

자바와 파이썬 웹으로 연결하기

자바와 파이썬 웹으로 연결하기

발 행 | 2021년 9월 24일

저 자 | 허진경

펴낸이 | 한건희

펴낸곳 | 주식회사 부크크

출판사등록 | 2014.07.15.(제2014-16호)

주 소 | 서울특별시 금천구 가산디지털1로 119 SK트윈타워 A동 305호

전 화 | 1670-8316

이메일 | info@bookk.co.kr

ISBN | 979-11-372-5701-6

www.bookk.co.kr

© 허진경 2021

본 책은 저작자의 지적 재산으로서 무단 전재와 복제를 금합니다.

CONTENT

1장. 자바 웹과 플라스크 RestAPI 연결하기	1
1절. RestAPI 구현하기	2
1.1. 파이썬 플라스크 프로젝트	2
1) 플라스크 RestAPI 구현	2
2) 플라스크 프로젝트 실행	2
3) RestAPI 실행 확인	3
1.2. 자바 프로젝트	3
1) 컨트롤러	3
2) 뷰	4
1.3. 요청 테스트	5
2절. POST 요청으로 파일 전송하기	6
2.1. 자바의 입력 폼	6
2.2. 파이썬의 요청 처리 함수	7
2.3. 테스트	7
1) 자바 웹서버 실행	7
2) 업로드된 파일 확인	8
3절. 비동기 요청 구현하기	9
3.1. 비동기 요청과 CORS	9
3.2. CORS 해결	11
1) 헤더 설정을 이용한 해결	11
2) 프락시를 이용한 해결	12
3.3. 테스트	12
4절. Restful 서버와 비동기로 이미지 주고받기	14
4.1. 파이썬 코드	14

1) 이미지 처리와 base64 인코딩	14
2) 파이썬 플라스크 코드	15
4.2. 자바 코드	15
1) 태그 이미지 표시	15
2) 실행 결과	17
5절. 동기방식으로 처리하기	18
5.1. 처리 구조	18
5.2. 설정	18
5.3. 요청 폼(jsp)	19
5.4. 결과를 출력할 JSP 페이지	20
5.5. 컨트롤러	20
5.6. 실행 결과	22
6절. 전체 코드	24
6.1. 플라스크 전체 코드	24
1) main.py	24
2) index.html	26
6.2. 자바 코드	26
1) 컨트롤러	26
2) servlet-context.xml	29

들어가기 전에...

이 책의 내용을 이해하려면 다음 사전지식이 있어야 한다.

- 자바 프로그래밍 언어
- 자바 스프링프레임워크를 이용한 웹 프로젝트
- 파이썬 프로그래밍 언어
- 파이썬 플라스크를 이용한 웹 프로젝트
- OpenCV를 이용한 영상처리

이 책은 아래의 수정 이력이 있습니다.

- 1차 수정일 : 2021년 12월 23일



1장. 자바 웹과 플라스크 RestAPI 연결하기

자바 웹 프로젝트에서 비동기 요청을 이용하여 파이썬의 플라스크 서버와 통신하는 방법을 설명한다.

1절. RestAPI 구현하기

1.1. 파이썬 플라스크 프로젝트

1) 플라스크 RestAPI 구현

프로젝트를 생성한 후 GET 방식 RestAPI 요청 테스트를 위한 /test1 라우트 함수를 추가한다. 이 함수의 결과는 JSON 형식이다.

main.py

```
1  # _*_ coding: utf-8 _*_
2  from flask import Flask, jsonify, render_template, request
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template("index.html")
8
9  @app.route('/test1', methods=['GET', 'POST'])
10 def rest_api_test1():
11     print(request.method)
12     data = {"0": "0.01", "1": "0.99"}
13     if request.method == 'GET':
14         param = request.args.get('data')
15         print(param)
16         data.update({"param": param})
17     return jsonify(data)
18
19
20 if __name__ == '__main__':
21     app.debug = True
22     app.run()
```


2) 플라스크 프로젝트 실행

파이썬 콘솔에서 플라스크 서버를 실행시킨다.

```
python main.py
```

3) RestAPI 실행 확인

GET 방식으로 요청을 테스트한다.

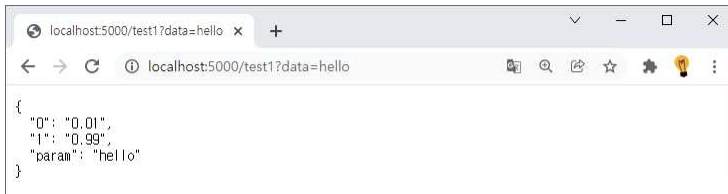


그림 1. http://127.0.0.1:5000/test1?data=hello

1.2. 자바 프로젝트

자바에서 다른 서버에 요청을 처리하려면 URL 클래스를 생성한 후 이를 이용해서 HttpURLConnection을 생성해야 한다.

1) 컨트롤러

Restful 서비스를 요청하는 컨트롤러를 작성한다.

RestReqController.java

```
1 package com.coderby.myapp;
2
3 import java.io.BufferedReader;
4 import java.io.InputStream;
```

```

5 import java.io.InputStreamReader;
6 import java.net.HttpURLConnection;
7 import java.net.URL;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.ui.Model;
10 import org.springframework.web.bind.annotation.RequestMapping;
11
12 @Controller
13 public class RestReqController {
14
15     @RequestMapping("/test_req")
16     public String testRestRequest(Model model) {
17         try {
18             URL url = new
19             URL("http://127.0.0.1:5000/test1?data=Hello");
20             HttpURLConnection con = (HttpURLConnection)
21             url.openConnection();
22             con.setRequestMethod("GET");
23             InputStream in = con.getInputStream();
24             InputStreamReader reader = new
25             InputStreamReader(in, "UTF-8");
26             BufferedReader response = new BufferedReader(reader);
27             String str = null;
28             StringBuffer buff = new StringBuffer();
29             while ((str = response.readLine()) != null) {
30                 buff.append(str + "\n");
31             }
32             String data = buff.toString().trim();
33             System.out.println("reqResult : " + data);
34             model.addAttribute("reqResult", data);
35         } catch (Exception e) {
36             e.printStackTrace();
37         }
38     }
39 }

