Spring Framework	-파일업로드 1.일반업로드 2.비동기업로드 3.이미지파일 불러오기 4.파일 다운로드

1.일반업로드

파일 업로드 방식에는 크게 3가지가 존재합니다.

1. cos.jar이용 (jsp에서 사용함)
2. commons-fileupload이용: 스프링에서 가장 일반적으로 많이 활용됨 (서블릿 스팩 3.0 이전, 이후 모두 사용가능
3. 서블릿 3.0이상에서 부터는 자체적인 파일업로드가 지원됨

commons-fileupload를 이용한방식을 사용하겠습니다.
준비사항

pom.xml 라이브러리 다운

```
<dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.3.1</version>
</dependency>
<dependency>
         <groupId>commons-io</groupId>
         <artifactId>commons-io</artifactId>
         <version>1.4</version>
</dependency>
```

servlet-context에 반드시 설정

```
<!-- 파일업로드 -->
<!-- 반드시 꼭 백번 id를 multipartResolver선언  -->
<beans:bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <!-- 최대업로드 가능한 바이트크기 -->
        <beans:property name="maxUploadSize" value="52428800" />
        <!-- defaultEncoding -->
        <beans:property name="defaultEncoding" value="utf-8" />
</beans:bean>
```

1.일반 업로드

파일 업로드에서는 enctype(인코딩타입)을 multipart/form-data으로 반드시 설정해야합니다

```
<form action="upload_ok" method="post" enctype="multipart/form-data">
     파일:<input type="file" name="file"><br>
     <input type="submit" value="전송"><br>
</form>
```

name명 / 파일타입으로 받음

```
@RequestMapping( "/upload_ok" )
public String upload(@RequestParam("file") MultipartFile file) {
     try {
          String fileRealName = file.getOriginalFilename(); //파일 정보
          Long size = file.getSize(); //파일사이즈

          String fileExtension = fileRealName.substring(fileRealName.lastIndexOf("."), fileRealName.length()); // 확장자
          String uploadFolder = " 업로드경로";

          File saveFile = new File(uploadFolder + "\\" + fileRealName);
          file.transferTo(saveFile);
     } catch (Exception e) {
          System.out.println("업로드중 에러발생:" + e.getMessage());
     }

     return "fileupload/upload_ok";
}
```

spring의 transferTo(파일타입) 은
FileWriter작업을 손쉽게 한방에 처리해준다

끝.

1.비동기 업로드

파일업로드 태그

```
<input type="file" name="file" id="file">
```

```
var data = $("#file")
var formData = new FormData(); //ajax폼전송의 핵심 FormData객체
console.log(formData); //폼데이터 확인
console.log(data[0].files[0]); //파일태그의 담긴 파일을 확인하는 키값

formData.append("file", data[0].files[0]); //파일타입으로 폼데이터에 추가

$.ajax({
	url: "upload",
	processData: false,
	contentType: false,
	data: formData, //폼데이터객체를 넘깁니다
	type: "POST",
	success: function(result) {
		alert("업로드성공");
	},
	error: function(status, er) {
		alert("업로드실패");
	}
})
```

폼을 &변수=값 형식으로 자동으로 변경되는 것을 막는 요소
ajax방식에서는 반드시 false로 처리 "multipart/form-data"로 선언됨

서버에서는

```
@RequestMapping("/upload")
@ResponseBody
public String upload(@RequestParam("file") MultipartFile file) {

	File saveFile = new File(업로드경로 + "WW" + 파일명);
	file.transferTo(saveFile);

	return 성공실패여부
}
```

1.비동기 업로드

업로드 후에는 비동기 식으로 목록을 불러와서 태그에 그리는 작업을 처리합니다

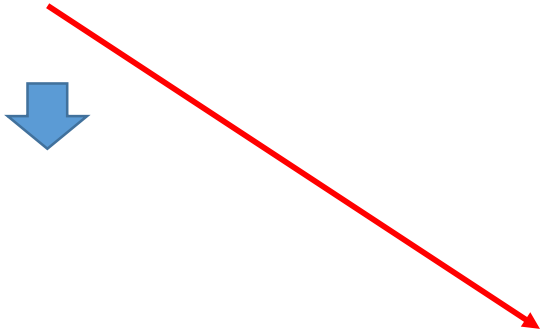
```
var str = "";
function getList() {
	$.getJSON(
		"getList",
		function(list) {

			for(var i = 0; i < list.length; i++) {
				str  += 태그그리기 작업
				...
				...
				...
				...
			}
		}
	)
}
```

