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	瞿久尧		院系	计算学部	
班级	2037101		学号	120L022314	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物 207		实验时间	2022.10.7	
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）

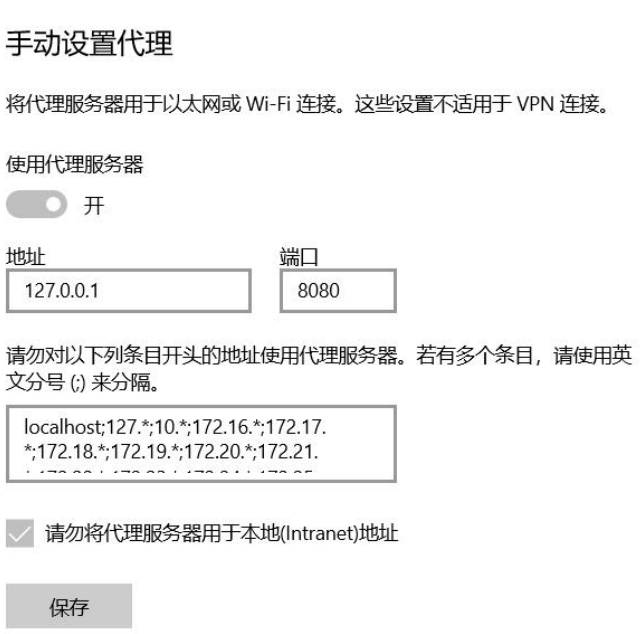
a) 网站过滤：允许/不允许访问某些网站；

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：

1. 在Edge浏览器中选择设置代理，并创建如下代理服务器：



2. 首先是HTTP 代理服务器。先是代理服务器初始化，

```python
def __init__(self):
    self.severPort = 8080  # 代理服务器端口
    self.main_sock = socket.socket(
        socket.AF_INET, socket.SOCK_STREAM)  # 创建TCP主套接字
    self.main_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.main_sock.bind(('', self.severPort))  # 绑定端口
    self.MAX_LISTEN = 20
    self.main_sock.listen(self.MAX_LISTEN)  # 最大连接数
    self.HTTP_BUFFER_SIZE = 4096  # http缓存大小
    self.default_cache_dir = r'D:\Rainbow\Study\大三秋\计算机网络\实验\lab1\120L022314-瞿久尧-实验1\cache\\'
    self.making_cache_dir()
```

接着是代理服务器链接函数，（返回文件的部分在cache部分展示）

```python
def proxy_connect(self, sock_to_web, address):
    message = sock_to_web.recv(
        self.HTTP_BUFFER_SIZE).decode('utf-8', 'ignore')

    print(f"Msg:{message}")
    msgs = message.split('\r\n')
    print(f"Format_msgs:{msgs}")
    request_line = msgs[0].strip().split()
    print(f"Request line:{request_line}")
    print(f"len(request line):{len(request_line)}")
    # print(f"request_line[1]:{request_line[1]}")
    if len(request_line) < 1:
        print("Request Line not contains url!")
        print(f"Full Request Message:{message}")
        sock_to_web.close()  # 关闭连接sock
        return

    else:
        # scheme://netloc/path;parameters?query#fragment url的一般形式 urlparse还可以包括 username password hostname port
        url = urlparse.urlparse(
            request_line[1][:-1] if request_line[1][-1] == '/' else request_line[1])
        print(f"url.scheme:{url.scheme},type:{type(url.scheme)}")
        print(f"url.hostname:{url.hostname},type:{type(url.hostname)}")
        print(f"url.port:{url.port},type:{type(url.port)}")
        print(f"url.scheme:{url.path},type:{type(url.path)}")
        print(f"url.netloc:{url.netloc},type:{type(url.netloc)}")
```

最后是在主函数中使用多线程

```python
# 主函数 使用threading实现多线程
def main():
    proxy = ProxyServer()
    while True:
        new_sock, address = proxy.main_sock.accept()
        print(address)
        threading.Thread(target=proxy.proxy_connect,
                         args=(new_sock, address)).start()
```

