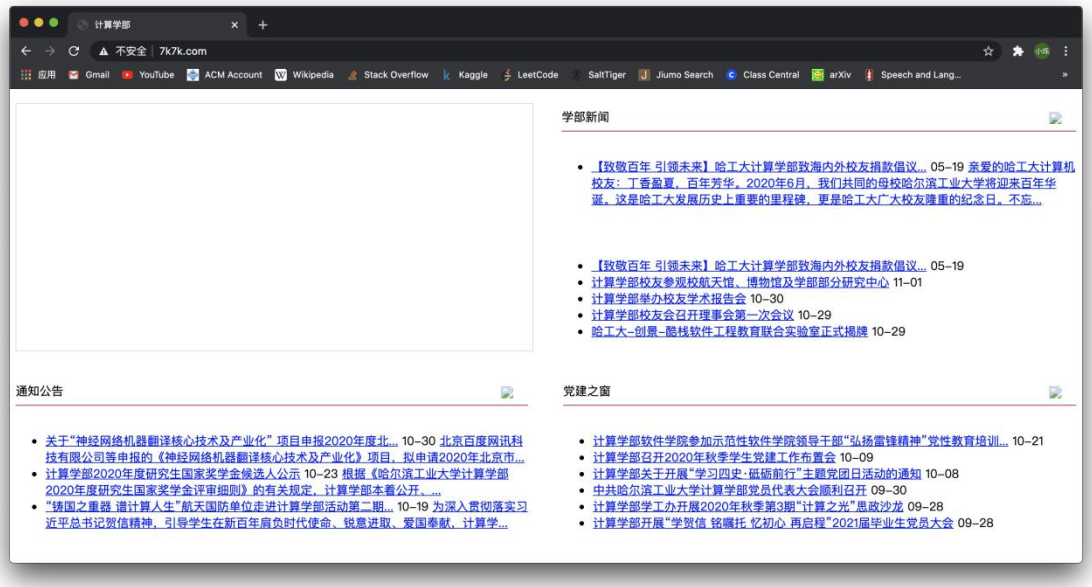
```
fake_list = {'www.7k7k.com': 'cs.hit.edu.cn'}    # 重定向网站列表
```

(1).我们首先访问today.hit.edu.cn:



