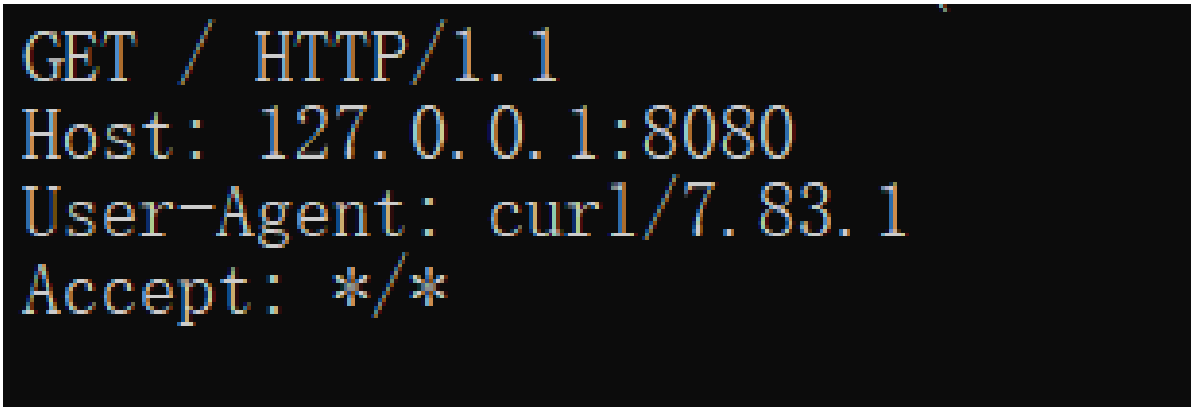
CS305 2022 Fall Lab Assignment--HTTP Server

Name: 吉辰卿　　ID: 11911303

Task 1:

- **HTTP Request**

在这一小节中，我们需要对命令行发过来的请求报文进行接收，并对接收到的请求报文进行分割解析以正确得到HTTPRequest类中相应的成员。在这之前，我们先看看命令行发送过来的请求报文：



根据RFC标准文档规定的格式，我们可以根据HTTP 请求报文中**请求行、首部行等相应的内容**为HTTP Request类中的对应成员进行赋值。注意的是每一行的结束都使用了回车换行符**"\r\n"**。因此我们对收到的报文进行切割操作时可以以此作为分割的依据。如下所示：

```
 def read_headers(self):
        """
        Read these structures from `self.socket`, format them and fill
HTTPRequest object fields.

        HTTP-message   = method SP request-target SP HTTP-version CRLF
                         *( header-field CRLF )
                         CRLF

        :return:
        """
        # TODO: Task1, read from socket and fill HTTPRequest object fields
        # read from socket and get the url of the request
        recv_data: bytes = b''
        count: int = 1
        body_count: int = 0
        body: str = ""
```

```
        # we wait for all the data is received, then we can resolve the request message
        while True:
            recv_sub_data = self.socket.recv(2048)
            recv_data = recv_data + recv_sub_data
            if len(recv_sub_data) < 2048:
                break
        data = recv_data.decode().split('\r\n')
        self.method = data[0].split(' ')[0]
        self.request_target = data[0].split(' ')[1]
        self.http_version = data[0].split(' ')[2]
        for sub_data in data[1:]:
            # print(sub_data)
            sub_data_spilt = sub_data.split(' ')
            # print(sub_data_spilt)
            if sub_data_spilt[0] != "":
                self.headers.append(HTTPHeader(sub_data_spilt[0].split(":")[0], sub_data_spilt[1]))
                count = count + 1
            else:
                count = count + 1
                break
        # Then came the entity part, and began to analyze the message
        for sub_body_data in data[count:]:
            if sub_body_data == data[-1]:
                # body_count +=1
                body = body + sub_body_data
                continue
            else:
                body = body + sub_body_data + "\r\n"
        # print(body_count)
        self.buffer = body.encode()
        # self.buffer = data[count:][0].encode()
        # Debug: print http request
        print(f"{self.method} {self.request_target} {self.http_version}")
        for h in self.headers:
            print(f"{h.name}: {h.value}")
        print()
        print(self.buffer.decode())
```

在读取请求报文的时候，需要**注意如下两个点**：

1. 我们不能直接这样写 `data = self.socket.recv(2048).decode()`，就认为 `data` 就是接收到的报文信息。由于我们发送的报文长度很可能大于2048 byte，因此我们需要写一个循环进行接收。直到某一次循环接收到的子报文长度小于2048 byte，我们才能认为是接收完了从而跳出循环。

2. 通过对上面代码的理解，我们用一个计数器count来确定什么时候切割到了实体体部分。`data[count: ]` 后面的部分都是请求报文中实体体的内容。但由于实体体中也有可能出现回车换行符**"\r\n"**, 因此我们不能简单地就取 `data[count:][0]` 就认为是全部的实体体报文，应该还需要设置一个计数器**body_count**来解决实体体中出现**\r\n**的时候对实体体信息的完全拼接。最后，完整的实体体被储存到了类成员 `self.buffer` 中。**(具体实现见上面的代码)**