3. 下面实现cache功能，在初始化时已经设置cache路径

```python
def making_cache_dir(self):  # 缓存路径
    if not os.path.exists(self.default_cache_dir):
        os.mkdir(self.default_cache_dir)
```

接下来是cache功能的实现，在客户端向服务器发起请求时，先查看本地是否有缓存，如果有的话，查看缓存的时间，并向服务器核实是否过期，如果未过期，则直接返回文件，如果已过期或不存在该缓存文件，则再向服务器申请，代码如下：

```
cache_path = self.default_cache_dir + \
    (str(url.hostname) + str(url.path)).replace('/', '_')
flag_modified = False  # 默认缓存没有更改
flag_exists = os.path.exists(cache_path)  # 检测缓存目录是否存在
sock_to_Client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

if flag_exists:
    cache_time = os.stat(cache_path).st_mtime  # 获取缓存的时间
    msgs = {
        'If-Modified-Since': time.strftime('%a, %d %b %Y %H:%M:%S GMT', time.gmtime(cache_time))}
    response = requests.get(url.geturl(), headers=msgs)
    if response.status_code == 304:  # 如果返回304 则无需进行重新访问
        print("Read From Cache" + cache_path)
        with open(cache_path, "rb") as f:
            sock_to_web.sendall(f.read())
    else:
        flag_modified = True  # 否则证明缓存已经过时
```

```
if not flag_exists or flag_modified:  # 如果没有缓存或者缓存文件已经发生变化
    print("Attempt to connect", url.geturl())
    sock_to_Client.connect(
        (url.hostname, url.port if url.port else 80))
    sock_to_Client.sendall(message.encode())
    temp_file = open(cache_path, 'w')  # 记录缓存
    while True:
        buff = sock_to_Client.recv(self.HTTP_BUFFER_SIZE)
        if not buff:
            temp_file.close()
            sock_to_Client.close()
            break
        temp_file.write(buff.decode('gbk', 'ignore'))
        sock_to_web.sendall(buff)
    sock_to_web.close()
```

4. 最后是网站过滤、用户过滤和网站引导三个功能。首先在配置文件中设置要过滤的网站、过滤用户的IP地址以及想要引导的网站

```
"host": [
    "www.tsinghua.edu.cn"
  ],
  "ip": [
    "127.0.0.1"

  ],
  "fishing": [
    "www.hit.edu.cn"
  ]
```

如果需要过滤某个网站，则返回404界面

```
if self.filter_web(url.hostname):  # 如果需要过滤某个网站
    with open("404.html") as f:
        sock_to_web.sendall(f.read().encode())
    sock_to_web.close()
    return
```

如果需要过滤某个IP，则返回403界面

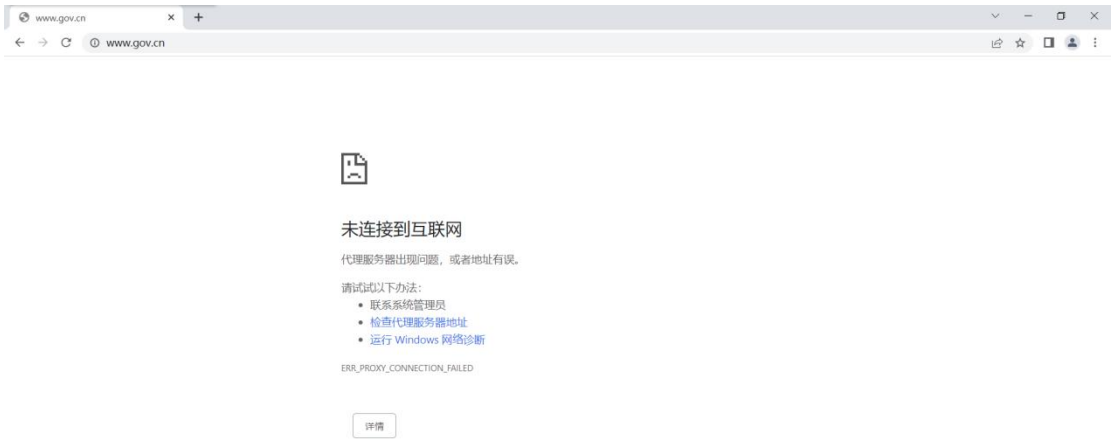
```
if self.filter_userip(address[0]):  # 如果需要过滤某个IP
    with open("403.html") as f:
        sock_to_web.sendall(f.read().encode())
    sock_to_web.close()
    return
```

