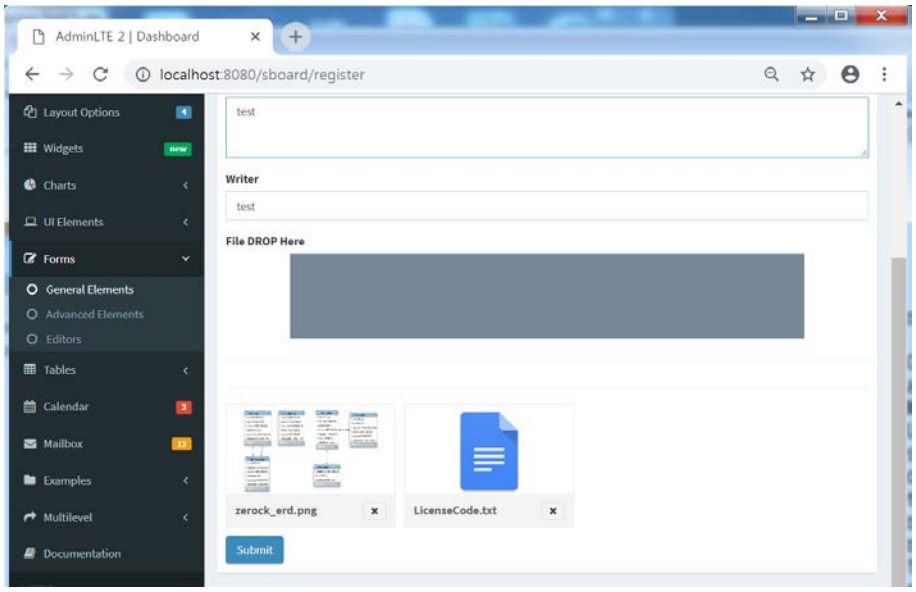
4장 게시물의 첨부파일 기능

4.1 스프링 MVC의 파일 업로드

4.1.1 파일 업로드의 활용

- 파일 업로드 기능은 크게 'JPG, GIF, PNG'와 같은 이미지 파일과 그렇지 않은 일반 파일들을 서버에 저장하고, 다른 사용자들이 보거나 다운로드할 수 있도록 구현하는 것이다.
- 게시물을 등록할 때 원하는 파일을 Drag & Drop을 이용해서 원하는 파일을 업로드 할 수 있다.
- 업로드 된 일반 파일은 아래쪽에서 확인할 수 있고, 이미지 파일의 경우에는 원래 이미지보다 작은 이미지 파일(썸네일)을 생성해서 처리한다.



4.1.2 예제 프로젝트의 생성

- 작성하는 프로젝트는 STS를 사용해서 'ex04Lab' 프로젝트로 생성한다.

(1) 실습을 위한 라이브러리 추가

- 이미지 파일의 경우에는 이미지를 깨끗하게 축소할 수 있는 imgScalr 라이브러리를 활용한다.

```
// pom.xml 파일의 추가 라이브러리
                    <dependency>
                              <groupId>commons-fileupload</groupId>
                              <artifactId>commons-fileupload</artifactId>
                              <version>1.3.1</version>
                    </dependency>
                    <dependency>
                              <groupId>org.imgscalr</groupId>
```

```
                        <artifactId>imgscalr-lib</artifactId>
                        <version>4.2</version>
                </dependency>
                <!—객체를 JSON으로 반환 -->
                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>2.5.4</version>
                </dependency>
```

(2) 파일 업로드 관련 <bean> 설정

■ 스프링 MVC에서 파일 업로드를 처리하기 위해서는 파일 업로드로 들어오는 데이터를 처리하는 객체가 필요하다. multipartResolver 객체의 설정은 웹과 관련 있기 때문에 root-context.xml이 아닌 servlet-context.xml을 이용해서 설정한다.