```

2) 뷰

결과를 보여줄 result.jsp 파일은 요청 결과를 출력하는 EL을 작성한다. 지금의 예는 요청한 결과를 그대로 텍스트 데이터로 출력만 한다.
WEB-INF/views/result1.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>Insert title here</title>
7 </head>
8 <body>
9   요청 결과 : ${reqResult}<p>
10 </body>
11 </html>
```

1.3. 요청 테스트

요청을 테스트한다. 서버의 컨텍스트 이름은 /로 했다.



그림 2. http://localhost:8080/test_req

2절. POST 요청으로 파일 전송하기

자바에서 GET 방식을 이용해서 플라스크의 특정 페이지를 실행시키는 것을 보았다. 그런데 GET 방식을 이용해서는 대량의 데이터나 파일을 보낼 수 없다.

이 절은 자바 웹서버를 이용해 선택한 파일을 파이썬 플라스크를 이용한 Rest 서버에 전송하는 것을 설명한다.

이것이 이용하면 기존 시스템이 자바 기반이고 자바에서 이미지 또는 대량의 텍스트 데이터를 파이썬 기반으로 동작하는 AI 서버에 보내서 예측 모델을 실행시키고 그 결과를 다시 자바에서 받아 자바 기반 웹으로 출력하는 시스템을 구축할 수 있다.

2.1. 자바의 입력 폼

파일을 전송할 수 있는 입력 양식을 작성한다. 다음은 /WEB-INF/views/req_form.jsp 파일이다. 파일을 전송하려면 form 태그에 enctype="multipart/form-data"를 추가해야 한다. 요청 버튼을 누르면 실행할 action의 주소는 플라스크 서버의 주소이다.

/WEB-INF/views/req_form.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>Form</title>
7 </head>
```

```

8 <body>
9 <form action="http://127.0.0.1:5000/test1" method="post"
  enctype="multipart/form-data">
10     데이터 : <input type="text" name="data"><p>
11     파일 : <input type="file" name="file"><p>
12     <input type="submit" value=" 요청 ">
13     <input type="reset" value=" 취소 ">
14 </form>
15 </body>
16 </html>

```

스프링 웹 빈 설정파일에 뷰 컨트롤러를 설정을 추가해야 한다

```
<view-controller path="/req_form" view-name="req_form"/>
```

2.2. 파이썬의 요청처리 함수

파이썬에서 요청을 받아 처리할 함수는 다음과 같다. 앞에서 작성한 코드에 POST 방식 요청을 위한 구문을 추가했다. POST 방식 파라미터를 받기 위해서는 request.args.get() 함수를 이용하며, 요청한 파일을 받기 위해서는 request.files[]을 이용한다.

```

1 # -*- coding: utf-8 -*-
2 from flask import Flask, jsonify, render_template, request
3
4 app = Flask(__name__)
5
6
7 @app.route('/')
8 def index():
9     return render_template("index.html")
10
11 @app.route('/test1', methods=['GET', 'POST'])
12 def rest_api_test1():
13     print(request.method)

```

```

14     data = {"0": "0.01", "1": "0.99"}
15     if request.method == 'GET':
16         param = request.args.get('data')
17         print(param)
18         data.update({"param": param})
19     elif request.method == 'POST':
20         param = request.form.get('data')
21         print(param)
22         f = request.files['file']
23         print(f.filename)
24         f.save(f.filename)
25         data.update({"param": param, "file": f.filename})
26     return jsonify(data)
27
28
29 if __name__ == '__main__':
30     app.debug = True
31     app.run()

```

2.3. 테스트

1) 자바 웹서버 실행

자바 웹서버에 연결해서 폼 페이지를 연다.

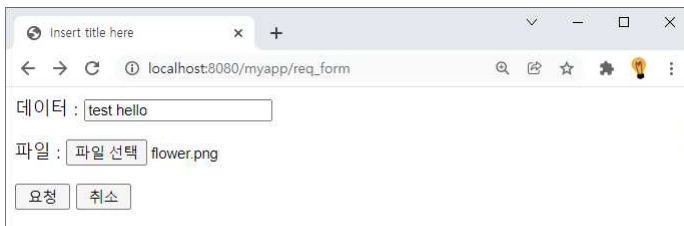


그림 3. http://localhost:8080/req_form

데이터를 입력하고 파일을 선택한 후 요청 버튼을 클릭한다. 그러면 POST 요청이 플라스크 서버에 요청된다. 브라우저에는 전송했던 데이터와 파일 이름이 표시되어야 한다.



그림 4. POST 요청처리 확인됨

2) 업로드된 파일 확인

프로젝트들 통해 플라스크 서버의 디렉토리에 파일이 저장된 것을 확인할 수 있다.

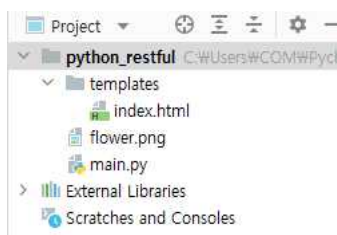


그림 5. 저장된 파일 확인

아직 완벽한 처리는 아니다. 위의 예는 자바에서 POST 방식으로 파일을 파이썬 플라스크에 보내는 것을 테스트할 뿐이다. 이 방법은 톱캣 화면의 제어가 플라스크 화면으로 넘어간다. 톱캣의 화면에서 결과를 받아 처리하려면 비동기 방식으로 요청하거나 스프링의 컨트롤러에 요청처리를 구현해야 한다.

3절. 비동기 요청 구현하기

앞의 동기 요청방식으로는 자바에서 파이썬 페이지를 요청한 후 그 결과를 받을 수 없다. 이 절은 자바에서 비동기 요청을 수행하는 방법을 알아본다.

3.1. 비동기 요청과 CORS

비동기 요청을 하기 위해서는 자바스크립트 코드를 작성해야 한다. 그런데 jQuery를 이용하면 더 쉽게 비동기 요청을 수행할 수 있다. 비동기 요청처리는 ajax() 함수를 이용하였다.

