스프링(Spring) 개발

간단한 게시판을 만들어 보면서 스프링의 MVC 구조와 DB, 트랜잭션, 파일 업로드, jQuery등을 살펴봅니다.

(13) 파일 업로드

첨부파일 업&다운로드의 기초적인 개념입니다.
실제 프로젝트에서는 이 내용을 바탕으로 좀 더 보완해야 됩니다.

13.1. SQL

13.1.1). 테이블 생성

먼저 다음의 쿼리를 실행시키자.

```
1   CREATE TABLE TB_FILE
2   (
3     IDX    NUMBER,
4     BOARD_IDX NUMBER NOT NULL,
5     ORIGINAL_FILE_NAME VARCHAR2(260 BYTE) NOT NULL,
6     STORED_FILE_NAME VARCHAR2(36 BYTE) NOT NULL,
7     FILE_SIZE NUMBER,
8     CREA_DTM   DATE DEFAULT SYSDATE NOT NULL,
9     CREA_ID    VARCHAR2(30 BYTE) NOT NULL,
10    DEL_GB     VARCHAR2(1 BYTE) DEFAULT 'N' NOT NULL,
11    PRIMARY KEY (IDX)
12  );
```

이 테이블은 첨부파일의 정보를 저장할 테이블이다.
몇가지 컬럼을 살펴보자.
BOARD_IDX 컬럼은 앞서 게시글 목록에서 작성한 TB_BOARD에서 사용된 컬럼이다.
각 게시글마다 여러개의 파일을 저장할 수 있으니, 해당 파일이 어떤 게시글에 속해있는지를 알기위한 컬럼으로 사용된다.
그 다음으로 **ORIGINAL_FILE_NAME과 STORED_FILE_NAME 컬럼**은 각각 원본 파일 이름과 변경된 파일 이름을 저장한다.
파일을 업로드하면 그 파일을 서버의 어딘가에 저장이 되어야하는데 만약 파일 이름이 같을 경우, 저장 중 문제가 발생하거나 파일 이름이 변경될 수 있다.
따라서 파일을 저장할 때, 원본파일의 이름을 저장해 놓고 서버에는 변경된 파일이름으로 파일을 저장한다.

나중에 파일 다운로드를 할때에는, 파일의 이름을 통해서 해당 파일에 접근하기 때문에 겹치지 않는 파일이름은 필수이다.
여기서 **ORIGINAL_FILE_NAME**은 **260**byte, **STORED_FILE_NAME은 36byte**로 잡았는데, 이유는 간단하다.
원본파일명은 windows에서는 최대 256글자 + 확장자인 260글자이고, STORED_FILE_NAME은 32글자 + 확장자로 저장할 계획이다.
여기서는 32글자의 랜덤문자를 생성할건데, 변경된 파일이름을 해당 날짜+시간 등으로 한다면 그에 해당하는 길이로 잡으면 된다.
마지막으로 **DEL_GB**는 해당 첨부파일이 삭제되었는지를 알려주는 컬럼이다.

13.1.2). 시퀀스 생성

다음의 쿼리를 실행시키자.

```
1   CREATE SEQUENCE SEQ_TB_FILE_IDX
2     START WITH 1
3     INCREMENT BY 1
4     NOMAXVALUE
5     NOCACHE;
```

당연히 위에서 생성한 TB_FILE 테이블의 PK로 사용할 시퀀스이다.

13.2. 설정파일

13.2.1). Maven Dependency 추가

pom.xml 을 열어서 다음의 dependency를 추가하자.

```
1   <!-- MultipartHttpServletRequset -->
2   <dependency>
3       <groupId>commons-io</groupId>
4       <artifactId>commons-io</artifactId>
5       <version>2.0.1</version>
6   </dependency>
7
8   <dependency>
9       <groupId>commons-fileupload</groupId>
10      <artifactId>commons-fileupload</artifactId>
11      <version>1.2.2</version>
12  </dependency>
```

앞서 프로젝트를 그대로 따라한 사람의 경우 해당 라이브러리는 이미 추가되어 있다.
현재 최신버전은 commons-io의 경우 2.4, commons-fileupload는 1.3.1이다.

13.2.2). context-common.xml 파일 추가

src/main/resources 폴더 밑의 config > spring 폴더 밑에 context-common.xml 파일을 생성한다.
context-common.xml에는 그 이름처럼 스프링의 공통적인 설정이 들어갈 예정이다.
생성 후, 다음의 내용을 작성한다.

