**README.MD**

SOCKET HTTP Server/Client

국민대학교 소프트웨어학부

20213050 이재원

**개요**

TCP 기반의 소켓 프로그래밍을 활용하여 HTTP 프로토콜 기반으로 한 Server와 Client 프로그램을 구현함으로써 HTTP의 Request와 Response에 대한 구조와 실행을 이해한다.

또한 HTTP의 여러가지 header들을 이해하고 접목시켜, 특히 Connection header를 통해 HTTP 1.1의 특성 중 하나인 persistent connection을 구현한다.

**개발환경**

- IDE: Visual Studio Code

- Language: Python 3.11.8 (64bit), HTML, json

- 그외: git, github, wireshark

**동작환경**

- OS: Windows, Ubuntu

- Language: Python 3.11.8 (64bit)

- localhost or 2 PC on different networks

**시작가이드**

* git clone https://github.com/2wodnjs7/Socket.git

[1대의 PC에서 구동 시]

$ python server.py

$ python client.py (python server.py를 한 환경과 다른 환경에서 실행)

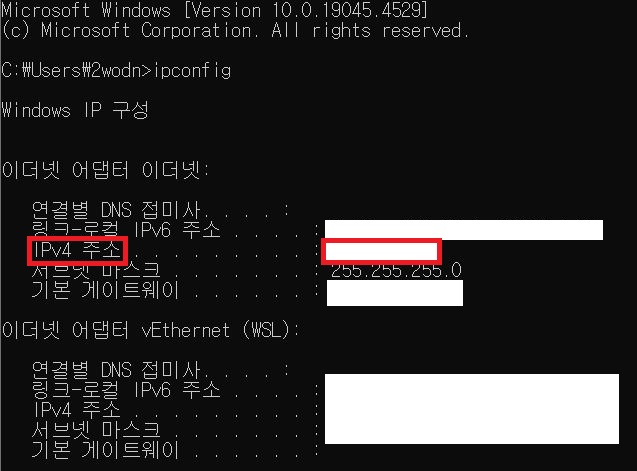
[2대의 PC에서 구동 시]

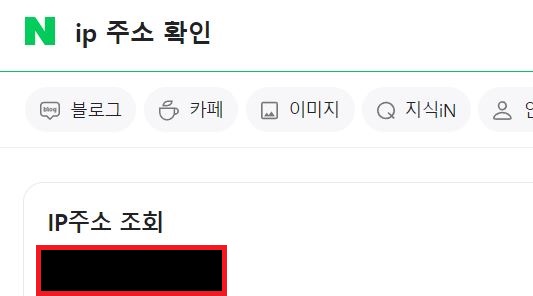
* server.py의 ‘serverIP’를 현재 PC의 Public IP (내부 IP)로 변경 후 저장한다.

(Hint: Cmd > ipconfig > 이더넷 IPv4)

* client.py의 ‘serverIP’를 현재 PC의 external IP (공인 IP)로 변경 후 저장한다.

(Hint: naver.com > ‘ip 주소 확인’ 검색)





$ python server.py (서버 환경에서 실행)

$ python client.py (외부 환경에서 실행, 예시: colab)

[2대의 PC에서 연결이 되지 않는 경우]

- 포트포워딩을 해줘야 한다.

[KT 인터넷 환경 기준]

* http://172.30.1.254 접속
* 장치설정 > 트래픽 관리 > 포트 포워딩 설정
* 외부 포트: 80 ~ 80
* 내부 IP 주소: [현재 PC의 Public IP (내부 IP)]
* 내부 포트: 80 ~ 80
* 프로토콜: TCP

텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

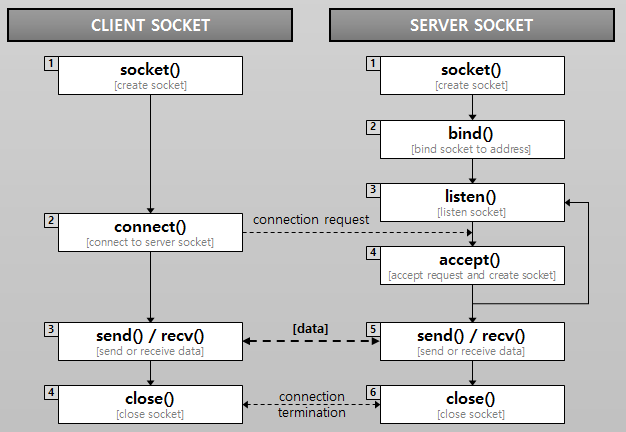
자동 생성된 설명

**프로그램 사용법**

* server.py 를 실행한 후 다른 환경에서 client.py를 실행한다.
* 출력되는 request list를 통해 requestNum와 keep\_alive에 대한 [y/n]여부를 입력한다.
* keep\_alive가 y이면 request 이후 계속해서 request를 이어나간다.
* keep\_alive가 n이면 request 이후 server/client 모두 종료된다.

**프로그램 구조**

server와 client의 연결을 위해 Socket을 이용하여 구현했다. 그리고 HTTP 프로토콜을 구현하여야 하기 때문에 TCP와 UDP 중 TCP Socket 프로그래밍을 진행했다.



먼저 서버, 클라이언트 모두 socket을 생성한다. 그리고 서버에서 서버의 IP와 Port를 소켓에 바인딩한다. 그 후 클라이언트의 접속을 기다리는 listening 상태에 들어간다.

클라이언트가 connect()를 통해 Server에 접속하면 서버가 accept()를 통해 클라이언트 와 연결할 새로운 소켓을 생성하고 서버와 클라이언트간의 send() / recv()로 서로 데이터를 주고받는다. 그러다 클라이언트의 keep\_alive 입력이 ‘n’이 되면 서버와 클라이언트 모두 close()를 통해 소켓을 닫고 연결을 종료한다.

서버

serverSocket = socket(AF\_INET, SOCK\_STREAM) # 서버 소켓 생성

serverSocket.bind((serverIP, serverPort)) # 서버 바인딩: 소켓을 서버 IP, Port에 바인딩

serverSocket.listen() # 서버 리스닝: 클라이언트 연결을 대기

connectionSocket, addr = serverSocket.accept() # 연결 수락: 클라이언트 연결을 수락 및 클라이언트와 연결할 새로운 소켓 생성

message = connectionSocket.recv(65535).decode().split('\r\n')

# 각 로직에 따라 구성된 sendMessage를 client에게 전송

connectionSocket.send(sendMessage.encode('utf-8'))

# keep\_alive가 아닐 경우 client와 연결된 socket 연결 종료 및 프로그램 종료

if not keep\_alive:

connectionSocket.close()

    break

클라이언트

clientSocket = socket(AF\_INET, SOCK\_STREAM) # 클라이언트 소켓 생성

clientSocket.connect((serverIP, serverPort)) # 특정 서버 IP, Port에 연결 요청

# 각 로직에 따라 구성된 request\_message를 server에게 전송

clientSocket.send(request\_message.encode('utf-8'))

# server에게 송신받은 데이터를 decode하여 저장

recieve\_message = clientSocket.recv(65535).decode()

# client 소켓 연결 종료 및 프로그램 종료

clientSocket.close()

**HTTP 1.1 Protocol**

HTTP는 하이퍼텍스트(HTML) 문서를 교환하기 위해 만들어진 Protocol.

HTTP는 TCP/IP 기반으로 되어있고, 기본적으로 request(요청)/response(응답) 구조.

클라이언트가 HTTP request를 서버에 보내면 서버는 HTTP response를 보내는 구조.

클라이언트와 서버의 모든 통신이 요청과 응답으로 이루어진다.

**Request Message**

HTTP Request Message는 3가지 부분으로 나누어진다.

**( Start Line, Headers, Body )**

**Start Line**: HTTP Request Message의 시작 라인으로 3가지 부분으로 구성된다.

* HTTP Method: 요청의 의도를 담고 있는 GET, POST, POST, PUT 등이 있다.
* Request target: HTTP Request가 전송되는 목표 주소이다.
* HTTP Version: version에 따라 Request의 구조, 데이터가 달라질 수 있어

version을 명시한다.

