

计算机网络专题实验 - 实验八报告

 姓名	 班级
熊原	计算机2201
李鑫瑞	计算机2204



一、实验名称



基于 HTTP 协议的客户端程序（文本浏览器）



二、实验原理

-  **TCP 原理：**

建立连接需三次握手，关闭连接需四次挥手。通过序列号、确认应答、窗口控制和重传机制，确保数据可靠有序传输。同时采用流量控制和拥塞控制优化网络传输。

-  **C/S 模式工作原理：**

-  **服务器端：**

创建 Socket → 绑定端口 → 监听连接 → 接受客户端 → 通信 → 验证用户信息 → 返回验证结果 → 关闭连接

-  **客户端：**

创建 Socket → 发起连接 → 输入用户名密码 → 等待验证 → 通信完成 → 主动关闭连接



三、实验目的

1.  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

四、实验内容

1. 基本功能

- 客户端程序基于标准的**HTTP/1.1**协议（**RFC 2616**等），能够把请求得到的资源保存在本地的文件中。

基本要求：

- 支持**GET**、**HEAD**和**POST**三种请求方法，支持**URI**的"%HEXHEX"编码，如对 <http://abc.com:80/~smith/> 和 <http://ABC.com/%7Esmith/> 两种等价的**URI**能够正确处理；支持**Connection: Keep-Alive**和**Connection: Close**两种连接模式；
- ⌚ 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；
- 🍪 支持**Cookie**（见**RFC 2109**）的基本机制，实现典型的网站登录；
- 👉 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；
- 📁 支持基础缓存机制

2. 高级功能

- 🔒 支持**HTTPS**；
- ✳️ 支持分块传输编码(**Chunked Transfer Encoding**)、**gzip**等内容编码；
- 📤 支持基于**POST**方法的文件上传；
- 🎯 支持把一个网页中特定对象（如多个**PDF**资源）一次全部获取。

五、实验实现

1. 人员分工

熊原：

- 完成了服务器端的相关配置以及POST服务中服务端代码
- 完成了项目整体框架设计
- 实现了 `cache.py` (缓存机制), `client.py` (客户端实现), `http_requset.py` (**POST**方法实现), `http_ui.py` (**POST**中不同类型文件选择部分ui实现)
- 测试部分： `test_uri` (**url**解析测试)，同时，协助测试代码**bug**解决
- 测试报告撰写

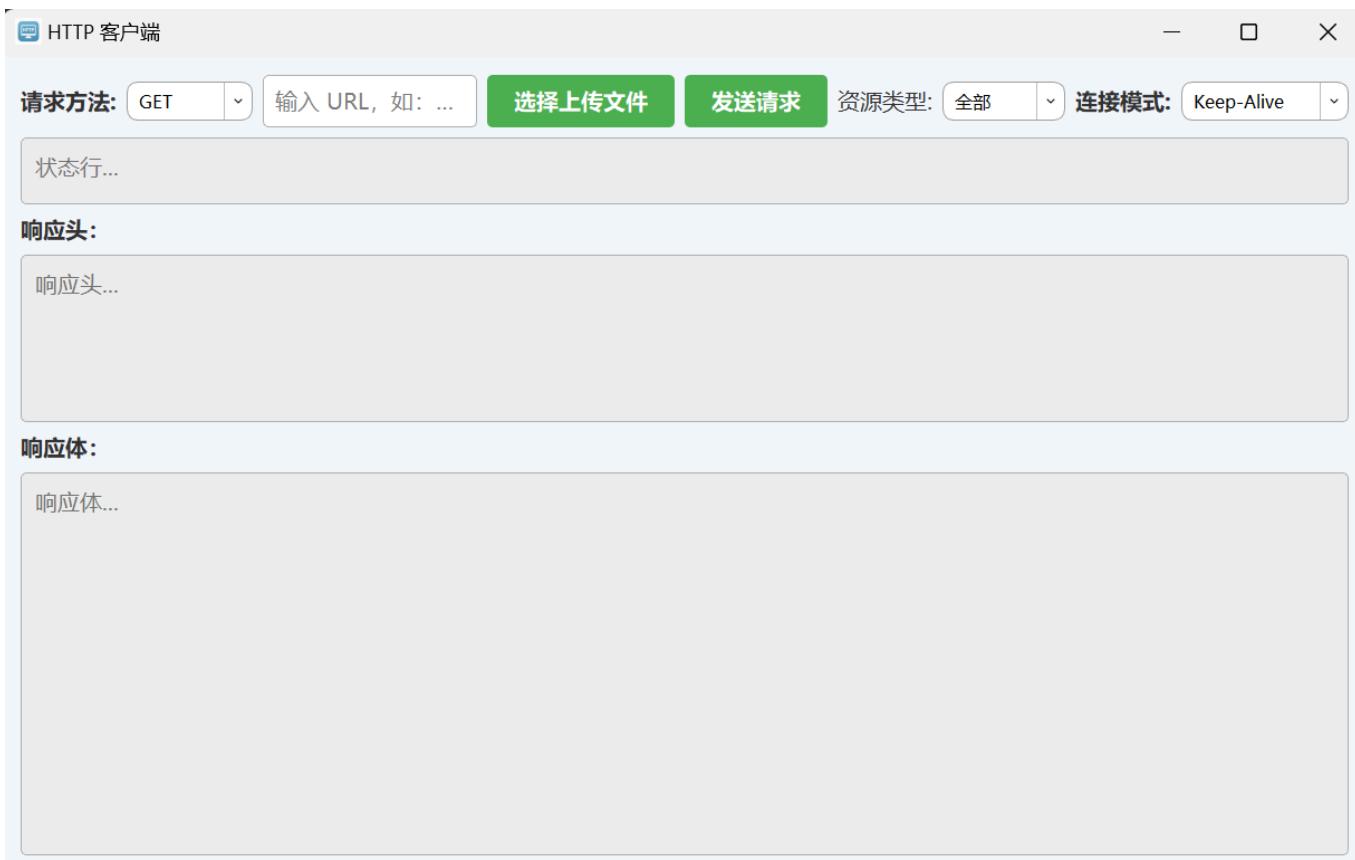


李鑫瑞：

- 实验报告撰写，部分测试报告撰写
- 实现了 `cookie_jar.py` (**cookie**机制的实现), `http_request.py` (**GET HEAD** 两种方法的接口), `http_response.py` (**chunked**传输, **Gzip**解码的实现, 重定向判断接口), `http_ui.py` (**ui**的设计与实现)
- 测试部分: `test_chunked.py` (**chunked**传输的测试), `test_cookie.py` (**cookie**注入, 存储, 访问的测试), `test_gzip.py` (**Gzip**解码测试), `test_request.py` (**GET**请求的测试), `test_response.py` (**Http**回复测试)

2. 实验设计

UI设计



设计如上：我们支持了请求方法（**GET HEAD POST**）的选择，支持输入URL以进行访问，在请求方法为**POST**时，支持在本地文件中选择需要上传的文件，同时支持选择文件类型来协助文件上传。同时支持连接模式的**Keep-Alive**和**Close**之间的选择。在返回部分，我们设计了**状态行**, **响应头**, **响应体**三个返回框，可以在用户发送请求后，返回**HTTP**报文的返回状态，响应头，和原始的响应体内容。

框架结构

我们的项目结构如下：

```
HTTP_CLIENT
|   cache.py
|   client.py
|   cookie_jar.py
|   http_request.py
|   http_response.py
|   uri_utils.py
|   utils.py
|   http_ui.py
|
├─report
|   └─picture
|       └─ui.png
|
└─tests
    |   test_cache.py
    |   test_chunked.py
    |   test_cookie.py
    |   test_gzip.py
    |   test_request.py
    |   test_response.py
    └─ test_uri.py
|
|
|
└─ui
    |
    └─picture
        Arrow-down.svg
        icon.png
```

📦 模块说明

模块文件名	功能描述
client.py	项目的核心模块，负责建立 HTTP/HTTPS 连接构造请求、发送请求、接收响应、处理缓存、重定向、Cookie、资源下载等。
http_request.py	提供 GET / HEAD / POST 请求报文的构造函数，支持文件上传、表单编码等。

模块文件名	功能描述
<code>http_response.py</code>	解析响应报文，提取状态行、响应头和响应体，支持分块传输（ <code>chunked</code> ）、 <code>gzip</code> 解压等，同时支持判断 HTTP 状态码是否为重定向，也支持获取重定向的 URL
<code>cookie_jar.py</code>	管理 Cookie 的存储与生命周期，实现从响应中提取和向请求头中注入 Cookie 。
<code>cache.py</code>	实现简单的缓存机制，支持根据URL计算缓存文件的路径，可以判断 URL 是否存在缓存，支持获取缓存的 <code>Last-Modified</code> 和 <code>ETag</code> ，用于 <code>If-Modified-Since</code> / <code>If-None-Match</code> ，支持存储 HTTP 响应到缓存，同时也支持从缓存加载响应体
<code>uri_utils.py</code>	实现 URI 的解析、标准化，确保资源定位和处理正确。
<code>utils.py</code>	存放通用工具函数，例如文件保存、路径生成、类型判断、编码处理，解析相对 <code>URL</code> ，转换成完整的绝对 <code>URL</code> 。

