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	石卓凡		院系	软件学院	
班级	2037101		学号	120L021011	
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

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。

一、理论原理

（1）Socket 编程的客户端和服务器端主要步骤；

TCP 客户端与服务器端的原理说明：

服务器端

1，Socket（）实例化创建

2，Bind（IP，PORT）绑定 IP 和 PORT

3，Listen（）使得 socket 处于监听状态

4，Accept（）进入阻塞状态等待连接

5，Recv()接收数据

6，Send（）发送数据

7，Close()关闭

客户端：

1，socket（）初始化 socket 对象

2，Connect（IP,PORT)连接服务器

3，Send（）给服务器发送数据

4，Recv（）接收数据

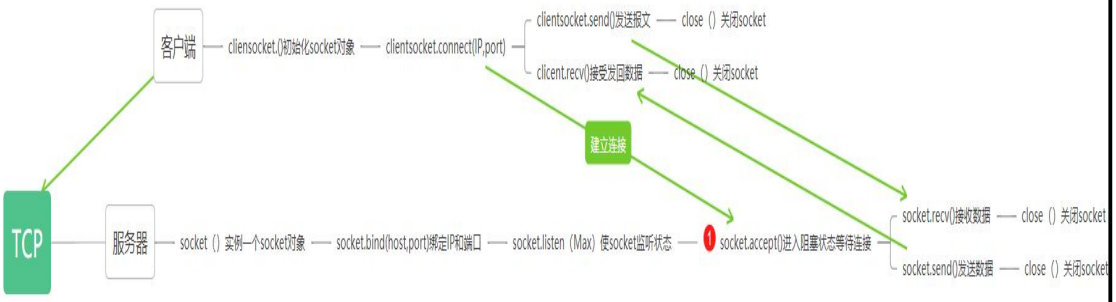
5，Clouse（）关闭

并发面向连接服务器基本流程

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）.

具体 TCP 流程图：

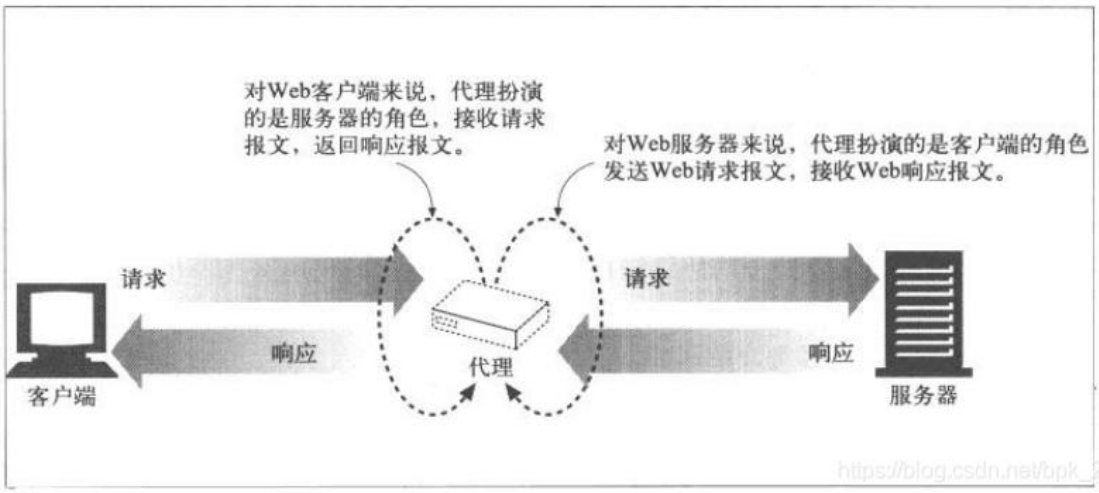
原图位于'./TCP 客户端和服务器端流程.png'



（2）HTTP 代理服务器的基本原理：

代理服务器的概念

一台主机想要访问某个网址，该主机不会直接向对应的服务器请求该网址，而是通过一个中间服务器，主机告诉中间服务器我想要哪个网址的内容，中间服务器代替主机去请求该网址，中间服务器得到响应后将该响应发送给主机。这里的中间服务器就是代理服务器(**Proxy Server**)



代理服务器的实现原理：

代理服务器启动并且设置一个 serversocket，

然后客户端启动 clientsocket 并且连接代理服务器端，

代理服务器获得这个客户端对应的 datasocket 此后可以通过 datasocket 向客户端发送数据，

客户端连接之后通过 clientsocket 发送请求，

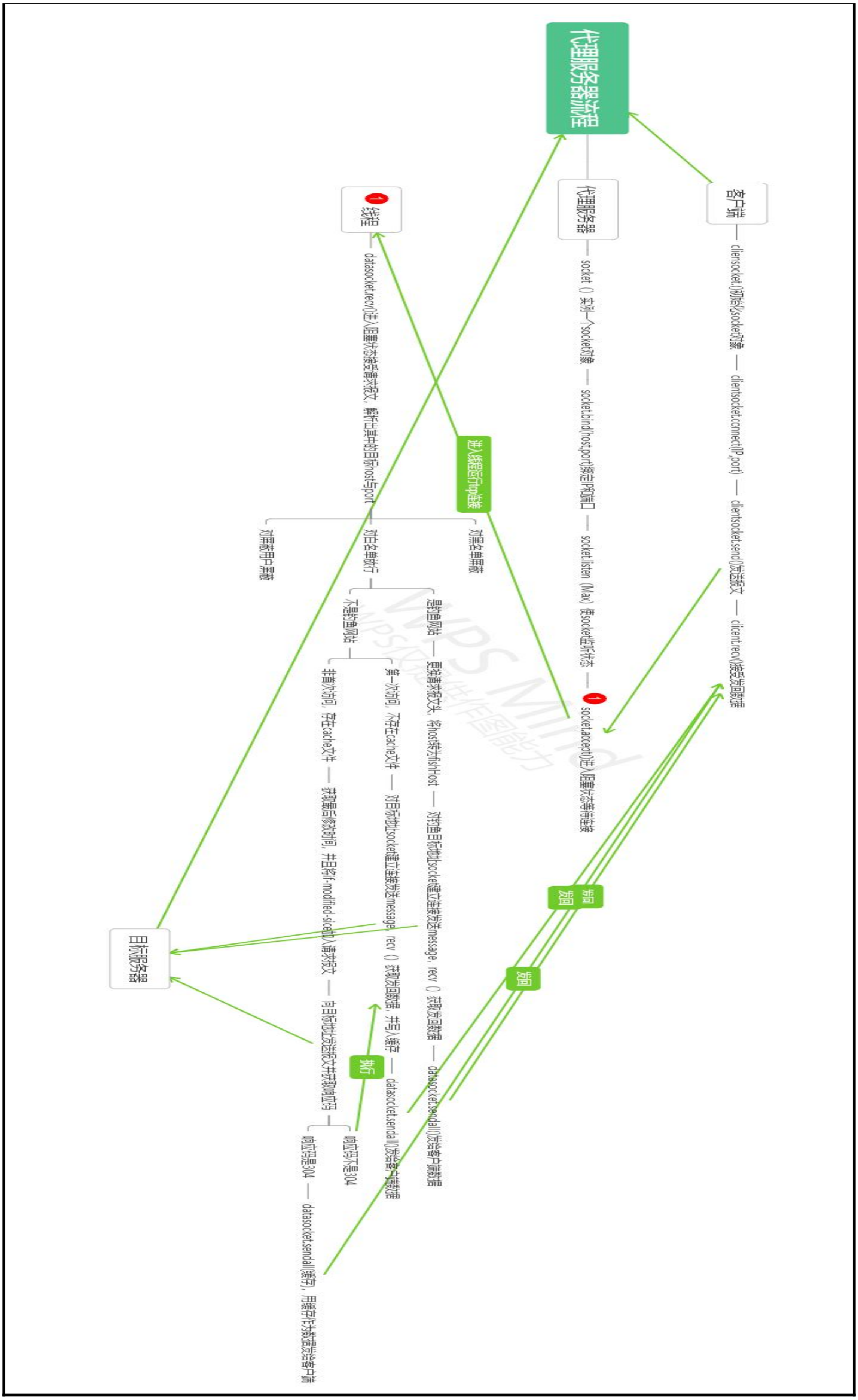
代理服务器通过 serversocket 收到客户端的请求，将请求转发给目的网站，然后 serversocket 收到目的网站返回的响应报文，