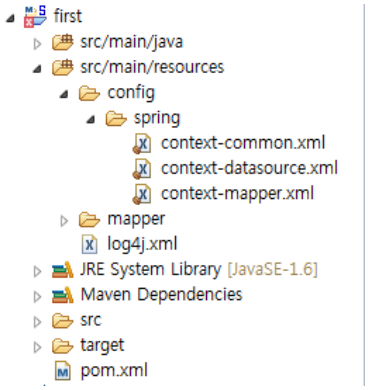
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:cache="http://www.springframework.org/schema/cache"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
                        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
                        http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd">

    <!-- MultipartResolver 설정 -->
    <bean id="multipartResolver"
 class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="maxUploadSize" value="100000000" />
        <property name="maxInMemorySize" value="100000000" />
    </bean>
</beans>
```

위에서 보듯이 **CommonsMultipartResolver**를 등록하는것이다.
COmmonsMultipartResolver는 스프링에서 파일업로드 기능을 구현해놓은 클래스다.
이 Resolver 덕분에 우리가 실제로 작성해야할 부분은 많이 없다.
value="100000000"이라고 되어있는데, 이는 서버에 **업로드할 수 있는 첨부파일의 최대 크기**를 의미한다. 단위는 byte로 10,000,000byte이기 때문에 **10MB**로 설정이 되어있다.
해당 크기 이상의 파일이 전송되면 에러가 발생한다.

이에 대한 에러처리는 나중에 다시 이야기를 하도록 하겠다.

여기까지의 폴더 구조는 다음과 같다.



여기까지 하면 설정은 완료된다. 다음은 구현 부분을 보도록 하자.

13.3. 파일 업로드

13.3.1). JSP

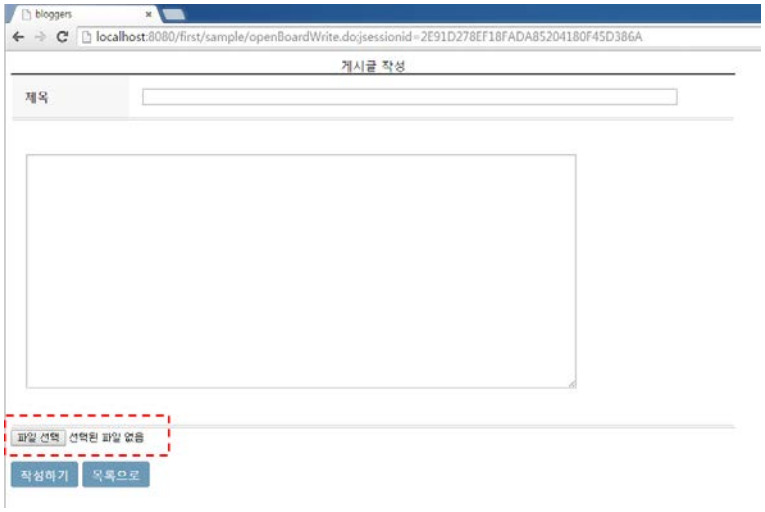
앞서 작성된 게시판 등록페이지를 변경해서 첨부파일도 등록할 수 있도록 해보자.
boardWrite.jsp를 다음과 같이 변경하자.

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2 pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html lang="ko">
5 <head>
6 <%@ include file="/WEB-INF/include/include-header.jspf" %>
7 </head>
8 <body>
9     <form id="frm" name="frm" enctype="multipart/form-data">
10        <table class="board_view">
11            <colgroup>
12                <col width="15%">
13                <col width="*"/>
14            </colgroup>
15            <caption>게시글 작성</caption>
16            <tbody>
17                <tr>
```

```
18                  <th scope="row">제목</th>
19                  <td><input type="text" id="TITLE" name="TITLE"
20class="wdp_90"></input></td>
21              </tr>
22              <tr>
23                  <td colspan="2" class="view_text">
24                      <textarea rows="20" cols="100" title="내용" id="CONTENTS"
25name="CONTENTS"></textarea>
26                  </td>
27              </tr>
28          </tbody>
29      </table>
30      <input type="file" name="file">
31      <br/><br/>
32
33      <a href="#this" class="btn" id="write">작성하기</a>
34      <a href="#this" class="btn" id="list">목록으로</a>
35  </form>
36
37  <%@ include file="/WEB-INF/include/include-body.jspf" %>
38  <script type="text/javascript">
39      $(document).ready(function(){
40          $("#list").on("click", function(e){ //목록으로 버튼
41              e.preventDefault();
42              fn_openBoardList();
43          });
44
45          $("#write").on("click", function(e){ //작성하기 버튼
46              e.preventDefault();
47              fn_insertBoard();
48          });
49      });
50
51      function fn_openBoardList(){
52          var comSubmit = new ComSubmit();
53          comSubmit.setUrl("<c:url value='/sample/openBoardList.do' />");
54          comSubmit.submit();
55      }
56
```

```
57        function fn_insertBoard(){
58            var comSubmit = new ComSubmit("frm");
59            comSubmit.setUrl("<c:url value='/sample/insertBoard.do' />");
60            comSubmit.submit();
61        }
      </script>
  </body>
  </html>
```

