백엔드 팀 실습

3주차

16 김성민

ARGOS

INDEX

백엔드 팀 3주차

1:1 채팅 프로그램 구현

- 개요
- chat_server.py 구현
- chat_client.py 구현
- 결과 화면

ARGOS

개요

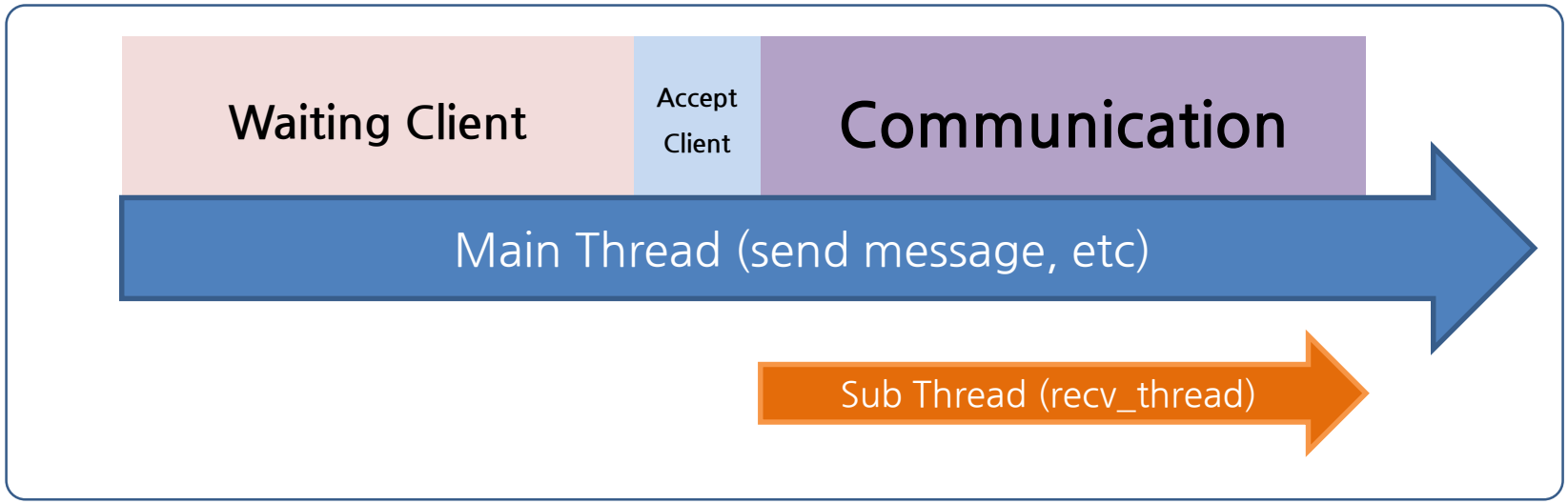
ARGOS

목표 : 소켓 통신을 이해하고, 이를 코드로 직접 구현 하여 본다.