UI 模块说明

模块/目录	功能描述
<code>ui/http_ui.py</code>	图形化用户界面主程序，使用 PyQt5 实现功能选择、 URL 输入、响应展示。
<code>ui/picture/</code>	存放图形界面所使用的图标与资源图片。

测试模块说明

测试脚本	测试内容
<code>test_cache.py</code>	测试缓存模块的读写、条件缓存请求（ <code>If-Modified-Since</code> ）逻辑。
<code>test_chunked.py</code>	测试分块传输（ <code>Chunked Transfer Encoding</code> ）内容的解析正确性。
<code>test_cookie.py</code>	测试 Cookie 的注入与提取机制。
<code>test_gzip.py</code>	测试在显性设置以 <code>gzip</code> 形式传输报文的情况下是否能正常传输和解压缩
<code>test_request.py</code>	测试 GET 是否能够正常执行，以此确定URL的解析是否正确
<code>test_response.py</code>	测试 HTTP 响应能否正常解析，测试获取重定向地址。
<code>test_uri.py</code>	测试 URI 工具函数的解析、重定向地址拼接、编码/解码等功能。

3. 关键代码的描述

(一) 关键代码1:

🔧 send_request

自主编写：大部分由熊原编写，李鑫瑞做了部分修改和添加

```
def send_request(uri, method="GET", body=None, file_path=None, depth=0,
resource_types=None):
    """发送 HTTP/HTTPS 请求，支持缓存与重定向"""
    if depth > 5:
        print("[!] 重定向次数过多")
        return

    uri_parsed = parse_uri(uri)
    host, port = get_host_port(uri_parsed)

    headers = {}
    inject_default_headers(headers, host, keep_alive=False,
user_agent=USER_AGENT)

    # 如果存在缓存，添加 `If-Modified-Since` 和 `If-None-Match` 头
    if is_cached(uri):
        cached_headers = get_cached_headers(uri)
        headers.update({k: v for k, v in cached_headers.items() if v})

    CookieJar.inject_into_headers(COOKIE_JAR, headers, uri)
    print("COOKIE_JAR:", COOKIE_JAR.getcookies())
    # 处理文件上传
    if method == "POST" and file_path:
        body, extra_headers = build_multipart_form_data(file_path)
        headers.update(extra_headers)

    # 构造请求
    if method == "GET":
        request = build_get_request(uri_parsed, headers)
    elif method == "HEAD":
        request = build_head_request(uri_parsed, headers)
    elif method == "POST":
        request = build_post_request(uri_parsed, body or "", headers,
file_path)
    else:
        raise ValueError("Unsupported HTTP method")

    use_https = uri_parsed.scheme == "https"
    sock = make_connection(host, port, use_https=use_https)
    # 发送请求
```

```
if isinstance(request, str):
    print("request:\n" + request)
    sock.sendall(request.encode("utf-8")) # 仅字符串需要 encode
else:
    print("request为bytes")
    sock.sendall(request) # 如果已经是 bytes, 直接发送

status_line, headers, body = read_response(sock)
sock.close()

if method == "HEAD":
    body = None

# 提取cookie
COOKIE_JAR.extract_from_headers(headers)

# 处理 304 Not Modified, 直接返回缓存内容
status_code = int(status_line.split()[1])
if status_code == 304:
    print("[CACHE] 服务器返回 304 Not Modified, 使用缓存")
    if method == "HEAD":
        return status_line, get_cached_headers(uri), None
    return status_line, get_cached_headers(uri), load_cached_body(uri)

# 处理重定向
if is_redirect(status_code):
    location = get_redirect_location(headers)
    redirect_uri = resolve_relative_url(uri, location)
    print(f"[+] 重定向到 {redirect_uri}")
    return status_line, headers, send_request(redirect_uri, method, body,
                                              depth + 1, resource_types=resource_types)

# 存储缓存
store_response(uri, headers, body)

# 保存主页面
save_to_file(uri_parsed, body, headers.get("Content-Type"))

# 如果是 HTML 页面则提取资源并下载
content_type = headers.get("Content-Type", "")
if "text/html" in content_type and method == "GET":
    download_embedded_resources(body, uri, resource_types=resource_types)
```

```
    return status_line, headers, body
```

该部分具有以下功能：

- 支持 **GET/HEAD/POST** 三种 HTTP 方法
- 处理文件上传 (**POST**)
- 支持 **Cookie**、缓存、重定向
- 可以自动下载 **HTML** 中嵌套的资源 (如 **CSS**、**JS**、图片)
- 支持最大重定向深度控制

(二) 关键代码2



parse_uri

自主编写：由熊原编写

```
def parse_uri(uri_str):  
    """  
        手动解析 URI 字符串，返回 URI 对象  
        支持 http://host:port/path?query  
    """  
  
    uri_str = uri_str.strip()  
  
    # 1. 提取 scheme  
    if "://" not in uri_str:  
        raise ValueError("URI 缺少 scheme")  
    scheme, rest = uri_str.split("://", 1)  
  
    # 2. 提取 host[:port]  
    path_start = rest.find("/")  
    if path_start == -1:  
        host_port = rest  
        path_query = ""  
    else:  
        host_port = rest[:path_start]  
        path_query = rest[path_start:]  
  
    # 3. 提取 host 和 port  
    if ":" in host_port:  
        host, port_str = host_port.split(":", 1)  
        port = int(port_str)  
    else:  
        host = host_port
```

```

        if scheme == 'http':
            port = 80
        if scheme == 'https':
            port = 443

    # 4. 提取 path 和 query
    if "?" in path_query:
        path, query = path_query.split("?", 1)
    else:
        path = path_query
        query = ""

    return URI(scheme, host.lower(), port, normalize_path(path), query)

```

该部分具有以下功能：

- 拆解 **scheme://host:port/path?query** 格式 **URI**
- 默认端口设置：**http=80, https=443**
- 为后续构造请求做准备（尤其用于分离主机与路径）

(三) 关键代码3

GET HEAD POST

自主编写：由李鑫瑞编写

```

# 构造 GET 请求
def build_get_request(uri, headers=None):
    path = uri.path or "/"
    if uri.query:
        path += f"?{uri.query}"

    request_line = f"GET {path} HTTP/1.1"
    default_headers = {
        "Host": uri.host,
        "User-Agent": "XiongYuan and LiXinRui",
        "Connection": "Keep-Alive"
    }

    if headers:
        default_headers.update(headers)

    headers_str = "\r\n".join(f"{key}: {value}" for key, value in
default_headers.items())
    return f"{request_line}\r\n{headers_str}\r\n\r\n"

```

```
# 构造 HEAD 请求 (同 GET, 但无 Body)
def build_head_request(uri, headers=None):
    path = uri.path or "/"
    if uri.query:
        path += f"?{uri.query}"

    request_line = f"HEAD {path} HTTP/1.1"
    default_headers = {
        "Host": uri.host,
        "User-Agent": "XiongYuan and LiXinRui",
        "Connection": "Keep-Alive"
    }

    if headers:
        default_headers.update(headers)

    headers_str = "\r\n".join(f"{key}: {value}" for key, value in
default_headers.items())
    return f"{request_line}\r\n{headers_str}\r\n\r\n"

# 构造 POST 请求
def build_post_request(uri, data, headers=None, file_path=None):
    path = uri.path or "/"
    if uri.query:
        path += f"?{uri.query}"

    request_line = f"POST {path} HTTP/1.1"

    default_headers = {
        "Host": uri.host,
        "User-Agent": "XiongYuan and LiXinRui",
        "Connection": "Keep-Alive"
    }

    # 处理 JSON 数据
    if data and not file_path:
        body = json.dumps(data)
        default_headers["Content-Type"] = "application/json"
        default_headers["Content-Length"] = str(len(body))

    # 处理文件上传
    elif file_path:
        boundary = "----WebKitFormBoundary7MA4YWxkTrZu0gW"
```

```

        file_name = os.path.basename(file_path)
        mime_type = mimetypes.guess_type(file_name)[0] or "application/octet-
stream"

        with open(file_path, "rb") as f:
            file_content = f.read()

        body = (
            f"--{boundary}\r\n"
            f'Content-Disposition: form-data; name="file"; filename="
{file_name}"\r\n'
            f"Content-Type: {mime_type}\r\n\r\n"
        ).encode() + file_content + f"\r\n--{boundary}--\r\n".encode()

        default_headers["Content-Type"] = f"multipart/form-data; boundary=
{boundary}"
        default_headers["Content-Length"] = str(len(body))

    else:
        raise ValueError("POST 请求需要提供 data 或 file_path")

# 合并用户自定义 Headers
if headers:
    default_headers.update(headers)

headers_str = "\r\n".join(f"{key}: {value}" for key, value in
default_headers.items())

# 如果是 JSON, 则返回字符串; 如果是文件上传, 则返回二进制数据
if file_path:
    request = f"{request_line}\r\n{headers_str}\r\n\r\n\r\n".encode() + body
else:
    request = f"{request_line}\r\n{headers_str}\r\n\r\n\r\n{body}"

return request