/WEB-INF/views/req_image_ajax1.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>요청 폼</title>
7 <script
  src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
8 <script type="text/javascript">
9 $(function() {
10     $("input[type=button]").click(function() {
11         console.log()
12         var form = $("#fileUploadForm")[0];
13         var form_data = new FormData(form);
14         $("input[type=button]").prop("disabled", true);
15         $.ajax({
16             url : "http://127.0.0.1:5000/test1",
17             async : true,
```



```

18         type : "POST",
19         data : form_data,
20         processData: false,
21         contentType: false,
22         success : function(data) {
23             console.log(data)
24             $("#result").text(JSON.stringify(data))
25             $("input[type=button]").prop("disabled", false);
26         },
27         error : function(e) {
28             console.log("ERROR : ", e);
29             alert("fail");
30         }
31     });
32 })
33 });
34 </script>
35 </head>
36 <body>
37 <form method="post" enctype="multipart/form-data"
38     id="fileUploadForm">
39     데이터 : <input type="text" name="data" value="test
40     hello"><p>
41     파일 : <input type="file" name="file"><p>
42     <input type="button" value=" 비동기 요청 ">
43 </form>
44 <div id="result">여기에 요청 결과가 출력되어야 합니다.</div>
45 </body>
46 </html>

```

이 페이지를 로드하려면 뷰컨트롤러 설정을 추가해야 한다.

```
<view-controller path="/req_image_ajax1" view-name="req_image_ajax1"/>
```

이 페이지를 실행시키고 [비동기 요청] 버튼을 클릭하면 다음과 같이 정상적으로 실행되지는 않는다.

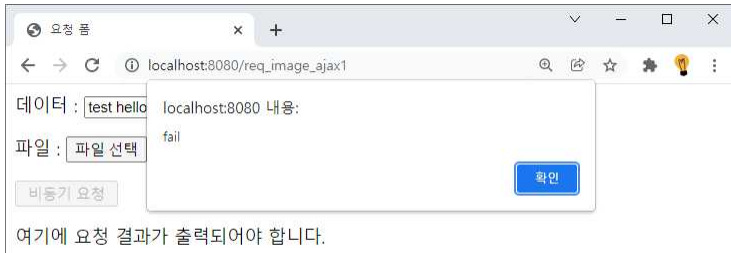


그림 6. 비동기 요청 실패

요청이 처리되지 않는 이유를 확인하기 위해서 크롬 브라우저에서 F12키를 눌러 확인하자. 오류 원인은 현재 접속한 주소가 아닌 다른 주소로 비동기 요청을 수행하기 때문에 브라우저가 이를 차단한 것이다.

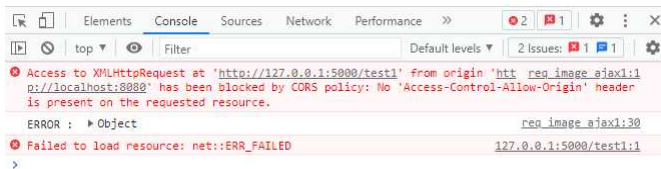


그림 7. CORS 오류

CORS(Cross-Origin Resource Sharing) 사양에 정의된 CORS는 HTTP 헤더를 사용하여 동일 도메인 출처 정책으로 제한되는 도메인 간 웹 요청을 가능하게 한다.

기본적으로 동일한 사이트 출처 정책은 웹 사이트가 다른 도메인에 있는 서버의 리소스를 요청하지 못하도록 한다. 그래서 XHR 객체(비동기 객체)는 요청할 수 있는 서버의 리소스 URL에 제한이 있다. 즉, 접근할 수 있는 서버 리소스 URL은 XHR 객체가 존재하는 도메인에 있어야 한다. XHR 객체가 자기가 속해있는 도메인이 아닌 그 밖에 있는 서버의 URL 을 호출하면 브라우저에서 요청을 차단한다.

3.2. CORS 해결

CORS를 해결하는 방법은 헤더 방법과 프락시 방법이 있다.

1) 헤더 설정을 이용한 해결

서버에서 응답할 때 수신할 도메인을 Access-Control-Allow-Origin 헤더에 설정하여 해결하는 방법이 있다. 헤더의 이름은 Access-Control-Allow-Origin, 그리고 값은 수신할 도메인이어야 한다. 이 헤더의 설정값을 *로 하면 모든 도메인에서 요청을 받아 처리할 수 있다. 이 작업은 정보 제공자가 해줘야 한다.

JSP 코드의 예

▶ `<% response.setHeader("Access-Control-Allow-Origin", "*"); %>`

파이썬 플라스크의 예

▶ `response.headers.add("Access-Control-Allow-Origin", "*")`

플라스크라면 다음과 같이 헤더를 추가할 수 있다. 다음 코드는 수정된 플라스크의 라우트 함수이다.

main.py.파일의 일부

```
1 @app.route('/test2', methods=['GET', 'POST'])
2 def rest_api_test2():
3     print(request.method)
4     data = {"0": "0.01", "1": "0.99"}
5     if request.method == 'GET':
6         param = request.args.get('data')
7         print(param)
8         data.update({"param": param})
9     elif request.method == 'POST':
```

