

Spring Boot에서 Flask 호출하기!

1. 의존성 추가하기

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.9</version>
</dependency>
```

httpcomponents는 외부 요청을 주고받을 수 있는 여러 변수형식, 메소드를 갖고 있는 의존성이다. 추가하도록한다.

2. Http연결 정의

HTTP형태 연결에 대하여 여러 설정을 할 수 있다. 코드를 참고한다.

```
import org.apache.http.client.HttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
import org.springframework.web.client.RestTemplate;

@Configuration
public class HttpConnectionConfig {

    @Bean
    public RestTemplate getCustomRestTemplate(){
        HttpComponentsClientHttpRequestFactory httpRequestFactory
        = new HttpComponentsClientHttpRequestFactory();
        httpRequestFactory.setConnectTimeout(2000);
        httpRequestFactory.setReadTimeout(3000);

        HttpClient httpClient = HttpClientBuilder.create()
            .setMaxConnTotal(200)
            .setMaxConnPerRoute(20)
            .build();

        httpRequestFactory.setHttpClient(httpClient);
        return new RestTemplate(httpRequestFactory);
    }
}
```

HttpConnectionConfig를 정의한 모습이다.

정의를 하지 않는다면 기본값으로 설정된다.

RestTemplate 형식으로 HTTP 요청을 수행한다. 이는 RestTemplate의 설정을 변경한다고 볼 수 있다.

3. RestTemplateService 정의하기

```
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import com.multicampus.finalproject.util.RestTemplateUtil;

import lombok.extern.slf4j.Slf4j;

@Service
@Slf4j
public class RestTemplateService {

    // public XmlVo getXmlData() {
    //     return RestTemplateUtil.getXmlResponse();
    // }

    // public JsonVo getJsonData() {
    //     return RestTemplateUtil.getJsonResponse();
    // }

    // public ResponseEntity<String> getEntity(String key) {
    //     return RestTemplateUtil.getResponseEntity(key);
    // }

    public ResponseEntity<String> addData(String imgString) {
        return RestTemplateUtil.post(imgString);
    }
}
```

이는 RestTemplateService 정의한 모습이다.

Return 형태를 변경함으로 Xml형식, Json형식, String형식 등 원하는 형태로 응답을 받을 수 있다. 물론! RestTemplateUtil에서 조작이 필요하다.

필자는 테스트를 위해 String형태로 요청 결과값을 받는 메소드만을 남겨놓았다.

4. RestTemplateUtil 정의하기

실질적으로 RestTemplate을 활용해 요청을 주고받는 동작을 하는 부분이다.

요청 방식은 restTemplate.방식For결과타입 메소드 명으로 구분되어있다.

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

@Component
public class RestTemplateUtil {

    private static RestTemplate restTemplate;

    @Autowired
    public RestTemplateUtil(RestTemplate restTemplate) {
        this.restTemplate=restTemplate;
    }

    // public static ResponseEntity<String> getResponseEntity(String key){
    //     //header setting
    //     HttpHeaders headers = new HttpHeaders();
    //     headers.add("Authentication", key);

    //     HttpEntity<Map<String, String>> httpEntity = new HttpEntity<>(headers);

    //     Map<String, String> params = new HashMap<>();
    //     params.put("name", "jaeyeon");

    //     //순서대로 url, method, entity(header, params), return type
    //     return restTemplate.exchange("http://localhost:8080/entity?name={name}", HttpMethod.GET, httpEntity, String.class, params);
    // }

    public static ResponseEntity<String> post(String imgString){
        // MultiValueMap<String,String> map = new LinkedMultiValueMap<String,String>();

        return restTemplate.postForEntity("http://localhost:5000/testapi",imgString, String.class);
        // return restTemplate.getForEntity("http://localhost:5000/testapi", String.class);
    }
}

```

필자는 post형태의 Entity결과값을 요청하는 메소드를 사용했다.

매개변수로 요청할 URI, 넘겨줄 데이터(필자는 사진 Base64 문자열을 보냈다.), 결과 형식을 입력해준다.

현재 Flask 테스트용 API의 URI, String형태로 변환한 이미지, 반환타입을 명시해주었다.

5. Flask 에서 요청 받기

```

@app.route('/testapi',methods=["POST","GET"])
def test():
    # data = request.get_json()
    # print(data)
    # print(request.get_data())
    image = base64.b64decode(request.get_data())
    im=Image.open(BytesIO(image))
    print(im)
    # return jsonify(message="응답 메시지")
    return "응답메시지"

```

Flask 소스에는 request.get_data() 라는 메소드가 보인다. 요청과 함께 넘겨준 데이터를 받아올 수 있다. 이는 request.get_json() 도 있는데 json형태로 여러 데이터를 넘겼을 경우 사용할 듯 싶다.

현재는 데이터를 1가지만 보내기 때문에 request.get_data()를 사용했다.

return 값으로 "응답메시지" 문자열을 반환한다.