代理服务器通过 datasocket 将响应报文发回给客户端

客户端 clinetsocket 接受代理服务器发来的数据

（3）HTTP 代理服务器的程序流程图；

原图位于’./代理服务器程序流程图.png’


代理服务器流程
客户端
cliensocket.()初始化socket对象
clientsocket.connect(IP,port)
clientsocket.send()发送报文
clicent.recv()接受发回数据
代理服务器
socket（）实例一个socket对象
socket.bind(host,port)绑定IP和端口
socket.listen（Max）使socket监听状态
socket.accept()进入阻塞状态等待连接
进入线程运行tcp连接
线程
datasocket.recv()进入阻塞状态接受请求报文，解析出其中的目标host与port
对黑名单屏蔽
对白名单放行
对屏蔽用户屏蔽
是钓鱼网站
更换请求报文头，将host转为fishHost
对钓鱼目标地址socket建立连接发送message，recv（）获取发回数据
datasocket.sendall()发给客户端数据
不是钓鱼网站
第一次访问，不存在cache文件
对目标地址socket建立连接发送message，recv（）获取发回数据，并写入缓存
datasocket.sendall()发给客户端数据
非首次访问，存在cache文件
获取最后修改时间，并且将if-modified-sice加入请求报文
向目标地址发送报文并获取响应码
响应码不是304
响应码是304
datasocket.sendall(缓存)，用缓存作为数据发给客户端
执行
发回
目标服务器

（4）**实现 HTTP 代理服务器的关键技术及解决方案；**

二、程序代码逻辑

1. 代理服务器基本功能

(1) 设定各参数

```
# 代理服务器相关参数
PARAMETERS = {
    'HOST': '127.0.0.1',
    'PORT': 10086,
    'MAX_LISTEN': 50,
    'MAX_LENGTH': 4096,
    'CACHE_SIZE': 65507
}
```

(2) bind，listen初始化服务器socket

```
print("初始化socket")

# 实例化一个socket对象
# 参数 AF_INET 表示该socket网络层使用IP协议
# 参数 SOCK_STREAM 表示该socket传输层使用TCP协议
listenSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 绑定IP地址和端口号
listenSocket.bind((PARAMETERS['HOST'], PARAMETERS['PORT']))
# 使socket处于监听状态，等待客户端的连接请求
# 参数 MAX_LISTEN 表示 最多可以有MAX_LISTEN处于排队等待连接
# socket的排队个数
listenSocket.listen(PARAMETERS['MAX_LISTEN'])

# 创建cache目录
if not os.path.exists(cache_dir):
    os.mkdir(cache_dir)

print('初始化成功')
# 线程id
threadId = 1
```

(3) 利用while True死循环等待连接，接到连接之后，发起多线程工作

```
while True:
    # 在循环中监听9999端口，接收到客户端请求则创建一个新线程处理
    print('等待连接')
    # datasocket数据socket
    # address--客户端IP地址和端口号
    dataSocket, address = listenSocket.accept()
    print('接受来自主机' + str(address) + '的连接')
    threading.Thread(target=socket_tcp, args=(dataSocket, address, threadId)).start()
    threadId = threadId + 1
```

进入socket_tcp(dataSocket, address, threadId)

- dataSocket对应的客户端socket
- Address对于的客户端IP地址和客户端端口
- ThreadId线程Id

接受请求报文message，然后将bytes解码为字符串，然后通过parseToHostnameAndPort(message)获得目标host和目标port

```
# 接受来自客户端的http请求报文
message = dataSocket.recv(PARAMETERS['MAX_LENGTH'])

# 如果返回空bytes，请求报文为空，不需要处理
if len(message) == 0:
    return

# 读取的字节数据是bytes类型，需要解码为字符串
message = message.decode('utf-8', 'ignore')  # 对报文进行解码，忽略错误

# print("message:")
# print(message)
# message.split('\r\n')[0]拿到请求头

# GET http://www.sina.com/ HTTP/1.1
request_line = message.split('\r\n')[0].split()  # 获得请求行，去掉前后空格\

target_host_without_port, target_port = parseToHostnameAndPort(message)
```

进入parseToHostnameAndPort(message):

Message请求报文由于请求报文格式固定，每一行结尾都是\r\n，所以可以拿到url

其中有两种情况

情况1 today.hit.edu.cn:443（https后续无需考虑）

情况2 cs.hit.edu.cn　(默认端口80)

再次利用split(:)划分，拿到host和port并返回

```python
def parseToHostnameAndPort(message):
    message_line = message.split('\r\n')[1].split()

    # 情况1 today.hit.edu.cn:443
    # 情况2 cs.hit.edu.cn  (默认端口80)
    target_host_with_port = message_line[1]

    target_host_line = target_host_with_port.split(':')
    target_host_without_port = target_host_line[0]

    # 情况1 today.hit.edu.cn:443
    if (len(target_host_line) == 2):
        target_port = int(target_host_line[1])
    else:  # 情况2 cs.hit.edu.cn,默认端口80
        target_port = 80

    return target_host_without_port, target_port
```

返回继续执行socket_tcp（）

1.用户屏蔽功能：

(1)设定过滤 IP

```python
# 过滤用户IP
Blocked_User_IP = [
    # '127.0.0.1'
]
```

(2)然后如果本次线程中的hostIP在应该被屏蔽的IP里面则Socket.close()

(3)关闭socket禁止访问

```python
if hostIP in Blocked_User_IP:  # 用户IP被过滤
    print('线程:' + str(threadId) + '----------用户已被禁止访问------------')
    print('线程:' + str(threadId) + '-->' + '该用户的IP' + str(hostIP) + '已经被代理服务器禁
    print('线程:' + str(threadId) + '----------用户已被禁止访问------------')
    dataSocket.close()
    return
```