```

10     param = request.form.get('data')
11     print(param)
12     f = request.files['file']
13     print(f.filename)
14     f.save(f.filename)
15     data.update({"param": param, "file": f.filename})
16     response = make_response(jsonify(data))
17     response.headers.add("Access-Control-Allow-Origin", "*")
18     return response

```

2) 프락시를 이용한 해결

같은 도메인 내에 외부 도메인 페이지를 포함시킬 서버페이지를 이용한다. 이 방법은 정보 요청자가 직접 해결할 수 있다. 그래서 요청되는 서버의 코드를 수정할 수 없을 때 사용한다. 이 설명서에는 프락시를 이용한 해결 방법은 사용하지 않았다.

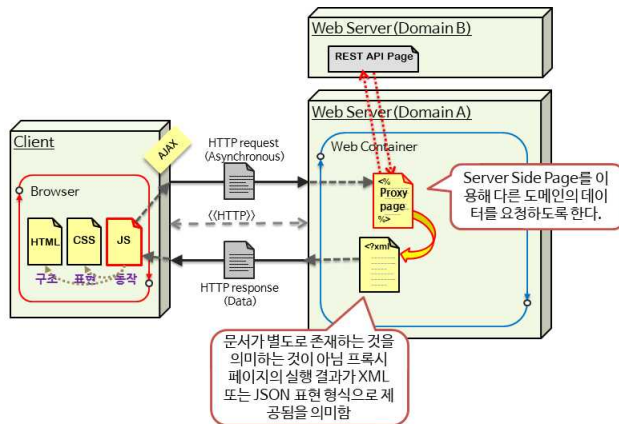


그림 8. 프락시를 이용한 CORS 해결

예) - JSP 코드(프락시 페이지로 이용된다.)

JSP 코드의 예

- ▶ `<%@ contentType="text/xml;charset=UTF-8" trimDirectiveWhitespaces="true"%>`
- ▶ `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>`
- ▶ `<c:import charEncoding="UTF-8" url="http://other/page"></c:import>`

3.3. 테스트

자바 JSP 코드의 수정은 없이 다시 요청을 해보자.

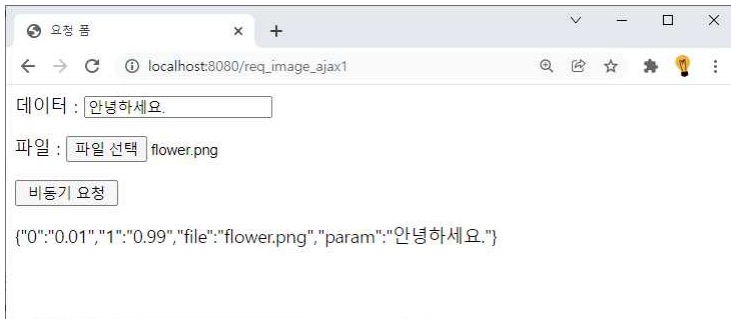


그림 9. 비동기 요청 성공

서버에 파일이 저장되고 브라우저에는 비동기 요청 결과가 출력되어야 한다.

CORS를 해결하기 위해서 앞의 두 방법이 아닌 스프링의 컨트롤러를 이용해서 동기방식으로 처리할 수 있다. 그 내용은 5절에서 설명한다.

다음 절은 Restful 서버와 비동기 방식으로 이미지 데이터를 주고받는 방법을 설명한다.

4절. Restful 서버와 비동기로 이미지 주고받기

4.1. 파이썬 코드

1) 이미지 처리와 base64 인코딩

POST 방식으로 요청된 이미지 데이터를 처리해서 JSON 형식으로 응답하려면 이미지는 base64 인코딩되어야 한다. 파일 시스템의 이미지를 인코딩하는 것은 파일을 불러와서 하면 되지만 이 경우에는 요청받은 메모리상의 이미지를 인코딩해야 한다. 그러기 위해서는 FileStorage의 이미지 데이터를 넘파이 배열로 만들고, 이를 다시 디코딩해서 이미지 객체로 변환한다. 이후 변환한 이미지를 OpenCV 영상처리 API를 이용하거나 AI를 이용해서 처리한 후 JSON으로 응답한다. 이미지 객체를 JSON으로 응답하려면 base64 인코딩을 해야 한다.

다음은 FileStorage의 이미지 객체를 OpenCV의 이미지로 변환한 후 그레이스케일 이미지로 변환한 후 base64 인코딩하는 코드의 설명이다.

요청객체에서 파일 정보를 불러온다.

```
▶ f = request.files['file']  
▶ filestr = f.read()
```

넘파이 배열로 변환한다.

```
▶ npimg = np.fromstring(filestr, np.uint8)
```

넘파이 배열을 OpenCV에서 사용하는 배열로 변환한다.

```
▶ img = cv2.imdecode(npimg, cv2.IMREAD_COLOR)
```

이미지를 처리한다. 여기에서는 cv2.cvtColor() 함수를 이용해서 컬러 이미지를 그레이스케일 이미지로 변환한다. 다른 처리를 해야 한다면 여기에서 이미지 처리를 수행한다.

```
▶ img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

서버에 파일을 저장하기를 원하면 cv2.imwrite() 함수를 이용한다.

```
▶ cv2.imwrite(f.filename, img_gray)
```

처리한 이미지를 base64 인코딩한다.

```
▶ img_str = base64.b64encode(cv2.imencode('.',jpg', img_gray)[1]).decode()
```

인코딩한 이미지를 딕셔너리 객체로 만든다.

```
▶ data = {"param": param, "file": img_str}
```

2) 파이썬 플라스크 코드

다음 코드는 POST 요청으로 이미지를 받아 OpenCV를 이용해서 이미지 처리한 후 JSON으로 응답하는 라우트(route) 함수이다.

main.py.파일의 일부

```
1 @app.route('/test_img', methods=['POST'])
2 def rest_img_test():
3     param = request.form.get('data')
4     print(param)
5     f = request.files['file']
6     filestr = f.read()
7     # FileStorage의 이미지를 넘파이 배열로 만들
8     npimg = np.fromstring(filestr, np.uint8)
9     # 넘파일 배열을 이미지 배열로 변환함
10    img = cv2.imdecode(npimg, cv2.IMREAD_COLOR)
11    # 여기에서 처리를 하면 됨
12    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```

13     cv2.imwrite(f.filename, img_gray)
14     img_str = base64.b64encode(
        cv2.imencode('.jpg', img_gray)[1]).decode()
15     data = {"param": param, "file": img_str}
16     response = make_response(jsonify(data))
17     response.headers.add("Access-Control-Allow-Origin", "*")
18     return response

```

4.2. 자바 코드

1) 태그 이미지 표시

비동기 요청을 수행해서 받은 이미지 데이터는 태그의 src 속성에 지정해서 브라우저 화면에 출력할 수 있다. 다만 이 데이터는 base64 인코딩된 데이터이므로 data:image/png;base64,를 붙여서 태그의 src 속성값으로 사용해야 한다.

