실습순서 - Spring boot

1. MultipartResolver 설정
   - Bean 등록
     - Auto-Configuration 지원
     - StandardServletMultipartResolver를 구현 클래스로 사용
   - application.properties 추가

```
# MultipartResolver Setting
spring.servlet.multipart.file-size-threshold=0B
spring.servlet.multipart.location=C:/temp
spring.servlet.multipart.max-file-size=1MB
spring.servlet.multipart.max-request-size=10MB
```

항목	설명	default
spring.servlet.multipart.enabled	멀티파트 업로드 지원여부	true
spring.servlet.multipart.file-size-threshold	파일이 메모리에 기록되는 임계값	0B
spring.servlet.multipart.location	업로드된 파일의 임시 저장 공간	
spring.servlet.multipart.max-file-size	파일의 최대 사이즈	1MB
spring.servlet.multipart.max-request-size	요청의 최대 사이즈	10MB

> **DataSize**
>
> 참고사이트 : https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/util/unit/DataSize.html
>
> - Spring의 클래스로 DataSize를 처리해주는 클래스
> - 특정한 DataSize를 작성하면 그에 맞게 원하는 데이터 형식으로 바꿔준다.

2. <form/>을 기반으로 파일 업로드
   - formUpload.html

```
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org/">
<head>
<meta charset="UTF-8">
<title>Form 방식</title>
</head>
<!-- formUpload.html -->
<body>
    <form th:action="@{/uploadForm}" method="post"
          enctype="multipart/form-data">
        <input type="file" name="files" multiple>
        <button type="submit">저장</button>
    </form>
</body>
</html>
```

   - UploadController

▼ 파일 업로드 경로 설정

> 프로젝트 내부에 파일을 업로드 경로를 설정할 경우
> 해당 프로젝트를 재배포할 때 사용자가 업로드한 파일이 손실될 가능성이 있음.
> ⇒ 프로젝트 바깥에 경로를 설정

> 다만, 메인 페이지에서 사용되는 정적인 파일의 경우
> ‘classpath:/static/’을 활용해 프로젝트와 함께 관리하는 방법도 사용

```
package com.yedam.app.upload.web;

import java.io.File;
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;

import lombok.extern.slf4j.Slf4j;

@Slf4j
@Controller
public class UploadController {
    private String uploadPath = "D:/upload/";

    @GetMapping("formUpload")
    public void formUploadPage() {}

    @PostMapping("uploadForm")
    public String formUploadFile(@RequestPart MultipartFile file) {
        log.info(uploadPath);
        log.info(file.getContentType()); // 파일의 종류 및 확장자
        log.info(file.getOriginalFilename()); // 파일 이름
        log.info(String.valueOf(file.getSize())); // 파일 크기

        String fileName = file.getOriginalFilename();
        String saveName = uploadPath + File.separator + fileName;
        log.debug("saveName : " + saveName);

        //Paths.get() 메서드는 특정 경로의 파일 정보를 반환(경로 정의)
        Path savePath = Paths.get(saveName);

        try {
             // 실제로 파일을 업로드(저장) 하는 메서드 transferTo(file)
            file.transferTo(savePath);
```

```
        } catch (IOException e) {
                e.printStackTrace();
        }

        return "redirect:formUpload";
    }
}
```

3. 파일 업로드 시 외부 경로 설정 ⇒ application.properties로 변경
    - 개발환경과 운영환경이 달라지므로 배포 시 지정된 업로드 변경을 고려
    - application.properties 추가

    ```
    # file upload path
    file.upload.path=D:/upload/
    ```

    - UploadController 변경

    ```
    //before
    private String uploadPath = "D:/upload/";

