Flask

Flask의 기본 구조

|  |  |
| --- | --- |
| 구조 | 이클립스 예 |
| 프로젝트명  flaskApp명(PyDev Package)  --- static (요청시 바로 해석되어 응답이 되는 파일들)  --- css  --- images  --- js  --- templates (view 페이지가 있는 곳)  --- \_\_init\_\_.py  start\_flask.py (요건 내 맘대로 쓴 것) | EvCar와 HRD라는 모듈 패키지를  등록한 예 |

Flask는 먼저 Python이 설치되어 있는 상태에 PIP패키지를 통해 설치할 수 있다.

그럼 window환경에서 cmd창을 열고 설치해 보자!

먼저 pip패키지를 업그레이드 하자!

|  |
| --- |
| python -m pip install pip --upgrade |

이제 프로젝트명으로 이동한 후 pip를 이용하여 Flask를 설치하도록 하자!(아무 곳에서 해도 됨)

|  |
| --- |
| pip install flask |

다음은 Spring환경 같은 곳에서 Flask를 호출할 때 CORS 크로스 도메인 이슈인 다음과 같은 오류가 발생한다.

**No 'Access-Control-Allow-Origin' header is present on the requested resource.**

이 부분을 해결하기 위해 Flask-cors패키지를 설치해 둬야 한다.

|  |
| --- |
| pip install -U flask-cors |

간단하게 확인하는 예제

EvCar안에 있는 \_\_init\_\_.py를 다음과 같이 작성하자!

|  |
| --- |
| FlaskTest/EvCar/\_\_init\_\_.py |
| from flask import Flask  app = Flask(\_\_name\_\_)  *@app.route*(*"/test"*)  def **test**():  return *"Hello Flask! "* |

사실은 각 모듈 위에서 실행해도 된다. 하지만 그렇게 되면 내부적인 포퍼먼스가 많이 생기므로 될 수 있으면 프로젝트명 안에서 만든 **start\_flask.py를 다음과 같이 정의하고, 실행**시키자!

그럼 프로젝트 이름에서 마우스의 우측버튼을 클릭하여 새로운 [PyDev Module] 를 만들어서 Package는 비워놓고, 파일명을 start\_flask.py로 하자! start\_flask.py의 내용을 다음과 같이 간단하게 정의해 보자.

|  |  |
| --- | --- |
| FlaskTest/start\_flask.py | 결과화면 |
| from EvCar import app  app.run(host=*'127.0.0.1'*) |  |

그리고 ServletContext와 같은 객체가 있다. 그것을 파이썬에서는 g로 불리며, Global Object로 인식된다. 예제로 살펴보자!

|  |  |
| --- | --- |
| FlaskTest/EvCar/\_\_init\_\_.py | 결과화면 |
| from flask import Flask, g  app = Flask(\_\_name\_\_)  #다음은 모든 요청에 앞서 수행하는 함수다.  *@app.before\_request*  def **before\_request**():  g.msg = *'g에 저장한 값'*  *@app.route*(*"/test"*)  def **test**():  return *"Hello Flask! "*+ getattr(g, *'msg'*) |  |

일반 문자열이 아니라 HTML코드를 반환하면 어떻게 될까?

이제 Flask앱에 다음과 같이 함수를 하나 추가하고, 실행해 보자!

|  |
| --- |
| FlaskTest/EvCar/\_\_init\_\_.py |
| from flask import Flask  app = Flask(\_\_name\_\_)  ...  *@app.route*(*'/'*)  def **hello\_world**():  str = *'''*  *<h1>Hello world</h1>*  *'''*  return str |
|  |
| 브라우저 에서 실행한 화면 ( **127.0.0.1:5000 또는 localhost:5000** ) |

그건 그렇고 우린 Spring환경과 어떻게 쓸 것인가? 이것이 중요한데… 이는 비동기식 통신이 가장 편리한 방법이다. 하지만 Spring환경에서 Ajax비동기식 통신을 이용하면 다음과 같은 오류가 발생한다.