▶

파일을 업로드하는 경우는 단순 문자열을 전송하는 경우와 조금 다르다. 일반적으로 서버에 전달되는 데이터는 query string이라는 형태로 전달된다. 그런데 파일을 업로드해야 하는 경우 data 매개변수로 전달된 데이터를 jQuery 내부적으로 query string으로 만들지 않아야 한다. 이때 사용하는 것이 processData: false이다.

contentType의 기본값은 "application/x-www-form-urlencoded; charset=UTF-8"인데, "multipart/form-data"로 전송이 되게 하려면 이 값을 false로 한다. 만일 false가 아닌 "multipart/form-data"를 사용하면 boundary string이 안 들어가게 되어 제대로 동작하지 않는다.

다음 코드는 비동기 요청을 수행한 후 처리된 이미지를 화면에 출력하는 코드이다. 앞에서 비동기 요청을 처리하기 위해 작성한 코드와 크게 다르지 않다.

/WEB-INF/views/req_image_ajax2.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>요청 폼</title>
7 <script
  src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js">
</script>
8 <script type="text/javascript">
9 $(function() {
10     $("input[type=button]").click(function() {
11         console.log()
12         var form = $("#fileUploadForm")[0];
13         var form_data = new FormData(form);
14         $("input[type=button]").prop("disabled", true);
15         $.ajax({
16             url : "http://127.0.0.1:5000/test_img",
17             async : true,
18             type : "POST",
19             data : form_data,
20             processData: false,
21             contentType: false,
22             success : function(data) {
23                 console.log(data)
24                 $("#result").text(data.param)
25                 img_src = "data:image/png;base64,"+data.file
26                 $("#result").append("<img src='"+img_src+"'>");
27                 $("input[type=button]").prop("disabled", false);
28             },
29             error : function(e) {
```

```

29             console.log("ERROR : ", e);
30             alert("fail");
31         }
32     });
33 })
34 });
35 </script>
36 </head>
37 <body>
38 <form method="post" enctype="multipart/form-data"
39     id="fileUploadForm">
40     데이터 : <input type="text" name="data" value="test
41     hello"><p>
42     파일 : <input type="file" name="file"><p>
43     <input type="button" value=" 비동기 요청 ">
44 </form>
45 <div id="result">여기에 요청 결과가 출력되어야 합니다.</div>
46 </body>
47 </html>

```

이 페이지를 실행하려면 스프링 설정파일에 뷰컨트롤러 설정을 추가해야 한다.

```
<view-controller path="/req_image_ajax2" view-name="req_image_ajax2"/>
```

2) 실행

비동기 요청을 테스트할 주소는 다음과 같다.

http://localhost:8080/myapp/req_image_ajax2

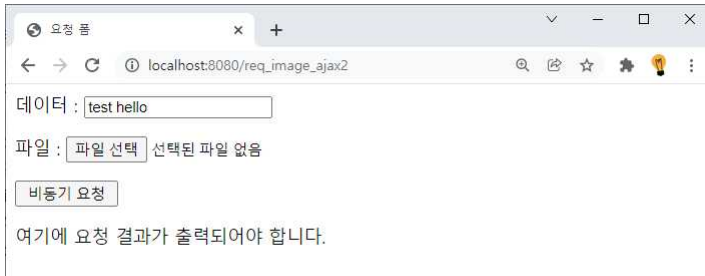


그림 10. http://localhost:8080/myapp/req_image_ajax2

비동기 요청 버튼을 클릭하면 선택한 이미지파일이 서버에 전송되고 서버에서 컬러 이미지가 회색조(grayscale) 이미지로 변환되어 응답한 이미지가 화면에 출력되어야 한다.

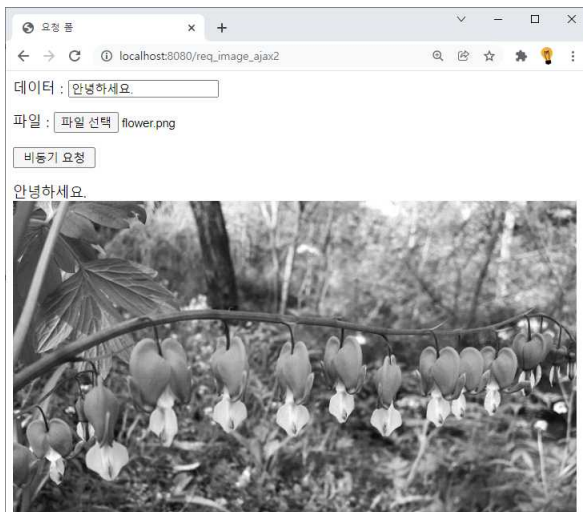


그림 11. Restful 서버에 비동기 요청한 결과

5절. 동기방식으로 처리하기

스프링의 컨트롤러가 폼 요청을 받아 다시 플라스크에 요청하는 방식으로 처리해 보자. 이 방법을 이용하면 플라스크에서 응답할 때 헤더에 “Access-Control-Allow-Origin” 속성을 추가하지 않아도 된다.

5.1. 처리 구조

동기방식으로 처리하기 위한 구조는 다음과 같다.

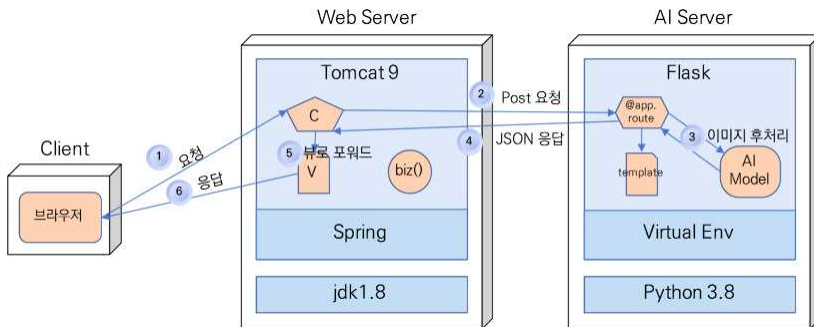


그림 12. 동기방식 요청/응답 구조

스프링의 컨트롤러는 브라우저에서 요청된 파라미터와 파일을 읽어 플라스크에 POST 방식 요청을 한다. 플라스크는 요청을 받아 실행한 후 JSON 형식으로 응답한다. 스프링 컨트롤러는 응답받은 JSON 문자열 데이터를 Map 객체로 변환한 후 모델에 저장하고 뷰(View)로 포워드(forward)해야 EL을 이용해서 원하는 키의 값만 추출할 수 있다.

5.2. 설정

스프링 컨트롤러가 업로드된 파일을 처리하려면 파일 업로드 컴포넌트 의존성 추가와 JSON 처리를 위한 의존성을 추가해야 하고 스프링 설정 파일에 멀티파트 설정을 추가해야 한다.

스프링 레거시 프로젝트라면 다음 내용을 pom.xml 파일에 추가한다.
pom.xml 파일의 일부