2.网站屏蔽功能（白名单和黑名单功能）：

设定白名单和黑名单

```
# 仅可访问白名单
White_url = [
    'cs.hit.edu.cn',  # 可访问，访问快
    'jwts.hit.edu.cn',  # 可访问且可以缓存，但是访问慢
    # 'today.hit.edu.cn',  # 这里默认是https，需要人工加上http，但是显示全
    'software.hit.edu.cn',  # 可访问
    'example.com',#可访问且可以缓存，多刷新几次，还挺快
    'www.7k7k.com',  # 可访问但是补全还有关联的子host "i1.7k7kimg.cn"
    'www.7k7kjs.cn',
    # 'info.cern.ch'
]

# 禁止访问的黑名单
Black_url = [
    # 'today.hit.edu.cn'
]
```

如果host解析出来之后，去检测是否在白名单和黑名单，然后根据内容进行处理

```
if target_host_without_port in Black_url:  # 主机名被禁止访问
    print('线程:' + str(threadId) + '-->' + str(target_host_without_port) + '在黑名单，禁止访问')
    dataSocket.close()
    return

# 只允许白名单内的可以被访问
if target_host_without_port not in White_url:
    # print('线程:' + str(threadId) + '-->' + str(target_host_without_port) + '不在白名单，禁止访
    dataSocket.close()
    return
# 黑名单内的不允许访问
# if target_host_without_port in Black_url:
#     print('线程:' + str(threadId) + '-->' + str(target_host_without_port) + '黑名单，禁止访问')
#     dataSocket.close()
#     return
```

3.钓鱼网站功能：

(1)判断host是否在钓鱼网站

```
# 钓鱼情况
if target_host_without_port in fishingWeb:  # 主机名为钓鱼网站
```

(2)如果是，则通过map拿到应该钓鱼目的host

```
fishing_host_without_port = fishingWeb[target_host_without_port]
# http默认80
```

(3)将message中的原host全改为钓鱼目的host

```
# 把host全改成钓鱼的
message = message.replace(target_host_without_port, fishing_host_without_port)
print('------------改之后')
print(message)
```

(4)利用修改后的message发送

```
fishDataSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # 初始化钓鱼网站的socket
fishDataSocket.connect((fishing_host_without_port, fishing_host_port))  # 与钓鱼网站的服务器建立连接
fishDataSocket.sendall(message.encode())  # 将报文编码发送到钓鱼网站服务器
```

(5)代理服务器接受钓鱼目的服务器发回数据，然后转发给客户端

```
while True:
    # 从钓鱼目的服务器接收数据,转发给客户端
    buff = fishDataSocket.recv(PARAMETERS['MAX_LENGTH'])
    if not buff:
        fishDataSocket.close()
        break
    dataSocket.sendall(buff)
print('线程:' + str(threadId) + '-->' + '接收数据完毕')
dataSocket.close()
fishDataSocket.close()
print('线程:' + str(threadId) + '----------钓鱼结束------------')
return
```

4.实现缓存，非钓鱼情况：

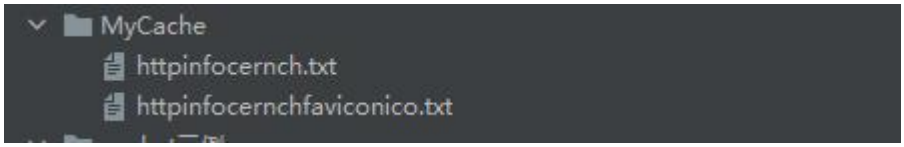
(1)通过拿到的url格式化为文件filename

比如http://cs.hit.edu.cn/page/jquery2.js格式化为cshiteducnpagejquery2js

最后路径为path = cache_dir + filename + '.txt'

```
filename = target_url.replace(":", '').replace('.', '').replace('/', '').replace('?', '').replace('*', '').replace('<', '').replace('>', '').replace('|', '')
path = cache_dir + filename + '.txt'  # 组合缓存路径和文件名
modified = False  # 第一次标记为未修改
```

path对应位置比如



(2)利用modified判断是否服务器资源是否修改，是否可以读取代理服务器缓存如果path路径已经存在文件，代表非第一次访问

```
# 不是第一次访问该网页
if os.path.exists(path):  # 当已经存在该文件，需要判断服务器是否修改过此网页
```

(3)获取path路径下系统的最后修改时候然后按照报文格式将if-modified-since插入报文

```
# 在path路径下调用系统stat拿到最后修改时间
modified_time = os.stat(path).st_mtime  # 缓存文件最后修改的时间

# 把modified-time按报文要求格式化
headers = str('If-Modified-Since: ' + time.strftime('%a, %d %b %Y %H:%M:%S GMT', time.gmtime(modified_time)))
```

```
# '\r'
# 回车，回到当前行的行首，而不会换到下一行，如果接着输出的话，本行以前的内容会被逐一覆盖；
# '\n'
# 换行，换到当前位置的下一行，而不会回到行首；

# message[:-2]去掉最后两行空白的\r\n\r\n
message = message[:-2] + headers + '\r\n\r\n'  # 把If-Modified-Since字段加入到请求报文中
```

(4)利用新报文message向目的服务器发送，然后只需要获取头部内容

```
# 向服务器发送报文
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.connect((target_host_without_port, target_port))
server_socket.sendall(message.encode())
data = server_socket.recv(PARAMETERS['MAX_LENGTH']).decode('utf-8', 'ignore')
# print(data)
server_socket.close()
```

(5)利用获取到头部内容，利用响应报文固定格式拿到响应码

(6)如果响应码为304，则读取path路径下的缓存，以二进制方式读，并且socket.sendall()发送给客户端，实现读取缓存

(7)如果不是304响应码，则将modified改为true代表服务器资源已修改需要继续执行

```
if data[9:12] == '304':  # 响应码为304，表示网页未变化，从cache中读取网页
    print('----------------------------------------------')
    print('线程:' + str(threadId) + '-->' + '响应码304-当前服务器资源未被修改，直接读取代理服务器cache返回')
    print('304')
    print('----------------------------------------------')
    print('线程:' + str(threadId) + '----------代理服务器，304读取缓存，工作结束------------')
    # 二进制文件就用二进制方法读取'rb'
    with open(path, "rb") as f:
        # 直接把代理服务器缓存里的目标服务器资源发送给客户端
        dataSocket.sendall(f.read())
else:  # 网页变化，标记为已修改
    modified = True
    # 使得进行下面的代码段运行
```

(8)如果发现path路径为空，或者刚才的modified为true代表需要目的服务器给的数据

```
if not os.path.exists(path) or modified:  # 如果没有该网页的缓存，或者网页已被修改（拿到modified=true）
```

直接用原message报文发送

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.connect((target_host_without_port, target_port))
server_socket.sendall(message.encode())
```

(1)通过while死循环中的recv保证能够全部接受到socket发过来的数据，如果缓存buff为空代表目标服务器发送完毕，可以退出while关闭连接

(2)接送到的数据写入path下的缓存，并且通过socket.sendall()发送给客户端