可以看到收到403 Forbidden消息，被禁止访问。

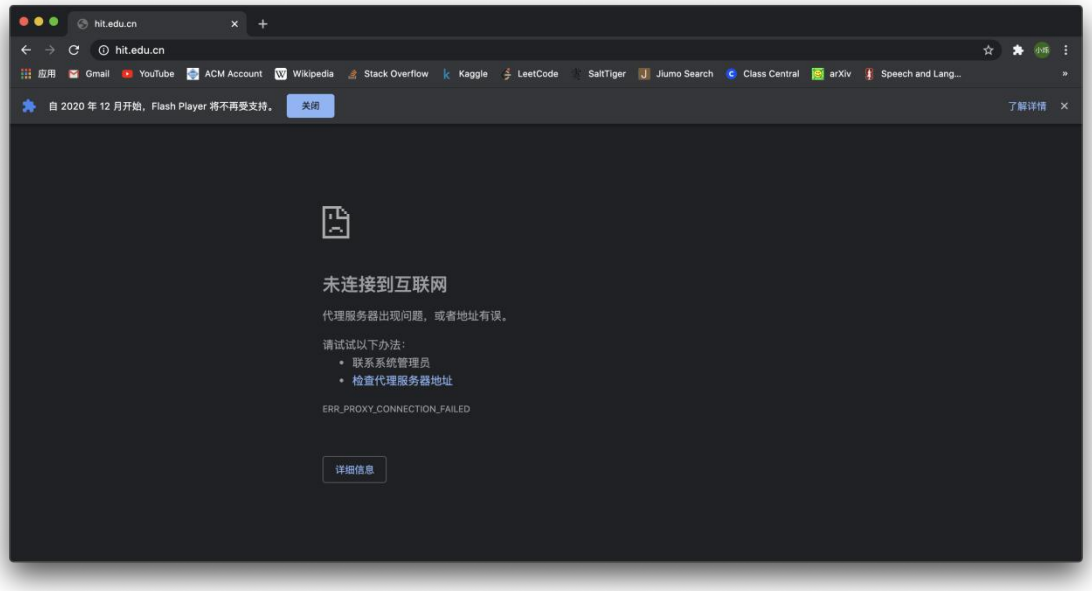
(2).访问钓鱼网站：www.7k7k.com



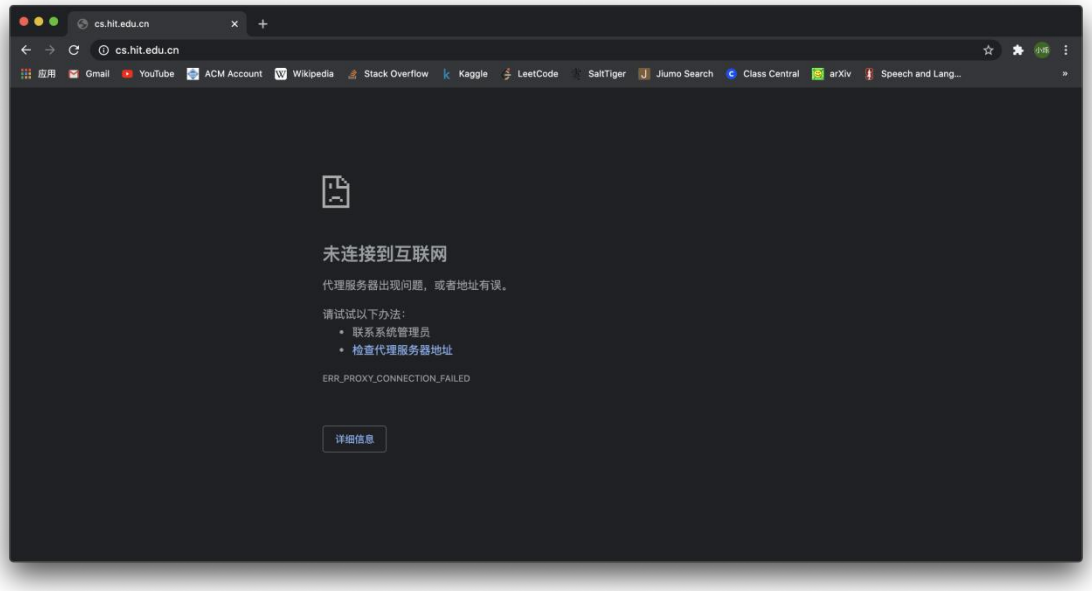
可以看到被导入到计算学部的主页中。

(3).用过滤用户127.0.0.1访问网站

访问hit.edu,cn,访问失败：



访问cs.hit.edu.cn，访问失败：



综上，所有基础功能以及附加功能均已实现。

问题讨论：

对实验过程中的思考问题进行讨论或回答。

1.如何处理recv()方法接收到的请求报文?

socket.recv()方法接收到的为连接对方发送的报文字节流，因此解析报文需要先用decode()方法将其转化为字符串的形式，并根据HTTP报文格式对其做相应的切分处理。

2.代理服务器实现缓存的原理和方法?

代理服务器通过将服务器的曾经发送的响应报文缓存在代理服务器存储文件中，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。

缓存通常是基于键值对来缓存的，键通过hash计算后，存放于内存某个空间，所以键可以理解

为索引。而值是存放在内存空间或是磁盘空间上。当用户的用户请求送达至Web服务器，Web服务器会对URL进行hash计算，然后比对缓存（hash表）中的键。如若命中，则根据与之对应的值找到数据存放的位置（这里的值可以理解为指针，指着对应数据存放的位置），从而获取到缓存的结果。
如果缓存命中，代理服务器在对应内存或硬盘空间上找到对应的内容数据，构建成响应报文，直接返回给客户端。具体的，代理服务器向目标服务器发送条件GET报文并等待响应，若响应报文的首部行状态码为304则说明无需更改缓存文件，直接发送给客户端即可；如果状态码为200，则需要进行更改缓存文件并转发报文给客户端。
如果缓存不明中，代理服务器会自行封装成请求报文，把自己当做http的客户端，向上游服务器发起请求。若内容存在，上游服务器会构建成响应报文，返回给代理服务器。当代理服务器收到响应之后，会检查该对象是否可以缓存，如若可以，会对URL进行hash之后生成一个键，存放到对应的hash表中。在相应的内存或磁盘空间上存储对应的内容数据，当操作完成之后，会将数据构建成相应报文，然后响应给客户端。