```
1 <!-- File upload -->
2 <dependency>
3     <groupId>commons-fileupload</groupId>
4     <artifactId>commons-fileupload</artifactId>
5     <version>1.3.1</version>
6 </dependency>
7 <!-- JSON 처리 -->
8 <dependency>
9     <groupId>com.fasterxml.jackson.core</groupId>
10    <artifactId>jackson-databind</artifactId>
11    <version>2.9.8</version>
12 </dependency>
```

다음은 멀티파트 설정을 위한 내용이다. 필자는 servlet-context.xml 파일에 아래 내용을 추가했다.

servlet-context.xml 파일의 일부

```
1 <beans:bean id="multipartResolver"
2     class="org.springframework.web.multipart.commons.CommonsMulti
3     partResolver">
4     <beans:property name="maxUploadSize">
5         <beans:value>5000000</beans:value>
6     </beans:property>
7 </beans:bean>
```

5.3. 요청 폼(jsp)

다음은 파일을 선택하고 전송할 수 있는 폼이다. 폼의 action값인 test_req_image는 컨트롤러의 URL이다.

WEB-INF/views/req_image.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>Form</title>
7 </head>
8 <body>
9 <form action="test_req_image" method="post"
  enctype="multipart/form-data">
10     데이터 : <input type="text" name="data"><p>
11     파일 : <input type="file" name="file"><p>
12     <input type="submit" value=" 요청 "
13     <input type="reset" value=" 취소 ">
14 </form>
15 </body>
16 </html>
```

이 페이지를 실행하려면 스프링 설정파일에 뷰컨트롤러 설정을 추가해야 한다.

```
<view-controller path="/req_image" view-name="req_image"/>
```

JSP 페이지에서 요청 버튼을 클릭하면 컨트롤러에 파라미터 data와 파일이 전송된다.

5.4. 결과를 출력할 JSP 페이지

컨트롤러가 실행한 결과를 출력할 JSP 페이지이다. EL을 이용해서 결과를 화면에 출력한다. 다만 이미지의 경우 base64 인코딩된 문자열을 받으므로 이미지를 브라우저 화면에 출력하려면 태그의 src 속성의 값에 data:image/png;base64,를 추가해야 한다.

WEB-INF/views/result2.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>Insert title here</title>
7 </head>
8 <body>
9 <h1>요청 결과</h1>
10 파라미터 : ${reqResult.param}<p>
11 <img src='data:image/png;base64,${reqResult["file"]}'><p>
12 </body>
13 </html>
```

5.5. 컨트롤러

컨트롤러는 POST 방식으로 플라스크의 지정한 URL을 요청해야 한다. 그렇게 하기 위해서 HttpURLConnection 객체를 이용해서 OutputStream을 얻은 후 이를 이용해서 DataOutputStream 객체를 생성해야 한다. 이 객체를 request라고 하면 이후 writeBytes() 메서드를 이용해서 파라미터와 파일 객체를 보내도록 설정한다.

폼의 파라미터들이 POST 방식으로 전송될 때 파라미터들을 구분하기

위한 구분자가 있는데 이 구분자의 형식은 다음과 같다. --뒤의 UUID(Universally Unique Identifier)는 java.util 패키지의 UUID 클래스의 randomUUID() 메서드를 이용하여 생성할 수 있다.

▶ --6abbb881-145a-4032-b39f-57b56389e718\r\n

다음은 폼데이터를 동기방식으로 요청하기 위한 컨트롤러의 메서드이다. 15라인과 19라인의 name 속성의 값이 POST 방식으로 요청할 파라미터의 이름이다.

```
1      @RequestMapping("/test_req_image")
2      public String testRestRequestImage(MultipartFile file,
3      String data, Model model) {
4          try {
5              URL url = new URL("http://127.0.0.1:5000/test_img");
6              HttpURLConnection con = (HttpURLConnection)
7                  url.openConnection();
8              String boundary = UUID.randomUUID().toString();
9              con.setRequestMethod("POST");
10             con.setDoOutput(true);
11             con.setRequestProperty("Content-Type",
12                 "multipart/form-data;boundary=" + boundary);
13             OutputStream out = con.getOutputStream();
14             DataOutputStream request = new
15                 DataOutputStream(out);
16             request.writeBytes("--" + boundary + "\r\n");
17             request.writeBytes("Content-Disposition:
18                 form-data; name=\"data\"\r\n\r\n"); // 파라미터 data를 전송함
19             request.writeBytes(data + "\r\n");
20             request.writeBytes("--" + boundary + "\r\n");
21             request.writeBytes("Content-Disposition:
22                 form-data; name=\"file\"; filename=\"" +
23                 file.getOriginalFilename() + "\"\r\n\r\n");
24             request.write(file.getBytes());
```

```

21         request.writeBytes("\r\n");
22
23         request.writeBytes("--" + boundary + "--\r\n");
24         request.flush();
25         int respCode = con.getResponseCode();
26
27         // 요청 결과 코드에 따라 처리
28         switch(respCode) {
29             case 200:
30                 System.out.println("OK");
31                 break;
32             case 301:
33             case 302:
34             case 307:
35                 System.out.println("Redirect");
36                 break;
37             default:
38                 //do something
39         }
40
41         // 요청 후 응답 결과를 받기 위한 코드
42         InputStream in = con.getInputStream();
43         InputStreamReader reader = new
InputStreamReader(in, "UTF-8");
44         BufferedReader response = new
BufferedReader(reader);
45         String str = null;
46         StringBuffer buff = new StringBuffer();
47         while ((str = response.readLine()) != null) {
48             buff.append(str + "\n");
49         }
50         String result = buff.toString().trim();
51
52         // 결과 문자열을 Map 객체로 변환
53         ObjectMapper mapper = new ObjectMapper();
54         Map<String, String> map =
mapper.readValue(result, Map.class);

```