기존의 boardWrite.jsp 에서 바뀐 것은 <form>이 변경되고, <input type="file" name="file"> 태그가 추가된 것이다.



화면에서 볼수 있는 것과 같이, 첨부파일을 올릴수 있는 <input type="file"> 태그를 추가하였다.
여기서는 먼저 단일파일을 기준으로 설명을 하고, 그 후에 다중파일에 대해서 다시 설명하도록 하겠다.
다음으로 중요한것은 <form> 태그이다.
기존글에서는 <form id="frm"> 만 되어 있었는데, 이번에는 form에 **enctype="multipart/form-data" 라는것이 추가**되었다.
이것은 해당 폼을 Multipart 형식임을 알려주는데, 사진, 동영상 등 글자가 아닌 파일은 모두 Multipart 형식의 데이터다.
enctype을 설정해주는 것을 잊으면 안된다.

13.3.2). 파일 업로드 확인

이제 화면에서는 파일을 서버로 전송하게 되었다. 이제 할일은 클라이언트(화면)에서 전송된 파일을 받아서, 저장하는 작업이다.
먼저 화면에서 전송해 준 파일이 정상적으로 서버에 전송이 되었는지 확인해 보고 그 방식에 대해서 간단히 알아본 후, 그 내용을 바탕으로 실제 구현을 어떻게 하는지 설명하도록 하겠다.

먼저 Controller부터 수정을 하자.

1) SampleController.java를 열어서 insertBoard를 다음과 같이 수정하자.

```
@RequestMapping(value="/sample/insertBoard.do")
public ModelAndView insertBoard(CommandMap commandMap, HttpServletRequest request) throws Exception{
    ModelAndView mv = new ModelAndView("redirect:/sample/openBoardList.do");

    sampleService.insertBoard(commandMap.getMap(), request);

    return mv;
}
```

기존의 insertBoard 에서 변경된 점은 파라미터로 HttpServletRequest를 추가로 받는다는 것이다.
우리가 화면에서 전송한 모든 데이터는 HttpServletRequest에 담겨서 전송되고, 그것을 HandlerMethodArgumentResolver를 이용하여 CommandMap에 담아주었다.
그렇지만 **첨부파일은 CommandMap에서 처리를 하지 않았기 때문에 HttpServletRequest를 추가로 받도록 하였다.** 이 HttpServletRequest에는 모든 텍스트 데이터 뿐만이 아니라 화면에서 전송된 파일정보도 함께 담겨있다. 우리는 CommandMap을 이용하여 텍스트 데이터는 처리하기 때문에, HttpServletRequest는 파일정보만 활용할 계획이다.
Controller는 단순히 웹 클라이언트에서 들어온 요청에 해당하는 비지니스 로직을 호출하고 응답을 해주는 Dispatcher 역할만 한다.
ServiceImpl 단에서 비지니스 로직을 처리하므로, ServiceImpl 단에서 해당 파일데이터를 이용하도록 request를 추가로 넘겨주었다.

2) SampleService 인터페이스를 수정하자.

