**计算机网络专题实验 实验八报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名： |  | 班级： |  |
| 姓名： |  | 班级： |  |
|  |  |  |  |

1. 实验名称

Socket网络编程实验，实现基于HTTP协议的客户端程序（文本浏览器）

1. 实验原理

HTTP协议是基于TCP/IP协议栈的应用层协议，采用请求/响应模型。客户端通过Socket建立与服务器的TCP连接，发送HTTP请求报文，服务器处理后返回响应报文。本实验通过Python的socket模块实现HTTP客户端，支持GET/POST/HEAD方法、URI编码、Cookie管理、重定向跟踪和基础缓存等功能，并通过分块传输解码、gzip解压、multipart文件上传等机制实现高级特性。

1. 实验目的

1)掌握Sockets的相关基础知识，学习Sockets编程的基本函数和模式、框架。

2)掌握UDP、TCP协议及 Client/Server和P2P两种模式的通信原理。掌握可靠数据传输协议（GBN和SR协议）

3)掌握socket编程框架

1. 实验内容
   1. 基本功能

A. 支持GET、HEAD和POST三种请求方法，支持URI的"%HEXHEX"编码；支持Connection: Keep-Alive和Connection: Close两种连接模式；

B. 能够把一个网页中所有的内嵌对象（如HTML中的IMG、CSS、JS等对象）一次全部获取；

C. 支持Cookie的基本机制，实现典型的网站登录；

D. 能够正确处理几种典型的应答（如200，100，301，304，404，500等），并支持重定向请求；

E. 支持基本的缓存处理。

* 1. 高级功能

A. 支持HTTPS；

B. 支持分块传输编码(Chunked Transfer Encoding)、gzip等内容编码；

C. 支持基于POST方法的文件上传；

D. 支持把一个网页中特定对象一次全部获取。

1. 实验实现
2. 人员分工

【说清楚每个人所做的具体工作，不限于设计、编程、调试、测试、写报告等。】

1. 实验设计
2. 协议及标准

|  |  |
| --- | --- |
| 应用层协议 | 基于HTTP/1.1（RFC 2616）和HTTPS（RFC 2818） |
| 传输层协议 | TCP协议 |
| 编码规范 | URI百分比编码（RFC 3986）、Chunked传输编码（RFC 7230） |
| 状态管理 | Cookie机制（RFC 6265） |
| 内容传输 | gzip/deflate压缩（RFC 1952）、multipart文件上传（RFC 7578） |

1. 框架结构

采用分层设计，网络I/O与协议逻辑分离，便于扩展HTTPS

核心类：HTTPClient类，封装所有HTTP功能

功能模块：

1.网络连接：\_connect()处理TCP连接、SSL加密、连接复用

\_close\_connection()关闭连接

2.URL处理：\_parse\_url()解析URL

3.HTTP协议处理：\_build\_request()构造HTTP请求报文

\_parse\_response()解析HTTP响应

4.数据处理: \_decode\_chunked()解码分块传输编码的数据

\_decompress()解压gzip或deflate编码的内容

\_parse\_headers()将原始响应头解析为字典

5.资源管理：处理Cookie和缓存

6.核心请求方法：GET、HEAD、POST方法

7.网页下载：download\_embedded()下载网页及所有嵌入资源

拓展功能：重定向处理、文件上传、错误处理等

1. 关键代码的描述
2. \_request()方法：

def request(self, method: str, url: str, headers: Dict[str, str] = None, body: bytes = None, max\_redirects: int = 5) -> Tuple[int, Dict[str, str], bytes]:

...

功能：处理HTTP请求

过程：先解析URL并检查缓存，再建立TCP/SSL连接，然后构建并发送HTTP请求，收到服务器响应后解析响应，最后处理重定向、Cookie和缓存

关键：支持持久连接和自动重定向，能自动处理SSL连接，也实现了请求缓存机制

1. \_parse\_response()方法：

def \_parse\_response(self, sock: socket.socket) -> Tuple[int, Dict[str, str], bytes]:

...

功能：解析HTTP服务器响应

过程：读取响应头数据并解析，然后处理内容压缩、内容编码，最后根据Content-Length确保读取完整的响应体

关键：支持HTTP分块传输编码解析,可以自动解压gzip/deflate压缩内容，并且添加了超时安全的数据读取机制

1. post()方法：

def post(self, url: str, data: Dict[str, str] = None, files: Dict[str, Tuple[str, bytes]] = None, save\_to: str = None) -> Tuple[int, Dict[str, str], bytes]:

    ...

功能：发送POST请求，包含表单提交和文件上传两种请求类型

表单提交：能自动编码字典数据为URL查询字符串格式，并设置正确的Content-Type头

文件上传：支持自动生成随机boundary分隔符，支持上传多部分表单数据，也支持同时上传文件和普通表单字段，并且能自动设置Content-Type和Content-Length

1. download\_embedded()方法：

def download\_embedded(self, url: str, output\_dir: str = 'downloads') -> None:

    ...

功能：下载网页内嵌资源

过程：先下载主HTML页面并保存为index.html，再通过正则表达式扫描HTML内容，识别以下嵌入资源：

图片（img）

JavaScript文件（script）

CSS文件（link）

CSS中的URL引用（url()）

然后将相对URL转换为绝对URL，再批量下载所有资源到指定目录

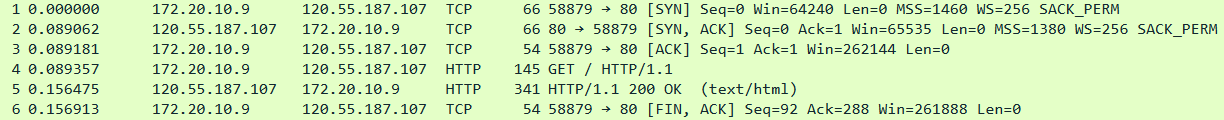
关键：可以自动创建输出目录，并智能生成文件名，使用unquote()处理URL编码的文件名，对无扩展名的资源根据Content-Type自动添加扩展名。支持相对URL自动转换处理，有错误处理机制保证单个资源下载失败不影响整体流程