```

55         System.out.println(map.keySet());
56         model.addAttribute("reqResult", map);
57     }catch(Exception e) {
58         e.printStackTrace();
59     }
60     return "result2";
61 }

```

서버로부터 응답한 데이터는 InputStream을 이용해서 받을 수 있다. 이렇게 받은 데이터는 문자열이므로 Jackson 라이브러리를 이용해서 맵(Map) 객체로 만들어 model에 저장한다. 이 데이터는 JSP 파일 안에서 EL(Expression Language)을 이용해서 출력할 수 있다.

5.6. 실행

요청을 테스트하기 위해서 http://localhost:8080/req_image를 실행시킨다.

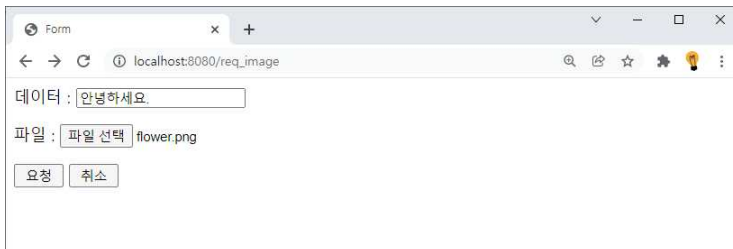


그림 13. 동기방식으로 요청할 폼

입력 폼에서 데이터 파라미터 입력 양식에 값을 입력하고 그림 파일을 선택한 후 요청 버튼을 클릭한다. 그러면 POST 요청을 컨트롤러가 받게 되고 컨트롤러는 전달받은 파라미터와 파일을 플라스크에

POST 방식으로 요청하기 위해서 파라미터와 파일을 재구성해서 OutputStream을 통해 출력한다.

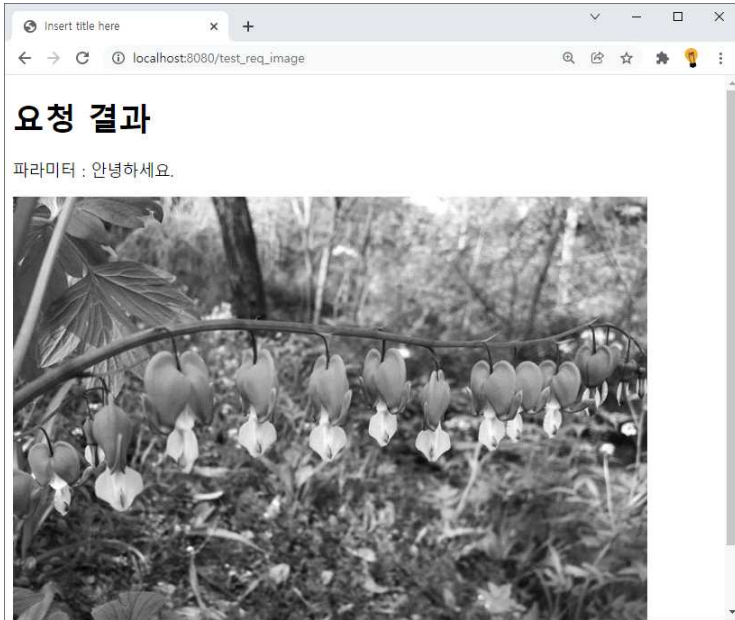


그림 14. 동기방식으로 실행한 결과

6절. 전체 코드

6.1. 파이썬

다음은 플라스크 전체 코드이다.

1) main.py

main.py 파일은 라우트 함수들을 정의하고 있다.

```
1  # -*- coding: utf-8 -*-
2
3  from flask import Flask, jsonify, render_template, request,
    make_response
4  import cv2
5  import base64
6  import numpy as np
7
8  app = Flask(__name__)
9
10
11 @app.route('/')
12 def index():
13     return render_template("index.html")
14
15
16 @app.route('/test1', methods=['GET', 'POST'])
17 def rest_api_test1():
18     print(request.method)
19     data = {"0": "0.01", "1": "0.99"}
20     if request.method == 'GET':
21         param = request.args.get('data')
22         print(param)
23         data.update({"param": param})
```

```

24     elif request.method == 'POST':
25         param = request.form.get('data')
26         print(param)
27         f = request.files['file']
28         print(f.filename)
29         f.save(f.filename)
30         data.update({"param": param, "file": f.filename})
31     return jsonify(data)
32
33
34 @app.route('/test2', methods=['GET', 'POST'])
35 def rest_api_test2():
36     print(request.method)
37     data = {"0": "0.01", "1": "0.99"}
38     if request.method == 'GET':
39         param = request.args.get('data')
40         print(param)
41         data.update({"param": param})
42     elif request.method == 'POST':
43         param = request.form.get('data')
44         print(param)
45         f = request.files['file']
46         print(f.filename)
47         f.save(f.filename)
48         data.update({"param": param, "file": f.filename})
49
50     response = make_response(jsonify(data))
51     response.headers.add("Access-Control-Allow-Origin", "*")
52     return response
53
54
55 @app.route('/test_img', methods=['POST'])
56 def rest_img_test():
57     param = request.form.get('data')
58     print(param)
59     f = request.files['file']
60     filestr = f.read()

```

```

61     # FileStorage의 이미지를 넘파이 배열로 만듦
62     npimg = np.fromstring(filestr, np.uint8)
63     # 넘파이 배열을 이미지 배열로 변환함
64     img = cv2.imdecode(npimg, cv2.IMREAD_COLOR)
65     # 여기에서 처리를 하면 됨
66     img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
67
68     cv2.imwrite(f.filename, img_gray)
69     img_str = base64.b64encode(cv2.imencode('.jpg',
img_gray)[1]).decode()
70     data = {"param": param, "file": img_str}
71     response = make_response(jsonify(data))
72     response.headers.add("Access-Control-Allow-Origin", "*")
73     return response
74
75
76 if __name__ == '__main__':
77     app.debug = True
78     app.run()
79
80

```

2) index.html

플라스크 서버를 실행시켰을 때 맨 처음 보이는 페이지인 index.html은 templates 디렉토리 아래에 만들어야 한다.

templates/index.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>index</title>