**No 'Access-Control-Allow-Origin' header is present on the requested resource.**

이 부분을 수행하기 위해 우린 앞서 Flask-cors패키지를 설치한 상태다. 소스코드를 다음과 같이 수정하자!

|  |  |
| --- | --- |
| FlaskTest/EvCar/\_\_init\_\_.py | |
| from flask import Flask, g**, jsonify**  **from flask\_cors import CORS**  app = Flask(\_\_name\_\_)  **CORS(app)**  ...  ...  *@app.route*(*'/'*)  def **hello\_world**():  str = *'''*  *<h1>Hello world</h1>*  *'''*  return **jsonify(code=str)** | |
|  | |
| 스프링 에서 실행할 View페이지 | |
| <%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>  <%@ page session=*"false"* %>  <html>  <head>  <title>Home</title>  <style>  **div***#box*{  width: *200px*;  height: *120px*;  border: *1px solid black*;  }  </style>  </head>  <body>  <h1>  Hello world!  </h1>  <input type=*"button"* value=*"Request"* id=*"req\_bt"*/>    <div id=*"box"*></div>    <script src=*"resources/js/jquery-3.4.1.min.js"*></script>  <script>  $(**function**(){    $("#req\_bt").bind("click",**function**(){  $.ajax({  url:"http://127.0.0.1:5000",  type: 'get'  }).done(**function**(data){  $("#box").html(data.code);  });  });  });  </script>  </body>  </html> | |
|  |  |

[Request]버튼을 클릭하면 Flask에서 전달하는 JSON값을 받아 div에 표현하는 것을 알 수 있다.

(아직은… 파이썬 소스코드를 수정한 후 반드시 Flask서버를 다시 실행해야 함)

공공데이터 활용

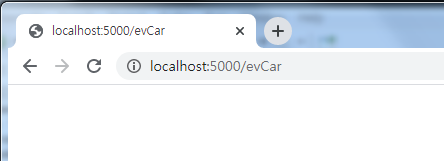
스프링에서 해 봤던 공공데이터(data.go.kr)의 자원을 파이썬에서 가공해서 전달하는 흐름을 구현해 보자!

|  |
| --- |
| 간단한 흐름도 |
| **Sping환경**  ActionController    View  JSON반환  **Flask환경**  Module  Public Data  router |

|  |
| --- |
| FlaskTest/EvCar/\_\_init\_\_.py |
| import xml.etree.ElementTree as ET  import pandas as pd  from pandas import Series, DataFrame  from urllib.request import urlopen  from flask import Flask, jsonify  from flask\_cors import CORS  app = Flask(\_\_name\_\_)  CORS(app)  ...  ...  spec = *"http://open.ev.or.kr:8080/openapi/services/EvCharger/getChargerInfo?serviceKey={개인인증키}"*  *@app.route*(*'/'*)  def **hello\_world**():  str = *'''*  *<h1>Hello world11</h1>*  *'''*  return jsonify(code=str)  *@app.route*(*'/evCar'*)  def **ev\_car**():  res = urlopen(spec).read()  xmlDoc = ET.fromstring(res)  print(*'------------------------------------------------------------------'*)  items = xmlDoc.find(*'body'*).find(*'items'*)  # print(items)  rows = []  for node in items:  statNm = node.find(*'statNm'*).text  chgerType = node.find(*'chgerType'*).text  addr = node.find(*'addr'*).text  rows.append({*'s\_name'*:statNm, *'type'*:chgerType, *'addr'*:addr})    df = DataFrame(rows)  print(df)  return *''* |

위와 같이 소스코드를 수정한 후 브라우저 창을 열어 다음과 같이 요청하면 브라우저 창에는 아무런 결과가 나타나지 않지만 Flask에서는 print(df) 한 결과가 나타난다.

**위 코드내용을 PostMan이나 브라우저에서 호출했을 때 오류가 난다면 우선  
공공데이터 url경로(spec)에 “https”로 시작하는지 확인한 후 http로 수정해야 함!**



|  |
| --- |
|  |

이제 Spring환경에서 Ajax비동기식 통신을 이용하여 호출하도록 하고, Flask쪽에서는 공공데이터에서 얻은 자원들을 DataFame으로 변환한 것을 다시 JSON으로 응답하도록 하자!

그렇게 되면 Flask에서 전달하는 JSON을 Spring에서 받아 JSP에서 표현하는 과정을 만들자!