[ex04/src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml]

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <beans:beans xmlns="http://www.springframework.org/schema/mvc"
03          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04  xmlns:beans="http://www.springframework.org/schema/beans"
05          xmlns:context="http://www.springframework.org/schema/context"
06          xsi:schemaLocation="http://www.springframework.org/schema/mvc
07  http://www.springframework.org/schema/mvc/spring-mvc.xsd
08                  http://www.springframework.org/schema/beans
09  http://www.springframework.org/schema/beans/spring-beans.xsd
10                  http://www.springframework.org/schema/context
11  http://www.springframework.org/schema/context/spring-context.xsd">
12
13          <!-- DispatcherServlet Context: defines this servlet's request-processing
14                  infrastructure -->
15
16          <!-- Enables the Spring MVC @Controller programming model -->
17          <annotation-driven />
18
19          <!-- Handles HTTP GET requests for /resources/** by efficiently serving
20                  up static resources in the ${webappRoot}/resources directory -->
21          <resources mapping="/resources/**" location="/resources/" />
22
23          <!-- Resolves views selected for rendering by @Controllers to .jsp resources
24                  in the /WEB-INF/views directory -->
25          <beans:bean
26                  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
27                  <beans:property name="prefix" value="/WEB-INF/views/" />
28                  <beans:property name="suffix" value=".jsp" />
29          </beans:bean>
30
31          <context:component-scan base-package="org.zerock.controller" />
32
33          <beans:bean id="multipartResolver"
34                  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
35                  <beans:property name="maxUploadSize" value="10485760"></beans:property>
36          </beans:bean>
37
38    <beans:bean id="uploadPath" class="java.lang.String">
39      <beans:constructor-arg value="C:||zzz||upload">
40      </beans:constructor-arg>
41    </beans:bean>
42
43  </beans:beans>
```

(3) 테이블 생성

```
-- 게시물 등록의 파일 업로드
create table tbl_attach (
          fullName varchar(150) not null,
          bno int not null,
          regdate timestamp default now(),
          primary key (fullName)
);

alter table tbl_attach add constraint fk_board_attach
foreign key (bno) references tbl_board (bno);
```

4.1.3 일반적인 파일 업로드 이해하기

(1) POST 방식의 파일 업로드 처리

- Multipart로 구성된 데이터의 처리는 프로젝트의 시작단계에서 적용한 multipartResolver 설정을 통해서 처리된다.

```
// ex04/src/main/java/org/zerock/controller/UploadController.java의 일부
@RequestMapping(value = "/uploadForm", method = RequestMethod.POST)
public String uploadForm(MultipartFile file, Model model) throws Exception {

    logger.info("originalName: " + file.getOriginalFilename());
    logger.info("size: " + file.getSize());
    logger.info("contentType: " + file.getContentType());

    String savedName = uploadFile(file.getOriginalFilename(), file.getBytes());

    model.addAttribute("savedName", savedName);

    return "uploadResult";
}
```

- 서버를 실행하고 파일을 입력하는 화면에서 한글 이름의 파일을 업로드 해 본다. 만일 출력한 파일의 이름이 깨지는 문제가 발생한다면 web.xml에 한글 처리용 필터를 적용해야만 한다.

```
// WEB-INF/web-xml의 일부

          <filter>
            <filter-name>encoding</filter-name>
            <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
            <init-param>
             <param-name>encoding</param-name>
             <param-value>UTF-8</param-value>
            </init-param>
          </filter>

          <filter-mapping>
            <filter-name>encoding</filter-name>
            <url-pattern>/*</url-pattern>
          </filter-mapping>
```

(2) 업로드 파일의 저장

- 서버외 파일 저장 경로 설정: 파일을 저장할 경로는 상수처럼 사용되기 때문에 servlet-context.xml 파일을 이용해서 특정 경로를 문자열로 설정한다.

```
// ex04/src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml의 일부
<beans:bean id="uploadPath" class="java.lang.String">
    <beans:constructor-arg value="C:\\zzz\\upload">
    </beans:constructor-arg>
</beans:bean>
```

- UploadController의 파일 저장

```
// org.zerock.controller.UploadController.java의 일부

@Resource(name = "uploadPath")
  private String uploadPath;

  @RequestMapping(value = "/uploadForm", method = RequestMethod.GET)
  public void uploadForm() {
  }

  @RequestMapping(value = "/uploadForm", method = RequestMethod.POST)
  public String uploadForm(MultipartFile file, Model model) throws Exception {

    logger.info("originalName: " + file.getOriginalFilename());
    logger.info("size: " + file.getSize());
    logger.info("contentType: " + file.getContentType());

    String savedName = uploadFile(file.getOriginalFilename(), file.getBytes());

    model.addAttribute("savedName", savedName);

    return "uploadResult";
  }

  private String uploadFile(String originalName, byte[] fileData) throws Exception {