**Headers**: 해당 Request에 대한 추가 정보를 담고 있는 부분

* Host : 요청하는 서버 호스트 이름과 포트 이름
* Connection: 연결을 유지할 것인지를 결정하는 헤더
* Cache-control: 불필요한 데이터 요청을 줄이기 위해 사용하는 헤더
* Accept: client에서 받을데이터 형식
* Accept-Charset: 클라이언트가 지원하는 문자 인코딩을 알리는 헤더
* Accept-Encoding: 요청에 대한 응답 메시지에 승인되는 인코딩을 표시하기 위한 헤더
* Accept-Language: 클라이언트가 서버로부터 받고자 하는 언어를 나타내는 헤더
* 그 외에도 사용 목적에 따라 많은 헤더가 존재한다.

**Body**: HTTP Request가 전송하는 데이터를 담고 있는 부분

* 전송하는 데이터가 없다면 body 부분은 비어 있다.
* 보통 POST/PUT 요청일 경우, body 데이터가 포함되어 있다.

**Response Message**

HTTP Response Message는 request와 동일하게 3가지 부분으로 나누어진다.

**( Status Line, Headers, Body )**

**Status Line**: HTTP Response의 상태를 간략하게 나타내며 3가지 부분으로 나누어진다.

* HTTP Version: Request와 같은 이유로 version을 명시한다.
* Status Code: Request에 대한 서버의 응답 코드이다. Status Code에 따라 요청의

성공/실패 여부를 세 자릿수(예시: 404)로 구분한다.

|  |  |
| --- | --- |
| 분류 | 내용 |
| 1xx Information | 요청을 받았으며 프로세스를 계속한다. |
| 2xx Successful | 요청을 성공적으로 받았으며 인식했고 수용하였다. |
| 3xx Redirection | 요청 완료를 위해 추가 작업 조치가 필요하다. |
| 4xx Client Error | 요청의 문법이 잘못되었거나 요청을 처리할 수 없다. |
| 5xx Server Error | 서버가 명백히 유효한 요청에 대해 충족을 실패했다. |

* Status Text: Status Code에 해당하는 상태 메시지 (예시: 200 OK, 404 – Not Found)

**Headers**: 해당 Response에 대한 추가 정보를 담고 있는 부분

**Body**: HTTP Response가 전송하는 데이터를 담고 있는 부분

* 전송하는 데이터가 없다면 body 부분은 비어 있다.
* 보통 GET/POST/PUT 요청일 경우, body 데이터가 포함되어 있다.

**HTTP Method**

* GET: 리소스를 조회하는 메서드로 서버에 전달하고 싶은 데이터는 query(쿼리 파라

미터, 쿼리 스트링)을 통해서 전달한다.

* HEAD: HEAD는 GET과 동일하나 body를 제외하고 상태 줄과 헤더만을 반환한다.

큰 파일이나 응답 본문을 다운로드하기 전과 같이 실제로 GET 요청을 하기

전에 GET 요청이 반환할 내용을 확인하는 데 유용하다.

if httpMethod == 'GET' or httpMethod == 'HEAD': # GET과 HEAD의 차이점은 body가 있고 없고의 차이일 뿐이기에 묶어서 구현한다.

    # version이 http 1.1이 아닐경우 응답코드 400 Bad Request 설정 및 BadRequest. html를 읽어 msgBody에 저장

    if version != "HTTP/1.1":

        sendMessage = 'HTTP/1.1 400 Bad Request\r\n'

        msgBody = readURL('/BadRequest.html')

    # client가 요청한 파일이 있는 경우 응답코드 200 OK 설정 및 요청파일을 읽어 msgBody에 저장

    elif inDir(url):

        sendMessage = 'HTTP/1.1 200 OK\r\n'

        msgBody = readURL(url)

    # client가 요청한 파일이 없는 경우 응답코드 404 Not Found 설정 및 NotFound.html를 읽어 msgBody에 저장

    else:

        sendMessage = 'HTTP/1.1 404 Not Found\r\n'

        msgBody = readURL('/NotFound.html')