|  |  |
| --- | --- |
| Flask소스 | |
| import xml.etree.ElementTree as ET  import pandas as pd  from pandas import Series, DataFrame  from urllib.request import urlopen  from flask import Flask, jsonify  from flask\_cors import CORS  app = Flask(\_\_name\_\_)  CORS(app)  spec = *"http://open.ev.or.kr:8080/openapi/services/EvCharger/getChargerInfo?serviceKey={개인인증키}"*  *@app.route*(*'/'*)  def **hello\_world**():  str = *'''*  *<h1>Hello world11</h1>*  *'''*  return jsonify(code=str)  *@app.route*(*'/evCar'*, **methods=[*'POST'*]**)  def **ev\_car**():  res = urlopen(spec).read()  xmlDoc = ET.fromstring(res)  print(*'-------------------------------------------------------------------'*)  items = xmlDoc.find(*'body'*).find(*'items'*)  # print(items)  rows = []  for node in items:  statNm = node.find(*'statNm'*).text  chgerType = node.find(*'chgerType'*).text  addr = node.find(*'addr'*).text  rows.append({*'s\_name'*:statNm, *'type'*:chgerType, *'addr'*:addr})    df = DataFrame(rows)  # print(df)  **json = df.to\_json(orient=*'records'*)**  **return json** | |
| Spring의 JSP소스 | |
| <%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>  <%@ page session=*"false"* %>  <html>  <head>  <title>Home</title>  <style>  **div***#box*{  width: *200px*;  height: *120px*;  border: *1px solid black*;  }  </style>  </head>  <body>  <h1>  Hello world!  </h1>  <input type=*"button"* value=*"Request"* id=*"req\_bt"*/>    <div id=*"box"*></div>  <p></p><p></p><p></p><p></p>    <input type=*"button"* value=*"Request"* id=*"req\_bt2"*/>  <ol id=*"ol"*></ol>  <script src=*"resources/js/jquery-3.4.1.min.js"*></script>  <script>  $(**function**(){    $("#req\_bt").bind("click",**function**(){  $.ajax({  url:"http://127.0.0.1:5000",  type: 'get'  }).done(**function**(data){  $("#box").html(data.code);  });  });    $("#req\_bt2").bind("click",**function**(){  $.ajax({  url:"http://127.0.0.1:5000/evCar",  type: 'post',  dataType: 'json'  }).done(**function**(data){  **var** str = '';  **for**(**var** i=0; i<data.length; i++)  str += '<li>'+data[i].s\_name+'</li>';    $("#ol").html(str);  });  });  });  </script>  </body>  </html> | |
|  |  |

[문제]

공공데이터에서 전기자동차 충전소 현황 서비스를 이용하여 전국 충전소 목록을 얻어 빅데이터 통계를 내어 전국 충전소 수를 구해서 JSON으로 제공하고, 그것을 Spring환경의 View에서 비동기식 통신으로 받아 차트로 표현하라!

다음은 하나의 도메인에서 두개의 Flask app을 실행하는 방법을 알아보자!

앞서 정의된 구조 EvCar와 HRD를 그대로 두고 각 모듈 패키지 안에 있는 \_\_init\_\_.py파일들의 내용을 다음과 같이 수정하자!

먼저 다음의 패키지를 하나 설치해야 한다.

|  |
| --- |
| pip3 install gunicorn |

디스패처 미들웨어 솔루션은 하나의 gunicorn 인스턴스 만 사용한다. (사실 다른 것도 많음!)

|  |  |
| --- | --- |
| 구조 | EvCar/\_\_init\_\_.py |
|  | from flask import Flask, request, g  from EvCar import PandasTest1  def **create\_app**():  app\_name = *'EvCar'*    app = Flask(\_\_name\_\_, instance\_relative\_config=True)    *@app.route*(*"/"*)  def **first**():  return *'Hello Flask! '*+app\_name+*' request.url='*+request.url    return app |

instance\_relative\_config=True 부분은 인스턴스가 생길 때의 인스턴스 폴더를 지정하는 것인데 생략하면 /instance다. 물론 기본적으로 절대경로로 지정하는 것을 기본으로 하는데 상대경로로 지정하고자 할 때는 위와 같이 기술해야 한다.

|  |  |
| --- | --- |
| 구조 | HRD/\_\_init\_\_.py |
|  | from flask import Flask, request  def **create\_app**():  app\_name = *'HRD'*    app = Flask(\_\_name\_\_)    *@app.route*(*"/"*)  def **first**():  return *'Hello Flask! '*+app\_name+*' request.url='*+request.url    return app |

|  |  |
| --- | --- |
| 구조 | FlaskTest/start\_flask.py |
|  | from werkzeug.middleware.dispatcher import DispatcherMiddleware  from werkzeug.serving import run\_simple  from EvCar import create\_app as evcar\_app\_create  evcar = evcar\_app\_create()  from HRD import create\_app as hrd\_app\_create  hrd = hrd\_app\_create()  #합치기  application = DispatcherMiddleware(  evcar,{*'/hrd'*:hrd}  )  if \_\_name\_\_ == *'\_\_main\_\_'*:  run\_simple(hostname=*'localhost'*, port=5000,  application=application,  use\_reloader=True, use\_debugger=True, use\_evalex=True) |

결과화면

|  |  |
| --- | --- |
|  |  |

위의 결과 화면에서 알 수 있듯이 마치 호출하는 URI의 구분으로 여러 개의 모듈패키지를 호출하는 것이 확인된다.