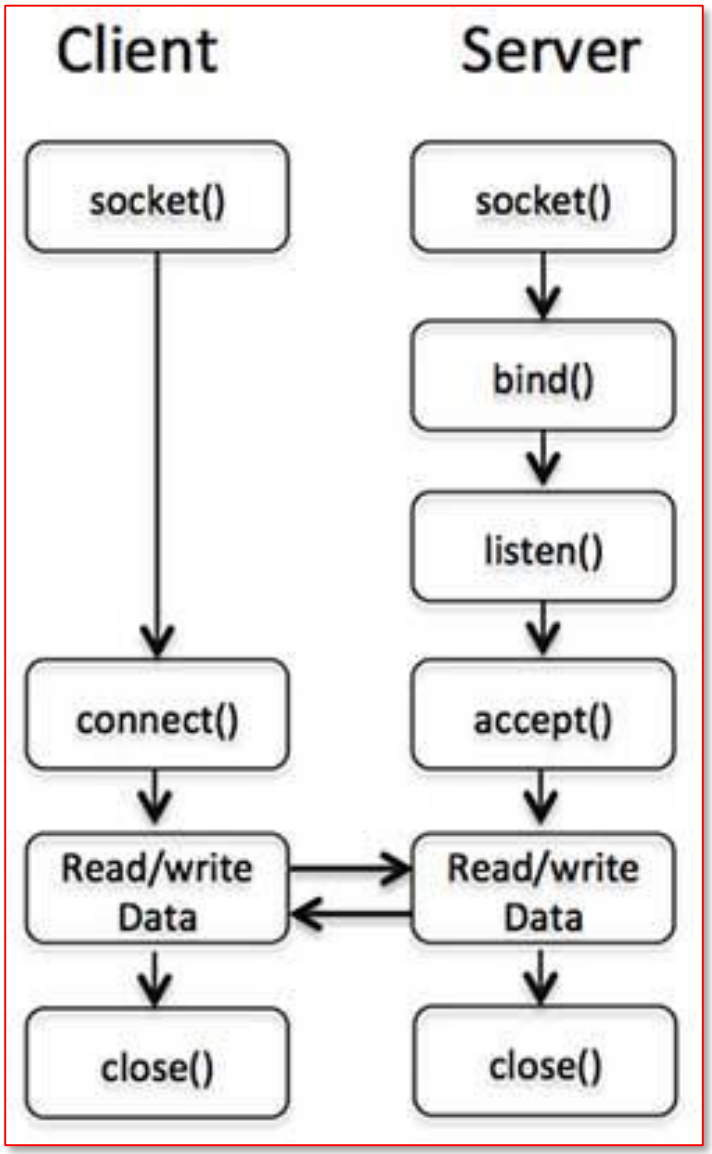
쓰레드, 소켓 개념을 이해한 뒤 간단한 1:1 통신 프로그램을 파이썬으로 구현한다.

Network socket (Wikipedia) : https://en.wikipedia.org/wiki/Network_socket

Thread (Wikipedia) : https://en.wikipedia.org/wiki/Thread_(computing)



< 서버 프로세스 흐름도 >


Client
Server
socket()
socket()
bind()
listen()
connect()
accept()
Read/write Data
Read/write Data
close()
close()

〈 소켓 통신 흐름도 〉

chat_server.py 구현

Abstract

1. 서버에서 socket을 생성한다.
2. 해당 socket을 자신의 IP와 특정 Port 번호에 bind 한다.
3. 그 뒤 listen 상태를 유지한다.
4. 클라이언트에서 해당 socket에 접속하면 이를 accept한 뒤,
   새로운 thread(recv_thread)를 생성한다.
5. main thread에서 표준 입력으로 메시지를 입력하면 클라이언트와 연결된
   소켓으로 메시지를 전달한다.
6. recv_thread는 클라이언트 소켓에서 메시지를 대기하다가, 메시지가 도착하면
   이를 표준 출력으로 출력한다.

```python
import socket, sys, threading

def recv_msg(message_socket):
    while True:
        msg = message_socket.recv(1024).decode()
        print(msg)

def main():
    if len(sys.argv) != 2:
        print("python3 %s [PORTNUMBER]" % sys.argv[0])
        sys.exit()

    PORT = int(sys.argv[1])

    # init socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('', PORT))
    server_socket.listen(1)
    print ("The chat server is started on port " + str(PORT))
    print ("Waiting for client...")

    (client_socket, addr) = server_socket.accept()
    print ("Client connected, start chat : ")

    recv_thread = threading.Thread(target = recv_msg, args = (client_socket,))
    recv_thread.start()

    while True:
        message = input('')
        client_socket.send(('[Server] '+ message).encode())

if __name__ == "__main__" :
    main ()
```

ARGOS

소스코드

chat_client.py 구현

Abstract

1. 특정한 IP 주소와 Port에 소켓을 연결한다.
2. 연결에 성공하였다면 새로운 thread (recv_thread)를 생성한다.
3. main thread에서 표준 입력으로 메시지를 입력하면 서버와 연결된 소켓으로 메시지를 전달한다.
4. recv_thread는 서버 소켓에서 메시지를 대기하다가, 메시지가 도착하면 이를 표준 출력으로 출력한다.