    # http header

    sendMessage += msgDATE + '\r\n'

    sendMessage += 'Server: Apache\r\n'

    sendMessage += 'Content-Type: text/html; charset=utf-8\r\n'

    sendMessage += 'Content-Length: {}\r\n\r\n'.format(len(msgBody))

    # http Method 가 GET이면 body에 client가 요청한 파일 저장, HEAD면 skip

    if httpMethod == 'GET':

        sendMessage += msgBody

    elif httpMethod == 'HEAD':

        pass

server.py 104 line ~ 131 line

* POST: POST는 요청 데이터를 처리하는 메소드로, 메시지 바디를 통해 서버로 요청 데이터를 전달하고 서버는 메시지 바디를 통해 들어온 데이터를 처리하여 응답한다. 주로 신규 리소스를 등록하고, 프로세스 처리에 사용한다. POST 요청을 반복적으로 호출하면 동일한 리소스를 여러 번 생성하는 부작용이 있다.

elif httpMethod == 'POST':

msgBody = ""

    # version이 http 1.1이 아닐경우

    if version != "HTTP/1.1":

        sendMessage = 'HTTP/1.1 400 Bad Request\r\n'

    # client가 요청한 파일이 있는 경우

    elif inDir(url):

        path = './server' + url

        with open(path, "r") as json\_file:      # json 파일 읽기

            json\_data = json.load(json\_file)

        # request의 body를 parsing하여 dictionary로 저장

        item\_dic = {}

        for item in body:

            item\_list = item.replace(" ","").replace("\"", "").split(':')

            item\_dic[item\_list[0]] = item\_list[1]

        # 새로운 유저를 추가하는 함수를 호출하고, response의 body를 저장

        msgBody = userAppend(path, json\_data, item\_dic)

        sendMessage = 'HTTP/1.1 201 Created\r\n'

    # client가 요청한 파일이 없는 경우

    else:

        sendMessage = 'HTTP/1.1 404 Not Found\r\n'

    # http header

    sendMessage += msgDATE + '\r\n'

    sendMessage += 'Server: Apache\r\n'

    sendMessage += 'Content-Type: application/json; charset=utf-8\r\n'

    sendMessage += 'Content-Length: {}\r\n\r\n'.format(len(msgBody))

    sendMessage += msgBody

server.py 133 line ~ 167 line

* PUT: PUT은 리소스를 만들거나/업데이트 하기 위해서 데이터를 서버에게 보내기

위해 사용된다. POST와의 차이점으로는 클라이언트가 리소스의 위치를 알고

URI를 지정한다는 점이다. 동일한 PUT 요청을 여러 번 호출하면 항상 동일한

결과가 생성된다.

elif httpMethod == 'PUT':

    msgBody = ""

    curr\_id = int(url.split('/')[-1])   # url에서 PUT될 user id를 파싱

    # version이 http 1.1이 아닐경우

    if version != "HTTP/1.1":

        sendMessage = 'HTTP/1.1 400 Bad Request\r\n'

    # client가 요청한 파일이 있는 경우

    elif inDir(url):

        # url에서 PUT될 user id를 제거한 문자열을 다시 설정하여 server의 경로 설정

        path = './server' + '/' + '/'.join(url.split('/')[1:len(url.split('/'))-1])

        with open(path, "r") as json\_file:      # json 파일 읽기

            json\_data = json.load(json\_file)

        # request의 body를 parsing하여 dictionary로 저장

        item\_dic = {}

        for item in body:

            item\_list = item.replace(" ","").replace("\"", "").split(':')

            item\_dic[item\_list[0]] = item\_list[1]