1. 测试及结果分析
2. GET请求测试

运行客户端程序，客户端请求http://120.55.187.107首页

客户端私网地址：172.20.10.9

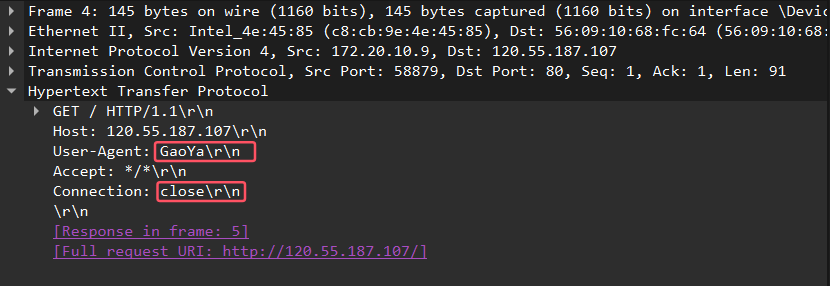
服务器公网地址：120.55.187.107

报文1-3：客户端与服务器进行TCP三次握手连接

报文4：客户端向服务器发送GET请求

User-Agent为GaoYa

请求的连接方式为close

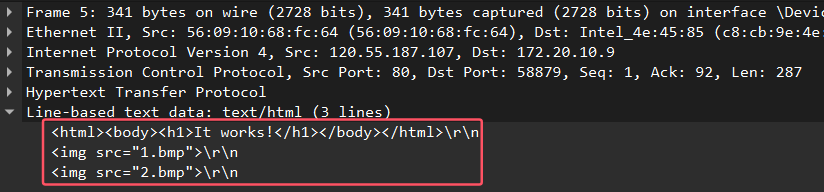


报文5：服务器向客户端返回HTTP应答报文

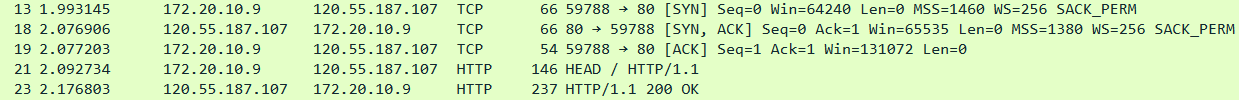
状态码为200，表示请求成功

返回的响应体内容类型为text/html

响应体中html内容即为WWW服务器的html文件内容

后续报文客户端与服务器基于TCP协议进行明文通信，验证了GET请求功能正常

1. HEAD请求测试

前三条报文：客户端与服务器进行TCP三次握手连接

报文21：客户端向服务器发送HEAD请求

报文23：服务器向客户端返回HTTP应答报文

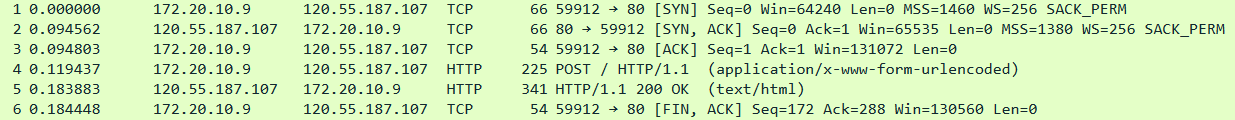
状态码为200，表示请求成功

只返回响应头，不返回响应体

HEAD请求功能正常

1. POST请求测试

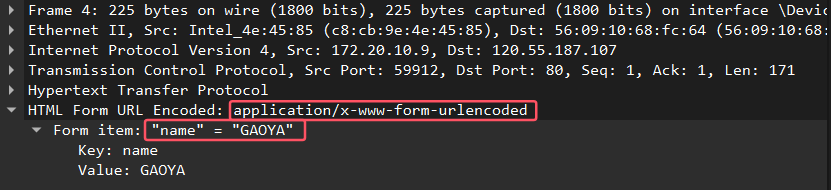
访问http://120.55.187.107，附加表单数据{"name":"GAOYA"}

报文1-3：客户端与服务器进行TCP三次握手连接

报文4：客户端向服务器发送POST请求

发送表单数据{"name":"GAOYA"}到服务器

application/x-www-form-urlencoded指提交简单表单



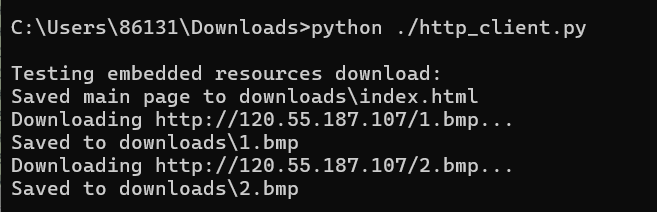
报文53：服务器向客户端返回HTTP应答报文

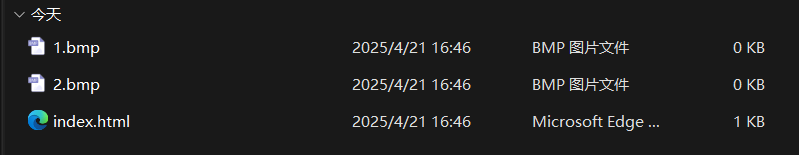
状态码为200，表示服务器能正确处理表单数据

POST请求和表单编码功能正常

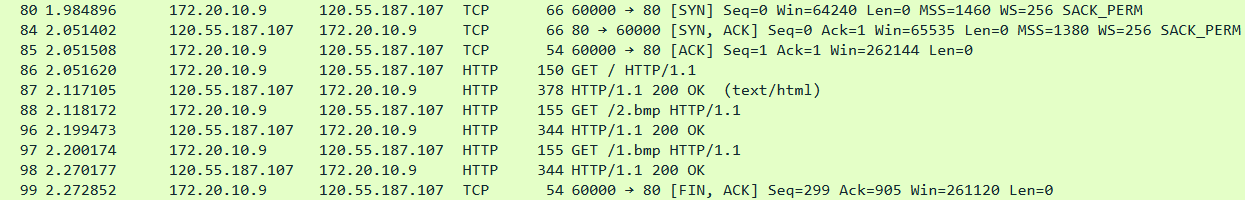
1. 把一个网页中所有的内嵌对象一次全部获取测试，以及keep-alive连接测试