如果访问的网址为被钓鱼网址，则返回钓鱼内容网址

```
if self.filter_fishing(url.hostname):  # 将需要钓鱼的网站重定向至中国作家网
    sock_to_web.sendall(requests.get("http://www.chinawriter.com.cn/").content)
    sock_to_web.close()
    return
```

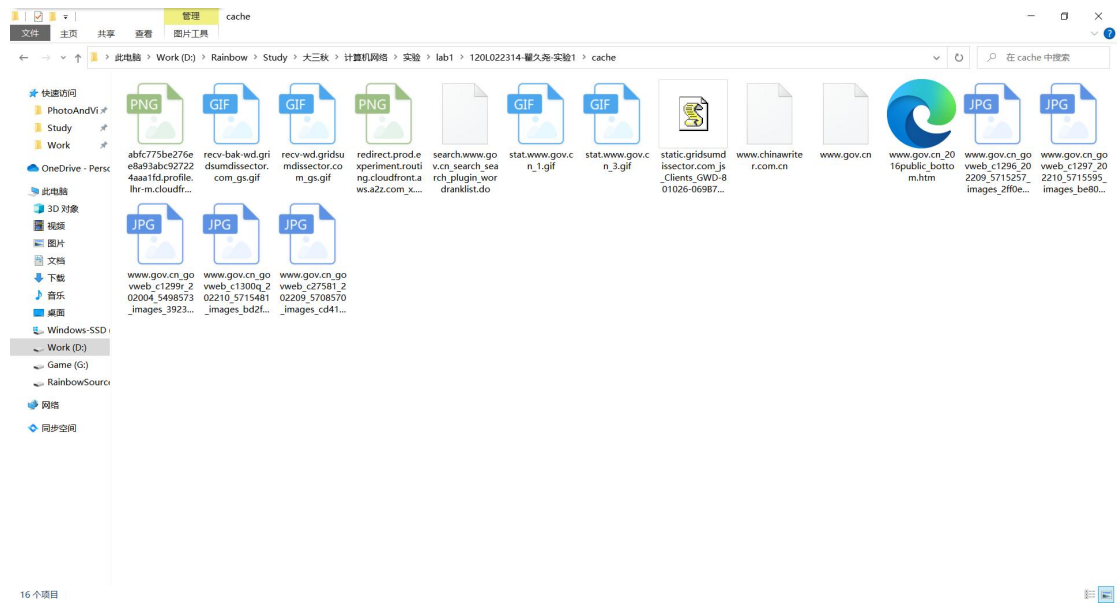
实验结果：

浏览器使用代理

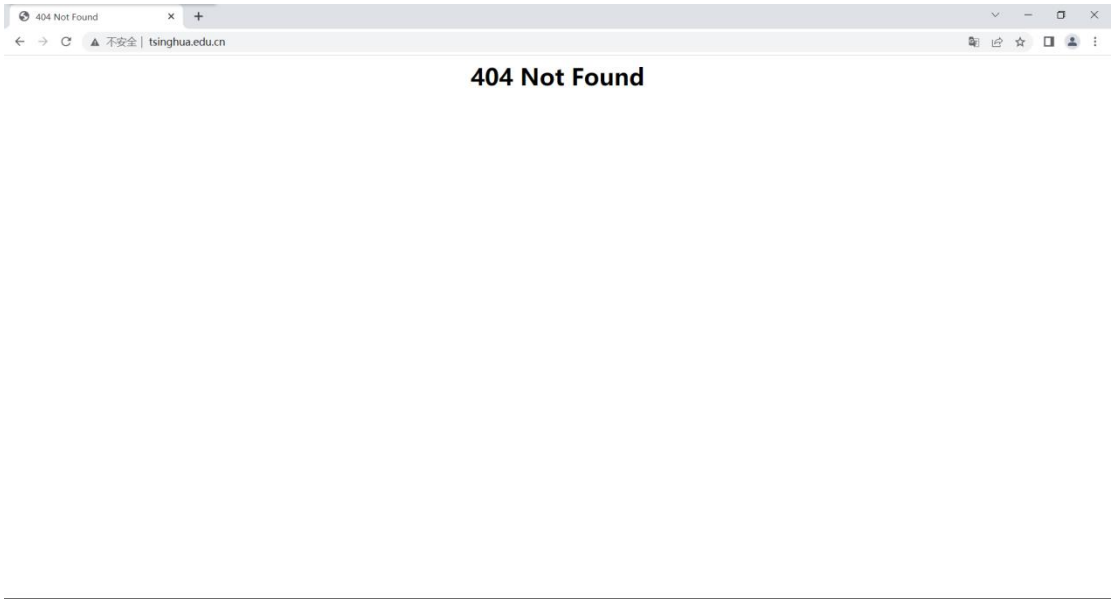


运行代码，访问www.gov.cn

cache文件夹



访问被过滤的清华大学网址



下面是用户屏蔽功能，本机无法正常访问

最后是钓鱼功能，把www.hit.edu.cn引导到http://www.chinawriter.com.cn/

问题讨论：

1. 对如何读取URL没有头绪，通过查找资料解决了问题
2. 用户过滤和网站过滤不知道如何进行，最后选择当满足条件时返回本地的HTML文件
3. 超时事件的设置，通过查找资料，采取实验指导书中的方式解决

心得体会：

结合实验过程和结果给出实验的体会和收获。

初步了解了 Socket 网络编程的过程与技术，帮助我课上的理解 HTTP 协议以及代理服务器的原理，同时知道了 IP 过滤、网站屏蔽以及钓鱼网站的原理，帮助我提升了网络安全意识。