```

该部分具有以下功能：

- 三个函数分别构建 **GET / HEAD / POST** 请求报文
- 支持手动追加 **Headers**, 如 **User-Agent Connection**
- **POST** 支持上传 **JSON** 数据 和 文件 (表单 **multipart** 上传)
- 文件上传时自动判断 **MIME** 类型

(四) 关键代码4

🔧 chunked Gzip

自主编写：李鑫瑞编写

```
def handle_chunked_body(body, sock):
    decoded_body = b""
    while True:
        chunk_size_line = read_line(sock)
        try:
            chunk_size = int(chunk_size_line, 16) # 16 进制解析 chunk 大小
        except ValueError:
            print(f"Skipping invalid chunk size: {repr(chunk_size_line)}")
            continue # 跳过无效的 chunk size
        if chunk_size == 0:
            break # 结束处理
        chunk_data = recv_exact(sock, chunk_size)
        sock.recv(2) # 读取 \r\n
        decoded_body += chunk_data
        print(f"[Chunk] {chunk_size} bytes:
{chunk_data.decode(errors='ignore')}") # 打印 chunk 内容
    return decoded_body

def decompress_gzip(body):
    """解压 Gzip 响应体"""
    try:
        return gzip.decompress(body)
    except Exception as e:
        print(f"[x] Gzip 解压失败: {e}")
    return body # 返回原始数据
```

该部分具有以下功能：

- `handle_chunked_body` 用于逐块解析 **chunked** 编码响应体，自动跳过非法块
- `decompress_gzip` 解压 **Gzip** 编码的内容，自动降级处理异常

(五) 关键代码5

🔧 cache

自主编写：由熊原编写

```
def get_cached_headers(uri):
    """获取缓存的 'Last-Modified' 和 'ETag'，用于 If-Modified-Since / If-None-Match"""
    pass
```

```

cache_path = _get_cache_path(uri)

if not os.path.exists(cache_path):
    return {}

with open(cache_path, "r", encoding="utf-8") as f:
    cache_data = json.load(f)

return {
    "If-Modified-Since": cache_data.get("Last-Modified"),
    "If-None-Match": cache_data.get("ETag"),
}
}

def store_response(uri, headers, body):
    """存储 HTTP 响应到缓存"""
    cache_path = _get_cache_path(uri)

    cache_data = {
        "Last-Modified": headers.get("Last-Modified"),
        "ETag": headers.get("ETag"),
        "Date": headers.get("Date"),
        "Body": body.decode("utf-8", errors="ignore"), # 转换为字符串
    }

    with open(cache_path, "w", encoding="utf-8") as f:
        json.dump(cache_data, f, indent=4)

def load_cached_body(uri):
    """从缓存加载响应体"""
    cache_path = _get_cache_path(uri)
    with open(cache_path, "r", encoding="utf-8") as f:
        cache_data = json.load(f)

    return cache_data["Body"].encode("utf-8")

```

该部分具有以下功能：

- 使用本地缓存目录保存请求响应的 **ETag / Last-Modified / Body**
- 可通过请求头 **If-Modified-Since** 和 **If-None-Match** 实现 **HTTP** 缓存机制
- 节省流量、提升响应速度



