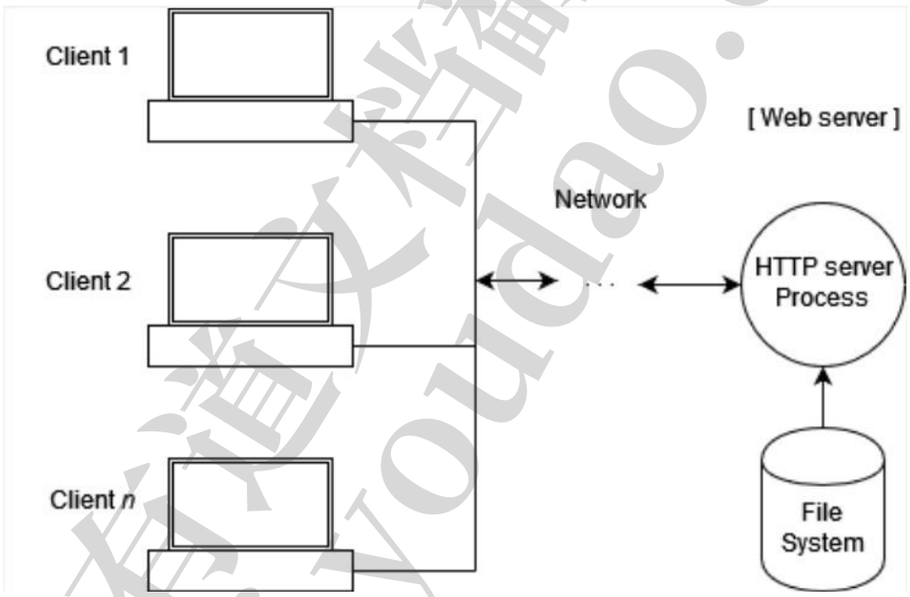
CS305 2022 秋季分配 1 - HTTP 服务器

介绍

HTTP 服务器是一种计算机(软件)程序，通过实现 HTTP 和/或 HTTPS 网络协议的服务器部分，在 C-S 模型中扮演服务器的角色。



本作业的目标是基于给定的框架实现一个 HTTP/1.1 服务器(如 flask)。为了实现这个目标，你需要按照说明，编写代码，并最终通过给定的 Python 单元测试。

HTTP / 1.1 rfc

A Request for Comments (RFC)是由主要的互联网技术开发和标准制定机构发布的一系列出版物。

在本作业中，鼓励您阅读以下 rfc 的一些章节。

- MDN 文档
- RFC9110: HTTP 语义
- RFC7230: HTTP/1.1 消息语法和路由
    - 第 3 章:消息格式
    - 第六章:连接管理
- RFC7231: HTTP/1.1 语义和内容
    - 第 4 章:请求方法
    - 第六章:响应状态码
        - 只有 200、300、301、400、
- RFC7232: HTTP/1.1 条件请求
- RFC7233: HTTP/1.1 范围请求
- RFC7234: HTTP/1.1 缓存

如果你对 RFC 中的语法感到困惑，你可以尝试阅读每个 RFC 的语法符号章节。
下面是该符号的一些实例。

- SP:空格
- CRLF:‘\ r

Tips:关于如何阅读 RFC，这里有一篇相关的文章。如何阅读 RFC

任务

Task 1: HTTP Message 的封装和解封装(20 分)在 task1 中，你需要完成框架中的代码，并直接操作二进制数据和套接字来处理 HTTP 消息。

需要完成的方法:

- ite_all()类
-

完成此部分后，您可以使用 curl 来测试您的 HTTP 服务器。

使用命令 python3 main.py 运行服务器，然后运行 curl -v

在一个新的终端上运行 http://127.0.0.1:8080。之后，您应该会得到预期的 404 Not Found。

为了简化 HTTP 服务器的实现，我们添加了报头 Connection: close 来要求客户端不要重用 TCP 连接。

```
$ curl -v http://127.0.0.1:8080/
*在 127.0.0.1:8080……
*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)> GET / HTTP/1.1
>主持人:127.0.0.1:8080
>用户代理:旋度/ 7.85.0
>接受:* / *
>
*标记 bundle 不支持多用途
< HTTP/1.1 404 Not Found
<连接:关闭
<

*关闭连接 0
```