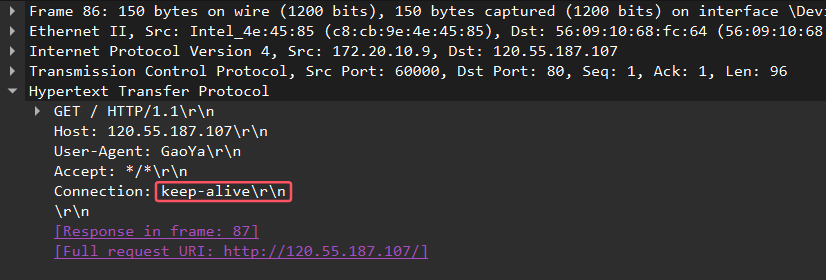
（1）请求WWW服务器http://120.55.187.107



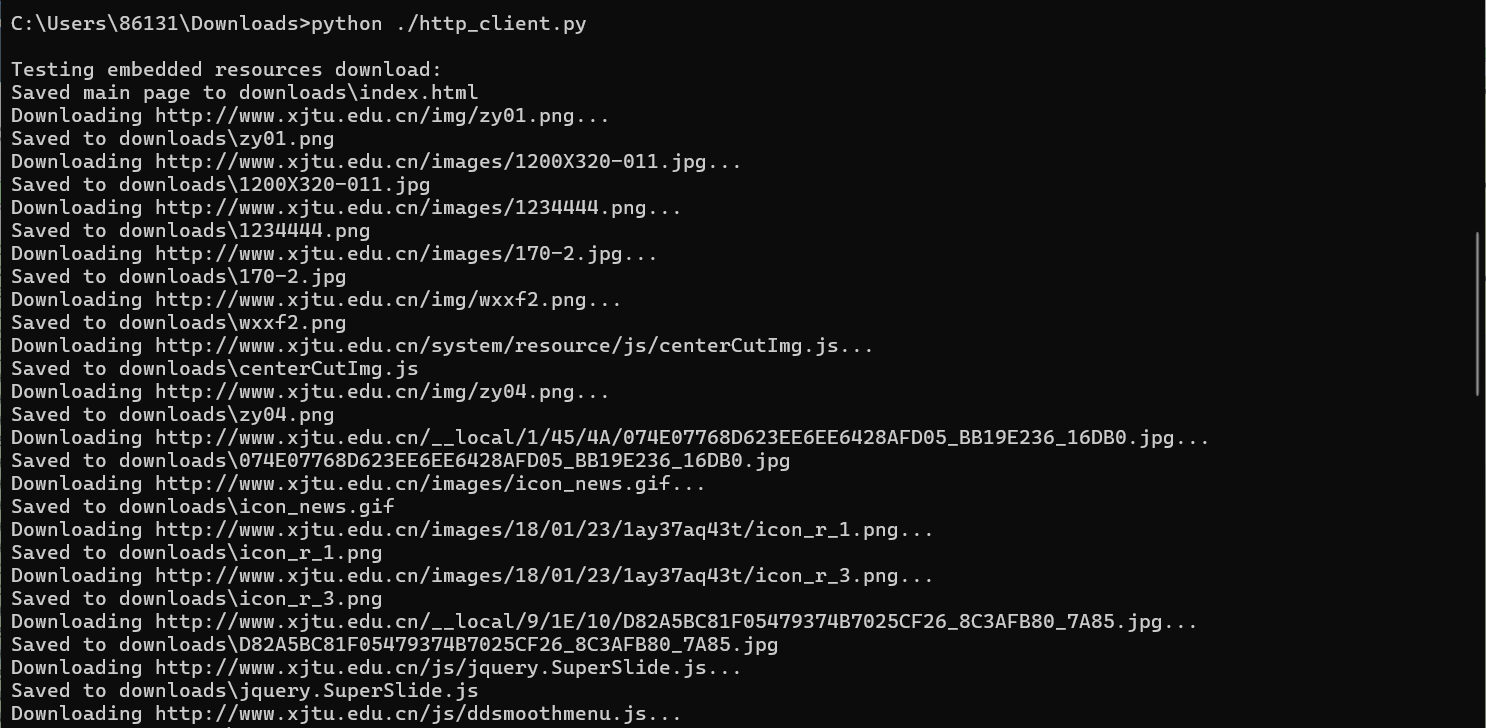


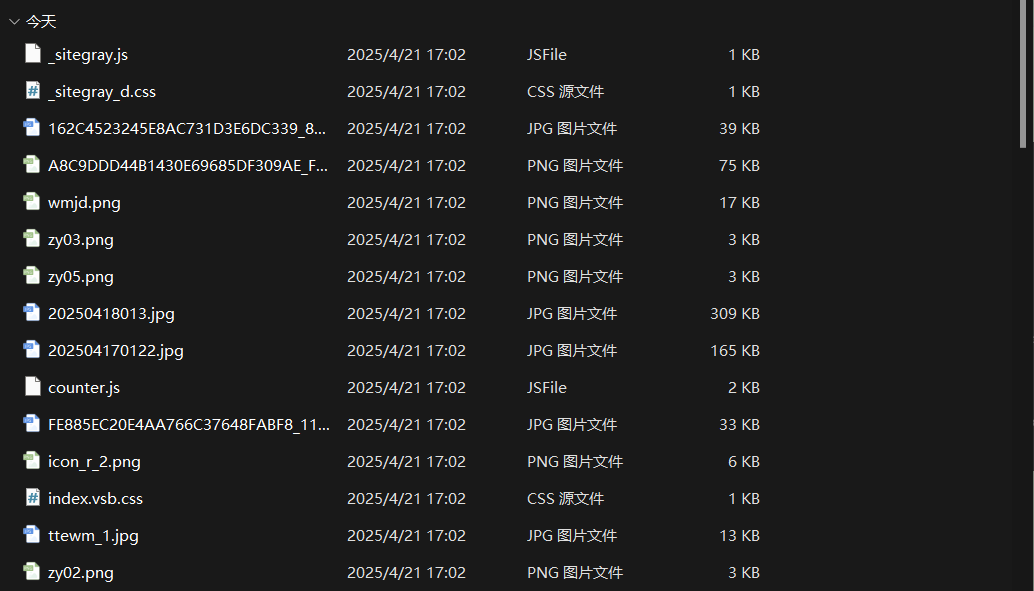
程序在当前目录创建了一个名为“download”的文件夹，并下载了网页中所有内嵌对象

从抓包分析，服务器与客户端建立TCP连接，设置连接类型为keep\_alive，然后依次请求网页内嵌的html、bmp文件，并得到响应，成功下载到本地，并且在几次GET请求之间复用TCP连接，不同再重新建立连接