```

```
6 </head>
7 <body>
8 <h1>Welcome</h1>
9 </body>
10 </html>
```

6.2. 자바 코드

1) 컨트롤러

RestReqController.java

```
1 package com.coderby.myapp;
2
3 import java.io.BufferedReader;
4 import java.io.DataOutputStream;
5 import java.io.InputStream;
6 import java.io.InputStreamReader;
7 import java.io.OutputStream;
8 import java.net.HttpURLConnection;
9 import java.net.URL;
10 import java.util.Map;
11 import java.util.UUID;
12
13 import org.springframework.stereotype.Controller;
14 import org.springframework.ui.Model;
15 import
    org.springframework.web.bind.annotation.RequestMapping;
16 import org.springframework.web.multipart.MultipartFile;
17
18 import com.fasterxml.jackson.databind.ObjectMapper;
19
20 @Controller
```

```

21 public class RestReqController {
22
23     @RequestMapping("/test_req")
24     public String testRestRequest(Model model) {
25         try {
26             URL url = new
27             URL("http://127.0.0.1:5000/test1?data=Hello");
28             HttpURLConnection con = (HttpURLConnection)
29             url.openConnection();
30             con.setRequestMethod("GET");
31             InputStream in = con.getInputStream();
32             InputStreamReader reader = new
33             InputStreamReader(in, "UTF-8");
34             BufferedReader response = new
35             BufferedReader(reader);
36             String str = null;
37             StringBuffer buff = new StringBuffer();
38             while ((str = response.readLine()) != null) {
39                 buff.append(str + "\n");
40             }
41             String data = buff.toString().trim();
42             System.out.println("reqResult : " + data);
43             model.addAttribute("reqResult", data);
44         } catch (Exception e) {
45             e.printStackTrace();
46         }
47         return "result1";
48     }
49
50     @RequestMapping("/test_req_image")
51     public String testRestRequestImage(MultipartFile file,
52     String data, Model model) {
53         try {
54             URL url = new URL("http://127.0.0.1:5000/test_img");
55             HttpURLConnection con = (HttpURLConnection)
56             url.openConnection();
57
58         }
59     }
60 }

```

```

52         String boundary = UUID.randomUUID().toString();
53         con.setRequestMethod("POST");
54         con.setDoOutput(true);
55         con.setRequestProperty("Content-Type",
"multipart/form-data;boundary=" + boundary);
56
57         OutputStream out = con.getOutputStream();
58         DataOutputStream request = new
DataOutputStream(out);
59         request.writeBytes("--" + boundary + "\r\n");
60         request.writeBytes("Content-Disposition:
form-data; name=\"data\"\r\n\r\n"); // 파라미터 data를 전송함
61         request.writeBytes(data + "\r\n");
62
63         request.writeBytes("--" + boundary + "\r\n");
64         request.writeBytes("Content-Disposition:
form-data; name=\"file\"; filename=\"" +
file.getOriginalFilename() + "\"\r\n\r\n");
65         request.write(file.getBytes());
66         request.writeBytes("\r\n");
67
68         request.writeBytes("--" + boundary + "--\r\n");
69         request.flush();
70         int respCode = con.getResponseCode();
71
72         // 요청 결과 코드에 따라 처리
73         switch(respCode) {
74             case 200:
75                 System.out.println("OK");
76                 break;
77             case 301:
78             case 302:
79             case 307:
80                 System.out.println("Redirect");
81                 break;
82             default:
83                 //do something

```

```

84         }
85
86         // 요청 후 응답 결과를 받기 위한 코드
87         InputStream in = con.getInputStream();
88         InputStreamReader reader = new
InputStreamReader(in, "UTF-8");
89         BufferedReader response = new
BufferedReader(reader);
90         String str = null;
91         StringBuffer buff = new StringBuffer();
92         while ((str = response.readLine()) != null) {
93             buff.append(str + "\n");
94         }
95         String result = buff.toString().trim();
96
97         // 결과 문자열을 Map 객체로 변환
98         ObjectMapper mapper = new ObjectMapper();
99         Map<String, String> map =
mapper.readValue(result, Map.class);
100         System.out.println(map.keySet());
101         model.addAttribute("reqResult", map);
102     } catch (Exception e) {
103         e.printStackTrace();
104     }
105     return "result2";
106 }
107 }

```

```

1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">

```

```

6 <title>요청 폼</title>
7 <script
  src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js">
</script>
8 <script type="text/javascript">
9 $(function() {
10     $("input[type=button]").click(function() {
11         console.log()
12         var form = $("#fileUploadForm")[0];
13         var form_data = new FormData(form);
14         $("input[type=button]").prop("disabled", true);
15         $.ajax({
16             url : "http://127.0.0.1:5000/test_img",
17             async : true,
18             type : "POST",
19             data : form_data,
20             processData: false,
21             contentType: false,
22             success : function(data) {
23                 console.log(data)
24                 $("#result").text(data.param)
25                 img_src = "data:image/png;base64,"+data.file
26                 $("#result").append("<img src='"+img_src+"'>");
27                 $("input[type=button]").prop("disabled", false);
28             },
29             error : function(e) {
30                 console.log("ERROR : ", e);
31                 alert("fail");
32             }
33         });
34     })
35 });
36 </script>
37 </head>
38 <body>
39 <form method="post" enctype="multipart/form-data"
  id="fileUploadForm">

```

```
40     데이터 : <input type="text" name="data" value="test
      hello"><p>
41     파일 : <input type="file" name="file"><p>
42     <input type="button" value=" 비동기 요청 ">
43 </form>
44 <div id="result">여기에 요청 결과가 출력되어야 합니다.</div>
45 </body>
46 </html>
```