最后，我们提供单元测试 TestTask1.py 供你检查你的实现。

任务 2:基本静态内容服务器(20 分)

在任务 2 中，当客户端为/data/下的任何文件 GET时，你被要求编写处理程序来服务本地文件。

例如，如果客户端为 http://127.0.0.1:8080/data/index.html GET，服务器应该响应文件 data/index.html 的内容。如果客户端 GET 一个不存在的文件，例如 http://127.0.0.1:8080/data/nosuchfile，服务器应该响应 HTTP 404 Not Found。

您应该拒绝它们的 POST 请求，并正确地响应 GET 和 HEAD 请求。此外，您应该正确设置响应头中的内容类型和内容长度。

在浏览器中打开 http://127.0.0.1:8080/data/index.html，你应该看到渲染的 HTML，一张图片和一个 javascript 警告。

Route:对不同的 URL 路径使用不同的处理程序

这个框架包含了一个非常简单的 Route 模块，它允许你为不同的 URL 路径注册不同的处理程序，Route 模块寻找匹配 URL 路径最长的处理程序。

```
def default_handler(server: HTTPServer, request: HTTPRequest,
response: HTTPResponse):
    #不匹配请求的默认处理程序
    响应。 status_code,响应。 reason = 400， 'Bad Request'
    print(f ''调用 url 的默认处理程序
{request.request_target}
”  )


def task2_file_handler(服务器:HTTPServer， 请求:HTTPRequest， 响
应:HTTPResponse):
    #独立的 URL 处理程序， 例如'/data/<any file> '
    #该处理程序将在客户端请求/data/…
    # TODO:任务 2:基于请求 URL 传递服务本地文件


#…
…
http_server。 register_handler("/",
default_handler) #在这里注册你的处理程序!
http_server。 register_handler("/data"， task2_file_handler,
allowed_methods=['GET'， 'HEAD'])
```

最后，我们提供了 unit-test TestTask2.py 供你检查你的实现。

任务 3:处理 POST 请求(20 分)

如果你不是很熟悉 POST 是什么，请参考这个文档。https://develo per.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

在任务 3 中，你只需要在类 HTTPRequest 中完成方法 read_message_body()。请注意二进制边界!

如果 HTTP 消息体存在，则保证 header Content-Length 始终存在于本次赋值的每个 POST 中，并且 HTTP 消息体的长度等于它的值。

客户端将 POST 一个 json，其中包含一个名为 data 的键和其他垃圾键，如下面的 json。你应该将数据的内容存储到服务器中。桶。保证关键数据出现在每个测试用例的 json 中。

```
{
  “junk1”：“
  ……”
  “data”：“< 随 机 数
  据 >”， “junk2”：“…”
}
```

当客户端 get http://127.0.0.1:8080/post 时，你应该返回上次 POST 中存储的数据。

```
{
  随机数据“数
  据”：“< >”
}
```

当你实现它时，结果将如下图所示。

```
美元 curl http://127.0.0.1:8080 - v / post -数据”｛“数据”：“测试”、“垃圾”：“忽略”｝”
*在 127.0.0.1:8080……
*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)
> POST / POST
HTTP/1.1
>主持
人:127.0.0.1:8080
```

```
>用户代理:旋度/7.85.0
>接受:*/*
内容长度>:18
>内容-类型:application/x-www-form-urlencoded >
*标记 bundle 不支持多用途<HTTP/1.1 200
OK
<连接:关闭
<
*关闭连接 0
$ curl -v http://127.0.0.1:8080/post——get
*正在尝试 127.0.0.1:8080…
*连接到 127.0.0.1(127.0.0.1)端口 8080(#0)>GET /post
HTTP/1.1
>主持
人:127.0.0.1:8080
>用户代理:旋度/7.85.0
>接受:*/*
>
*标记 bundle 不支持多用途<HTTP/1.1 200
OK
<连接:关闭
< Content-Type: application/json<
Content-Length: 16
<
*关闭连接 0{"data":
"test"}
```