六、测试及结果分析

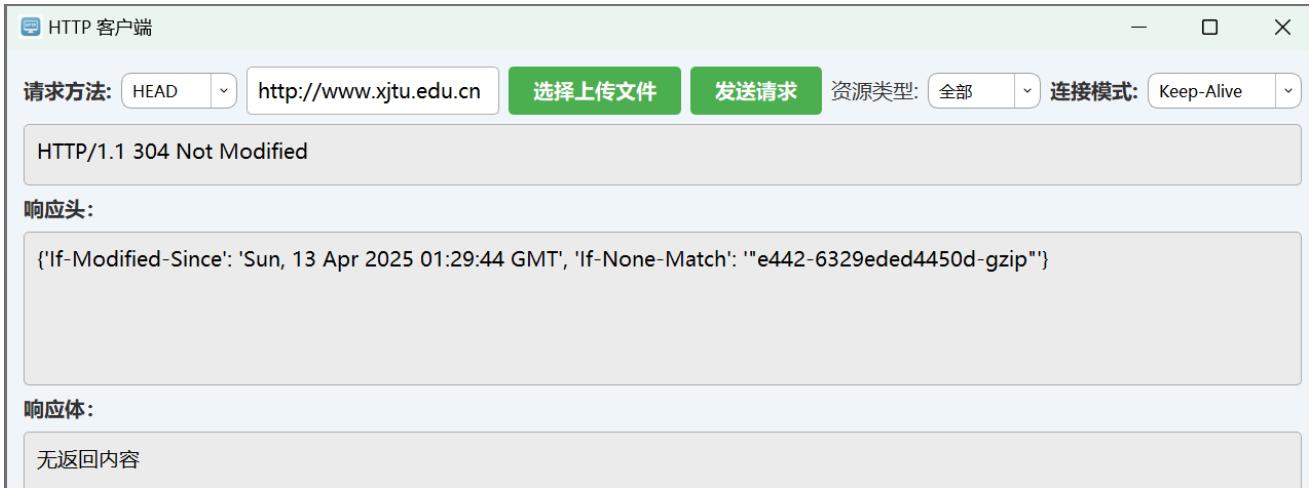
1. GET 请求测试

使用常规 GET 请求测试客户端的请求与响应功能。

如图，访问了学校网页，成功返回了对应的响应头和响应体

2. HEAD 请求测试

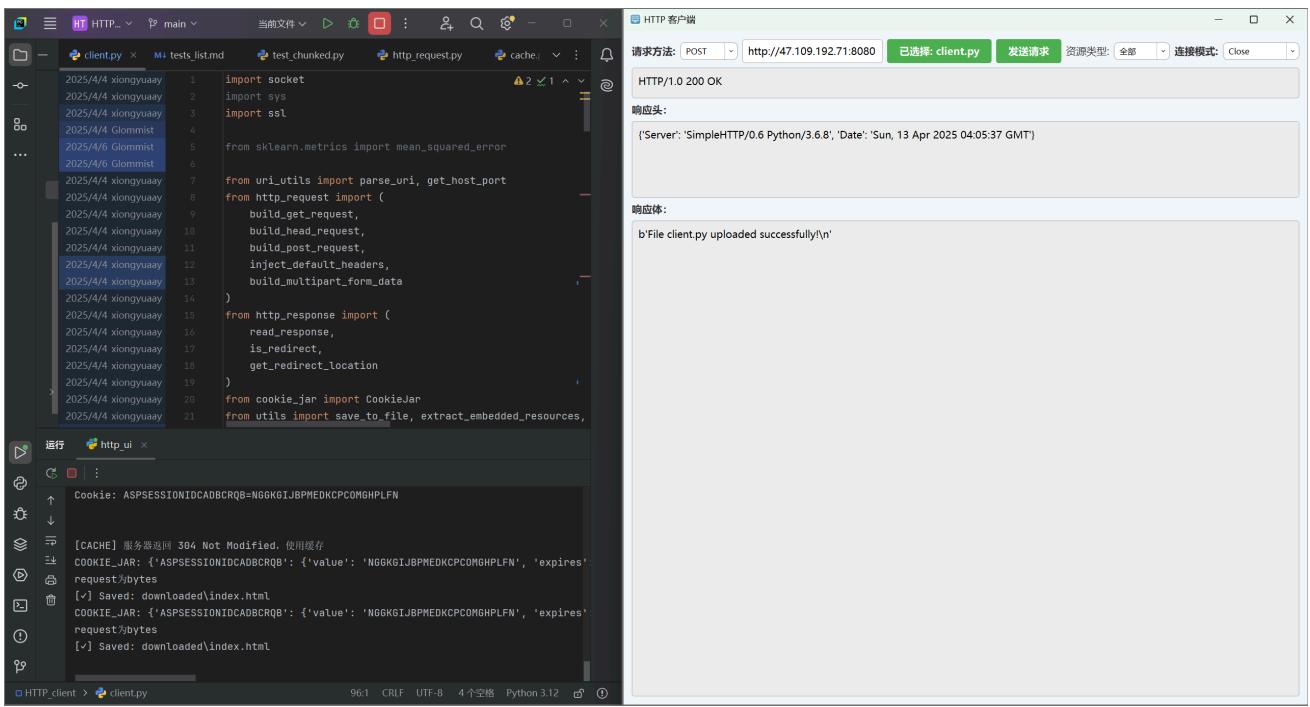
验证是否能仅请求响应头部信息，无响应体内容。



如图，使用HEAD时，由于之前已经访问过了对应的网站，此时再次使用HEAD访问时，会触发缓存，但是HEAD的使用没有问题。

3. POST 请求测试

包括文件上传、多字段提交等 POST 操作，验证参数解析和服务器响应情况。



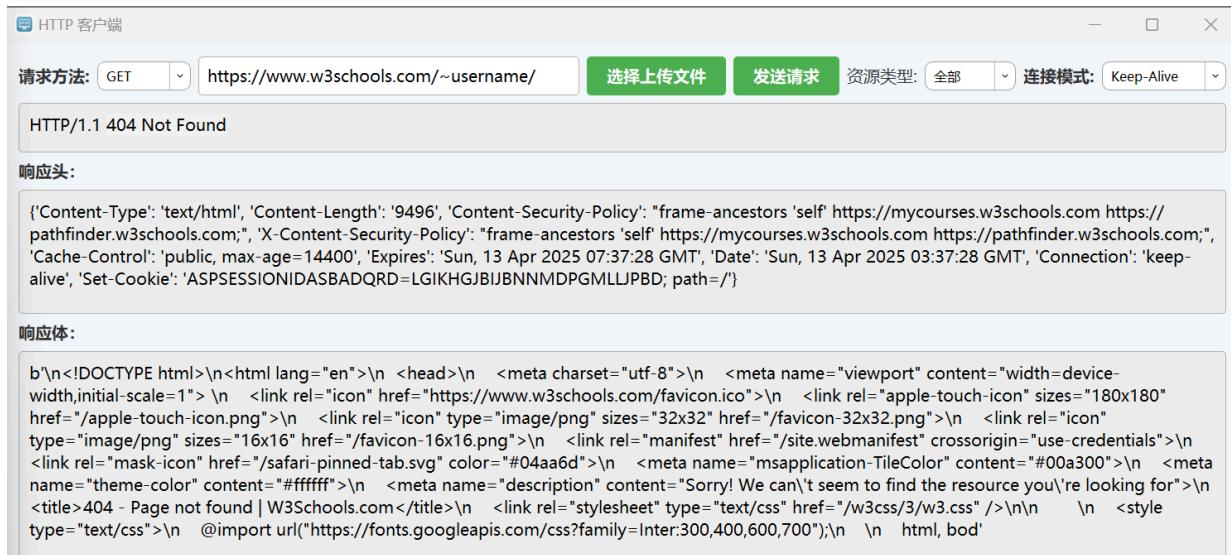
如图，使用阿里云服务器，通过编写python代码搭建简易的服务器，通过客户端发送POST请求，选择上传本地的client.py文件，通过响应体可见POST请求使用正常。

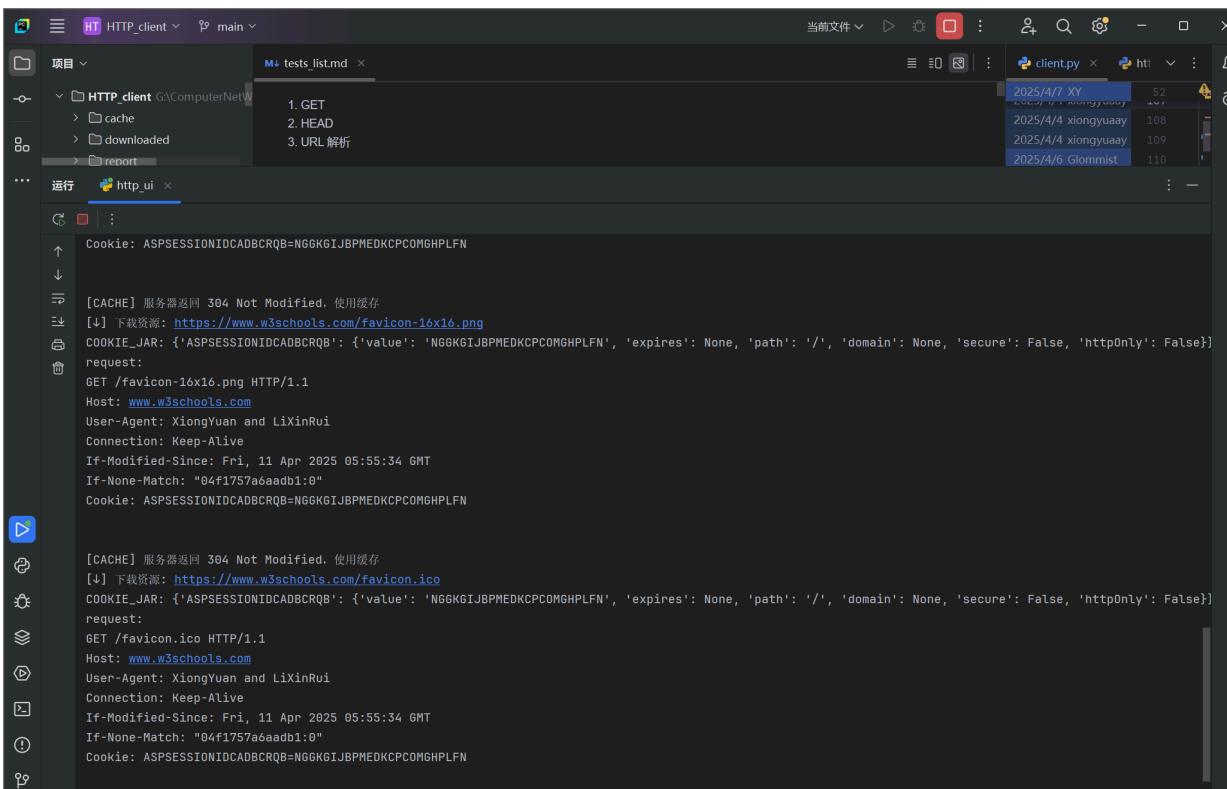
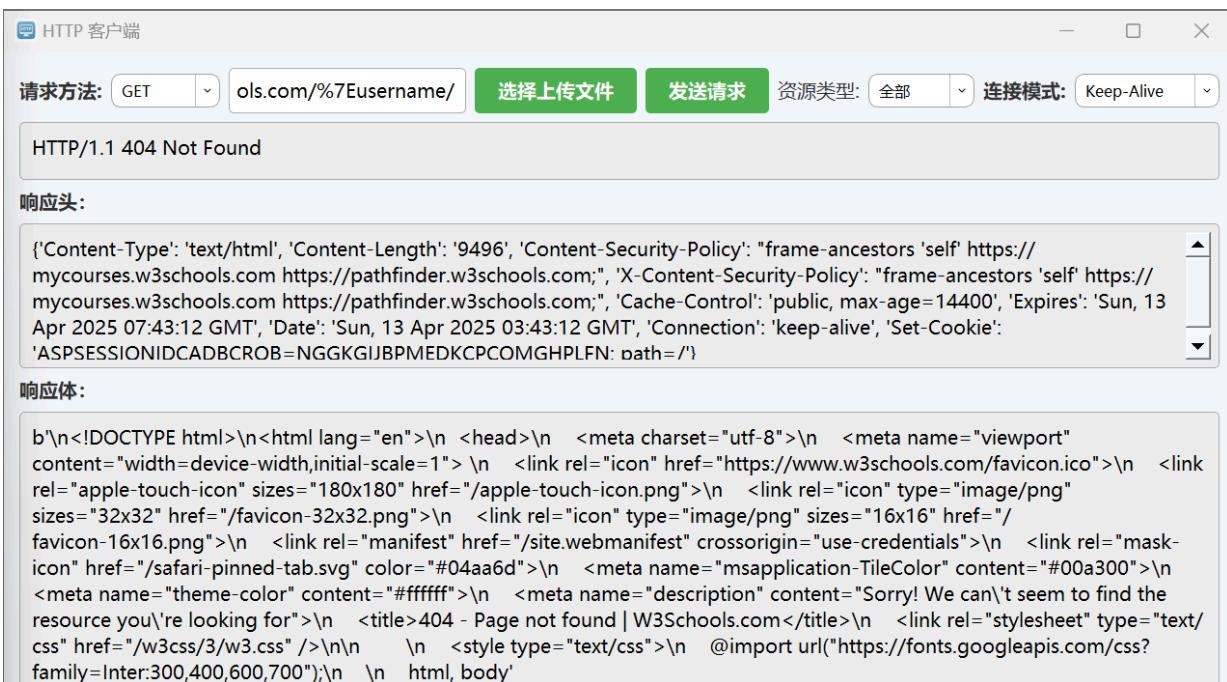
4. URL 解析功能测试