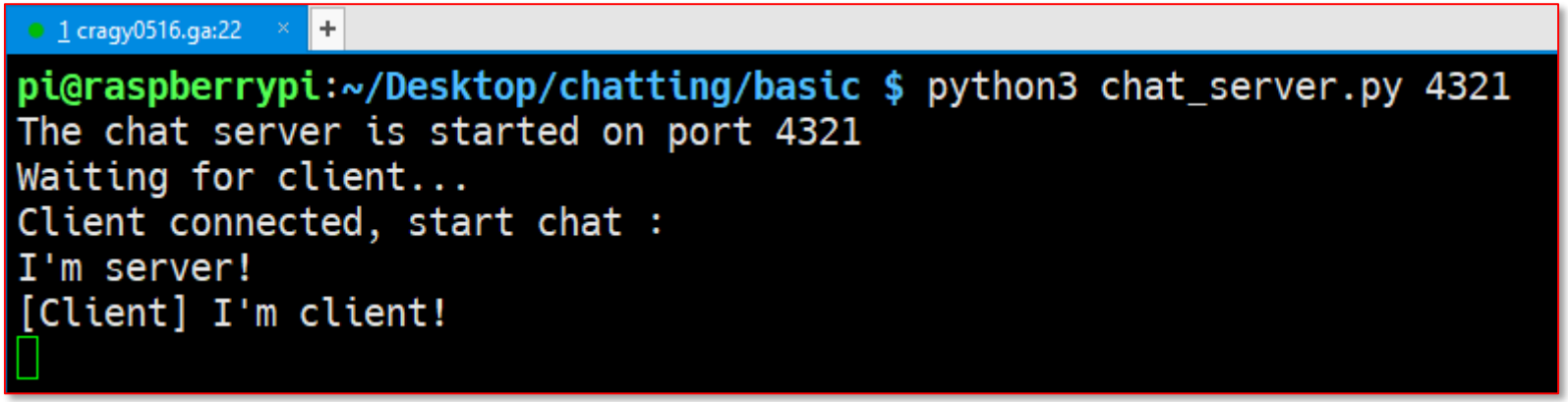
```python
import socket, sys, threading

def recv_msg (message_socket) :
    while True:
        msg = message_socket.recv(1024).decode()
        print(msg)

def main():
    if len(sys.argv) != 3:
        print("python3 %s [IPADDRESS] [PORTNUMBER]" % sys.argv[0])
        sys.exit()

    IP_ADDR = sys.argv[1]
    PORT = int(sys.argv[2])

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((IP_ADDR, PORT))

    print ("Server connected")
    print ("Start chat : ")

    recv_thread = threading.Thread(target=recv_msg, args=(client_socket,))
    recv_thread.start()

    while True:
        message = input('')
        client_socket.send(('[Client] ' + message).encode())

if __name__ == "__main__" :
    main()
```

ARGOS

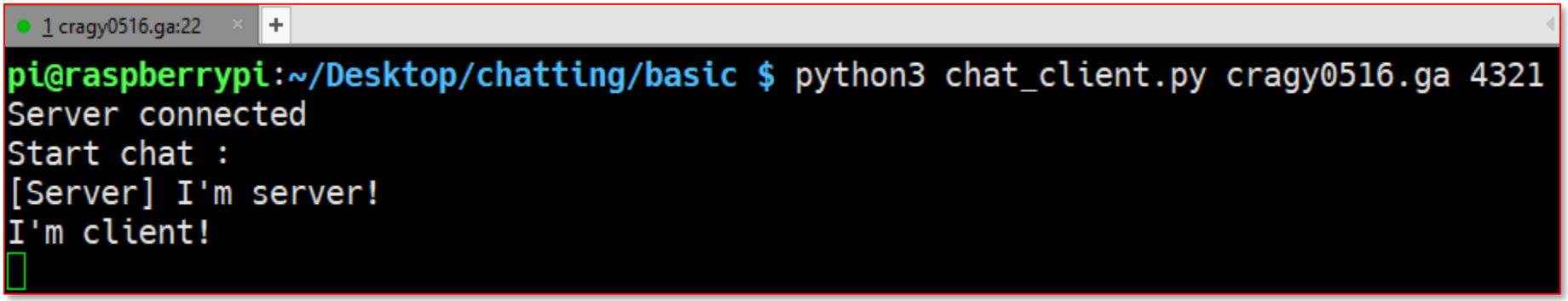
소스코드

결과 화면

ARGOS



chat_server.py 실행



chat_client.py 실행

추가 사항

- python3 로 진행해야 함
  python --version 으로 버전 체크.