附录：程序源代码

120L022314-瞿久尧-实验.py

```
import json
import os
import socket
import threading
import time
import urllib.parse as urlparse

import requests


class ProxyServer(object):
    def __init__(self):
        self.severPort = 8080   # 代理服务器端口
        self.main_sock = socket.socket(
            socket.AF_INET, socket.SOCK_STREAM)   # 创建 TCP 主套接字
        self.main_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.main_sock.bind(('', self.severPort))   # 绑定端口
        self.MAX_LISTEN = 20
        self.main_sock.listen(self.MAX_LISTEN)   # 最大连接数
        self.HTTP_BUFFER_SIZE = 4096   # http 缓存大小
        self.default_cache_dir = r'D:\Rainbow\Study\大三秋\计算机网络\实验\lab1\120L022314-瞿久尧
-实验 1\cache\\'
        self.making_cache_dir()

    def making_cache_dir(self):   # 缓存路径
        if not os.path.exists(self.default_cache_dir):
            os.mkdir(self.default_cache_dir)

    def filter_web(self, url):   # 过滤网站
        with open("config.json", "r") as f:
            filter_json = json.load(f)
            host_denied = filter_json['host']
            for url_denied in host_denied:
                if url in url_denied:
                    return True
            return False

    def filter_userip(self, ip):   # 过滤 ip
        with open("config.json", "r") as f:
            filter_json = json.load(f)
            user_denied = filter_json['ip']
            if ip in user_denied:
```

```
                    return True
            return False

    def filter_fishing(self, url):    # 钓鱼
        with open("config.json", "r") as f:
            filter_json = json.load(f)
            fishing = filter_json['fishing']
            for fishes in fishing:
                if url in fishes:
                    return True
            return False

    def proxy_connect(self, sock_to_web, address):
        message = sock_to_web.recv(
            self.HTTP_BUFFER_SIZE).decode('utf-8', 'ignore')

        print(f"Msg:{message}")
        msgs = message.split('\r\n')
        print(f"Format_msgs:{msgs}")
        request_line = msgs[0].strip().split()
        print(f"Request line:{request_line}")
        print(f"len(request line):{len(request_line)}")
        # print(f"request_line[1]:{request_line[1]}")
        if len(request_line) < 1:
            print("Request Line not contains url!")
            print(f"Full Request Message:{message}")
            sock_to_web.close()    # 关闭连接 sock
            return

        else:
            # scheme://netloc/path;parameters?query#fragment url 的一般形式  urlparse 还可以包括
username password hostname port
            url = urlparse.urlparse(
                request_line[1][:-1] if request_line[1][-1] == '/' else request_line[1])
            print(f"url.scheme:{url.scheme},type:{type(url.scheme)}")
            print(f"url.hostname:{url.hostname},type:{type(url.hostname)}")
            print(f"url.port:{url.port},type:{type(url.port)}")
            print(f"url.scheme:{url.path},type:{type(url.path)}")
            print(f"url.netloc:{url.netloc},type:{type(url.netloc)}")

        if self.filter_web(url.hostname):    # 如果需要过滤某个网站
            with open("404.html") as f:
                sock_to_web.sendall(f.read().encode())