正确处理包含特殊字符的 URL 编码。

- 示例 URL：

- <https://www.w3schools.com/~username/>
- <https://www.w3schools.com/%7Eusername/>





如图可见，通过两种不同的URL进行访问，获得了同样的返回内容，说明客户端可以成功解析这两种URL。此处的404状态值是由于网页请求资源本身不存在所导致的，但访问的页面是同一个。

5. 应答码测试

验证不同状态码的处理能力。

- 500 错误码: <https://httpstat.us/500>

The screenshot shows an HTTP client interface with the following details:

- 请求方法:** GET
- 资源地址:** https://httpstat.us/500
- 响应头:**

```
{"Content-Length": "25", "Content-Type": "text/plain", "Date": "Sun, 13 Apr 2025 03:45:08 GMT", "Server": "Kestrel", "Set-Cookie": "ARRAffinitySameSite=1c4ce6b282a4edef63e94171500d99e8b18888422937ab7168b9007141be8730; Path=/; HttpOnly; SameSite=None; Secure; Domain=httpstat.us", "Strict-Transport-Security": "max-age=2592000", "Request-Context": "appId=cid-v1:3548b0f5-7f75-492f-82bb-b6eb0e864e53"}
```
- 响应体:**

```
b'500 Internal Server Error'
```

- 3xx 重定向: <https://www.python.org>

The screenshot shows an HTTP client interface with the following details:

- 请求方法:** GET
- 资源地址:** http://www.python.org
- 响应头:**

```
{"Connection": "close", "Content-Length": "0", "Server": "Varnish", "Retry-After": "0", "Location": "https://www.python.org/", "Accept-Ranges": "bytes", "Date": "Sun, 13 Apr 2025 03:27:36 GMT", "Via": "1.1 varnish", "X-Served-By": "cache-itm1220055-ITM", "X-Cache": "HIT", "X-Cache-Hits": "0", "X-Timer": "S1744514857.852620,VS0,VE0", "Strict-Transport-Security": "max-age=63072000; includeSubDomains; preload"}
```
- 响应体:**

```
('HTTP/1.1 200 OK', {'Connection': 'keep-alive', 'Content-Length': '12079', 'via': '1.1 varnish, 1.1 varnish, 1.1 varnish', 'x-frame-options': 'SAMEORIGIN', 'content-type': 'text/html; charset=utf-8', 'content-encoding': 'gzip', 'Accept-Ranges': 'bytes', 'Date': 'Sun, 13 Apr 2025 03:27:37 GMT', 'Age': '492', 'X-Served-By': 'cache-iad-kiad7000081-IAD, cache-iad-kiad7000081-IAD, cache-itm1220030-ITM', 'X-Cache': 'MISS, HIT, HIT', 'X-Cache-Hits': '0, 429, 3', 'X-Timer': 'S1744514857.312072,VS0,VE0', 'Vary': 'Cookie', 'Strict-Transport-Security': 'max-age=63072000; includeSubDomains; preload'}, b'')
```

如图，客户端对不同的返回值可以正确的返回结果。对于500，会显示出错误代码。对于301，会自动重定向，最终成功访问目标网站。

6. 连接模式测试

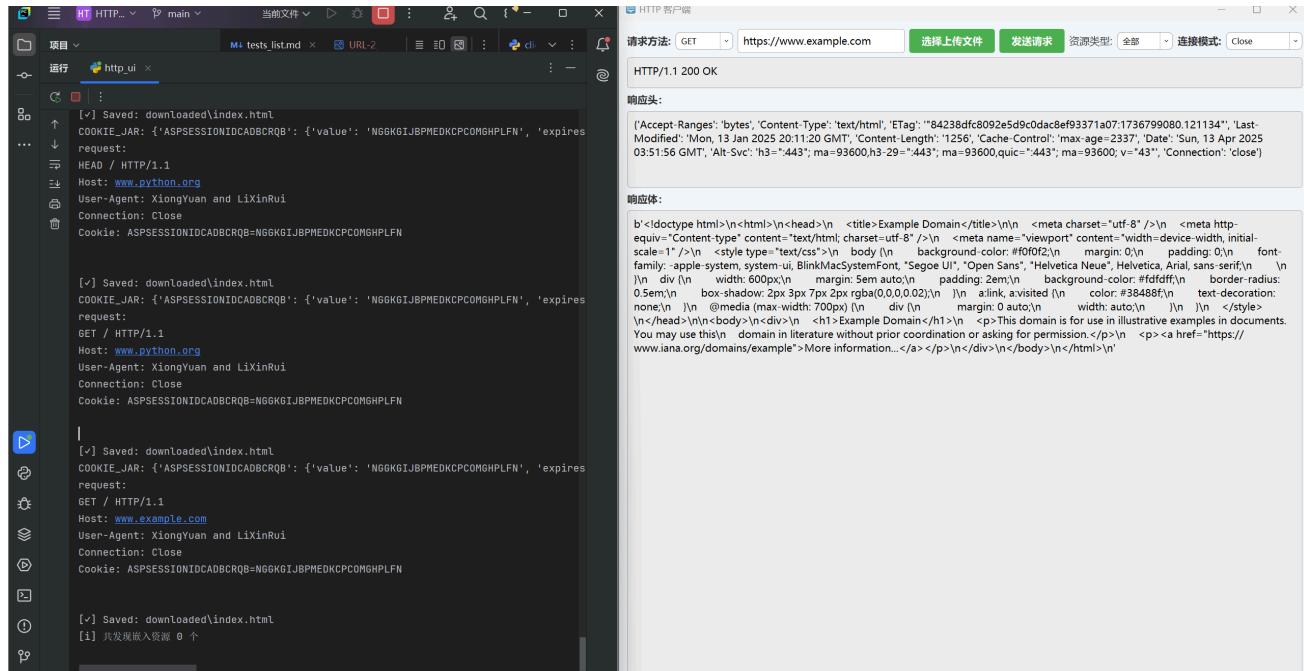
支持 Keep-Alive 与 Close 模式。

如图，这两种连接模式都能够正常使用。

7. HTTPS 支持测试

检查是否能通过 TLS 访问 HTTPS 网站。

- 示例网站：<https://www.python.org>



如图可见，对于https的网址，该客户端程序仍然成功的返回了正确的响应内容，状态码为200。

8. 重定向处理测试

客户端能否自动跳转到目标地址。



如图，对于网址<http://www.python.org>，返回码是301，在客户端中成功通过重定向，得到了正确的URL，并返回了正确值。

9. Chunked 传输测试

使用 chunked 分块传输响应的测试。

- 测试脚本： test_chunked.py

The screenshot shows a PyCharm IDE window. The project navigation bar at the top has 'HTTP_client' selected. Below it, there are several files: HEAD.png, Https.png, KEEPALIVE.png, URL-1.png, URL-2 1mm, tests.list.md, test_chunked.py, URL-2.png, and 300.png. The main code editor contains the following Python script:

```
E:\python\py\python.exe: No module named test.test_chunked
PS G:\ComputerNetwork\Lab\Lab8\HTTP_client> python -m tests.test_chunked
COOKIE_JAR: {}
request:
GET /stream/20 HTTP/1.1
Host: httpbin.org
User-Agent: XiongYuan and LiXinRui
Connection: Close

[Chunk] 458 bytes: {"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 7}
{"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 8}

[Chunk] 229 bytes: {"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 9}
[Chunk] 230 bytes: {"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 10}
[Chunk] 230 bytes: {"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 11}
[Chunk] 460 bytes: {"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 12}
{"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c0b29fb9c87", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 13}
```

The terminal output shows the execution of the script and the resulting chunks of data received from the server. The status bar at the bottom indicates the file is 'HTTP_client > tests > test_chunked.py'.