- ## HTTP Response

在**HTTP Server**类中我们对请求报文进行处理以生成与相应报文有关的属性字段**(比如状态代码、首部行等)**。之后，我们在**HTTP Response**中的`write_all()`方法通过`socket`发送生成好的HTTP响应报文信息。如下所示：

```
  def write_all(self):
        """
        set status_line, and write status_line, headers and message body (if exists) into self.socket
        :return:
        """
        # TODO: Task1, construct response from fields and write binary data to socket
        self.response = '{} {} {}\r\n'.format(self.http_version, self.status_code, self.reason).encode()
        self.socket.send(self.response) # send response
        for h in self.headers:
            self.socket.send('{}: {}\r\n'.format(h.name, h.value).encode()) # send response headers
        self.socket.send(b'\r\n') # send response \r\n for response body
        # Then if have any file we should transmit them
        self.socket.send(self.body)
        pass
```

Task 2:

这一问我刚开始还准备用路径匹配去做，即切割请求报文中我的发送路径，和现有的文件(相对路径)进行匹配。如果匹配成功则在相应报文的实体体返回对应文件，如匹配失败则挂出**404 NOT Found**状态码。但是实现的时候发现很麻烦，于是我直接写一个`try...except...`结构：如果根据**我发送过来的url没有找到相应文件**，也就是`FileNotFoundError`，此时便挂出**404 NOT Found**状态码即可。这一问的处理代码如下：

```
def task2_data_handler(server: HTTPServer, request: HTTPRequest, response: HTTPResponse):
    # TODO: Task 2: Serve static content based on request URL (20%)
    # get the request url from thr request message
    request_target = request.request_target
    if request.method == "GET":
        # match that request_target if our local url have this request_target url, so we should firstly try to open our local file
        try:
            f = open("." + request_target, 'rb')
            response.add_header("Content-Type", mimetypes.guess_type(request_target)[0])
            # If there is not any except(file is exist), we should return the response code "200"
            response.status_code, response.reason = 200, 'OK'
            response.body = f.read()
            response.add_header("Content-Length" , str(len(response.body)))
            f.close()
        except FileNotFoundError :
            # If the file doesn't exist we should return the response code "404"
```

```
            response.status_code, response.reason = 404, 'Not Found'
        print(f"calling task2_data_handler for url {request_target}")
    elif request.method == "HEAD":
        # match that request_target if our local url have this
request_target url, so we should firstly try to open our local file
        try:
            f = open("." + request_target, 'rb')
            response.add_header("Content-Type",
mimetypes.guess_type(request_target)[0])
            # If there is not any except(file is exist), we should return
the response code "200"
            response.status_code, response.reason = 200, 'OK'
            response.add_header("Content-Length" , str(len(f.read())))
            f.close()
        except FileNotFoundError :
            # If the file doesn't exist we should return the response code
"404"
            response.status_code, response.reason = 404, 'Not Found'
        print(f"calling task2_data_handler for url {request_target}")
    pass
```

通过上面的代码，我们可以看出，做这一问还是要区分一下**“GET”**和**“HEAD”**请求，即**“HEAD”**请求返回空的实体体，**"GET"**请求需要返回完整的实体体。

Task 3：

做这一问，先要拿到完整的消息体，直接返回 `self.buffer` 即可，如下代码：

```
def read_message_body(self) -> bytes:
        return self.buffer
```

之后，在 `task3_json_handler` 方法中解析消息体中的json字符串，之后把得到的**data键值对信息**写入一个本地文件post后储存起来，以便下一次用**"GET"**方法请求后能够获取到上一次储存的data键值对信息。代码如下所示：

```
def task3_json_handler(server: HTTPServer, request: HTTPRequest, response:
HTTPResponse):
    # TODO: Task 3: Handle POST Request (20%)
    response.status_code, response.reason = 200, 'OK'
    if request.method == 'POST':
        binary_data = request.read_message_body()
        obj = json.loads(binary_data)
        # TODO: Task 3: Store data when POST
        server.task3_data = str(obj["data"])
        real_data = ("{"+ f"'data': '{server.task3_data}'" +
"}").replace('\'', '"')
        with open(".\\post","w") as f:
            f.write(real_data)
        pass
    elif request.method == "GET":
        obj = {'data': server.task3_data}
        json_return = json.dumps(obj)
```

```
        response.add_header("Content-Type",
mimetypes.guess_type("json_return.json")[0])
        return_binary = json_return.encode()
        response.add_header("Content-Length", str(len(return_binary)))
        response.body = return_binary
    elif request.method == "HEAD":
        obj = {'data': server.task3_data}
        json_return = json.dumps(obj)
        response.add_header("Content-Type",
mimetypes.guess_type("json_return.json")[0])
        return_binary = json_return.encode()
        response.add_header("Content-Length", str(len(return_binary)))
        pass
```

通过上面的代码，我们可以看出，做这一问仍然是要区分一下**“GET”**和**“HEAD”**请求，即**“HEAD”**请求返回空的实体体，**"GET"**请求需要返回完整的实体体。

Task 4:

在做这一问时，我们需要在客户端访问的资源url路径为**/redirect**时候，在**HTTP Response**类中设置相应的成员属性如HTTP Version、status_code等， 同时在HTTP返回报文的首部行添加一个**资源重定向的头部字段**即可。代码如下所示：

```
def task4_url_redirection(server: HTTPServer, request: HTTPRequest, response:
HTTPResponse):
    # TODO: Task 4: HTTP 301 & 302: URL Redirection (10%)
    response.status_code, response.reason = 302, 'Found'
    response.add_header("Location", "http://127.0.0.1:8080/data/index.html")
    pass
```

这一问，由于HTTP 响应报文没有实体体，因此**“GET”**和**“HEAD”**请求没有任何区别，我们不需要写`if...else...`来判断请求的方法是什么。

Task 5:

这一问有两个小问，对应**Cookie**和**Session**，但是实现起来并不难。但是需要注意：这一问仍然是要区分一下**“GET”**和**“HEAD”**请求，即**“HEAD”**请求返回空的实体体，**"GET"**请求需要返回完整的实体体。下面分别展示一下这两问的核心处理代码：

- ## Cookie

```
def task5_cookie_login(server: HTTPServer, request: HTTPRequest, response:
HTTPResponse):
    # TODO: Task 5: Cookie, Step 1 Login Authorization
    obj = json.loads(request.read_message_body())
    if obj["username"] == 'admin' and obj['password'] == 'admin':
        response.status_code, response.reason = 200, 'OK'
        response.add_header("Set-Cookie", "Authenticated=yes")
        pass
    else:
```

```
        response.status_code, response.reason = 403, 'Forbidden'
        pass

def task5_cookie_getimage(server: HTTPServer, request: HTTPRequest,
response: HTTPResponse):
    # TODO: Task 5: Cookie, Step 2 Access Protected Resources
    # 检查请求头部有没有Cookie，如果有就返回相应的图片资源
    if request.method == "GET":
        for h in request.headers:
            if f"{h.name}" == "Cookie" and f"{h.value}" ==
"Authenticated=yes":
                response.status_code, response.reason = 200, 'OK'
                with open(".\\data\\test.jpg", 'rb') as f :
                    response.add_header("Content-Type",
mimetypes.guess_type(".\\data\\test.jpg")[0])
                    response.body = f.read()
                    response.add_header("Content-Length" ,
str(len(response.body)))
            else:
                response.status_code, response.reason = 403, 'Forbidden'
    elif request.method == 'HEAD':
         for h in request.headers:
            if f"{h.name}" == "Cookie" and f"{h.value}" ==
"Authenticated=yes":
                response.status_code, response.reason = 200, 'OK'
                with open(".\\data\\test.jpg", 'rb') as f :
                    response.add_header("Content-Type",
mimetypes.guess_type(".\\data\\test.jpg")[0])
                    response.add_header("Content-Length" ,
str(len(f.read())))
            else:
                response.status_code, response.reason = 403, 'Forbidden'
    pass
```

- **Session**

```
def task5_session_login(server: HTTPServer, request: HTTPRequest, response:
HTTPResponse):
    # TODO: Task 5: Cookie, Step 1 Login Authorization
    obj = json.loads(request.read_message_body())
    if obj["username"] == 'admin' and obj['password'] == 'admin':
        response.status_code, response.reason = 200, 'OK'
        session_key = random_string()
        while session_key in server.session:
            session_key = random_string()
        pass
        server.session = {"SESSION_KEY=":f"{session_key}"}
        response.add_header("Set-Cookie", "SESSION_KEY="+ f"{session_key}")
    else:
        response.status_code, response.reason = 403, 'Forbidden'

def task5_session_getimage(server: HTTPServer, request: HTTPRequest,
response: HTTPResponse):
    # TODO: Task 5: Cookie, Step 2 Access Protected Resources
```

```
    # 检查请求头部有没有Cookie，如果有就返回相应的图片资源
    if request.method == "GET":
        for h in request.headers:
            if f"{h.name}" == "Cookie" and f"{h.value}" == "SESSION_KEY="+server.session["SESSION_KEY="]:
                response.status_code, response.reason = 200, 'OK'
                with open(".\\data\\test.jpg", 'rb') as f :
                    response.add_header("Content-Type", mimetypes.guess_type(".\\data\\test.jpg")[0])
                    response.body = f.read()
                    response.add_header("Content-Length" , str(len(response.body)))
            else:
                response.status_code, response.reason = 403, 'Forbidden'
    elif request.method == 'HEAD':
         for h in request.headers:
            if f"{h.name}" == "Cookie" and f"{h.value}" == "SESSION_KEY="+server.session["SESSION_KEY="]:
                response.status_code, response.reason = 200, 'OK'
                with open(".\\data\\test.jpg", 'rb') as f :
                    response.add_header("Content-Type", mimetypes.guess_type(".\\data\\test.jpg")[0])
                    response.add_header("Content-Length" , str(len(f.read())))
            else:
                response.status_code, response.reason = 403, 'Forbidden'
    pass
```