（2）请求http://www.xjtu.edu.cn





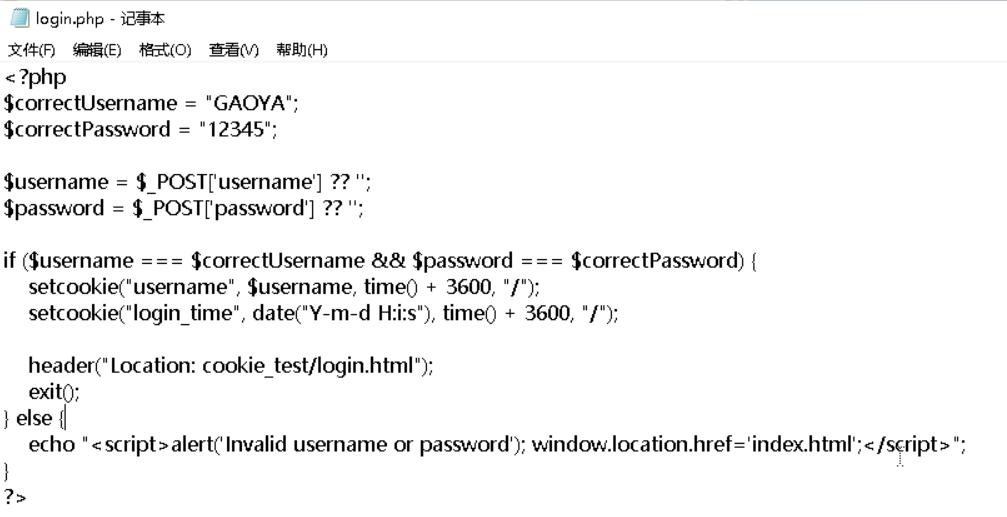


成功一次下载网页中所有IMG、CSS、JS等对象

1. 支持Cookie的基本机制，实现典型网站登录测试，以及重定向功能测试

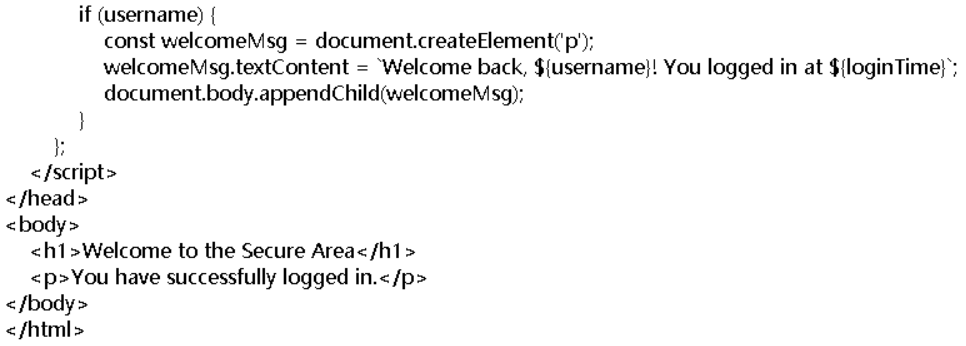
在服务器/cookie\_test目录下创建login.php文件，用于处理登录验证，文件中定义了正确的用户名GAOYA和密码12345，可以接收客户端通过POST方法提交的表单数据，验证成功则重定向到欢迎页面，失败则返回错误信息

内容如下：

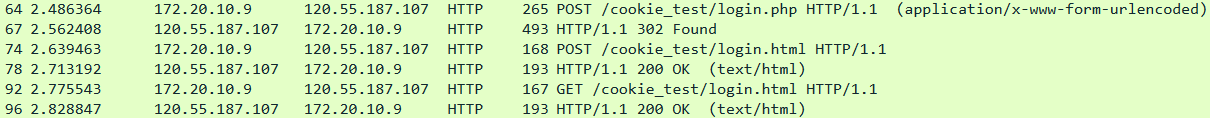


在服务器/cookie\_test目录下创建login.html文件，内容如下：

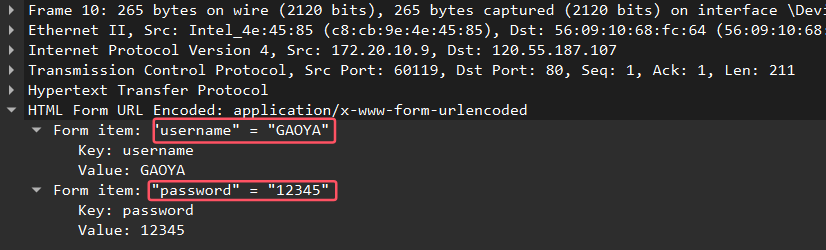




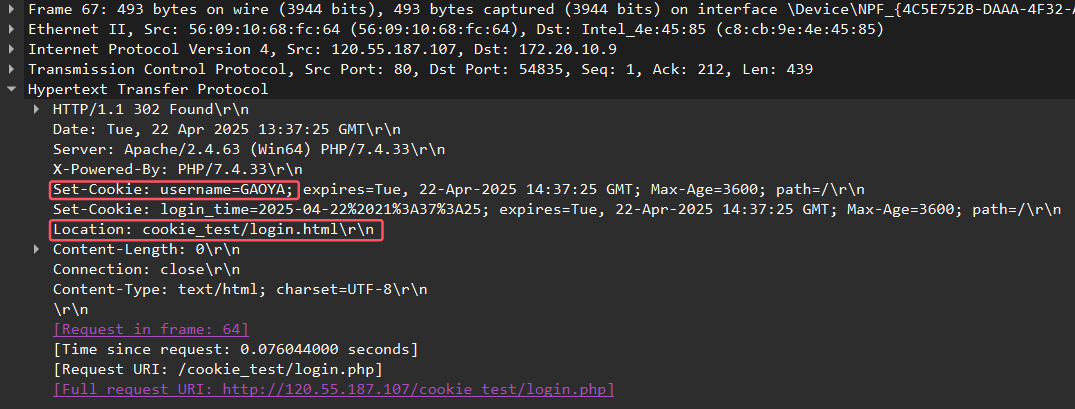
客户端发送两次请求，第一次POST请求提交正确的用户名和密码表单数据，第二次使用GET请求访问登陆后的页面



报文64：客户端向服务器发送POST请求，提交用户名GAOYA和密码12345表单数据



报文67：服务器收到请求后验证用户名密码正确，返回302 Found重定向响应，响应头中包含 Location: cookie\_test/login.html，Set-Cookie: username=GAOYA