        # PUT될 curr\_id가 이미 json파일에 있을 경우 수정, 없을 경우 새로 추가하는 로직

        # bool 변수로 수정이 된 경우 새로 추가 X, 수정이 안 된 경우 새로 추가 O

        bool = True

        for i in range(len(json\_data['user'])):

            if json\_data['user'][i]['id'] == curr\_id:

                if 'name' in item\_dic:

                    json\_data['user'][i]['name'] = item\_dic['name']

                if 'age' in item\_dic:

                    json\_data['user'][i]['age'] = int(item\_dic['age'])

                with open(path, 'w') as outfile:

                    json.dump(json\_data, outfile, indent=4)

                bool = False

        # 수정이 안 된 경우 유저를 새로 추가하는 함수를 호출하고, 201 Created

메시지를 저장

        if bool:

            msgBody = userAppend(path, json\_data, item\_dic, curr\_id)

            sendMessage = 'HTTP/1.1 201 Created\r\n'

        # 수정된 경우 204 No Content 메시지를 저장

        else:

            sendMessage = 'HTTP/1.1 204 No Content\r\n'

    # client가 요청한 파일이 없는 경우

    else:

        sendMessage = 'HTTP/1.1 404 Not Found\r\n'

    # http header

    sendMessage += msgDATE + '\r\n'

    sendMessage += 'Server: Apache\r\n'

    sendMessage += 'Content-Type: application/json; charset=utf-8\r\n'

    sendMessage += 'Content-Length: {}\r\n\r\n'.format(len(msgBody))