            sock_to_web.close()
```

```
                return


            #if self.filter_userip(address[0]):    # 如果需要过滤某个 IP
            #       with open("403.html") as f:
            #               sock_to_web.sendall(f.read().encode())
            #       sock_to_web.close()
            #       return

            if self.filter_fishing(url.hostname):    # 将需要钓鱼的网站重定向至中国作家网
                sock_to_web.sendall(requests.get("http://www.chinawriter.com.cn/").content) #也可以尝试
将钓鱼网站换成其他的，
                sock_to_web.close()
                return

            cache_path = self.default_cache_dir + \
                (str(url.hostname) + str(url.path)).replace('/', '_')
            flag_modified = False    # 默认缓存没有更改
            flag_exists = os.path.exists(cache_path)    # 检测缓存目录是否存在
            sock_to_Client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            if flag_exists:
                cache_time = os.stat(cache_path).st_mtime    # 获取缓存的时间
                msgs = {
                    'If-Modified-Since': time.strftime('%a, %d %b %Y %H:%M:%S GMT',
time.gmtime(cache_time))}
                response = requests.get(url.geturl(), headers=msgs)
                if response.status_code == 304:    # 如果返回 304 则无需进行重新访问
                    print("Read From Cache" + cache_path)
                    with open(cache_path, "rb") as f:
                        sock_to_web.sendall(f.read())
                else:
                    flag_modified = True    # 否则证明缓存已经过时

            if not flag_exists or flag_modified:    # 如果没有缓存或者缓存文件已经发生变化
                print("Attempt to connect", url.geturl())
                sock_to_Client.connect(
                    (url.hostname, url.port if url.port else 80))
                sock_to_Client.sendall(message.encode())
                temp_file = open(cache_path, 'w')    # 记录缓存
                while True:
                    buff = sock_to_Client.recv(self.HTTP_BUFFER_SIZE)
                    if not buff:
                        temp_file.close()
```

```
                            sock_to_Client.close()
                            break
                        temp_file.write(buff.decode('gbk', 'ignore'))
                        sock_to_web.sendall(buff)
                sock_to_web.close()


# 主函数 使用 threading 实现多线程
def main():
    proxy = ProxyServer()
    while True:
        new_sock, address = proxy.main_sock.accept()
        print(address)
        threading.Thread(target=proxy.proxy_connect,
                                    args=(new_sock, address)).start()


# 程序入口
if __name__ == '__main__':
    main()
```

config.json

```
{
    "host": [
        "www.tsinghua.edu.cn"
    ],
    "ip": [
        "127.0.0.1"

    ],
    "fishing": [
        "www.hit.edu.cn"
    ]
}
```

301.html

```
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
</body>
</html>
```

403.html

```
<html>
<head><title>403 Forbidden </title></head>
<body bgcolor="white">
<center><h1>403 Forbidden </h1></center>
</body>
</html>
```

404.html

```
<html>
<head><title>404 Not Found</title></head>
<body bgcolor="white">
<center><h1>404 Not Found</h1></center>
</body>
</html>
```