心得体会：

结合实验过程和结果给出实验的体会和收获。
课程知识方面：通过学习socket编程，更深刻的掌握socket编程有关知识结构，更加深入了理解了http协议中的通信过程，对http报文格式也有了更深刻的印象。通过实现代理服务器的缓存功能，对缓存这一技术有了更深刻的连接，并且尝试实现了过滤和钓鱼等技术。
工程实践能力方面，通过使用python语言进行socket编程，对http通讯过程中所需要的方法有了基本的掌握，加强拓宽了自己在这方面的工程实践能力。

源代码；

```
from socket import *

import threading

import urllib.parse

import os

import hashlib

import time

'''

    this is the proxy server

'''
```

```
def prox_the_connect(new_conect_soc, addr):
    """
    实现代理的方法  可被线程调度启动
    Args:
        new_conect_soc: 传入的为客户创建的新的 socket
        addr: 用户的 ip 地址
    Returns:
        none
    """

    # 从客户端接受报文
    banned_web_list = ['today.hit.edu.cn']  # 禁止网站列表
    banned_user_list = ['127.0.0.1']  #    # 禁止用户列表
    fake_list = {'www.7k7k.com': 'cs.hit.edu.cn'}    # 重定向网站列表

    cache_size = 1000    # 设定缓存文件最多为 1000 个
    cache_dir = os.path.join(os.path.dirname(__file__), 'Cache')  # 指定缓存文件所在文
件夹
    if not os.path.exists(cache_dir):
        os.mkdir(cache_dir)
```

```
new_conect_soc.settimeout(2)
mes = new_conect_soc.recv(4096)
try:
    while True:
        rec_buf = new_conect_soc.recv(4096)
        if rec_buf:
            mes += rec_buf
        else:
            break
except:
    pass
if not mes:  # 如果报文为空 断开该连接并返回
    new_conect_soc.close()
    return
str_mes = mes.decode('gbk', 'ignore')
print(str_mes)
http_mes = str_mes.split('\r\n')
req_line = http_mes[0]  # 获取首部字段请求行
if 'CONNECT' in req_line:  # 本次实验不考虑 CONNECT
    new_conect_soc.close()
    return
print(req_line)
```

```
    target_url = req_line.split()[1]
    target_url = urllib.parse.urlparse(target_url)
    target_host = target_url.hostname
    print(target_host)

    target_port = 80 if target_url.port is None else target_url.port

    if addr[0] in banned_user_list:  # 判断请求用户是否在过滤列表内 若是则拒绝并关闭
连接
        new_conect_soc.send(str.encode('HTTP/1.1 403 Forbidden\r\n'))  # 向客户端
发送拒绝访问报文
        new_conect_soc.close()
        return
    if target_host in banned_web_list:  # 判断目标主机是否被禁止 处理同上
        new_conect_soc.send(str.encode('HTTP/1.1 403 Forbidden\r\n'))  # 通上，发
送拒绝访问报文
        new_conect_soc.close()
        return
    if target_host in fake_list.keys():  # 判断是否在钓鱼网站列表中
        temp = mes.decode().replace(req_line.split()[1],
"http://"+fake_list[target_host]+"/")
```

```
            temp = temp.replace(target_host, fake_list[target_host])
            http_mes = temp.split('\r\n')
            req_line = http_mes[0]
            target_url = req_line.split()[1]
            target_url = urllib.parse.urlparse(target_url)   # 重新解析 url 用于与目标 server
连接
            mes = str.encode(temp)   # 重新生成字节形式报文



    m = hashlib.md5()
    m.update(str.encode(target_url.netloc + target_url.path))
    filename = os.path.join(cache_dir, m.hexdigest() + '.cache')
    if os.path.exists(filename):   # 如果之前已经建立过缓存
            real_ser_sock = socket(AF_INET, SOCK_STREAM)
            real_ser_sock.settimeout(120)
            real_ser_sock.connect((target_url.hostname, target_port))

            temp = req_line + '\r\n'

            t = (time.strptime(time.ctime(os.path.getmtime(filename)),   # 获取缓存文件的
最后修改时间并指定格式
                               "%a %b %d %H:%M:%S %Y"))
```

```