最后，我们提供单元测试 TestTask3.py 供你检查你的实现。

Task 4: HTTP 302 Found: URL 重定向(10 分)

如果你不是很熟悉什么是重定向，请参考这个文档。https://d eveloper.mozilla.org/en-US/docs/Web/HTTP/Redirections。

URL 重定向用于将浏览器重定向到另一个 URL。HTTP 服务器应该将状态码设置为 30x，并在 Header 中设置 Location 属性。

实现后，当客户端 get /HEADs http://127.0.0.1:8080/redirect 时，HTTP 服务器生成 302 并重定向到

http://127.0.0.1:8080 /数据/ index .
html。

旋度结果

```
$ curl -v http://127.0.0.1:8080/redirect
*在 127.0.0.1:8080……
*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)> GET /重
定向 HTTP/1.1
>主持
人:127.0.0.1:8080
>用户代理:旋度/ 7.85.0
>接受:* / *
>

*标记包不支持多用途< HTTP/1.1 302 发现
<连接:关闭
<地点:http://127.0.0.1:8080/data/index.html <
*关闭连接 0
```

最后，我们提供单元测试 TestTask4.py 供你检查你的实现。

任务 5:HTTP Cookie 和会话(30 分)

> *HTTP cookie (web cookie，浏览器 cookie)是服务器发送给用户的网络浏览器的一小段数据。浏览器可能会存储 cookie，并在以后的请求中将其发送回相同的服务器。通常，HTTP cookie 用于判断两个请求是否来自同一个浏览器——例如，保持用户登录。它会记住无状态 HTTP 协议的有状态信息。*

如果你不是很熟悉 Cookie 是什么，请参考这个文档。https://develo per.mozilla.org/en-US/docs/Web/HTTP/Cookies

在任务 5 中，你被要求保护一个只有经过身份验证的 users 才能访问的图像文件。

在第一步中，客户端将把他们的凭据 POST 到一个登录 API，你应该验证它并将其标记为已验证或未验证。由于 HTTP 协议是无状态的，所以使用 Cookie 来保存客户端的状态。

在第二步中，客户端将使用第一步中服务器设置的 Cookie 请求受保护的图像，您应该只向经过认证的客户端响应受保护的图像。

在这一部分，你不需要关心 CORS 问题，你可以直接忽略 Cookie 中的 Domain、Path 属性。

饼干(15 分)

在收到 HTTP 请求后，服务器可以发送一个或多个 Set-Cookie 报头和响应。在接收到 cookie 后，浏览器通常存储 cookie，并在 cookie HTTP 报头中向同一服务器发送请求。

步骤 1。登录身份验证

客户端将 POST 带有以下 json 结构的 http://127.0.0.1:8080/api/login，服务器应该验证用户名和密码是否等于 admin, admin。

```
{“用户名”:“admin”,“密码”:“admin”}
```

如果相等，服务器应该返回 HTTP 200 OK，并设置 cookie Authenticated=yes。否则，它应该返回 HTTP 403 Forbidden。

为了简化代码，我们可以忽略 Cookie 的任何其他属性。**步骤 2。访问受保护的资源**

客户端将 GET/HEAD http://127.0.0.1:8080/api/getimage。

如果客户端在 Step1 中授权，它将使用在最后一个 Response 中指定的 Cookies 请求这个 URL，即。， Request Cookie 中的 Authenticated 字段应该是 yes。

如果 Authenticated 字段是 yes，服务器应该返回图像数据/test.jpg，并正确设置 Content-Length 和 Content-Type。否则，如果 Authenticated Cookie 不存在或不为 yes，它应该返回 HTTP 403 Forbidden。