```
while True:
    buff = server_socket.recv(PARAMETERS['MAX_LENGTH'])
    if not buff:
        # print(buff)
        # 如果返回空bytes，表示对方关闭了连接
        print('线程:' + str(threadId) + '-->' + '对方服务器发送完毕数据，对方服务器关闭连接')
        f.close()
        server_socket.close()
        break
    f.write(buff)  # 将接收到的数据写入缓存
    dataSocket.sendall(buff)  # 通过代理服务器的dataSocket将接收到的数据转发给客户端
print('线程:' + str(threadId) + '-->' + '代理服务器转发给客户端数据，转发已完毕')
dataSocket.close()
```

(3) 关闭socket

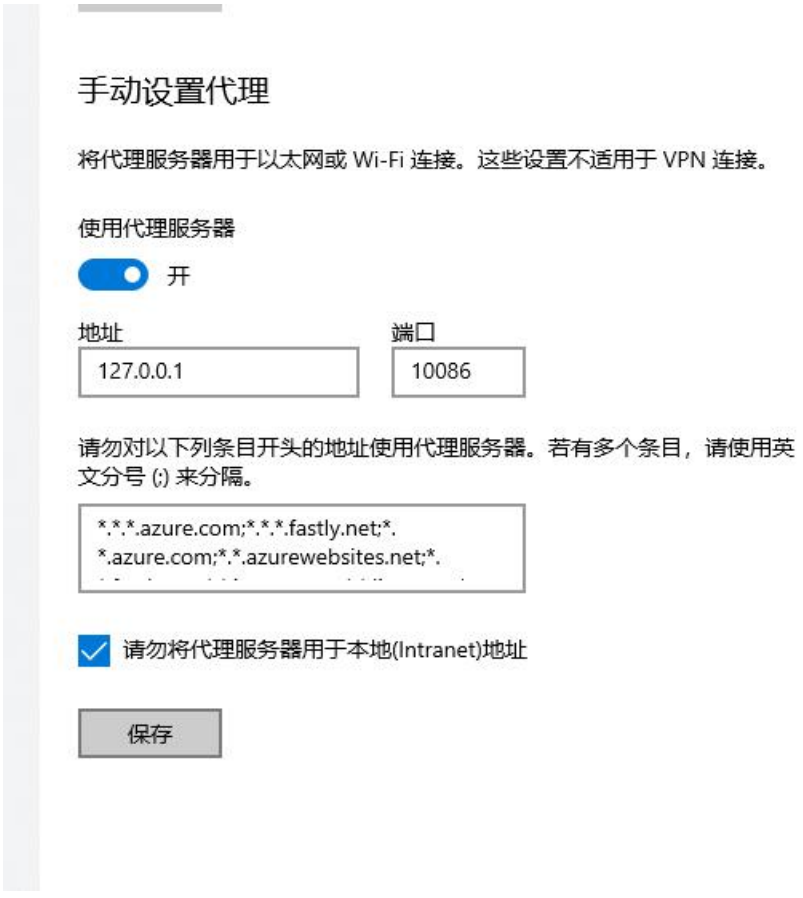
完成实验 1 所有必做与选做功能

实验结果：

（5）HTTP 代理服务器实验验证过程以及实验结果；

基本 HTTP 代理服务器功能：

1 打开代理服务器，ip 设为 127.0.0.1,端口为 10086



2 运行代理服务器，等待客户端连接

3 打开浏览器访问

cs.hit.edu.cn

发现网页正常访问连接成功

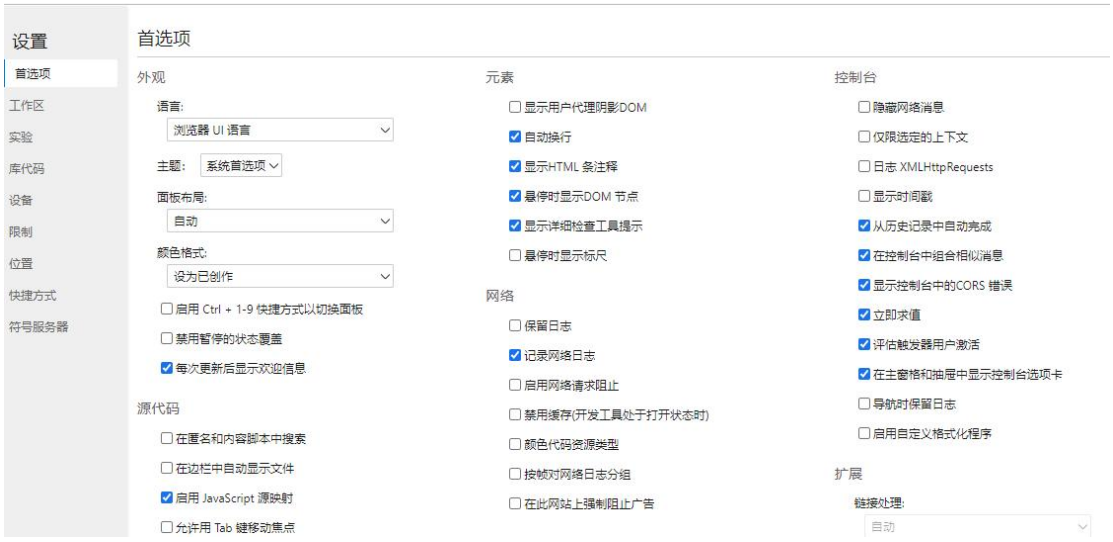
4 控制台显示日志，代理服务器工作，进行访问请求，转发数据

```
------target_host: ------
目的host:cs.hit.edu.cn
目的port:80
------target_host: ------
线程:4---------代理服务器工作开始------------
线程:4-->target:目标host:cs.hit.edu.cn   线程:4-->port:目标端口:80   线程:4-->source:客户端hostIP:127.0.0.1
线程:4-->真实要访问的地址http://cs.hit.edu.cn/
线程:4-->urlparse为: ParseResult(scheme='http', netloc='cs.hit.edu.cn', path='/', params='', query='', fragment='')
线程:4-->给真正要访问的地址:cs.hit.edu.cn发送访问请求
线程:4-->发送完毕,代理服务器准备接受来自cs.hit.edu.cn:80的数据，随后转发给客户端，并且更新代理服务器cache
------target_host: ------
目的host:cs.hit.edu.cn
目的port:80
------target_host: ------
线程:5---------代理服务器工作开始------------
线程:5-->target:目标host:cs.hit.edu.cn   线程:5-->port:目标端口:80   线程:5-->source:客户端hostIP:127.0.0.1
线程:5-->真实要访问的地址http://cs.hit.edu.cn/_upload/tpl/02/b2/690/template690/images/menu_line.png
线程:5-->urlparse为: ParseResult(scheme='http', netloc='cs.hit.edu.cn', path='/_upload/tpl/02/b2/690/template690/images/menu_line.png', params='', query='', fragment='')
线程:5-->响应码为404
线程:5-->给真正要访问的地址:cs.hit.edu.cn发送访问请求
线程:5-->发送完毕,代理服务器准备接受来自cs.hit.edu.cn:80的数据，随后转发给客户端，并且更新代理服务器cache
```