    //after
    @Value("${file.upload.path}")
    private String uploadPath;
    ```

    > @Value
    >
    > - 환경변수 혹은 Properties에 등록된 변수 값을 불러옴

4. AJAX를 기반으로 파일 업로드
    - upload.jsp

    ```
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>AJAX</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
    </head>
    <body>
        <div>
            <input type="file" name="uploadFiles" multiple>
            <button class="uploadBtn">upload</button>
        </div>
        <script>
            $('.uploadBtn').on('click', function(event){
                let formData = new FormData();
                // 1) <from/>태그 내부의 입력태그 정보를 한번에 받음
                // 2) AJAX를 이용하여 'Content-type:multipart/form-data'를 보내는 경우

                let input = $(event.currentTarget).prev()[0];
                let fileList = input.files;

                for(let file of fileList){
                    formData.append(input.name, file);
                }
    ```

```
            console.log(formData.get("uploadFiles"));

            // 실제 업로드 부분
        });

    </script>
</body>
</html>
```

▼ 실제 업로드 부분

```
        // == Javascript, fetch
        fetch('uploadsAjax', {
                method : 'post',
                body : formData
            })
            .then(res => res.json())
            .then(result => {
                console.log(result);
            })
            .catch(err => console.log(err));

        // == jQuery, ajax
        $.ajax('uploadsAjax',{
            type : 'post',
            data : formData,     // data 속성의 값(객체 타입)을
            processData : false, // QueryString으로 변환하는 설정
            contentType : false  // 기본 contentType을 사용하지 않겠다고 설정
        })
        .done(result => {
            for (let image of result){
                let imgTag
                   = `<img src='/yedam/images/${image}' style="width : 50%;">`;
                $('div').append(imgTag);
            }
        })
        .fail(err => console.log(err));
```

- UploadController 에 추가

```
    @GetMapping("upload")
    public void uploadPage() {}

    @PostMapping("/uploadsAjax")
    @ResponseBody
    public List<String> uploadFile
            (@RequestPart MultipartFile[] uploadFiles) {

        List<String> imageList = new ArrayList<>();

        for(MultipartFile uploadFile : uploadFiles){
            if(uploadFile.getContentType().startsWith("image") == false){
                System.err.println("this file is not image type");
                return null;
            }

            String fileName = uploadFile.getOriginalFilename();
```

```
        System.out.println("fileName : " + fileName);

        //날짜 폴더 생성 : 파일을 관리하기 위해 폴더별로 분리
        String folderPath = makeFolder();
        //UUID : 파일명이 중복되는 부분 방지
        String uuid = UUID.randomUUID().toString();
        //저장할 파일 이름 중간에 "_"를 이용하여 구분

        String uploadFileName = folderPath +File.separator + uuid + "_" + fileName;

        String saveName = uploadPath + File.separator + uploadFileName;

        Path savePath = Paths.get(saveName);

        System.out.println("path : " + saveName);
        try{
            uploadFile.transferTo(savePath);
            } catch (IOException e) {
                 e.printStackTrace();
            }
            // DB에 해당 경로 저장
            // 1) 사용자가 업로드할 때 사용한 파일명
            // 2) 실제 서버에 업로드할 때 사용한 경로
            imageList.add(setImagePath(uploadFileName));
         }

        return imageList;
    }

    private String makeFolder() {
        String str = LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));
        // LocalDate를 문자열로 포멧
        String folderPath = str.replace("/", File.separator);
        File uploadPathFoler = new File(uploadPath, folderPath);
        // File newFile= new File(dir,"파일명");

        // 해당 경로의 존재유무를 확인
        if (uploadPathFoler.exists() == false) {
            // mkdirs(): 디렉토리의 상위 디렉토리가 존재하지 않을 경우에는
            //           상위 디렉토리까지 모두 생성하는 함수
            uploadPathFoler.mkdirs();
        }
        return folderPath;
    }

    private String setImagePath(String uploadFileName) {
        return uploadFileName.replace(File.separator, "/");
    }
```

- WebMvcConfig

```
package com.yedam.app.upload.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
```

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Value("${file.upload.path}")
    private String uploadPath;

    //리소스 핸들링
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/images/**") // URL
                .addResourceLocations("file:///"+uploadPath, ""); // 실제 경로
                // => 하나의 URL에 여러 경로를 매핑하는 게 가능함
    }
}
```