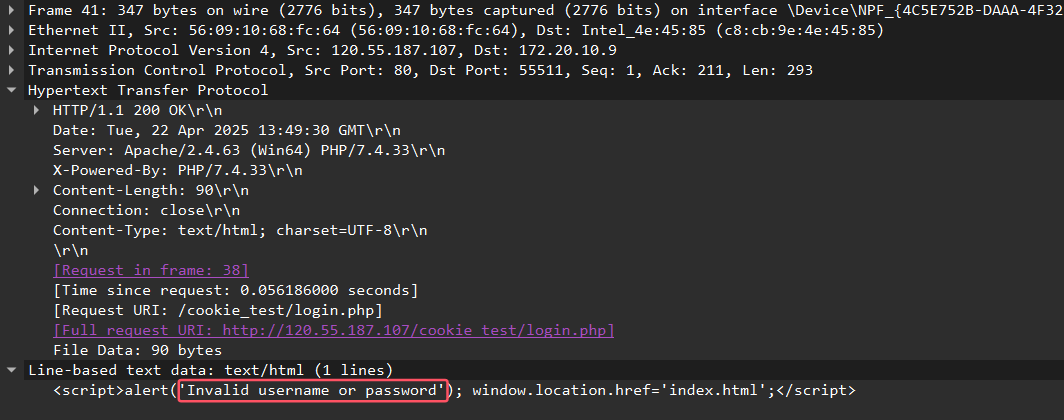


报文74、78：客户端自动跟随重定向，向新地址发送请求，服务器返回200 OK和login.html的内容

报文92、96：第二次直接向登陆后的页面发送GET请求，服务器返回200 OK和login.html的内容，表示请求成功

如果提交错误的用户名和密码表单：

客户端能够成功访问login.php，但用户名密码不匹配，服务器返回错误信息Invalid username or password



1. 能够正确处理几种典型的应答测试

上文已测试200、302应答

测试100：访问https://httpbin.org/status/100



测试301：访问<http://github.com>

会重定向到https://github.com



测试404：访问一个错误的地址

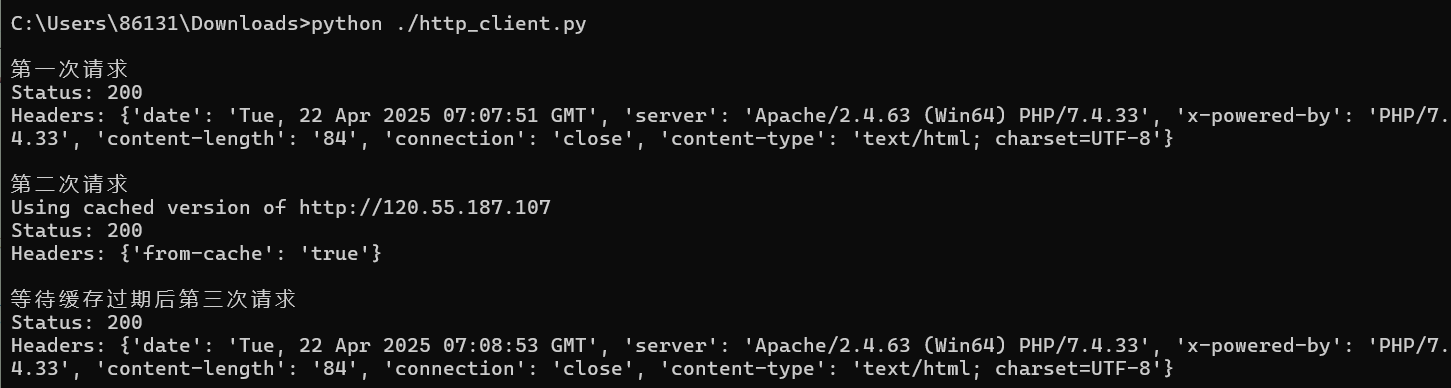


测试500：访问<https://httpbin.org/status/500>

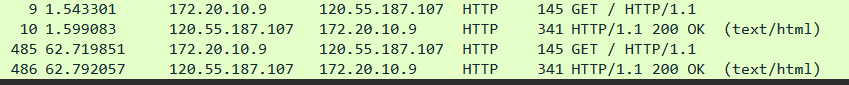


1. 支持基本的缓存处理测试

第一次访问http://120.55.187.107，紧接着第二次访问，此时应该从缓存读取，等60秒缓存失效后第三次访问



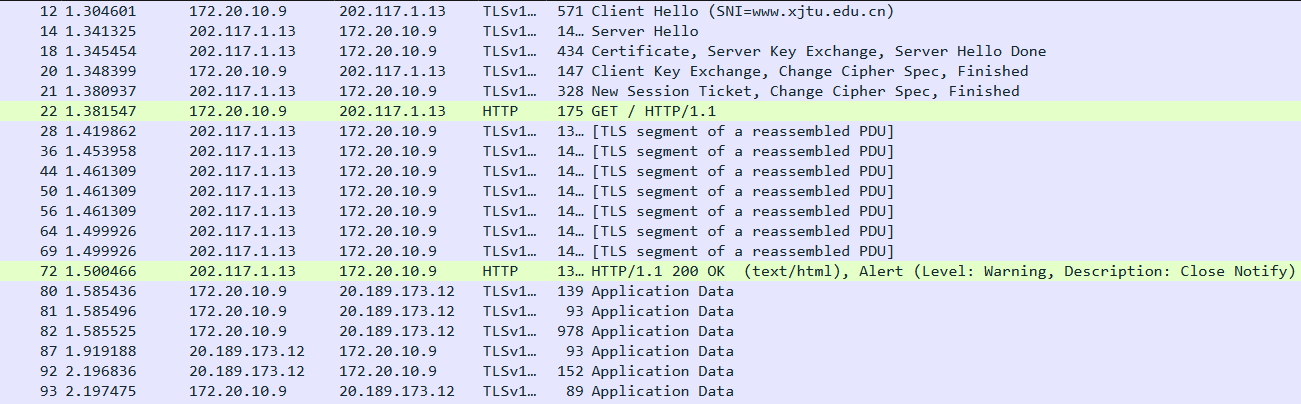
只有第二次访问收到的响应头中出现了'from-cache': 'true'



从抓包结果看，客户端只在第一次访问和缓存失效后访问网页才向服务器发送了GET请求，第二、三次访问时间相差60秒。说明第二次访问从缓存中读取了数据。

1. HTTPS请求测试

访问https://www.xjtu.edu.cn

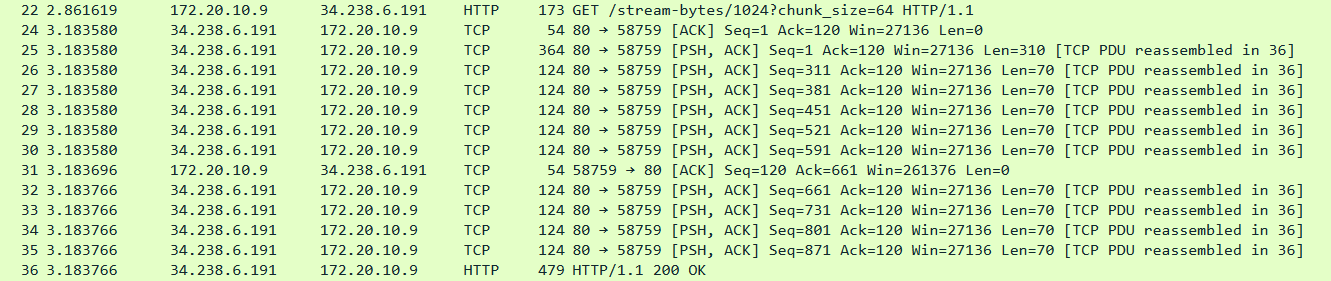


客户端发送HTTP请求前，先与服务器进行TLS握手，前五行报文可以看到完整的握手过程，包含ClientHello、ServerHello、Certificate、Key Exchange、Finished。