如图，成功实现了chunked 分块传输。

10. Gzip 解码测试

是否能够自动识别并解码 gzip 编码的响应体。

- 测试脚本： test_gzip.py

```

[Chunk] 230 bytes: {"url": "http://httpbin.org/stream/20", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67fb3578-1489ff9f3e9a3c8b29fb9c07", "User-Agent": "XiongYuan and LiXinRui"}, "origin": "113.201.132.190", "id": 19}

[*] Saved: downloaded\20.json
PS G:\ComputerNetwork\lab\lab8\HTTP_client> python -m tests.test_gzip
[*] HTTP 状态行: HTTP/1.1 200 OK
[*] 前应头: {'Date': 'Sun, 13 Apr 2025 03:55:37 GMT', 'Content-Type': 'application/json', 'Content-Length': '210', 'Connection': 'close', 'Server': 'unicorn/19.9.0', 'Content-Encoding': 'gzip', 'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Credentials': 'true'}
[*] 响应是 gzip 压缩的
[*] 解压成功，内容如下:
{
    "gzipped": true,
    "headers": {
        "Accept-Encoding": "gzip",
        "Host": "httpbin.org",
        "User-Agent": "CustomClient/1.0",
        "X-Amzn-Trace-Id": "Root=1-67fb35b9-26a31c9544f15297502746fa"
    },
    "method": "GET",
    "origin": "113.201.132.190"
}
PS G:\ComputerNetwork\lab\lab8\HTTP_client>

```

如图，成功使用gzip编码，并解码得到了正确的响应内容。

11. Cookie 机制测试

验证客户端是否能正确处理和发送 Cookie。

- 测试脚本： test_cookie.py

```

[TEST] 发送请求以设置 Cookie
... COOKIE_JAR: {}
request:
GET /cookies/set?session_id=123456 HTTP/1.1
Host: httpbin.org
User-Agent: XiongYuan and LiXinRui
Connection: Close

[*] 重定向到 http://httpbin.org/cookies
COOKIE_JAR: {'session_id': {'value': '123456', 'expires': None, 'path': '/', 'domain': None, 'secure': False, 'httpOnly': False}}
request:
GET /cookies HTTP/1.1
Host: httpbin.org
User-Agent: XiongYuan and LiXinRui
Connection: Close
[TEST] 发送请求以读取服务器存储的 Cookie
COOKIE_JAR: {'session_id': {'value': '123456', 'expires': None, 'path': '/', 'domain': None, 'secure': False, 'httpOnly': False}}
request:
GET /cookies HTTP/1.1
Host: httpbin.org
User-Agent: XiongYuan and LiXinRui
Connection: Close
Cookie: session_id=123456

[*] Saved: downloaded\cookies.json
PS G:\ComputerNetwork\lab\lab8\HTTP_client>

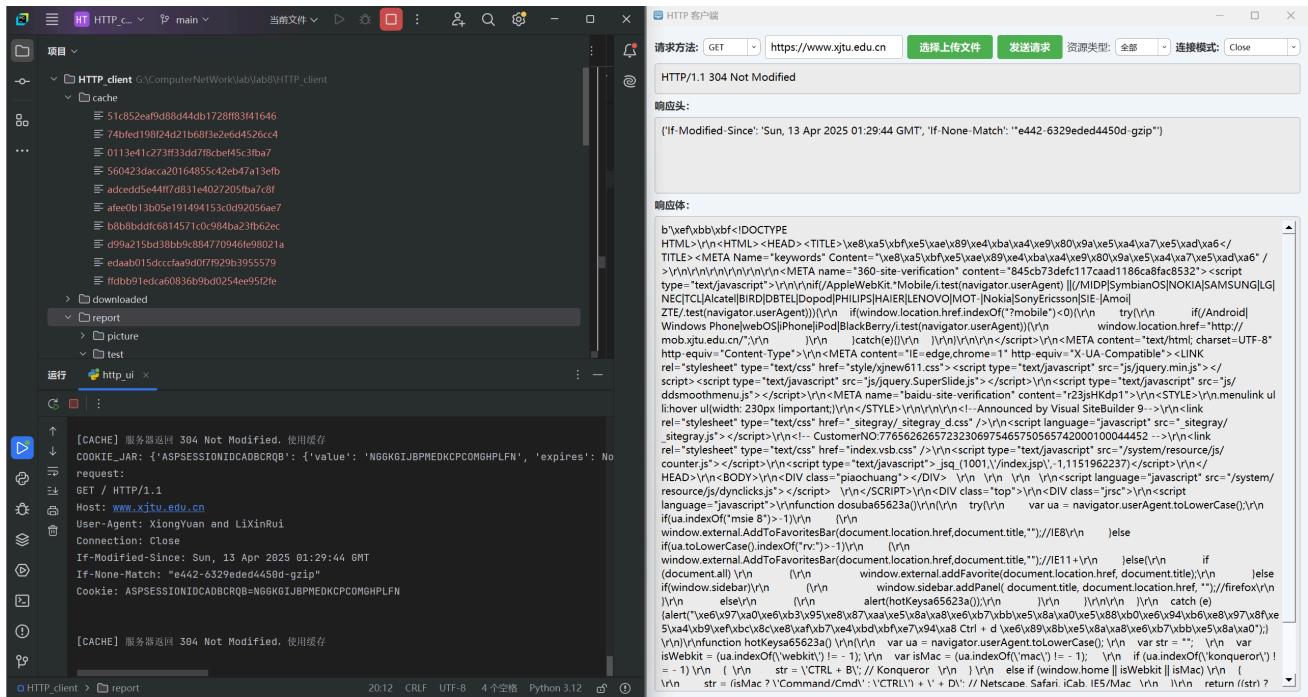
```

如图，第一次通过访问 http://httpbin.org/cookies/set?session_id=123456. 设置了 session_id，
第二次通过 <http://httpbin.org/cookies> 访问对应网址，并成功打印出上一次访问时所存储的 cookie 值。

12. 缓存机制测试

检查 Last-Modified 与 ETag 缓存策略的支持。

- 连续访问同一页面两次，查看是否使用缓存。



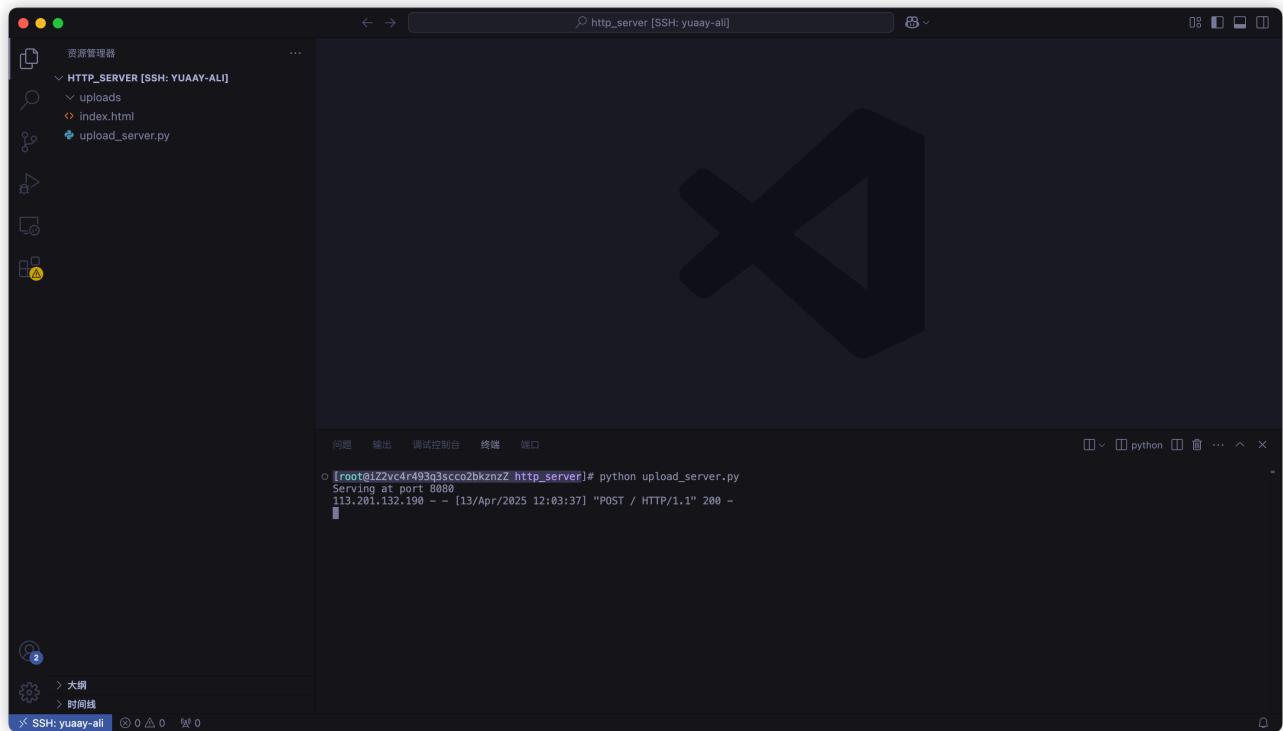
如图，在之前已经访问过一次学校网站后再次访问，会返回304，并通过本地存储的资源直接返回。

13. 文件上传测试 (POST)