```
线程:4-->对方服务器发送完毕数据，对方服务器关闭连接
线程:4-->代理服务器转发给客户端数据，转发已完毕
线程:4----------代理服务器，工作结束------------
线程:5-->对方服务器发送完毕数据，对方服务器关闭连接
线程:5-->代理服务器转发给客户端数据，转发已完毕
线程:5----------代理服务器，工作结束------------
```

验证 Cache 功能：

0. 关闭浏览器缓存



1. py 自动建立空 MyCache 文件夹

2 访问 jwts.hit.edu.cn 网页等待加载

3. 首次访问成功

4 控制台日志显示首次访问完毕，更新缓存

```
------target_host: ------
目的host:jwts.hit.edu.cn
目的port:80
------target_host: ------
线程:3----------代理服务器工作开始------------
线程:3-->target:目标host:jwts.hit.edu.cn    线程:3-->port:目标端口:80    线程:3-->source:客户端hostIP:127.0.0.1
线程:3-->真实要访问的地址http://jwts.hit.edu.cn/
线程:3-->urlparse为: ParseResult(scheme='http', netloc='jwts.hit.edu.cn', path='/', params='', query='', fragment='')
线程:3-->给真正要访问的地址:jwts.hit.edu.cn发送访问请求
线程:3-->发送完毕,代理服务器准备接受来自jwts.hit.edu.cn:80的数据，随后转发给客户端，并且更新代理服务器cache
```

5.生成 Mycache 中的缓存文件

5. 再次访问 **jwts.hit.edu.cn** 网页

6 发现响应码为 304，直接读取的缓存内容

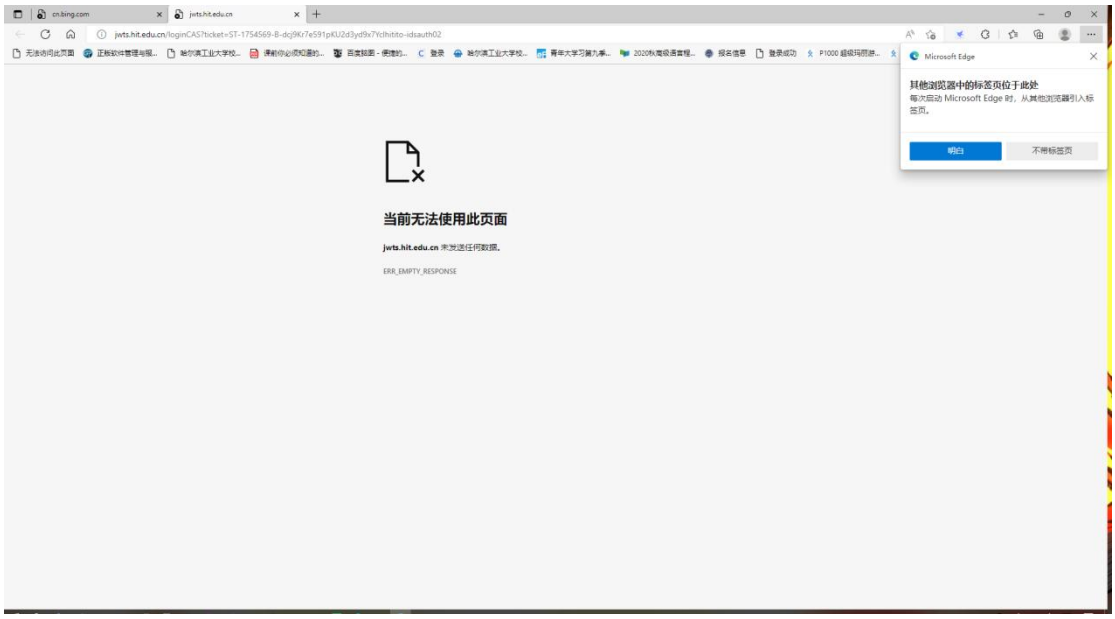
```
线程:21---------代理服务器工作开始------------
线程:21-->target:目标host:jwts.hit.edu.cn    线程:21-->port:目标端口:80    线程:21-->source:客户端hostIP:127.0.0.1
线程:21-->真实要访问的地址http://jwts.hit.edu.cn/resources/css/common/ydy.css
线程:21-->urlparse为: ParseResult(scheme='http', netloc='jwts.hit.edu.cn', path='/resources/css/common/ydy.css', params='', query='', fragment='')
线程:21-->响应码为304
--------------------------------------------
线程:21-->响应码304-当前服务器资源未被修改，直接读取代理服务器cache返回
304
--------------------------------------------
线程:21---------代理服务器，读取缓存，工作结束------------
```

测试屏蔽用户 IP:

1. 打开过滤功能

```
32    # 过滤用户IP
33    Blocked_User_IP = [
34        '127.0.0.1'
35    ]
36
```

2.以 127.0.01IP 访问任意网页均显示无法使用



3 日志显示：用户 IP 被禁止访问

```
接受来自主机('127.0.0.1', 5822)的连接
等待连接线程:46----------用户已被禁止访问------------

线程:46-->该用户的IP127.0.0.1已经被代理服务器禁止
线程:46----------用户已被禁止访问------------
接受来自主机('127.0.0.1', 5836)的连接
等待连接
线程:47----------用户已被禁止访问------------
线程:47-->该用户的IP127.0.0.1已经被代理服务器禁止
线程:47----------用户已被禁止访问------------
```

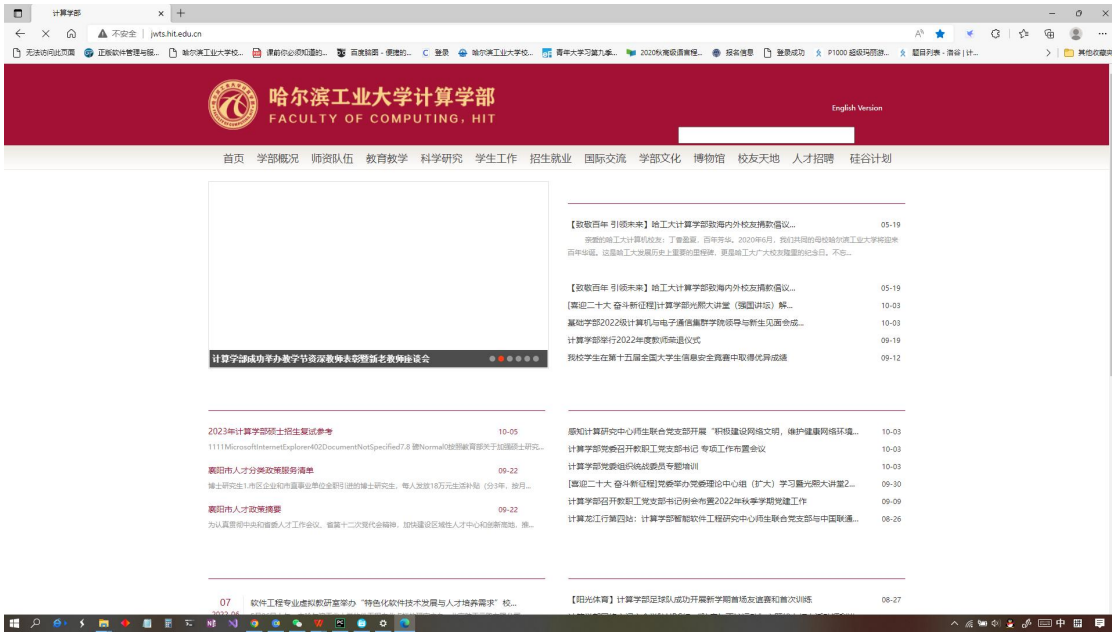
钓鱼功能测试：

1.指定被钓鱼网站以及钓鱼目的网站