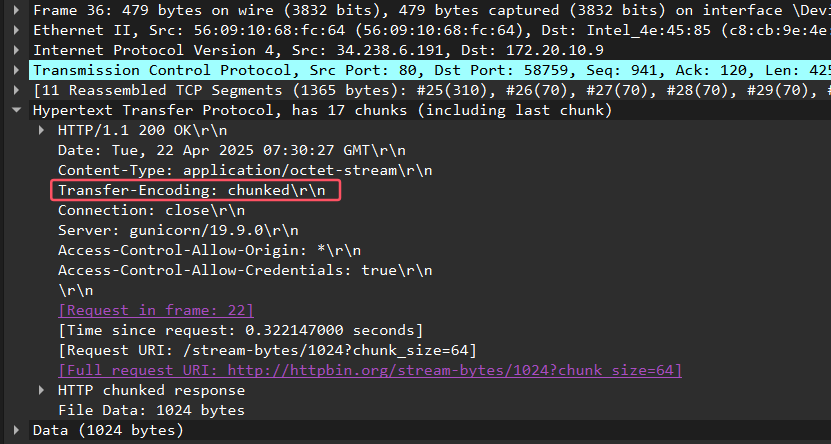
后续只能看到Application Data，而无法看到通信内容，说明浏览器支持HTTPS加密通信。

1. 分块传输编码、gzip内容编码测试

访问http://httpbin.org/stream-bytes/1024?chunk\_size=64，后面附上了参数chunk\_size=64，要求分块传输，每块大小为64B



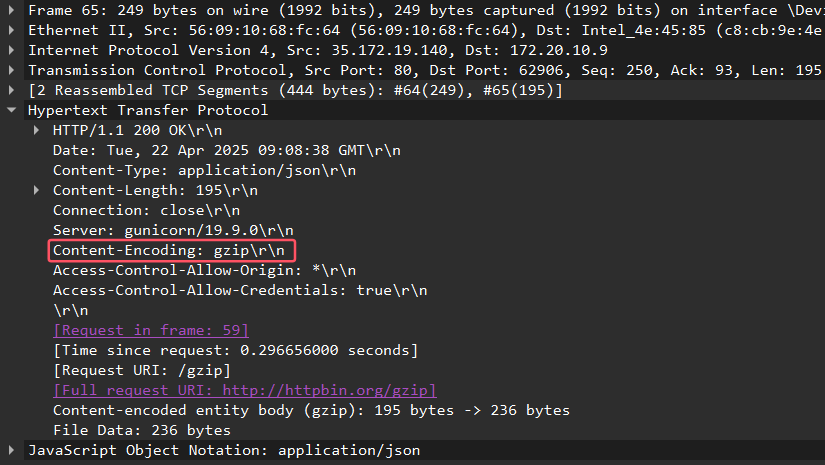
报文36：HTTP响应头中出现了Transfer-Encoding: chunked



访问http://httpbin.org/gzip



报文65：HTTP响应头中出现了Content-Encoding: gzip



1. 基于POST方法的文件上传测试

在服务器/upload目录下增加upload.php文件，用于接收客户端通过HTTP POST请求上传的文件，并将其保存到服务器上的指定目录，内容如下：



向服务器上传a.txt文件

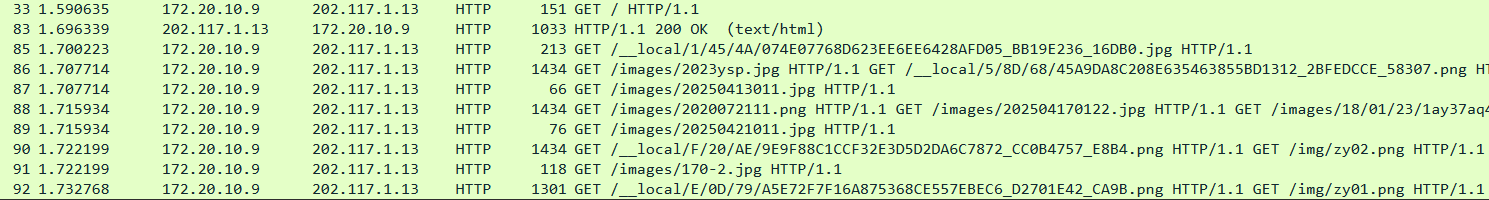




成功上传

1. 把一个网页中特定对象一次全部获取测试

获取http://www.xjtu.edu.cn中所有img对象



1. 实验结论

本次实验，我们成功实现了一个HTTP客户端程序，并且尽可能多的拓展了一些功能：

支持HTTP/1.1基本协议功能，包括GET、POST、HEAD方法

实现了Cookie管理、缓存处理

支持HTTPS安全连接和内容编码

具备文件上传和网页资源下载能力

完善错误处理和重定向机制

1. 总结及心得体会

实验过程中，我们深入理解了HTTP协议细节和Socket编程技术，掌握了浏览器的基本原理和实现方法，完成了从设计到实现的全过程，达到了预期的实验目标。

附件

1.源码文件

import socket

import re

import os

import ssl

from urllib.parse import urlparse, unquote, quote

from typing import Dict, List, Tuple, Optional

import zlib

import time

class HTTPClient:

    def \_\_init\_\_(self):

        self.cookies: Dict[str, str] = {}

        self.cache: Dict[str, Tuple[bytes, float]] = {}

        self.user\_agent = "GaoYa"

        self.default\_timeout = 10

        self.keep\_alive = False

        self.connection = None

    def \_connect(self, host: str, port: int, ssl\_context: bool = False) -> socket.socket:

        """建立与服务器的连接"""

        if self.keep\_alive and self.connection:

            return self.connection

        sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        sock.settimeout(self.default\_timeout)

        if ssl\_context:

            context = ssl.create\_default\_context()

            sock = context.wrap\_socket(sock, server\_hostname=host)

        sock.connect((host, port))

        if self.keep\_alive:

            self.connection = sock

        return sock

    def \_close\_connection(self):

        """关闭当前连接"""

        if self.connection:

            try:

                self.connection.close()

            except:

                pass

            self.connection = None

    def \_parse\_url(self, url: str) -> Tuple[str, int, str, bool]:

        """解析URL，返回主机、端口、路径和是否使用SSL"""

        parsed = urlparse(url)

        scheme = parsed.scheme.lower()

        host = parsed.hostname

        port = parsed.port

        path = parsed.path or '/'

        if parsed.query:

            path += '?' + parsed.query

        if not port:

            port = 443 if scheme == 'https' else 80

        use\_ssl = (scheme == 'https')

        return host, port, path, use\_ssl

    def \_decode\_chunked(self, data: bytes) -> bytes:

        """解码分块传输编码的数据"""

        result = b''

        while data:

            chunk\_size\_hex, \_, chunk = data.partition(b'\r\n')

            try:

                chunk\_size = int(chunk\_size\_hex, 16)

            except ValueError:

                break

            if chunk\_size == 0:

                break

            chunk\_start = len(chunk\_size\_hex) + 2

            chunk\_end = chunk\_start + chunk\_size

            result += data[chunk\_start:chunk\_end]

            data = data[chunk\_end + 2:]

        return result

    def \_decompress(self, data: bytes, encoding: str) -> bytes:

        """解压缩数据"""

        if encoding == 'gzip':

            return zlib.decompress(data, 16 + zlib.MAX\_WBITS)

        elif encoding == 'deflate':

            return zlib.decompress(data)

        return data

    def \_parse\_headers(self, headers\_str: str) -> Dict[str, str]:

        """解析HTTP头"""

        headers = {}

        for line in headers\_str.split('\r\n'):

            if ':' in line:

                key, value = line.split(':', 1)

                headers[key.strip().lower()] = value.strip()

        return headers

    def \_parse\_cookies(self, headers: Dict[str, str]):

        """正确处理多个 Set-Cookie 头（即使服务器合并成一个）"""

        if 'set-cookie' in headers:

            # 处理所有 Set-Cookie 头（Python 的 headers 可能合并多个同名头）

            set\_cookie\_lines = headers['set-cookie'].split(', ')

            for cookie\_line in set\_cookie\_lines:

                # 提取 name=value（忽略属性）

                cookie = cookie\_line.split(';')[0]

                if '=' in cookie:

                    name, value = cookie.split('=', 1)

                    self.cookies[name.strip()] = value.strip()

    def \_build\_request(self, method: str, path: str, host: str, headers: Dict[str, str] = None,

                       body: bytes = None) -> bytes:

        """构建HTTP请求"""

        if headers is None:

            headers = {}

        # 添加默认头

        headers['Host'] = host

        headers['User-Agent'] = self.user\_agent

        headers['Accept'] = '\*/\*'

        headers['Connection'] = 'keep-alive' if self.keep\_alive else 'close'

        # 添加Cookie

        if self.cookies:

            cookie\_str = '; '.join([f'{k}={v}' for k, v in self.cookies.items()])

            headers['Cookie'] = cookie\_str

        # 构建请求

        request = f'{method} {path} HTTP/1.1\r\n'

        for key, value in headers.items():

            request += f'{key}: {value}\r\n'

        request += '\r\n'

        if body:

            request = request.encode() + body

        else:

            request = request.encode()

        return request

    def \_parse\_response(self, sock: socket.socket) -> Tuple[int, Dict[str, str], bytes]:

        """解析HTTP响应"""

        # 读取响应头

        response = b''

        while b'\r\n\r\n' not in response:

            try:

                data = sock.recv(4096)

                if not data:

                    break

                response += data

            except socket.timeout:

                break

        header\_end = response.find(b'\r\n\r\n')

        if header\_end == -1:

            raise ValueError("Invalid HTTP response")

        headers\_str = response[:header\_end].decode('latin-1')

        body = response[header\_end + 4:]

        # 解析状态行

        status\_line = headers\_str.split('\r\n')[0]

        status\_code = int(status\_line.split()[1])

        # 解析头

        headers = self.\_parse\_headers(headers\_str)

        # 处理分块传输

        if 'transfer-encoding' in headers and 'chunked' in headers['transfer-encoding'].lower():

            body = self.\_decode\_chunked(body)

        # 处理内容编码

        if 'content-encoding' in headers:

            body = self.\_decompress(body, headers['content-encoding'].lower())

        # 如果Content-Length存在但body不完整，继续读取

        if 'content-length' in headers:

            content\_length = int(headers['content-length'])

            while len(body) < content\_length:

                try:

                    data = sock.recv(4096)

                    if not data:

                        break

                    body += data

                except socket.timeout:

                    break

        return status\_code, headers, body

    def \_handle\_redirect(self, method: str, url: str, headers: Dict[str, str], max\_redirects: int = 5) -> Tuple[

        int, Dict[str, str], bytes]:

        """处理重定向"""

        if max\_redirects <= 0:

            raise ValueError("Too many redirects")

        location = headers['location']

        if not location.startswith(('http://', 'https://')):

            # 相对路径重定向

            parsed = urlparse(url)

            base\_url = f"{parsed.scheme}://{parsed.netloc}"

            location = base\_url + ('/' if not location.startswith('/') else '') + location

        print(f"Redirecting to: {location}")

        return self.request(method, location, max\_redirects=max\_redirects - 1)

    def request(self, method: str, url: str, headers: Dict[str, str] = None, body: bytes = None,

                max\_redirects: int = 5) -> Tuple[int, Dict[str, str], bytes]:

        """发送HTTP请求"""

        if max\_redirects <= 0:

            raise ValueError("Too many redirects")

        # 解析URL

        host, port, path, use\_ssl = self.\_parse\_url(url)

        # 检查缓存

        cache\_key = f"{method}:{url}"

        if method.upper() == 'GET' and cache\_key in self.cache:

            cached\_data, expiry = self.cache[cache\_key]

            if expiry > time.time():

                print(f"Using cached version of {url}")

                return 200, {'from-cache': 'true'}, cached\_data

        # 建立连接

        sock = self.\_connect(host, port, use\_ssl)

        # 构建请求

        request = self.\_build\_request(method, path, host, headers, body)

        # 发送请求

        sock.sendall(request)

        # 获取响应

        status\_code, headers, body = self.\_parse\_response(sock)

        # 处理重定向

        if status\_code in (301, 302, 303, 307, 308):

            self.\_close\_connection()

            return self.\_handle\_redirect(method, url, headers, max\_redirects)

        # 处理Cookie

        self.\_parse\_cookies(headers)

        # 处理连接

        if not self.keep\_alive or headers.get('connection', '').lower() == 'close':

            self.\_close\_connection()

        # 缓存响应

        if method.upper() == 'GET' and status\_code == 200:

            cache\_control = headers.get('cache-control', '')

            if 'no-store' not in cache\_control and 'no-cache' not in cache\_control:

                max\_age = 30  # 默认缓存1小时

                if 'max-age' in cache\_control:

                    try:

                        max\_age = int(cache\_control.split('max-age=')[1].split(',')[0])

                    except (IndexError, ValueError):

                        pass

                self.cache[cache\_key] = (body, time.time() + max\_age)

        return status\_code, headers, body

    def get(self, url: str, save\_to: str = None) -> Tuple[int, Dict[str, str], bytes]:

        """发送GET请求"""

        status\_code, headers, body = self.request('GET', url)

        if save\_to:

            with open(save\_to, 'wb') as f:

                f.write(body)

        return status\_code, headers, body

    def head(self, url: str) -> Tuple[int, Dict[str, str], bytes]:

        """发送HEAD请求"""

        return self.request('HEAD', url)

    def post(self, url: str, data: Dict[str, str] = None, files: Dict[str, Tuple[str, bytes]] = None,

             save\_to: str = None) -> Tuple[int, Dict[str, str], bytes]:

        """发送POST请求"""

        body = None

        headers = {}

        if files:

            boundary = '----WebKitFormBoundary' + ''.join([str(i) for i in os.urandom(16)])

            #分隔符字符串

            headers['Content-Type'] = f'multipart/form-data; boundary={boundary}'

            body\_parts = []

            if data:

                for key, value in data.items():

                    body\_parts.append(

                        f'--{boundary}\r\nContent-Disposition: form-data; name="{key}"\r\n\r\n{value}\r\n')

# 文件上传

            for field\_name, (filename, file\_data) in files.items():

                body\_parts.append(

                    f'--{boundary}\r\nContent-Disposition: form-data; name="{field\_name}"; filename="{filename}"\r\n'

                    f'Content-Type: application/octet-stream\r\n\r\n'

                )

                body\_parts.append(file\_data)

                body\_parts.append(f'\r\n')

            body\_parts.append(f'--{boundary}--\r\n')

            body = b''.join([part.encode() if isinstance(part, str) else part for part in body\_parts])

        elif data:

            # 普通表单数据

            headers['Content-Type'] = 'application/x-www-form-urlencoded'

            body = '&'.join([f'{k}={quote(str(v))}' for k, v in data.items()]).encode()

        headers['Content-Length'] = str(len(body))

        status\_code, headers, body = self.request('POST', url, headers, body)

        if save\_to:

            with open(save\_to, 'wb') as f:

                f.write(body)

        return status\_code, headers, body

    def download\_embedded(self, url: str, output\_dir: str = 'downloads') -> None:

        """下载网页中的所有嵌入对象"""

        # 创建输出目录

        os.makedirs(output\_dir, exist\_ok=True)

        # 获取主页面

        status, headers, html = self.get(url)

        if status != 200:

            print(f"Failed to fetch main page: HTTP {status}")

            return

        # 保存主页面

        main\_filename = os.path.join(output\_dir, 'index.html')

        with open(main\_filename, 'wb') as f:

            f.write(html)

        print(f"Saved main page to {main\_filename}")

        # 查找嵌入资源

        html\_str = html.decode('utf-8', errors='ignore')

        # 查找各种资源

        patterns = {

            'img': r'<img[^>]+src="([^"]+)"',

            'script': r'<script[^>]+src="([^"]+)"',

            'link': r'<link[^>]+href="([^"]+)"',

            'css\_url': r'url\(["\']?([^)"\']+)["\']?\)',

        }

        resources = set()

        for tag, pattern in patterns.items():

            for match in re.finditer(pattern, html\_str, re.IGNORECASE):

                resource\_url = match.group(1)

                # 处理相对URL

                if not resource\_url.startswith(('http://', 'https://', 'data:')):

                    parsed = urlparse(url)

                    base\_url = f"{parsed.scheme}://{parsed.netloc}"

                    resource\_url = base\_url + ('/' if not resource\_url.startswith('/') else '') + resource\_url

                if not resource\_url.startswith('data:'):  # 忽略内联数据

                    resources.add(resource\_url)

        # 下载所有资源

        for i, resource\_url in enumerate(resources, 1):

            try:

                # 创建资源文件名

                parsed = urlparse(resource\_url)

                path = unquote(parsed.path)

                filename = os.path.basename(path) or f'resource\_{i}'

                # 添加扩展名

                if '.' not in filename:

                    content\_type = ''  # 可以从响应头获取

                    ext = {

                        'text/css': '.css',

                        'application/javascript': '.js',

                        'image/jpeg': '.jpg',

                        'image/png': '.png',

                        'image/gif': '.gif',

                    }.get(content\_type, '')

                    filename += ext

                save\_path = os.path.join(output\_dir, filename)

                print(f"Downloading {resource\_url}...")

                self.get(resource\_url, save\_path)

                print(f"Saved to {save\_path}")

            except Exception as e:

                print(f"Failed to download {resource\_url}: {e}")

if \_\_name\_\_ == '\_\_main\_\_':

    client = HTTPClient()

    base\_url = "http://120.55.187.107"

    # GET请求

    # status, headers, body = client.get(base\_url)

    # print(f"Status: {status}\n")

    # print(f"Headers: {headers}\n")

    # HEAD请求

    # status, headers, body = client.head(base\_url)

    # print(f"Status: {status}")

    # print(f"Headers: {headers}")

    # POST请求

    # post\_url = "http://120.55.187.107/cookie\_test/login.php"

    # login\_data = {'username': 'GAOYA', 'password': '1234'}

    # status, headers, body = client.post(post\_url, data=login\_data)

    #文件上传 (高级功能D)

    # with open('a.txt', 'rb') as f:

    #     file\_content = f.read()

    # url = 'http://120.55.187.107/upload/upload.php'

    # files = {'file': ('a.txt', file\_content)}

    # status\_code, headers, body = client.post(url, files=files)

    # print('Status Code:', status\_code)

    # 下载网页所有嵌入资源 (基本功能B)

    # client.keep\_alive = True

    # client.download\_embedded(base\_url)

    # 重定向测试 (基本功能D)

    # status, headers, body = client.get('http://github.com')

    # print(f"Final status: {status}")

    # 缓存测试 (基本功能E)

    # print("\n第一次请求")

    # status, headers, body = client.get(base\_url)

    # print(f"Status: {status}")

    # print(f"Headers: {headers}")

    # print("\n第二次请求")

    # status, headers, body = client.get(base\_url)

    # print(f"Status: {status}")

    # print(f"Headers: {headers}")

    # print("\n等待缓存过期后第三次请求")

    # time.sleep(61)  # 等待缓存过期

    # status, headers, body = client.get(base\_url)

    # print(f"Status: {status}")

    # print(f"Headers: {headers}")

2.参考资料

<https://blog.csdn.net/python03011/article/details/134699414>