- 같은 서버에서 진행할 경우 도메인 대신 localhost로도 가능

- 클라이언트와 서버가 다른 네트워크에 있을 경우 도메인을 통해 접근
  같은 네트워크, 다른 네트워크, 같은 서버 등 자유롭게 해보며 익히길 권장

- Address already in use 에러가 뜰 경우 다른 포트로 진행
  종료 시 연결을 끊어주는 부분이 완성되어 있지 않아서 발생.
  어느정도 시간이 지나면 죽은 연결은 운영체제가 복구하므로 일단은
  그냥 진행해도 무방함.

백엔드 팀 3주차

Group 채팅 프로그램 구현

- 개요
- chat_server.py 구현
- chat_client.py 구현
- 결과 화면

ARGOS

개요

목표 : Select를 이해한 후, 이를 이용해 1:1 통신 프로그램을 그룹 채팅 프로그램으로 확장 구현한다.

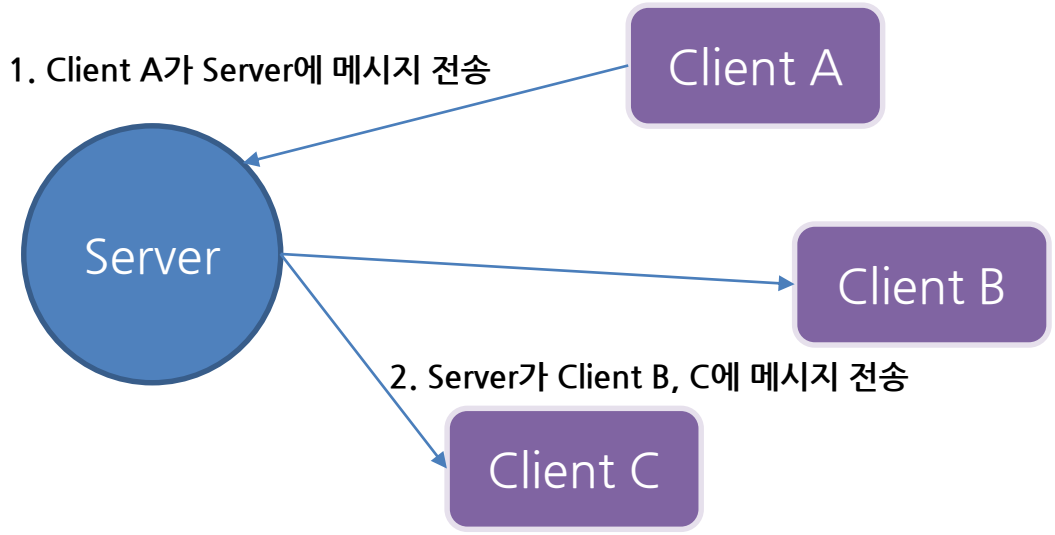
Select : File description들의 상태 변화를 감지할 수 있음.
여러 socket에 대한 다중 처리 시, socket들을 감시하고 원할 때 제어할 수 있음.
Event가 발생했을 때 이를 감지할 수 있음.

Python Select Documentation : https://docs.python.org/3/library/select.html

브로드캐스트 (Braodcast)

네트워크 내에서 연결된 모든 사용자에게 메시지를 송신하는 것.

채팅 프로그램에서, 서버는 접속한 클라이언트들의 소켓을 관리한다.
서버-클라이언트는 메시지를 1:1로 주고받으므로, 서버는 클라이언트로부터 받은 메시지를 다른 클라이언트들에게 모두 뿌려주어야 할 필요가 있다.
따라서 이를 브로드캐스트로 전송한다.



chat_server.py 구현

ARGOS

```
1 import socket, sys, select
2
3 USER_LIST = []
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5
6 def broadcast (socket, msg) :
7     global USER_LIST, server_socket
8     # send broadcast except server and target
9     print ("[Broadcast] ", msg)
10    for sock in USER_LIST :
11        if sock != server_socket and sock != socket and sock != sys.stdin:
12            sock.send(msg.encode( ))
```

broadcast 구현

해당 함수는 인자로 받은 **socket** 및 서버를 제외한 모든 **USER_LIST**에 메시지를 전송함.