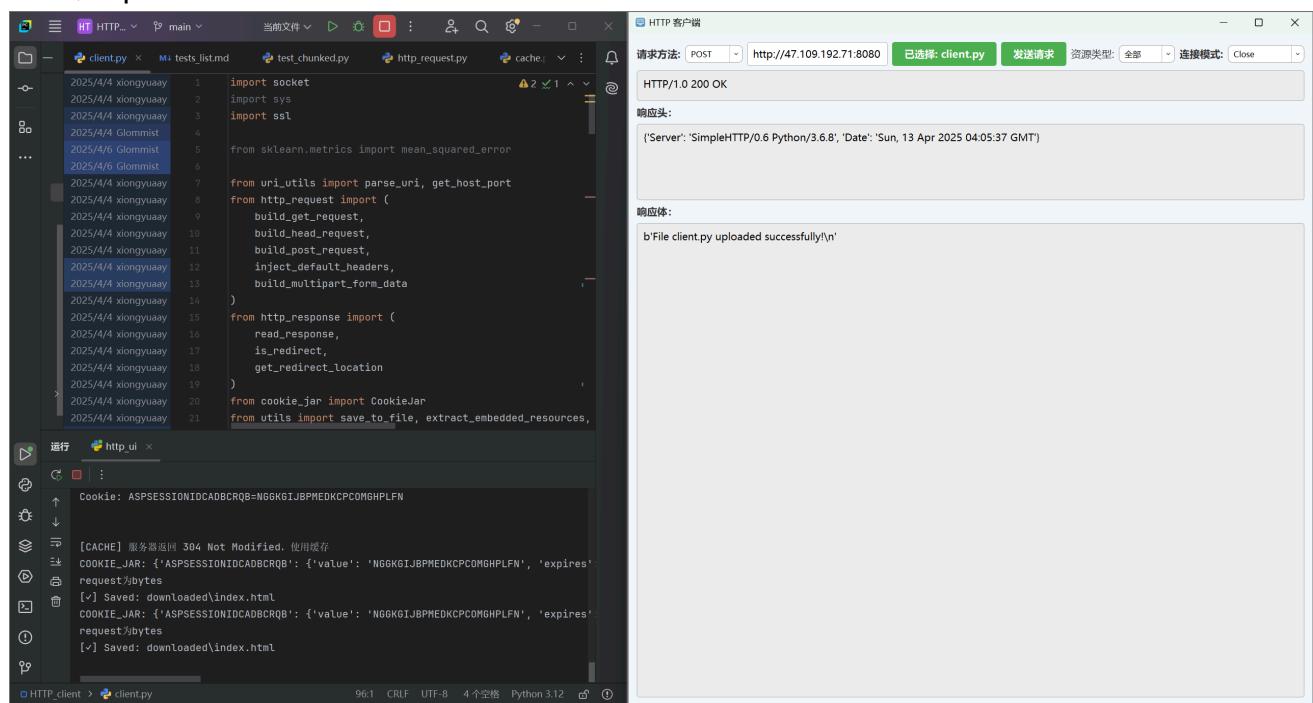
通过 POST 方法上传本地文件。

- 测试服务器：<http://47.109.192.71:8080>

服务器上传前，内容如下：



此时，uploads文件夹为空。



运行POST，选择文件进行上传，由返回内容可见上传成功。

上传文件后，内容如下：

The screenshot shows a terminal window titled "http_server [SSH: yuaya-all]". It displays Python code for a file upload server. The code defines a class `UploadHandler` that handles POST requests for file uploads. It uses `FieldStorage` to parse multipart form-data and saves files to a specified directory. The terminal shows the server running on port 8899 and receiving two POST requests from 113.201.132.198.

```
#!/usr/bin/python
# coding: utf-8

# 导入必要的模块
import os
import cgi
from http.server import SimpleHTTPRequestHandler, HTTPServer

# 定义上传目录
UPLOAD_DIR = "./uploads"

# 定义处理类
class UploadHandler(SimpleHTTPRequestHandler):
    def do_POST(self):
        """Handle file upload via POST request"""
        content_type, pdict = cgi.parse_header(self.headers.get('Content-Type'))

        if content_type == 'multipart/form-data':
            # 使用 FieldStorage 解析 multipart 表单数据
            form = cgi.FieldStorage(fp=self.rfile, headers=self.headers, environ={'REQUEST_METHOD': 'POST'})
            uploaded_file = form['file'] # "file" 是表单字段名

            if uploaded_file.filename:
                file_name = os.path.basename(uploaded_file.filename) # 获取上传的原始文件名
                os.makedirs(UPLOAD_DIR, exist_ok=True) # 确保目录存在

                file_path = os.path.join(UPLOAD_DIR, file_name)
                with open(file_path, "wb") as f:
                    f.write(uploaded_file.file.read()) # 读取并保存文件

            self.end_headers()

# 定义主类
class MyHTTPServer(HTTPServer):
    pass

# 定义处理类
def handle_request():
    handler = UploadHandler
    server = MyHTTPServer(("0.0.0.0", 8899), handler)

    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print("收到 KeyboardInterrupt, 停止服务")

# 主函数
if __name__ == '__main__':
    handle_request()
```

命令行输出：

```
[root@i22vc4d493a3cc02bkznzZ http_server]# python upload_server.py
Serving at port 8899
113.201.132.198 -- [13/Apr/2025 12:03:37] "POST / HTTP/1.1" 200 -
113.201.132.198 -- [13/Apr/2025 12:05:37] "POST / HTTP/1.1" 200 -
```

如图，在对应的服务器中，也出现了被上传的文件。

14. 不同类型的文件获取：

- img：

The screenshot shows a debugger interface with two panes. The left pane shows a file tree under "downloaded" containing various image files. The right pane shows an "HTTP客户端" (HTTP Client) window with a request for "HTTP/1.1 200 OK". The response body contains the HTML content of the image file "accelerating-innovation.jpg".

请求方法: GET /reports 已选择: client.py 发送请求 资源类型: Img 连接模式: Close

HTTP/1.1 200 OK

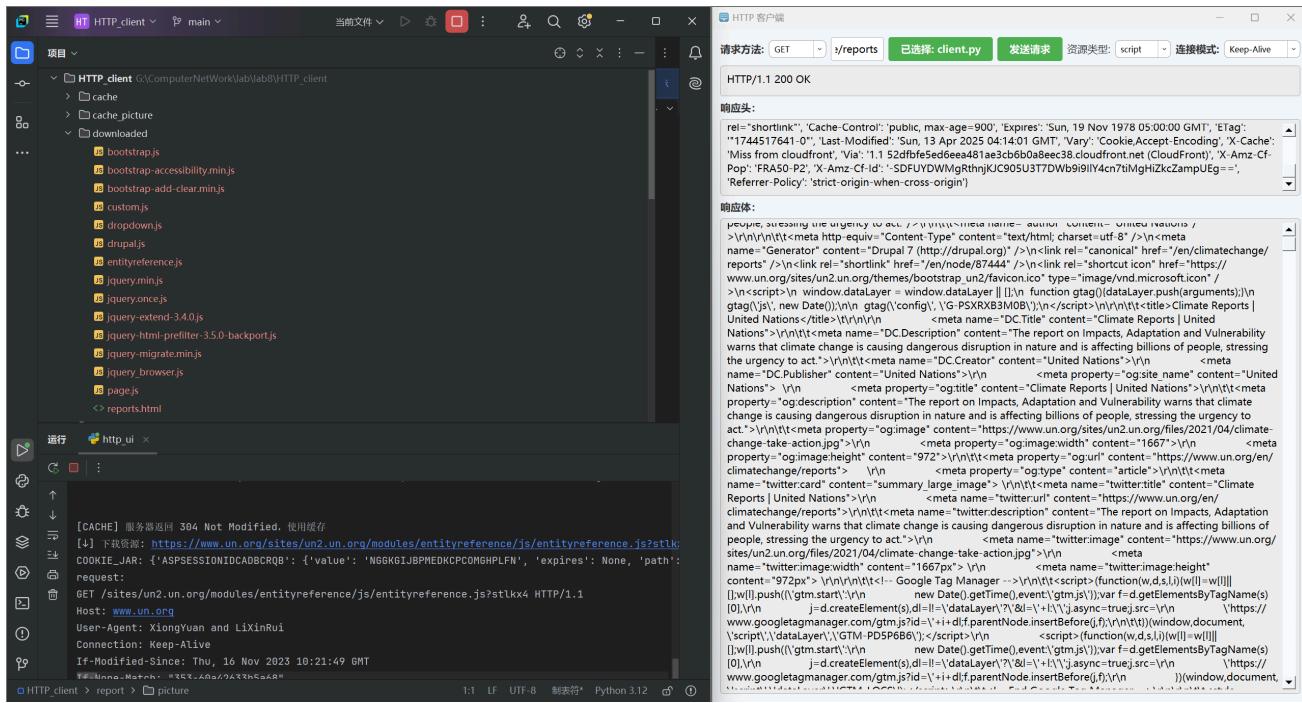
响应头:

```
(Content-Type: text/html; charset=utf-8; Transfer-Encoding: chunked; Connection: close; Date: Sun, 13 Apr 2025 04:08:50 GMT; Set-Cookie: AWSELBCORS=gU90EJbeuP/hRwyJzqag4eB62u8+2PIATzvdsLQ07Hv0/U4jxeQD20YQ31R6YosGSFafSaZQ+x51m77u3LukxCli3vs5vn7Ao8AJGpyjN0uTpM; Expires=Sun, 20 Apr 2025 04:08:50 GMT; Path=/; SameSite=None; Server: Apache; X-Dru... Cache: HIT; Content-Security-Policy: ...)
```