如果你正确地实现了这一部分，你应该传递 unittest TestTask5Cookie.py。你也可以在浏览器中打开 http://127.0.0.1:8080/api/test 进行测试。

旋度结果

```
$ curl -v http://127.0.0.1:8080/api/login——data
'{"username":"admin"， "password":"admin"}' *
Trying 127.0.0.1:8080···

*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)> POST
/api/login HTTP/1.1
>主持
人:127.0.0.1:8080
>用户代理:旋度/ 7.85.0
>接受:* / *
内容长度>:40
>内容-类型:application/x-www-form-urlencoded >
*标记 bundle 不支持多用途
< http /1.1 200 ok
<连接:关闭
< set - cookie:验证=yes
<

*关闭连接 0


$ curl -v http://127.0.0.1:8080/api/getimage *
正在尝试 127.0.0.1:8080···

*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)> GET
/api/getimage HTTP/1.1
>主持
人:127.0.0.1:8080
>用户代理:旋度/ 7.85.0
>接受:* / *

>

*标记 bundle 不支持多用途< HTTP/1.1 403
Forbidden
<连接:关闭
```

```
<

*关闭连接 0


$ curl -v http://127.0.0.1:8080/api/getimage——header "Cookie: Authenticated=yes"
*在 127.0.0.1:8080……
*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)
> GET /api/getimage HTTP/1.1
>主持
人:127.0.0.1:8080
>用户代理:旋度/7.85.0
>接受:*/*
>饼干:验证= yes

>

*标记 bundle 不支持多用途<HTTP/1.1 200
OK
<连接:关闭
<内容长度:44438
<-type:/jpeg 图像
<

警告:二进制输出会弄乱你的终端。使用“——output -”来判断
警告:卷曲输出到您的终端，或考虑“——output
警告:<FILE>”保存到一个文件。
*将输出写入目标文件失败
*关闭连接 0
```

之后，你应该传递 unit-test TestTask5Cookie.py。你可以在浏览器中打开 http://127.0.0.1:8080/api/test 来玩你的服务器。

会话(15 分)

以上方法中的认证方法是不安全的，恶意客户端在不知道真实用户名和密码凭据的情况下，只需要设置 cookie Authenticated=yes 就可以访问受保护资源。

为了实现会话，服务器会初始化一个字典，使用一个随机字符串作为密钥，并将客户端无法访问的数据存储在字典中。

服务器会在响应的 set - cookie 中设置这个随机字符串作为会话 key，客户端在以后的请求中会带来给定的 cookie，服务器会通过 cookie 中的 key 找到存储的数据。

步骤 1。登录身份验证

客户端用以下 json 结构 post http://127.0.0.1:8080/apiv2/login，服务器应该验证用户名和密码是否都等于 admin。

```
{“用户名”:“admin”,“密
码”:“admin”}
```

如果它们相等，服务器应该生成一个随机字符串作为 SESSION_KEY，并确保它在存储的会话密钥中是唯一的。服务器应该返回 HTTP 200 OK，并设置 cookie SESSION_KEY=<随机字符串>。否则，它应该返回 HTTP 403 Forbidden。

步骤 2。访问受保护的资源

客户端将 GET/HEAD http://127.0.0.1:8080/apiv2/getimage, cookie 中是否有 SESSION_KEY。

如果客户端在 Step1 中授权成功，它将在最后的响应中请求包含 cookie 的这个 URL。

如果 SESSION_KEY 字段存在，并且服务器识别它是有效的，服务器应该返回图像数据/test.jpg，并正确设置 Content-Length 和 Content-Type。否则，服务器应该返回 HTTP 403 Forbidden。

如果你正确地实现了这一部分，你应该通过 unit-test TestTask5Session.py。你也可以在浏览器中打开 http://127.0.0.1:8080/apiv2/test 进行测试。