            temp += 'If-Modified-Since: ' + time.strftime(
                '%a, %d %b %Y %H:%M:%S GMT', t) + '\r\n'
            for line in http_mes[1:]:
                temp += line + '\r\n'
            real_ser_sock.sendall(str.encode(temp))


            flag = True
            while True:
                data = real_ser_sock.recv(4096)
                if flag:
                    if data.decode('iso-8859-1').split()[1] == '304':   # 若响应报文条件码为
304 不必更新
                        print('Cache hit: {path}'.format(path=target_url.hostname +
target_url.path))
                        new_conect_soc.send(open(filename, 'rb').read())
                        break
                    else:
                        cac_file = open(filename, 'wb')   # 原服务器已经更新
                        print('Cache updated: {path}'.format(path=target_url.hostname +
target_url.path))
                        if len(data) > 0:
                            new_conect_soc.send(data)
```

```
                        cac_file.write(data)
                    else:
                        break
                    flag = False
            else:  # 前面没有 hit 则多次从原服务器读取内容，直到没有内容可读
                cac_file = open(filename, 'ab')
                if len(data) > 0:
                    new_conect_soc.send(data)
                    cac_file.write(data)
                else:
                    break

    else:  # 否则重新从源服务器获取数据
        real_ser_sock = socket(AF_INET, SOCK_STREAM)  # 为连接至目标服务器创建
套接字
        real_ser_sock.settimeout(120)  # 设置 120s 的超时时延
        real_ser_sock.connect((target_url.hostname, target_port))  # 连接至原服务器
        real_ser_sock.sendall(mes)
        cac_file = open(filename, 'ab')
        while True:
            buf = real_ser_sock.recv(4096)
            # 可以加入缓存
```

```
            if len(buf) > 0:
                new_conect_soc.send(buf)
                cac_file.write(buf)
            else:
                break
        cac_file.close()  # 关闭缓存文件、向目标服务器的连接以及客户端的连接
        real_ser_sock.close()
        new_conect_soc.close()


cache_counter = 0
cache_files = []
for file in os.listdir(os.path.join(os.path.dirname(__file__), 'cache')):
    if file.endswith('.cache'):  # 计算 cache 文件的数量
        cache_counter += 1
        cache_files.append(file)
if cache_counter > 1000:  # 当数量超过最大限度时删除一定量 cache 文件
    for i in range(len(cache_files) - 1):
        for j in range(i + 1, len(cache_files)):
            if os.path.getmtime(cache_files[i]) < os.path.getmtime(cache_files[j]):
                temp = cache_files[i]
                cache_files[i] = cache_files[j]
                cache_files[j] = temp
```

```
            for file in cache_files[cache_size:]:
                os.remove(file)
        return



# 生成套接字，绑定至本地指定端口并开始 listen
serv_ip = '127.0.0.1'  # local ip
serv_port = 10240
soc = socket(AF_INET, SOCK_STREAM)
soc.bind((serv_ip, serv_port))
soc.listen(30)  # 最多允许同时接听 30 个请求连接


try:
    while True:
        # 接受一个新的连接请求
        new_conect_soc, addr = soc.accept()
        # 创建一个字线程，并调用 prox_the_connect 函数进行代理
        pro_threa = threading.Thread(target=prox_the_connect,
args=(new_conect_soc, addr))
        pro_threa.start()
finally:
    soc.close()
```