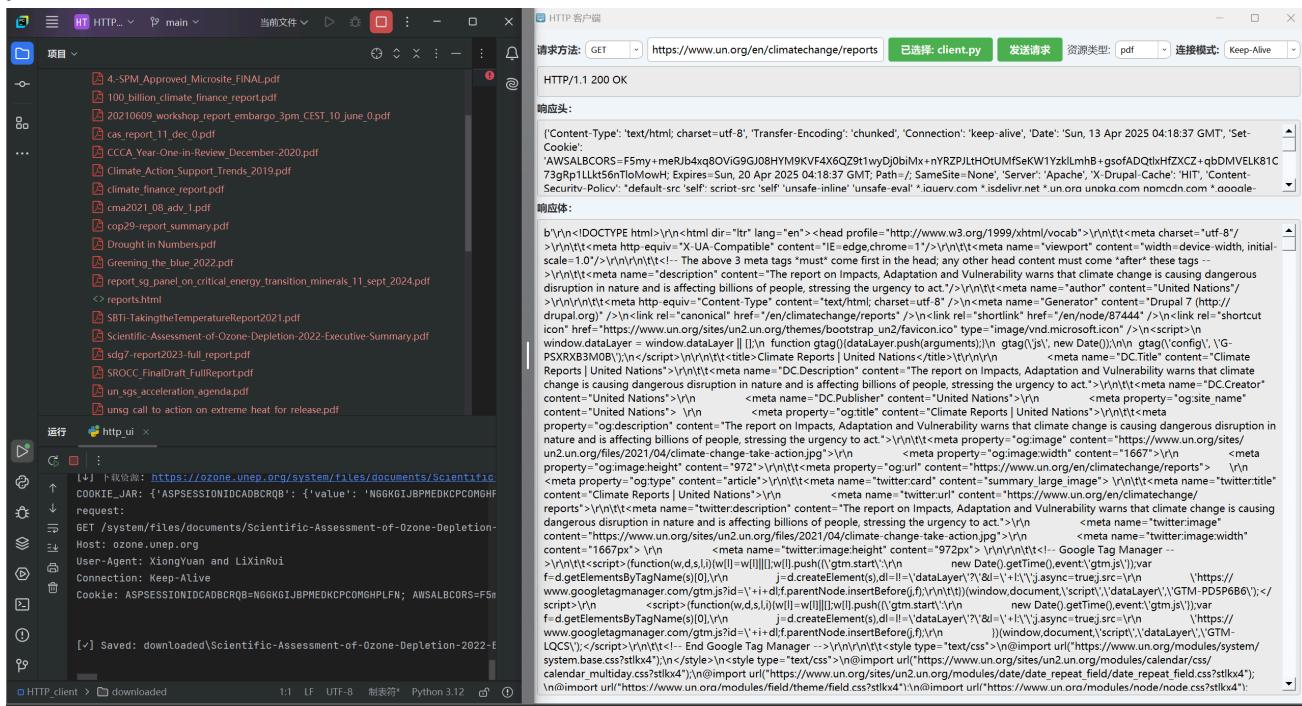
响应体:

```
b'<!DOCTYPE html><html lang="en"><head profile="http://www.w3.org/1999/xhtml/vocab"><meta charset="utf-8"/><meta http-equiv="X-UA-Compatible" content="IE=edge,chrome-1"/><meta name="viewport" content="width=device-width,initial-scale=1.0"/><title>- The above 3 meta tags must come first in the head, any other head content must come after these tags --><meta name="description" content="The report on Impacts, Adaptation and Vulnerability warns that climate change is causing dangerous disruption in nature and is affecting billions of people, stressing the urgency to act."/><meta name="author" content="United Nations" /><meta name="Generator" content="Drupal 7 (https://drupal.org)" /><link rel="canonical" href="/en/climatechange-reports/" /><link rel="shortlink" href="/en/node/87444" /><link rel="shortcut icon" href="https://www.un.org/sites/un2.un.org/themes/bootstrap_2n/favicon.ico" type="image/vnd.microsoft.icon" /><script>window.dataLayer = window.dataLayer || [];function gtag(){dataLayer.push(arguments);}gtag('js', new Date());gtag('config', 'G-PSXRXB3M0B');</script><title>Climate Reports | United Nations</title><meta name="DC.Creator" content="Climate Reports | United Nations" /><meta name="DC.Description" content="The report on Impacts, Adaptation and Vulnerability warns that climate change is causing dangerous disruption in nature and is affecting billions of people, stressing the urgency to act." /><meta name="DC.Publisher" content="United Nations" /><meta property="og:site_name" content="United Nations" /><meta property="og:title" content="Climate Reports | United Nations" /><meta property="og:description" content="The report on Impacts, Adaptation and Vulnerability warns that climate change is causing dangerous disruption in nature and is affecting billions of people, stressing the urgency to act." /><meta property="og:image" content="https://www.un.org/sites/un2.un.org/files/2021/04/climate-change-take-action.jpg" /><meta property="og:image:width" content="1667" /><meta property="og:image:height" content="972" /><meta property="og:type" content="article" /><meta property="twitter:card" content="summary_large_image" /><meta name="twitter:site" content="https://www.un.org/en/climatechange/reports" /><meta name="twitter:label0" content="The report on Impacts, Adaptation and Vulnerability warns that climate change is causing dangerous disruption in nature and is affecting billions of people, stressing the urgency to act." /><meta name="twitter:image" content="https://www.un.org/sites/un2.un.org/files/2021/04/climate-change-take-action.jpg" /><meta name="twitter:image:width" content="1667px" /><meta name="twitter:image:height" content="972px" /><!-- Google Tag Manager --><script>(function(w,d,s,l,w||l){w||l).push([l, s, w||l]);new Date().setTime(new Date().getTime() + event.utm.in)var f=d.createElement('script').setAttribute('src', 'https://www.un.org/sites/un2.un.org/files/2021/04/climate-change-take-action.jpg')w||l.push(f);})();</script>
```

- script:



- pdf:



如图，访问网址后，通过资源类型选择下载资源文件，可以在对应的download文件夹得到该种类型的文件。

本实验通过自主设计和实现一个简易的 **HTTP** 客户端程序，深入理解了 **HTTP** 协议的基本原理和交互过程，掌握了基于 **TCP Socket** 的底层网络编程方法。在不借助第三方库（如 **requests**、**urllib**）的前提下，我们完成了 **URI** 解析、请求构造、响应解析、缓存与 **Cookie** 管理等核心模块，并对用户界面进行了简单设计，使客户端具备基本的“文本浏览器”功能。

通过多个模块的协作与调试，验证了各功能模块的正确性与可扩展性，实现了多种 **HTTP** 请求的发送和响应的解析，具备一定的实用性。实验不仅提升了我们动手能力，也锻炼了我们将网络协议与编程实现相结合的能力。

八、总结及心得体会

通过本次实验，我们深入理解了 **HTTP** 协议的工作机制，包括请求方法（如 **GET**、**POST**）、状态码的含义、请求报文与响应报文的结构等。在动手实现的过程中，我们遇到了诸多挑战，例如：

- 如何构造合法且完整的 **HTTP** 请求报文；
- 如何解析响应中的状态行、头部字段与实体主体；
- 如何实现简易的缓存与 **Cookie** 管理功能；
- 如何处理 **URL** 中的路径、参数与编码；
- 如何设计程序结构，使各模块职责清晰、易于扩展。

这些问题促使我们查阅了许多文献、文档和协议规范，也加强了对网络编程和软件设计原则的理解。

本项目由熊原与李鑫瑞合作完成，双方在设计思路、代码编写与调试过程中密切配合，互相交流，不仅提升了编程技能，也锻炼了团队协作能力。总的来说，这是一次非常有收获的实验，既巩固了理论知识，也提高了解决实际问题的能力。

附件

1. 源码文件

[github地址](#)

2. 相关文档

[python项目开发规范](#)

3. 参考资料（链接）

[HTTP资源与规范](#)