    sendMessage += msgBody

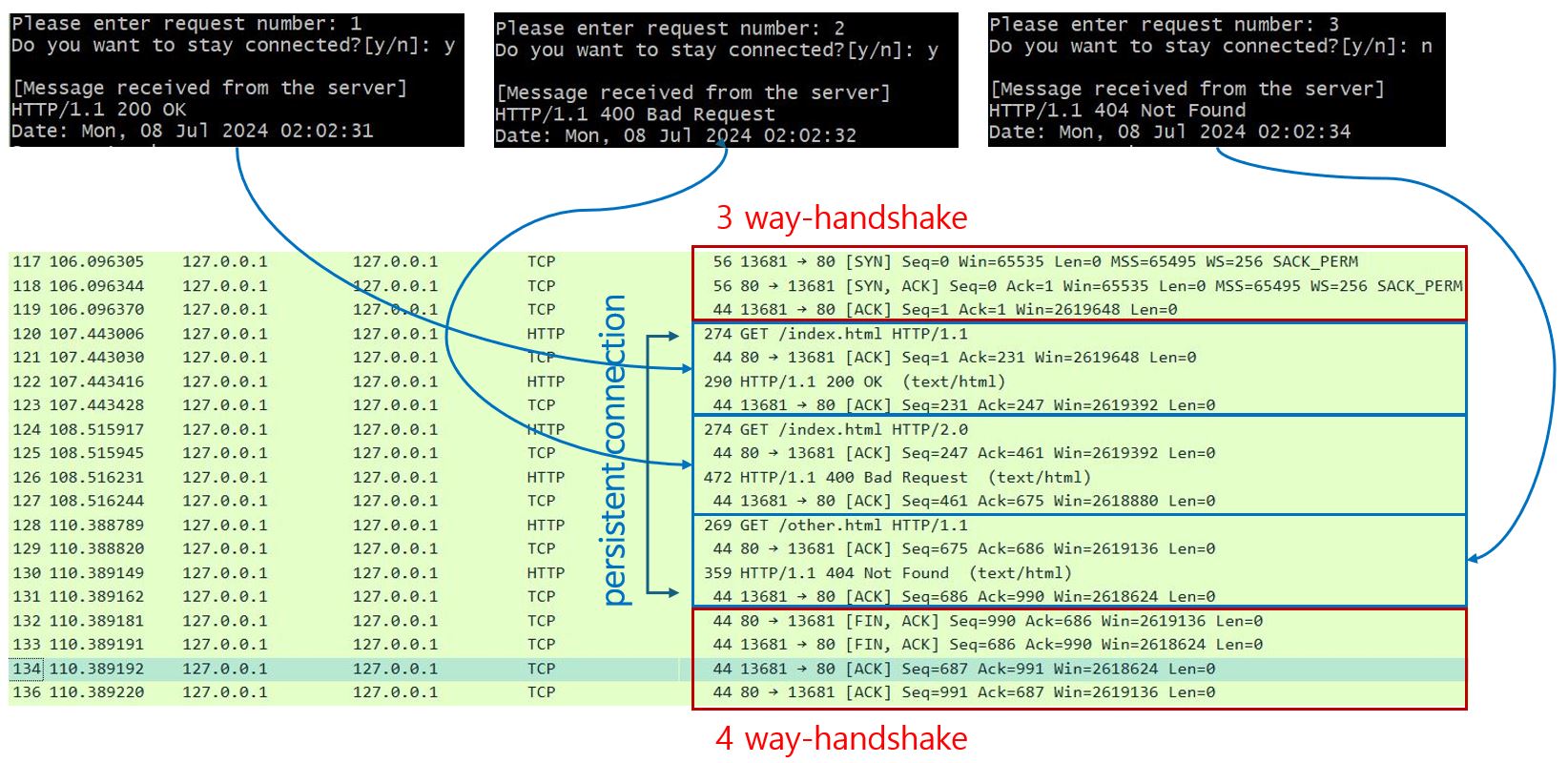
server.py 169 line ~ 223 line

**Connection Header**

http header에는 Connection이 있다. Connection에는 close, Keep-Alive가 있다.

- close는 메세지 교환 후 TCP 연결 종료, Keep-Alive는 메세지 교환 후 TCP 연결 유지

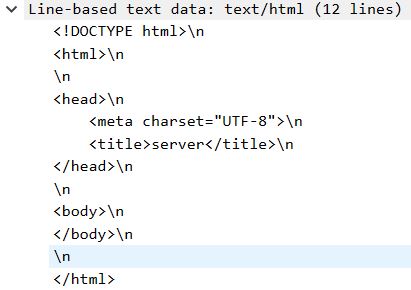
즉, Connection header를 통해 HTTP 1.1의 특성 중 하나인 persistent connection을 구현할 수 있다.



**WireShark 분석**

1. GET /index.html HTTP/1.1

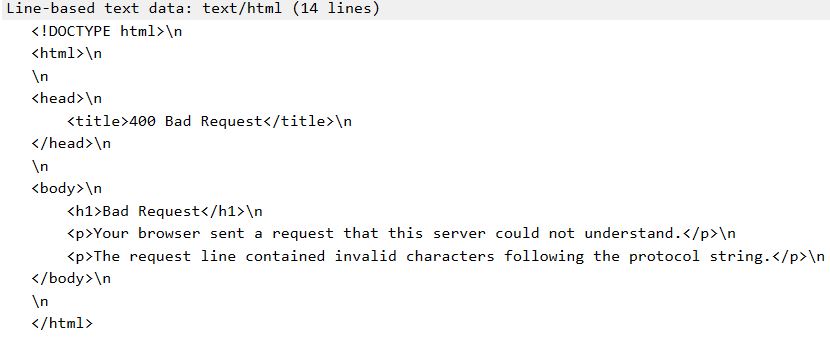




올바른 request이므로 200 OK, 그리고 요청한 index.html을 body로 전달한다.

2. GET /index.html HTTP/2.0



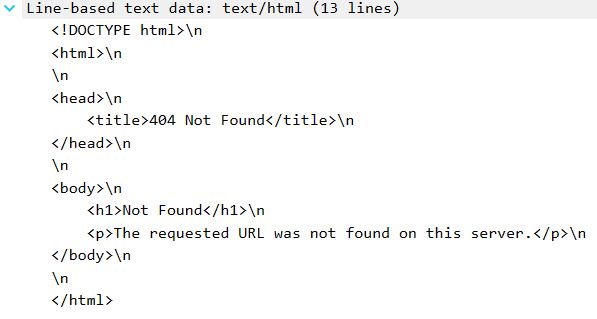


request의 http version이 2.0으로 1.1과 맞지 않으므로 400 Bad Request를 전송한다.

그리고 BadRequest.html를 body로 전달한다.