3.이미지파일 불러오기(방법1)

이미지 태그가 다음과 같을 때, src가 경로를 읽어 호출하는 원리를 이용합니다

```
<img src="view?fileLoca=폴더경로&filename=파일명">
```



서버에서는

```
@RequestMapping("/view")
@ResponseBody
public byte[] getFile(@RequestParam("fileLoca") String fileLoca, @RequestParam("fileName")String fileName) {

	File file = new File("업로드경로\\" + fileLoca + "\\" + fileName);

	byte[] result = null;
	try {
		result = FileCopyUtils.copyToByteArray(file);

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

	return result;
}
```

스프링의 파일데이터를 읽어서 바이트배열형으로 리턴하는 메서드 (매개변수로 File타입을 받는다)

3.이미지파일 불러오기(방법2)

```
@RequestMapping("/view")
@ResponseBody
public ResponseEntity<byte[]> getFile(@RequestParam("fileLoca") String fileLoca, @RequestParam("fileName")String fileName) {

	File file = new File("업로드경로\\" + fileLoca + "\\" + fileName);
	ResponseEntity<byte[]> result = null;
	try {
		HttpHeaders header = new HttpHeaders();
		header.add("Content-Type", Files.probeContentType(file.toPath()) );

		result = new ResponseEntity<>(FileCopyUtils.copyToByteArray(file), header, HttpStatus.OK);

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

파일의 경로를 읽어 파일의 MIME타입(변환된 타입)을 헤더에 저장

RestFul API를 개발하기 위해, (반환 데이터, HttpHeader, 상태코드) 모두 지정해서 반환 해주기 위해 사용하는 것입니다

ResponseEntity<>(바디에 담을내용, 헤더에 담을내용, 상태 메세지)

네트워크를 통해 binary파일(음악,동영상 등) 을 전송하는 경우들이 생기게 됩니다.
이런 파일들을 전송할 때 binary파일 형태로 전송할 때, 파일을 정상적으로 전송하지 못하는 경우가 발생하게 됩니다

그래서 binary파일을 문서로 변환해서 보내도록 됩니다.
텍스트파일로 변환 하는작업(Encoding), 다시 바이너리 파일로 변환하는 작업(Decoding)을 거치게 됩니다

MIME이란 이런 파일의 변환을 뜻합니다. (파일을 주고 받을 때 파일 변환을 해서 보냄)

클라이언트와 서버간에 데이터를 전송할 때, 전송하는 문서의 헤더에 파일이나, 데이터의 바이트데이터를 보냅니다.
이런 헤더에는 많은 정보(보내는 데이터의 Content-Type, 서버의 시간, 프로토콜 등등등)을 담고 있습니다.

웹에서 이런 헤더에 담는 가장 기본적인 MIME타입은 text/html 이며,
파일을 전송할 때는 이런 MIME타입을 헤더에 담아서 보낼 수 있습니다

3. 파일 다운로드

클릭시 비동기로 요청보내는 형식을 이용합니다

```
<a href='download?fileLoca=경로&fileName=이름'>파일다운로드</a>;
```

```
@RequestMapping(value = "/download")
@ResponseBody
public ResponseEntity<byte[]> download(@RequestParam("fileLoca") String fileLoca,
                                       @RequestParam("fileName") String fileName) {

    File file = new File("업로드경로\\" + fileLoca + "\\" + fileName);


    ResponseEntity<byte[]> result = null;
    try {
        HttpHeaders header = new HttpHeaders();
        header.add("Content-Disposition", "attachment; filename=" + fileName );

        byte[] fileCopy = FileCopyUtils.copyToByteArray(file);
        result = new ResponseEntity<>(fileCopy, header, HttpStatus.OK);

    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
```

해당경로의 파일의 바이트데이터를 읽어서
응답객체에, (파일데이터, 헤더정보, 상태코드) 저장

주의할점

응답헤더에 아래를 반드시 담아줘야 합니다
ex)
`Content-Disposition, attachment; filename=filename.csv`

`Content-Disposition, attachment;`
브라우저에게 어떻게 처리할지 알려주는 아주 중요한 정보 입니다
`attachment`**인 경우 다운로드됩니다**

`filename=`파일명.확장자
해당 파일의 확장자가 반드시 포함되어야 합니다

수고하셨습니다