USER_LIST는 서버 및 클라이언트의 소켓(및 서버 stdin)을 리스트로 저장하고 있음.


```
14 def main () :
15     global USER_LIST, server_socket
16     if len(sys.argv) != 2 :
17         print ("python3 %s [PORTNUMBER]" % sys.argv[0])
18         sys.exit()
19
20     PORT = int(sys.argv[1])
21
22     # init socket and the list
23     server_socket.bind(('', PORT))
24     server_socket.listen(5)
25     USER_LIST.append(sys.stdin)
26     USER_LIST.append(server_socket)
27
28     print ("Chat server start on port " + str(PORT))
29     print ("Start chat : ")
30
```

처음 서버를 실행하면 **server_socket**을 포트에 bind & listen 하며

USER_LIST에 **서버 소켓** 및 **stdin** (표준 입력) File Descriptor를 추가한다.

```
31      while True :
32          try: r_sockets, w_sockets, e_sockets = select.select(USER_LIST, [], [])
33          except: break
34
35          for sock in r_sockets :
36              if sock ==[redacted] :
37                  # When Server says...
38                  data = sock.readline( )
39                  if data[:4] == "quit" :
40                      server_socket.close( )
41                  else :
42                      broadcast (None, "\r" + "[SERVER] " + data)
43                      continue
44              elif sock == [redacted] :
45                  # When new client should be accepted
46                  sockfd, addr = [redacted].accept( )
47                  USER_LIST.append(sockfd)
48                  broadcast (sockfd, "\r" + "Client (%s:%s) entered room\n" % addr)
49
50              else:
51                  data = [redacted].recv(4096).decode( )
52                  # data from client
53                  if data[:4].lower( ) == "quit" :
54                      broadcast(sock, "\r" + "Client " + str([redacted].getpeername( )) + " is disconnected.\n")
55                      [redacted].close( )
56                      USER_LIST.remove([redacted])
57                  elif data :
58                      broadcast([redacted], "\r" + "<" + str([redacted].getpeername( )) + "> " + data)
59
60      print ("Server closed")
61
62
63  if __name__ == "__main__" :
64      main( )
```

r_socket : 읽을 것이 있는 fd List

서버가 채팅을 쳤을 때

(어디에 읽을 것이 있는가??)

새로운 클라이언트가 접속했을 때

다른 클라이언트가 채팅 했을 때

클라이언트 퇴장 (quit)

그 외 (일반 채팅)

chat_client.py 구현

ARGOS

```python
import socket, sys, select

def prompt() :
    sys.stdout.write("<You> ")
    sys.stdout.flush()

def main () :
    if len(sys.argv) != 3 :
        print("python3 %s [IPADDRESS] [PORTNUMBER]" % sys.argv[0])
        sys.exit()

    ip_address = sys.argv[1]
    port_number = int(sys.argv[2])

    # init client socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((ip_address, port_number))

    print ("Hello, host %s" % ip_address)
    prompt()
```

```
22      while True :
23          # observe server and stdin
24          socket_list = [                ,                    ]
25          r_sockets, w_sockets, e_sockets = select.select(socket_list, [], [])
26          for sock in r_sockets :
27              if sock ==                   :
28                  # from server message
29                  data = sock.recv(4096).decode( )
30                  if not data :
31                      print ("bye")
32                      sys.exit( )
33                  else :
34                      sys.stdout.write(data)
35                      prompt( )
36              elif sock ==                 :
37                  # client inputs something
38                  msg =              .readline( )
39                                     .send(msg.encode( ))
40                  prompt( )
41
42  if __name__ == "__main__" :
43      main( )
```

빈 칸을 채워보세요~

결과 화면

ARGOS

```
pi@raspberrypi:~/Desktop/chatting/select $ python3 chat_server.py 12345
Chat server start on port 12345
Start chat :
Client (168.188.123.210:49684) entered room

Client (127.0.0.1:53312) entered room

Client (168.188.123.189:39078) entered room
```