    UUID uid = UUID.randomUUID();

    String savedName = uid.toString() + "_" + originalName;

    File target = new File(uploadPath, savedName);

    FileCopyUtils.copy(fileData, target);

    return savedName;

  }
```

(3) <iframe>을 이용한 파일 업로드의 결과 처리

- 최근의 게시물들은 게시물을 작성할 때 내용과 별개로 파일을 필요할 때마다 추가하는 형태가 일반적이므로, 업로드 된 결과 역시 현재의 창에서 바로 사용할 수 있도록 하는 것이 좋다.

- 문제는 <form> 태그의 경우 기본이 현재 브라우저의 창에서 전송이 일어나기 때문에 화면 전환을 피할 수 없다는 점인데, <form> 태그에 target 속성을 주고, <iframe>을 이용하면 화면 전환 효과를 없앨 수 있다.

- <iframe>을 이용해서 파일을 업로드 하는 절차: 1)<form> 태그의 전송은 화면에 포함된 <iframe>으로 전송 -> 2)결과 페이지는 <iframe>내에 포함되므로 화면의 변화 없음 -> 3) 결과 페이지에서 다시 바깥쪽(parent)의 JavaScript 함수 호출

4.1.4 Ajax 방식의 파일 업로드

(1) Ajax 업로드용 컨트롤러와 JSP작성하기

```
// ex04/src/main/java/org/zerock/controller/UploadController.java의 일부
@RequestMapping(value = "/uploadAjax", method = RequestMethod.GET)
public void uploadAjax() {
}
```

(2) 이벤트 처리하기

```
// ex04/src/main/webapp/WEB-INF/views/uploadAjax.jsp의 일부
                    $(".fileDrop").on("drop", function(event){
                              event.preventDefault();

                              var files = event.originalEvent.dataTransfer.files;

                              var file = files[0];

                              //console.log(file);

                              var formData = new FormData();

                              formData.append("file", file);

                              $.ajax({
                                        url: '/uploadAjax',
                                        data: formData,
                                        dataType:'text',
                                        processData: false,
                                        contentType: false,
                                        type: 'POST',
                                        success: function(data){

                                                  var str ="";

                                                  if(checkImageType(data)){
                                                            str ="<div><a
href=displayFile?fileName="+getImageLink(data)+">"
                                                                      +"<img
src='displayFile?fileName="+data+"'/>"
                                                                      +"</a><small data-
src="+data+">X</small></div>";
                                                  }else{
                                                            str = "<div><a
href='displayFile?fileName="+data+"'>"
                                                                      +
getOriginalName(data)+"</a>"
```

```
                                                                    +"<small data-
src="+data+">X</small></div></div>";
                                                    }

                                                    $(".uploadedList").append(str);
                                                }
                                            });
                        });
```

(3) FormData를 이용한 서버 호출

- Ajax 처리하기
- UploadController에서 Ajax 처리하기

4.2 전송된 파일의 저장

4.2.1 파일 업로드용 클래스 설계하기

(1) 업로드 기능의 설계

- 파일을 업로드하기 위해서는 적어도 다음과 같은 3개의 데이터가 필요하다.
  - 파일의 저장 경로 (uploadPath)
  - 원본 파일의 이름 (originalName)
  - 파일 데이터 (byte[])

(2) 업로드 폴더의 생성 처리

- 작성하려는 uploadFile()은 다음과 같은 과정을 통해서 진행돼야 한다.
  - UUID를 이용한 고유한 값 생성
  - UUID와 결합한 업로드 파일 이름 생성
  - 파일이 저장될 '/년/월/일' 정보 생성
  - 업로드 기본 경로(uploadPath)와 '/년/월/일' 폴더 생성
  - 기본 경로 + 폴더 경로 + 파일 이름으로 파일 저장

(3) 썸네일 생성하기

- 업로드 되는 파일은 크게 브라우저에서 보여지는 이미지 종류의 파일과 사용자가 다운로드 하는 형태의 파일로 구분될 수 있다.
  - 이미지 타입의 파일: JPG, GIF, PNG
  - 다운로드 타입의 파일: 이미지 타입의 파일을 제외한 모든 종류

4.3 UploadController의 재구성

```
// ex04/src/main/java/org/zerock/controller/UploadController.java의 일부
@ResponseBody
```