SampleService.java를 열고 다음과 같이 수정한다.

```
void insertBoard(Map<String, Object> map, HttpServletRequest request) throws Exception;
```

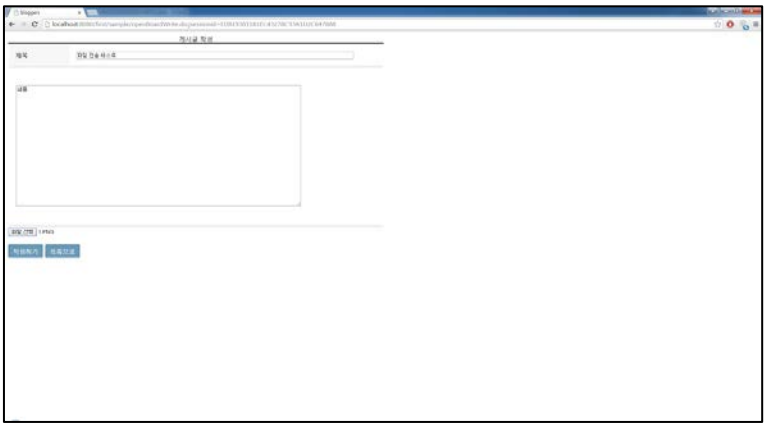
3) SampleServiceImpl.java

먼저 HttpServletRequest에 화면에서 보내준 파일이 정상적으로 전송이 되었는지 확인해 보자.
insertBoard를 다음과 같이 작성한다.

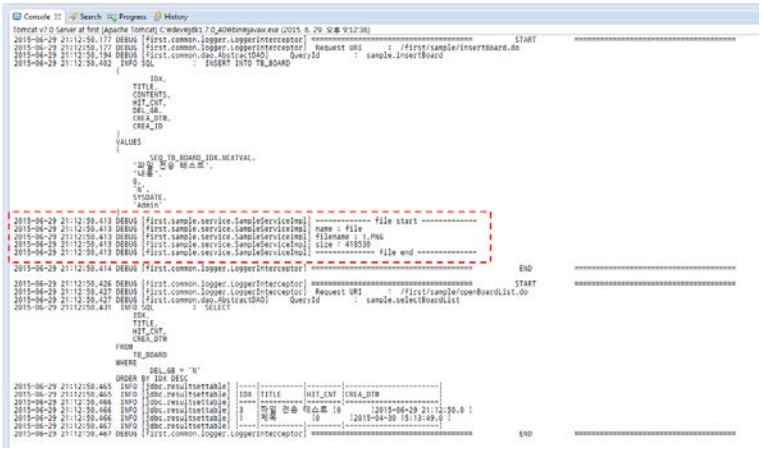
```
@Override
public void insertBoard(Map<String, Object> map, HttpServletRequest request) throws
Exception {
    sampleDAO.insertBoard(map);
```

```
5
6     MultipartHttpServletRequest multipartHttpServletRequest =
7                     (MultipartHttpServletRequest)request;
8     Iterator<String> iterator = multipartHttpServletRequest.getFileNames();
9     MultipartFile multipartFile = null;
10    while(iterator.hasNext()){
11        multipartFile = multipartHttpServletRequest.getFile(iterator.next());
12        if(multipartFile.isEmpty() == false){
13            log.debug("------------- file start -------------");
14            log.debug("name : "+multipartFile.getName());
15            log.debug("filename : "+multipartFile.getOriginalFilename());
16            log.debug("size : "+multipartFile.getSize());
17            log.debug("-------------- file end --------------₩n");
18        }
      }
  }
```

어떤 결과가 나오는지 먼저 확인을 해보자.
서버를 실행시키고 파일을 전송해보자.



위에서 보는것과 같이 1.PNG 파일을 전송하려고 한다.
파일선택을 누르고 아무 이미지를 업로드하면 된다.
그 후, 작성하기를 누른 후, 이클립스의 로그를 확인해보자.



위와 같은 로그를 확인할 수 있다.
여기서 우리가 봐야할것은 로그 중간쯤에 있는 파일 부분이다.
로그를 보면 화면에서 전송한 파일의 정보가 출력됨을 볼 수 있다.
아까 화면에서 1.PNG 라는 파일을 선택했었는데, 화면에서 선택한 파일의 이름과 크기가 출력되는 것을 볼 수 있다. 만약 이렇게 로그가 출력되면, 파일은 성공적으로 서버에 전송되었음을 알 수 있다.

이제 위의 소스에 대해서 간단히 살펴보자.
먼저 6번째 줄
MultipartHttpServletRequest multipartHttpServletRequest =
(MultipartHttpServletRequest)request; 부분이다.
위에서 첨부파일은 Multipart 형식의 데이터이며, HttpServletRequest에 담겨서 서버로 전송된다고 하였다. HttpServletRequest 자체로는 Multipart형식의 데이터를 조작하는데 어려움이 있기 때문에, MultipartHttpServletRequset 형식으로 형변환을 하였다.