```
# 钓鱼
fishingWeb = {

    'jwts.hit.edu.cn': 'cs.hit.edu.cn'
}
```

2. 访问 jwts.hit.edu.cn 发现拿到的内容是 cs.hit.edu.cn



3.查看控制台日志发现，启动钓鱼功能，代理服务器将请求报文的 jwts.hit.edu.cn 修改为 cs.hit.edu.cn

```
线程:6---------钓鱼启动------------
线程:7-->对jwts.hit.edu.cn进行钓鱼，钓鱼到-> cs.hit.edu.cn
线程:6-->对jwts.hit.edu.cn进行钓鱼，钓鱼到-> cs.hit.edu.cn------------改之前

------------改之前GET http://jwts.hit.edu.cn/_js/_portletPlugs/datepicker/css/images/dp/glass-bg.gif HTTP/1.1
Host: jwts.hit.edu.cn
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36 Edg/103.0.1264.71
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://jwts.hit.edu.cn/_js/_portletPlugs/datepicker/css/datepicker.css
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Cookie: _ga=GA1.3.908923564.1606903375; JSESSIONID=8B90912FEAE88EE0977B59E4B7A0F700
```

```
------------改之后
GET http://cs.hit.edu.cn/_js/_portletPlugs/datepicker/css/images/dp/glass-bg.gif HTTP/1.1
Host: cs.hit.edu.cn
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36 Edg/103.0.1264.71
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://cs.hit.edu.cn/_js/_portletPlugs/datepicker/css/datepicker.css
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Cookie: _ga=GA1.3.908923564.1606903375; JSESSIONID=8B90912FEAE88EE0977B59E4B7A0F700
```

4. 钓鱼成功

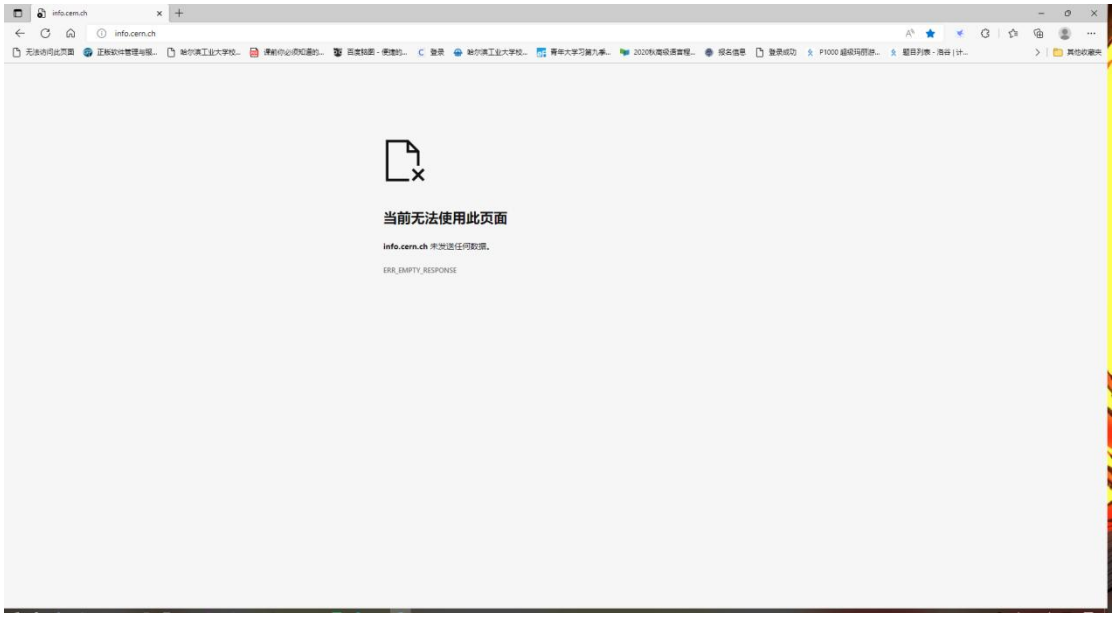
```
线程:20-->正在从钓鱼服务器接收数据并转发,请稍后
线程:8-->接收数据完毕
线程:8----------钓鱼结束------------
```

测试网站过滤功能（白名单/黑名单）：

1. 将 info.cern.ch 移除白名单

```
# 仅可访问白名单
White_url = [
    'cs.hit.edu.cn',   # 可访问，访问快
    'jwts.hit.edu.cn',   # 可访问且可以缓存，但是访问慢
    'today.hit.edu.cn',   # 这里默认是https，需要人工加上http，但是显示全字符
    'software.hit.edu.cn',   # 可访问
    'example.com',#可访问且可以缓存，多刷新几次，还挺快
    'www.7k7k.com',   # 可访问但是补全还有关联的子host "i1.7k7kimg.cn"
    'www.7k7kjs.cn',
    # 'info.cern.ch'
]
```

2 发现无法访问



3 控制台日志

```
等待连接
线程:6-->www.bing.com不在白名单，禁止访问
接受来自主机('127.0.0.1', 7130)的连接
等待连接
接受来自主机('127.0.0.1', 7131)的连接
等待连接
线程:7-->info.cern.ch不在白名单，禁止访问
线程:8-->info.cern.ch不在白名单，禁止访问
接受来自主机('127.0.0.1', 7132)的连接
等待连接线程:9-->info.cern.ch不在白名单，禁止访问