旋度结果

```
$ curl -v http://127.0.0.1:8080/apiv2/getimage  *
Trying 127.0.0.1:8080…

*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)> GET
/apiv2/getimage HTTP/1.1

>主持
人:127.0.0.1:8080

>用户代理:旋度/7.85.0

>接受:*/*

>

*标记 bundle 不支持多用途< HTTP/1.1  403
Forbidden

<连接:关闭

<

*关闭连接 0

$ curl -v http://127.0.0.1:8080/apiv2/login——data
'{"username":"admin",  "password":"admin"}' *正
在尝试 127.0.0.1:8080…

*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)> POST
/apiv2/login HTTP/1.1

>主持
人:127.0.0.1:8080

>用户代理:旋度/7.85.0

>接受:*/*

内容长度>:40

>内容-类型:application/x-www-form-urlencoded  >

*标记 bundle 不支持多用途< HTTP/1.1  200
OK

<连接:关闭

< set - cookie:  SESSION_KEY  = DJK5LAFTY8NEQN6JNRIA

<

*关闭连接 0

$ curl -v http://127.0.0.1:8080/apiv2/getimage——header “Cookie:
SESSION_KEY=DJK5LAFTY8NEQN6JNRIA”

*在 127.0.0.1:8080……

*连接到 127.0.0.1(127.0.0.1)端口 8080 (#0)

> GET /apiv2/getimage  HTTP/1.1

>主持
人:127.0.0.1:8080

>用户代理:旋度/7.85.0

>接受:*/*

>饼干:SESSION_KEY  = DJK5LAFTY8NEQN6JNRIA
```

```
>

*标记 bundle 不支持多用途< HTTP/1.1 200
OK
<连接:关闭
<内容长度:44438
< - type: / jpeg 图像

<

警告:二进制输出会弄乱你的终端。使用“——output -”来判断
警告:卷曲输出到您的终端，或考虑“——output
警告:<FILE>”保存到一个文件。
*将输出写入目标文件失败
*关闭连接 0
```

最后，我们还提供了 unit-test TestTask5Session.py，让你在这部分检查你的实现。请注意，在我们的测试环境中，/data 目录下的测试文件将是不同的。

你可以在浏览器中打开 http://127.0.0.1:8080/apiv2/test 来玩你的服务器。

如何运行代码

服务器

在项目根目录下运行 python main.py，然后在新终端下运行 curl -v http://127.0.0.1:8080。

单元测试

VSCode

运行 python -m unittest tests。在项目根目录下运行 TestTask1，执行 TestTask1.py。

对于其他测试，执行命令是类似的。

PyCharm

转到分配的根目录，然后执行测试。

要提交什么

你必须提供你的实现的 zip 文件，包括 main.py, framework.py 和其他文件(如果需要的话)。

一个 PDF 文件，其中包括一些必要的屏幕截图，以显示您的代码工作和一些简短的解释。

分数

作业的分数将由一组扩展的单元测试来决定。你可以使用给定的单元测试来检查你的实现的正确性。但是即使你通过了所有给定的单元测试，你仍然可能得不到 100 分。

注意!!

请在 main.py 中将 YOUR_STUDENT_ID = 12010000 修改为您的 SID，该字段用于自动测试。否则，sa 在处理你的提交时会很痛苦。

分数的环境

请注意，您的代码最终将在基于以下环境的平台上进行测试和评分。

- Python 3.9
- **LIB限制:你只能使用Python标准库，不包括http模块。**

Dockerfile 来构建测试环境(你不需要构建这个环境
本地):

```
从 python 3.9 圆心
运行 apt update && apt install -y curl
运行 pip3 install requests
体积/分
WORKDIR /分
CMD ["/bin/bash"， "-c"， "python3 -m tests. sh"TestAll > result.txt 2 > stderr.txt”)
# Build: docker Build . txt- t ass1
#运行:docker 运行-v $PWD:/score - ass1:最新
```

问答环节

如果你对本次作业有任何问题，请到上面的链接提出你的问题。并且我们会及时查看和回复。

https://github.com/yuk1i/cs305-2022fall-homework1-student/issues

享受自己!