서버를 열고 클라이언트에서 접속

```
pi@raspberrypi:~/Desktop/chatting/select $ python3 chat
_client.py cragy0516.ga 12345
Hello, host cragy0516.ga
Client (127.0.0.1:53312) entered room
Client (168.188.123.189:39078) entered room
[SERVER]
<You>
```

클라이언트에서 접속

백엔드 팀 3주차

flask 입문

- flask 설치
- flask + nginx

ARGOS

flask 설치

flask (Wikipedia) : https://en.wikipedia.org/wiki/Flask_(web_framework)

파이썬으로 작성된 마이크로 웹 프레임워크

가상환경 설치 : sudo apt-get install python3-venv

mkdir myproject

cd myproject

python3 -m venv venv

가상환경 활성화 :

. venv/bin/activate

비활성화 : deactivate

ARGOS

Flask 설치 (가상환경 내에서)

pip install Flask

디렉토리 & 파일 생성

./myproject

./myproject/hello.py

./myproject/templates

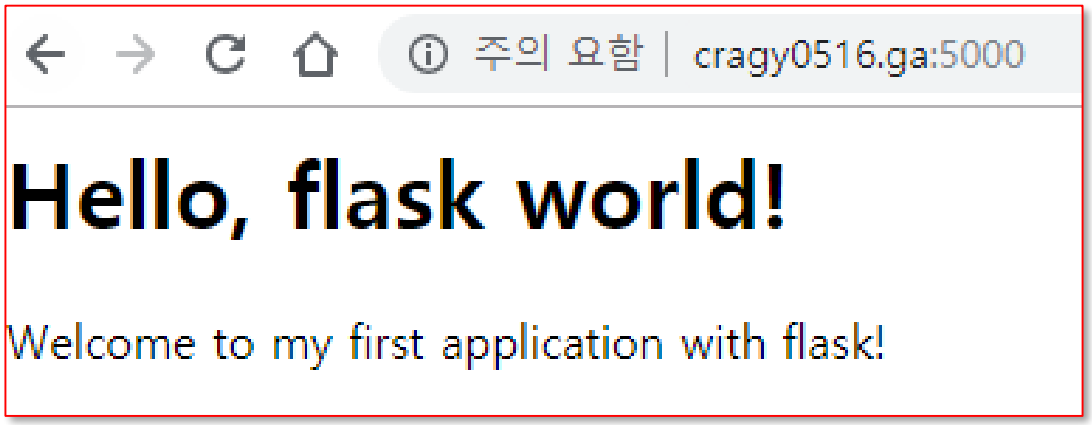
./myproject/templates/hello.html

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

from flask import Flask, render_template, session

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('hello.html')

if __name__ == '__main__':
    app.run()
```

hello.py

```
(venv) pi@raspberrypi:~/Desktop/chatting/flask $ export FLASK_APP=hello.py
(venv) pi@raspberrypi:~/Desktop/chatting/flask $ export FLASK_ENV=development
(venv) pi@raspberrypi:~/Desktop/chatting/flask $ flask run --host 168.188.123.210 --
port 5000
 * Serving Flask app "hello.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://168.188.123.210:5000/ (Press CTRL+C to quit)
 * Restarting with stat
```

flask 내장 웹 서버 실행



웹 페이지 접속 확인

현재 HTTPS 통신은 안 됨.

flask + nginx

uWSGI

WSGI : 파이썬 웹 어플리케이션과 웹 서버간의 연결을 위한 파이썬 프레임워크
WSGI 인터페이스는 파이썬 표준이며, 이 인터페이스에 맞추어서 제작된
프레임워크가 uWSGI

웹 어플리케이션과 웹 서버를 연결하여 서비스할 수 있도록 해 보자

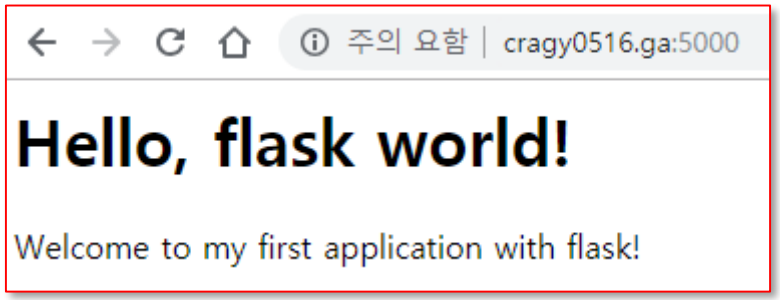
pip install uwsgi

./myproject/wsgi.py 파일 작성

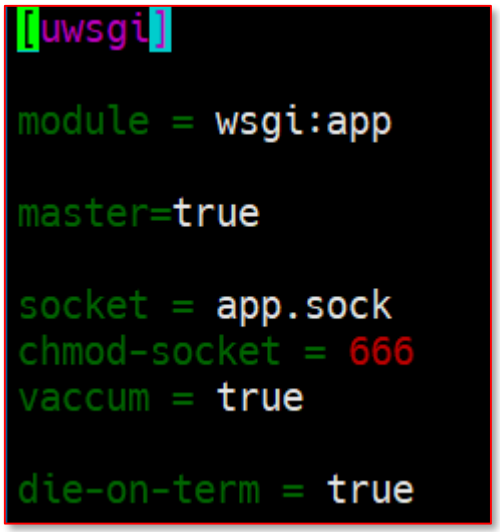
```
from hello import app