3. GET /other.html HTTP/1.1





요청한 other.html 파일이 server 디렉터리 안에 존재하지 않기 때문에 404 Not Found로 응답한다. 그리고 NotFound.html를 body로 전달한다.

4. HEAD /index.html HTTP/1.1



올바른 request이므로 200 OK, HEAD 이므로 요청한 /index.html을 전달하지 않고

header만 전달한다.

5. HEAD /index.html HTTP/2.0

****

request의 http version이 2.0으로 1.1과 맞지 않으므로 400 Bad Request를 전송한다.

그리고 BadRequest.html를 전달하지 않고 header만 전달한다.

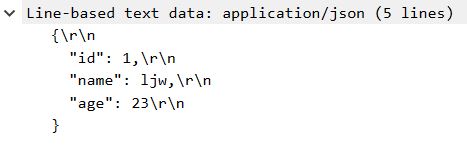
6. HEAD /other.html HTTP/1.1



요청한 other.html 파일이 server 디렉터리 안에 존재하지 않기 때문에 404 Not Found로 응답한다. 그리고 NotFound.html를 body로 전달하지 않고 header만 전달한다.

7. POST /user.json HTTP/1.1 | body: "name": "ljw", "age": 23





올바른 request이고 생성에 성공했으므로 201 Created, 그리고 생성된 user 정보 json

을 전달한다.

8. POST /user.json HTTP/2.0 | body: "name": "ljw", "age": 23



request의 http version이 2.0으로 1.1과 맞지 않으므로 400 Bad Request를 전송한다.

그리고 생성에 실패했으므로 user json을 전달하지 않는다.

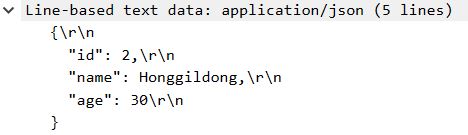
9. POST /other.json HTTP/1.1 | body: "name": "ljw", "age": 23



요청한 other.json파일이 server 디렉터리 안에 존재하지 않기 때문에 404 Not Found로 응답한다. 그리고 생성에 실패했으므로 user json을 전달하지 않는다.

10. PUT /user.json/2 HTTP/1.1 | body: name": "Hong gil dong", "age": 30





요청한 other.json파일이 server 디렉터리 안에 존재하고, json파일에 id가 2인 user가 없어 생성하기 때문에 201 Created, 그리고 생성된 user 정보 json을 전달한다.

11. PUT /user.json/2 HTTP/1.1 | body: "age": 15



요청한 other.json파일이 server 디렉터리 안에 존재하고, json파일에 id가 1인 user가 이미 있어 수정하기 때문에 204 Content, 수정된 user 정보는 전달되지 않는다.

12. PUT /user.json/1 HTTP/2.0 | body: "age": 47



request의 http version이 2.0으로 1.1과 맞지 않으므로 400 Bad Request를 전송한다.

그리고 생성 및 수정에 실패했으므로 user json을 전달하지 않는다.

13. PUT /other.json/1 HTTP/1.1 | "age": 47



요청한 other.json파일이 server 디렉터리 안에 존재하지 않기 때문에 404 Not Found로 응답한다. 그리고 생성 및 수정에 실패했으므로 user json을 전달하지 않는다.

**참고사이트**

wireshark - <https://en.wikiversity.org/wiki/Wireshark/HTTP>

http message - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

status code - <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Connection - <https://developer.mozilla.org/ko/docs/Web/HTTP/Headers/Connection>

KT 포트포워딩 -

<https://atnbt.com/%EC%BC%80%EC%9D%B4%ED%8B%B0-%EA%B3%B5%EC%9C%A0%EA%B8%B0-%ED%8F%AC%ED%8A%B8%ED%8F%AC%EC%9B%8C%EB%94%A9/>

서로 다른 네트워크 PC에서 socket 연결하기 -

<https://stackoverflow.com/questions/62631176/python-socket-to-connect-over-global-public-ip-address>

출력 buffering 해결하기 –

<https://reddit.com/r/AskProgramming/comments/nis6uu/python_sockets_clientserver_why_is_my_server_not/>