6. Controller 정의하기

```

@RequestMapping("/testing")
public String testjson(){

    return "testing";
}

@RequestMapping("/upload")
public String upload(Model model,@RequestParam("file") MultipartFile img){
    for(int i=0;i<10;i++){
        System.out.print("이미지는 이거임!!"+img);
    }
    byte[] imgtext;
    String imgtext2;
    try{
        imgtext = Base64.encodeBase64(img.getBytes());
        imgtext2 = new String(imgtext);
        // mav.addObject("uploadedImage",imgtext2);

        StringBuilder sb = new StringBuilder();
        sb.append("data:image/jpg;base64,");
        sb.append(imgtext2);
        model.addAttribute("uploadedImage",sb);
        ResponseEntity<String> a=restTemplateService.addData(imgtext2);
        System.out.print(a.getBody());
    }
    catch(IOException except){
        System.out.println("파일이 이상함!");
    }

    return "upload";
}

```

필자는 testing.html로 /testing태그를 매핑해두었다.

←
→
↻
ⓘ localhost:8080/testing

경로 적어야댐

파일 선택

선택된 파일 없음

업로드

추후 Flask API 서버로 이미지를 전송하기 위하여 Form형태로 기본 UI를 구성했다.
 이 Form의 action값은 "upload"이다.
 Controller의 upload로 매핑되어 있는 부분을 호출한다.

```

ResponseEntity<String> a=restTemplateService.addData(imgtext2);
System.out.print(a.getBody());

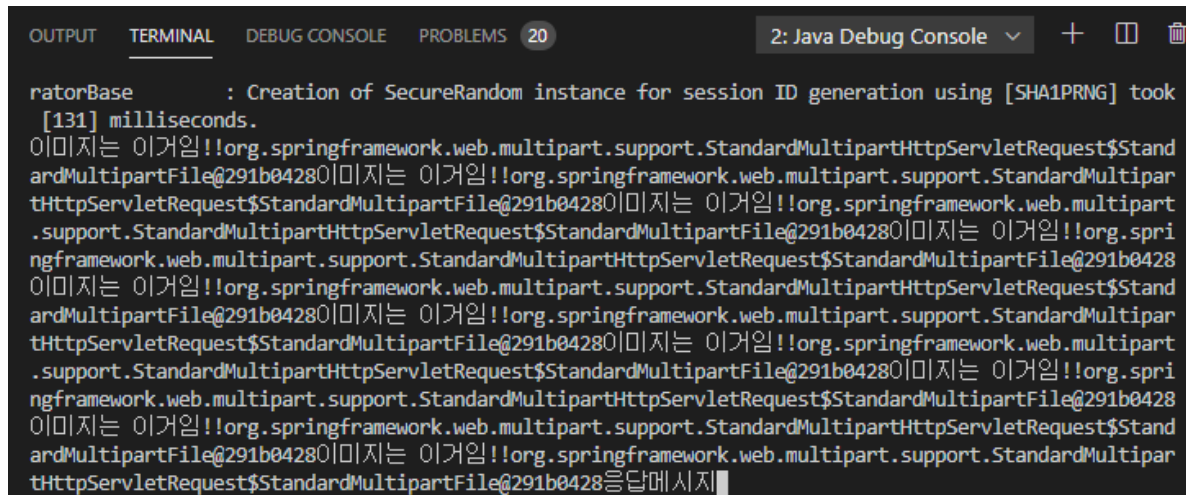
```

집중할 부분인 이 부분이다. Autowired 되어있는 restTemplateService의 addData를 호출한다.

호출한 값을 ResponseEntity<String>형태로 받아왔다.
또한 이를 Terminal에 print해보았다.

restTemplateService의 addData를 호출하고 -> RestTemplateUtil의 post를 호출한다. -> Util에서는 restTemplate.postForEntity를 통해 Flask API의 URI를 호출한다.
이는 Flask의 testapi로 정의된 route를 호출한다.
요청을 받은 FlaskAPI는 return 값으로 "응답메시지" 문자열을 반환한다.
결과값은 ResponseEntity<String> a 에 들어온다.

이를 Terminal에 출력해봤다.



```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS  20  2: Java Debug Console  +  [icon] [icon]
ratorBase      : Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took
[131] milliseconds.
이미지는 이거임!!org.springframework.web.multipart.support.StandardMultipartHttpServletRequest$Stand
ardMultipartFile@291b0428이미지는 이거임!!org.springframework.web.multipart.support.StandardMultipart
tHttpServletRequest$StandardMultipartFile@291b0428이미지는 이거임!!org.springframework.web.multipart
.support.StandardMultipartHttpServletRequest$StandardMultipartFile@291b0428이미지는 이거임!!org.spr
ingframework.web.multipart.support.StandardMultipartHttpServletRequest$StandardMultipartFile@291b0428
이미지는 이거임!!org.springframework.web.multipart.support.StandardMultipartHttpServletRequest$Stand
ardMultipartFile@291b0428이미지는 이거임!!org.springframework.web.multipart.support.StandardMultipart
tHttpServletRequest$StandardMultipartFile@291b0428이미지는 이거임!!org.springframework.web.multipart
.support.StandardMultipartHttpServletRequest$StandardMultipartFile@291b0428이미지는 이거임!!org.spr
ingframework.web.multipart.support.StandardMultipartHttpServletRequest$StandardMultipartFile@291b0428
이미지는 이거임!!org.springframework.web.multipart.support.StandardMultipartHttpServletRequest$Stand
ardMultipartFile@291b0428이미지는 이거임!!org.springframework.web.multipart.support.StandardMultipart
tHttpServletRequest$StandardMultipartFile@291b0428응답메시지
```

결과값을 확인하기 위해 난잡하게 print를 찍어냈지만
"응답메시지" 문자열을 출력한 모습이다.

Spring Boot에서 RestTemplate를 통해 Flask API를 호출했다!!!

Made by 장경석