```
@RequestMapping(value ="/uploadAjax", method=RequestMethod.POST,
                produces = "text/plain;charset=UTF-8")
public ResponseEntity<String> uploadAjax(MultipartFile file)throws Exception{

    logger.info("originalName: " + file.getOriginalFilename());


    return
      new ResponseEntity<>(
          UploadFileUtils.uploadFile(uploadPath,
                file.getOriginalFilename(),
                file.getBytes()),
          HttpStatus.CREATED);
}
```

4.4 전송된 파일을 화면에 표시하기

4.4.1 파일 데이터 전송하기

■ 여기서 주의해야 하는 점은 파일 데이터를 보내 줄 때의 MIME 타입이다. 이미지 파일인 경우 적절한 포맷에 맞는 MIME 타입을 지정해 줘야 하고, 일반 파일의 경우는 다운로드를 할 수 있도록 처리해야 한다.

(1) UploadController의 파일 전송 기능 구현

```
// ex04/src/main/java/org/zerock/controller/UploadController.java의 일부
@ResponseBody
@RequestMapping("/displayFile")
public ResponseEntity<byte[]>  displayFile(String fileName)throws Exception{

    InputStream in = null;
    ResponseEntity<byte[]> entity = null;

    logger.info("FILE NAME: " + fileName);

    try{

      String formatName = fileName.substring(fileName.lastIndexOf(".")+1);

      MediaType mType = MediaUtils.getMediaType(formatName);

      HttpHeaders headers = new HttpHeaders();

      in = new FileInputStream(uploadPath+fileName);

      if(mType != null){
        headers.setContentType(mType);
      }else{

        fileName = fileName.substring(fileName.indexOf("_")+1);
        headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);
        headers.add("Content-Disposition", "attachment; filename=\""+
          new String(fileName.getBytes("UTF-8"), "ISO-8859-1")+"\"");
      }

        entity = new ResponseEntity<byte[]>(IOUtils.toByteArray(in),
          headers,
```

```
            HttpStatus.CREATED);
        }catch(Exception e){
          e.printStackTrace();
          entity = new ResponseEntity<byte[]>(HttpStatus.BAD_REQUEST);
        }finally{
          in.close();
        }
          return entity;
    }
```

(2) 일반 파일인 경우의 다운로드 테스트

- http://localhost:8080/displayFile?fileName=/2017/10/29/e67a4639-d80b-44ad-94bb-47bddea8015b_scott.sql

(3) 이미지 타입의 파일인 경우의 테스트

- http://localhost:8080/displayFile?fileName=/2017/07/26/0ec1f995-487f-4a13-928e-17003a480a0b_user5-128x128.jpg

4.4.2 JSP에서 파일 출력하기

- Ajax로 호출된 결과를 이미지나 파일의 링크로 생성하면 사용자가 업로드 한 파일을 다운로드 하거나 확인할 수 있다. 작업은 다음과 같은 순서로 진행된다.
  - Ajax의 결과로 전송된 파일이 이미지 타입의 파일인지를 구분한다.
  - 이미지 타입인 경우 <img> 태그를 생성한다. 만들어진 <img> 태그를 원하는 요소에 넣어준다.
  - 이미지 타입이 아닌 경우 태그와 <a> 태그를 이용해서 처리한다.

4.5 첨부파일의 삭제

4.5.1 UploadController의 삭제 처리

```
// ex04/src/main/webapp/WEB-INF/views/uploadAjax.jsp의 일부
                            $.ajax({
                                        url: '/uploadAjax',
                                        data: formData,
                                        dataType:'text',
                                        processData: false,
                                        contentType: false,
                                        type: 'POST',
                                        success: function(data){

                                                var str ="";

                                                if(checkImageType(data)){
                                                        str ="<div><a
href=displayFile?fileName="+getImageLink(data)+">"

                                                                                +"<img
src='displayFile?fileName="+data+"'/>"

                                                                                +"</a><small data-
```

```
src="+data+"'>X</small></div>";
                                                    }else{
                                                              str = "<div><a
href='displayFile?fileName="+data+"'>"
                                                                                  +
getOriginalName(data)+"</a>"
                                                                                  +"<small data-
src="+data+"'>X</small></div></div>";
                                                    }

                                                    $(".uploadedList").append(str);
                                          }
                                        });
```