if __name__ == "__main__":
    app.run()
```

wsgi.py 파일

uwsgi --socket 168.188.123.210:5000 --protocol=http -w wsgi --callable app



웹 서버 실행 후 접속 확인



옵션을 미리 지정 하도록 ini 파일 작성

./myproject/app.ini 파일 작성

```
164 server {
165         listen 5000;
166         server_name cragy0516.ga;
167         location / {
168                 try_files $uri @app;
169         }
170         location @app {
171                 include uwsgi_params;
172                 uwsgi_pass unix:/home/pi/Desktop/chatting/flask/app.sock;
173         }
174 }
```

새로운 서버 설정 추가

sudo vi /etc/nginx/sites-available/default

```
[uwsgi]
chidr = /home/pi/Desktop/chatting/flask
uid = pi
gid = pi
chmod-socket = 666
socket = /home/pi/Desktop/chatting/flask/app.sock
module = hello
callable = app
virtualenv = /home/pi/Desktop/chatting/venv
```

ARGOS

module : 파이썬 파일 이름 (hello.py)

callable : Flask 인스턴스 생성 시 이름

sudo vi /etc/uwsgi/apps-available/uwsgi.ini 작성

available 파일을 enable로 설정 (심볼릭 링크 설정)

sudo ln -s /etc/uwsgi/apps-available/uwsgi.ini /etc/uwsgi/apps-enabled/

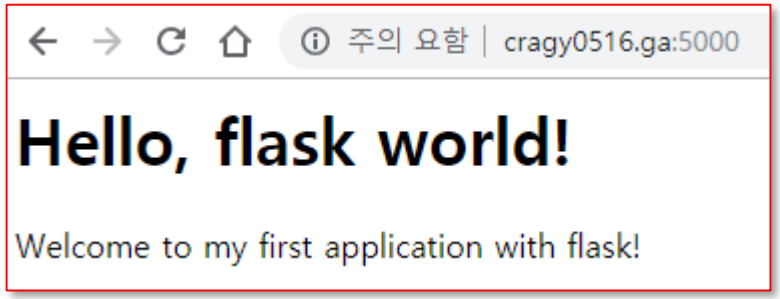
서비스 재시작

sudo service nginx restart

sudo service uwsgi restart

uwsgi 실행

uwsgi --ini app.ini



웹 서버 실행 후 접속 확인

서비스로 등록 : 일일이 실행시키지 않도록 서비스로 등록.

sudo vi /etc/systemd/system/app.service

```
[source]

[Unit]
Description=uWSGI instance to serve flask application
After=network.target

[Service]
User=pi
Group=pi
WorkingDirectory=/home/pi/Desktop/chatting/flask
Environment="PATH=/home/pi/Desktop/chatting/flask/venv/bin"
ExecStart=/home/pi/Desktop/chatting/flask/venv/bin/uwsgi --ini app.ini

[Install]
WantedBy=multi-user.target
```

sudo service app start

Q & A

Thank You for Listening

Created By cragy0516
Thanks To ccoma