接受来自主机('127.0.0.1', 7133)的连接
等待连接线程:10-->www.bing.com不在白名单，禁止访问

接受来自主机('127.0.0.1', 7134)的连接
等待连接
线程:11-->www.bing.com不在白名单，禁止访问
```

黑名单同理

```
# 禁止访问的黑名单
Black_url = [
    # 'today.hit.edu.cn'
]
```

问题讨论：

Q1:实现钓鱼网站时候，需要在代理服务器修改请求报文中的URL和Host，但是第一次修改之后（比如http://jwts.hit.edu.cn改为http://cs.hit.edu.cn)并发回客户端，客户端后续发回的请求报文仍是http://jwts.hit.edu.cn/page/jquery2.js，而不是http://cs.hit.edu.cn/page/jquery2.js

A1:本来以为代码错了，后面思考认为，在客户端中被代理服务器钓鱼到cs网站，但是客户端以为是jwts，所以在后续发出的请求js等报文中，由客户端填写为cs.hit.edu.cn的host，还需要再次被代理服务器修改

Q2:在进行缓存时候，将某个网站的一系列的请求缓存文件比如http://cs.hit.edu.cn与http://cs.hit.edu.cn/page/jquery2.js都以cshiteducn.txt缓存是否可行

A2:个人尝试之后，发现这种想法不够好，当再次访问该网站时候，很难实现响应304，因为如果某一个比如http://cs.hit.edu.cn的请求中不满足读取缓存要求，则会修改txt缓存，由于多线程打开同一个文件机制，可能会有多线程安全问题，较为复杂。

不如以cshiteducnpagejquery2js再单独存一份靠谱

Q3:有时候无法访问www.baidu.com,拿到的是443端口，有时候可以访问www.baidu.com拿到80端口

A3:现在很多网站都支持http和https同时存在,而https是443端口需要使用SSL/TLS证书进行加密传输，按照指导书上说明无需考虑443端口

心得体会：

结合实验过程和结果给出实验的体会和收获。

1. 明白了http和https的区别所在
2. 对于socket有了更深层次的体会和理解，对于利用socket作代理服务器，作为网络编程的技术更加的精进了些
3. 再次复习了代理服务器的原理和机制

附录：程序源代码

```
import os

import socket

import threading

import time

from urllib.parse import urlparse



# 代理服务器相关参数

PARAMETERS = {

    'HOST': '127.0.0.1',

    'PORT': 10086,

    'MAX_LISTEN': 50,

    'MAX_LENGTH': 4096,

    'CACHE_SIZE': 65507

}

# 仅可访问白名单
```

```
White_url = [
    'cs.hit.edu.cn',  # 可访问，访问快
    'jwts.hit.edu.cn',  # 可访问且可以缓存，但是访问慢
    # 'today.hit.edu.cn',  # 这里默认是 https，需要人工加上 http，但是显示全字符
    'software.hit.edu.cn',  # 可访问
    'example.com',#可访问且可以缓存，多刷新几次，还挺快
    'www.7k7k.com',  # 可访问但是补全还有关联的子 host "i1.7k7kimg.cn"
    'www.7k7kjs.cn',
    # 'info.cern.ch'
]


# 禁止访问的黑名单
Black_url = [
    # 'today.hit.edu.cn'
]


# 过滤用户 IP
Blocked_User_IP = [
    # '127.0.0.1'
]


# 钓鱼
```

```
fishingWeb = {
    #访问快
    'jwts.hit.edu.cn': 'cs.hit.edu.cn'
}

# 缓存目录
cache_dir = './MyCache/'

"""
从请求报文来获得 hostname,port
"""


def parseToHostnameAndPort(message):
    message_line = message.split('\r\n')[1].split()

    # 情况 1 today.hit.edu.cn:443
    # 情况 2 cs.hit.edu.cn  (默认端口 80)
    target_host_with_port = message_line[1]

    target_host_line = target_host_with_port.split(':')
    target_host_without_port = target_host_line[0]
```

```
    # 情况 1 today.hit.edu.cn:443
    if (len(target_host_line) == 2):
        target_port = int(target_host_line[1])
    else:  # 情况 2 cs.hit.edu.cn,默认端口 80
        target_port = 80

    return target_host_without_port, target_port


def socket_tcp(dataSocket, address, threadId):
    """
    建立 TCP 连接
    :param dataSocket: socket
    :param address: IP 地址和端口号组成的元组
    :return: 无
    """
    # 接受来自客户端的 http 请求报文
    message = dataSocket.recv(PARAMETERS['MAX_LENGTH'])

    # 如果返回空 bytes，请求报文为空，不需要处理
    if len(message) == 0:
```

```
    return


# 读取的字节数据是 bytes 类型，需要解码为字符串
message = message.decode('utf-8', 'ignore')  # 对报文进行解码，忽略错误

# print("message:")
# print(message)
# message.split('\r\n')[0]拿到请求头

# GET http://www.sina.com/ HTTP/1.1
request_line = message.split('\r\n')[0].split()  # 获得请求行，去掉前后空格\


target_host_without_port, target_port = parseToHostnameAndPort(message)
# print('------target_host：------')
# print(target_host_without_port)
# print(target_port)
# print('------target_host：------')


# message.split('\r\n')[0].split()再以空格切分，request_line[1]拿到 url 地址
target_url = request_line[1]
# 拿到情况 1，http://cs.hit.edu.cn/
# 情况 2，today.hit.edu.cn:443
```

```
    url = urlparse(request_line[1])  # 获得 URLparse 划分以辅助对比
    hostIP = address[0]  # 获得原主机 IP

    if hostIP in Blocked_User_IP:  # 用户 IP 被过滤
        print('线程:' + str(threadId) + '----------用户已被禁止访问------------')
        print('线程:' + str(threadId) + '-->' + '该用户的 IP' + str(hostIP) + '已经被代理服务器禁止')
        print('线程:' + str(threadId) + '----------用户已被禁止访问------------')
        dataSocket.close()
        return

    if target_host_without_port in Black_url:  # 主机名被禁止访问
        print('线程:' + str(threadId) + '-->' + str(target_host_without_port) + '在黑名单，禁止访问')
        dataSocket.close()
        return

    # 只允许白名单内的可以被访问
    if target_host_without_port not in White_url:
        # print('线程:' + str(threadId) + '-->' + str(target_host_without_port) + '不在白名单，禁止访问')
        dataSocket.close()
```

```
        return


    # if target_host_without_port in Black_url:
    #     print('线程:' + str(threadId) + '-->' + str(target_host_without_port) + '黑名单,
禁止访问')
    #     dataSocket.close()
    #     return
    print('------target_host: ------')
    print("目的 host:" + target_host_without_port)
    print("目的 port:" + str(target_port))
    print('------target_host: ------')
    # print('线程:' + str(threadId) + '-->' + '真实要访问的地址' + str(target_url))
    # print('线程:' + str(threadId) + '-->' + 'urlparse 为: ' + str(url))
    # print(request_line[1])


    # 钓鱼情况
    if target_host_without_port in fishingWeb:  # 主机名为钓鱼网站
        fishing_host_without_port = fishingWeb[target_host_without_port]
        # http 默认 80
        fishing_host_port = 80


        print('线程:' + str(threadId) + '----------钓鱼启动------------')
```

```
        print('线程:' + str(threadId) + '-->' + '对' + str(target_host_without_port) + '进行
钓鱼，钓鱼到-> ' + str(
            fishing_host_without_port))


        print('------------改之前')
        print(message)
        # # 更换报文头请求地址
        # message = message.replace(request_line[1], 'http://' +
fishing_host_without_port + '/')
        # 把 host 全改成钓鱼的
        message = message.replace(target_host_without_port,fishing_host_without_port)
        print('------------改之后')
        print(message)


        fishDataSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # 初始
化钓鱼网站的 socket
        fishDataSocket.connect((fishing_host_without_port, fishing_host_port))  # 与钓
鱼网站的服务器建立连接
        fishDataSocket.sendall(message.encode())  # 将报文编码发送到钓鱼网站服务器


        print('线程:' + str(threadId) + '-->' + '正在从钓鱼服务器接收数据并转发,请稍后')
        while True:
            # 从钓鱼目的服务器接收数据,转发给客户端
```

```
            buff = fishDataSocket.recv(PARAMETERS['MAX_LENGTH'])
            if not buff:
                fishDataSocket.close()
                break
            dataSocket.sendall(buff)
        print('线程:' + str(threadId) + '-->' + '接收数据完毕')
        dataSocket.close()
        fishDataSocket.close()
        print('线程:' + str(threadId) + '----------钓鱼结束------------')
        return


    #######
    # 非钓鱼情况
    if target_host_without_port is not None:
        print('线程:' + str(threadId) + '----------代理服务器工作开始------------')
        print('线程:' + str(threadId) + '-->' + 'target:目标 host:' +
target_host_without_port,
              end='    ')
        print('线程:' + str(threadId) + '-->' + 'port:目标端口:' + str(target_port),
              end='    ')
        print('线程:' + str(threadId) + '-->' + 'source:客户端 hostIP:' + hostIP)
        print('线程:' + str(threadId) + '-->' + '真实要访问的地址' + str(target_url))
```

```
        print('线程:' + str(threadId) + '-->' + 'urlparse 为: ' + str(url))


    if target_host_without_port is None:  # 主机名为空
        print('线程:' + str(threadId) + '-->' + 'target_host_without_port 为空,关闭该连接')
        dataSocket.close()
        return


    # 拿到情况 1，http://cs.hit.edu.cn/
    # 情况 2，today.hit.edu.cn:443


    filename = target_url.replace(':', '').replace('.', '').replace('/', '').replace('?',
'').replace('*', '').replace('<', '').replace('>', '').replace('|', '')
    path = cache_dir + filename + '.txt'  # 组合缓存路径和文件名
    modified = False  # 第一次标记为未修改


    # 不是第一次访问该网页
    if os.path.exists(path):  # 当已经存在该文件，需要判断服务器是否修改过此网页


        # 在 path 路径下调用系统 stat 拿到最后修改时间
        modified_time = os.stat(path).st_mtime  # 缓存文件最后修改的时间


        # 把 modified-time 按报文要求格式化
        headers = str('If-Modified-Since: ' + time.strftime('%a, %d %b %Y %H:%M:%S
GMT', time.gmtime(modified_time)))
```

```
        # '\r'
        # 回车，回到当前行的行首，而不会换到下一行，如果接着输出的话，本行以前的内容会被逐一覆盖；
        # '\n'
        # 换行，换到当前位置的下一行，而不会回到行首；

        # message[:-2]去掉最后两行空白的\r\n\r\n
        message = message[:-2] + headers + '\r\n\r\n'  # 把 If-Modified-Since 字段加入到请求报文中

        # 向服务器发送报文
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.connect((target_host_without_port, target_port))
        server_socket.sendall(message.encode())
        data = server_socket.recv(PARAMETERS['MAX_LENGTH']).decode('utf-8', 'ignore')
        # print(data)
        server_socket.close()

        # data 是 HTTP/1.1 304 ok
        print('线程:' + str(threadId) + '-->' + '响应码为' + data[9:12])
        if data[9:12] == '304':  # 响应码为 304，表示网页未变化，从 cache 中读取网页
            print('-----------------------------------------------')
```

```
            print('线程:' + str(threadId) + '-->' + '响应码 304-当前服务器资源未被修改，直
接读取代理服务器 cache 返回')
            print('304')
            print('-----------------------------------------')
            print('线程:' + str(threadId) + '----------代理服务器，304 读取缓存，工作结
束-------------')
            # 二进制文件就用二进制方法读取'rb'
            with open(path, "rb") as f:
                # 直接把代理服务器缓存里的目标服务器资源发送给客户端
                dataSocket.sendall(f.read())
        else:  # 网页变化，标记为已修改
            modified = True
            # 使得进行下面的代码段运行

    if not os.path.exists(path) or modified:  # 如果没有该网页的缓存，或者网页已被修改（拿
到 modified=true)
        # 向服务器发送数据，才能接收到服务器发回来的数据
        print('线程:' + str(threadId) + '-->' + '给真正要访问的地址:' +
target_host_without_port + '发送访问请求')
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.connect((target_host_without_port, target_port))
        server_socket.sendall(message.encode())
```

```
        print('线程:' + str(
            threadId) + '-->' + '代理服务器准备接受来自' + target_host_without_port + ':' +
str(
            target_port) + '的数据，随后转发给客户端，并且更新代理服务器 cache')
        f = open(path, 'wb')  # 重写缓存
        while True:
            buff = server_socket.recv(PARAMETERS['MAX_LENGTH'])
            if not buff:
                # print(buff)
                # 如果返回空 bytes，表示对方关闭了连接
                print('线程:' + str(threadId) + '-->' + '对方服务器发送完毕数据，对方服务器
关闭连接')
                f.close()
                server_socket.close()
                break
            f.write(buff)  # 将接收到的数据写入缓存
            dataSocket.sendall(buff)  # 通过代理服务器的 dataSocket 将接收到的数据转发
给客户端
        print('线程:' + str(threadId) + '-->' + '代理服务器转发给客户端数据，转发已完毕')
        dataSocket.close()
        print('线程:' + str(threadId) + '----------代理服务器，工作结束------------')
```

```
def main():
    print("初始化 socket")

    # 实例化一个 socket 对象
    # 参数 AF_INET 表示该 socket 网络层使用 IP 协议
    # 参数 SOCK_STREAM 表示该 socket 传输层使用 TCP 协议
    listenSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 绑定 IP 地址和端口号
    listenSocket.bind((PARAMETERS['HOST'], PARAMETERS['PORT']))
    # 使 socket 处于监听状态，等待客户端的连接请求
    # 参数 MAX_LISTEN 表示 最多可以有 MAX_LISTEN 处于排队等待连接
    # socket 的排队个数
    listenSocket.listen(PARAMETERS['MAX_LISTEN'])

    # 创建 cache 目录
    if not os.path.exists(cache_dir):
        os.mkdir(cache_dir)

    print('初始化成功')
    # 线程 id
    threadId = 1
    while True:
```

```
        # 在循环中监听 9999 端口，接收到客户端请求则创建一个新线程处理
        print('等待连接')
        # datasocket 数据 socket
        # address--客户端 IP 地址和端口号
        dataSocket, address = listenSocket.accept()
        print('接受来自主机' + str(address) + '的连接')
        threading.Thread(target=socket_tcp, args=(dataSocket, address,
threadId)).start()
        threadId = threadId + 1



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