그다음 8번째 줄
Iterator<String> iterator = multipartHttpServletRequest.getFileNames(); 부분이다.
이터레이터(Iterator) 패턴을 이용하여 request에 전송된 모든 name을 이용하려고 하는 부분이다. Iterator는 어떤 데이터들의 집합체에서 컬렉션(Collection)으로부터 정보를 얻어올 수 있는 인터페이스다. Iterator를 사용하면 집합체와 개별원소들을 분리시켜서 생각할 수가 있는데, 그 집합체가 어떤 클래스의 인스턴스인지 신경쓰지 않아도 되는 장점이 있다.
예를 들어 List나 배열에서는 순차적으로 데이터의 접근이 가능하지만, Map, Set 등의 클래스들은 순차적인 접근을 할수가 없다.
Map에서 데이터를 가져오려면 그 Map에 있는 키(Key)를 알아야 접근이 가능하다거나, for문 등의 반복문을 사용할 수가 없는 것을 생각하면 된다.
여기서는 Iterator를 이용하여 Map에 있는 데이터를 while문을 이용하여 순차적으로 접근하려고 한다.
아까 JSP에서 <input type="file" name="file">이라는 태그를 추가했었다. 여기서 name="file"을 봐

야한다.
JSP내에서 작성된 데이터가 서버로 전송될 때에는 태그의 name값을 키(key)로 해서 값(value)가 전송된다. 즉, request에 값이 전달될 때에도 Map과 마찬가지로 key, value 쌍의 형식으로 데이터가 저장된다. 위의 태그에서 name은 "file" 이라는 값이었고, reqeust에서 "file"이라는 키를 통해서 데이터를 가져올 수 있는데, 이 경우 우리는 "file"이라는 키를 알고 있지만, 실제로 개발을 하면, name값은 여러가지 다른 이름으로 넘어올 수 있다. 따라서 Iterator를 통해서 모든 name값을 알아서 가져오게 하면, 개발자는 name이 무엇인지를 몰라도, 쉽게 그 값을 사용할수 있다.

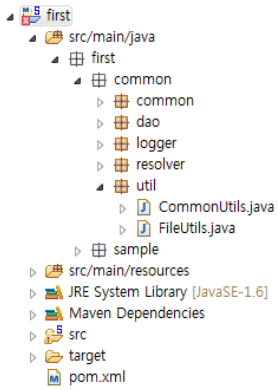
그 다음으로 10번째줄 **while(iterator.hasNext())** 부분이다.
Iterator 인터페이스의 hasNext 메서드를 통해서 Iterator에 다음 값이 있는동안 반복해서 다른 작업을 수행한다(while문).

그 다음 11번째줄
multipartFile = multipartHttpServletRequest.getFile(iterator.next()); 부분이다.
MultipartFile 객체에 request에서 파일 객체를 가져오는 부분이다. multipartHttpServletRequest의 getFile() 메서드는 request에 저장된 파일의 name을 인자로 받는다.
그런데 우리는 이 name을 Iterator를 통해서 가져온다고 이야기를 하였다. 그것이 Iterator.next() 메서드다. hasNext() 메서드는 Iterator 내에 그 다음 데이터의 존재 유무를 알려주고, next() 메서드는 Iterator 내의 데이터를 가져온 후, 커서를 다음 위치로 이동시킨다.

그 다음으로 **multipartFile.isEmpty()**를 통해서 실제 파일정보가 있는지 검사한 후에, getName(), getOriginalFilename(), getSize() 메서드 등을 통해서 파일의 정보를 출력하고 있다.

13.3.3). FIleUtils 클래스 생성

이제 위에서 설명한 내용을 바탕으로 실제로 파일을 저장하고, DB에 데이터를 저장하는 것을 살펴보자.
먼저 첨부파일의 정보를 이용하여 여러가지 조작을 할 클래스를 하나 만들자.
src/main/java 밑의 **first > common** 패키지 밑에 **util 이라는 패키지를 만든 후, FileUtils, CommonUtils 라는 클래스를 생성**하자.



먼저 **CommonUtils.java** 에 다음의 내용을 작성하자.

```
1       package first.common.util;
2
3       import java.util.UUID;
4
5       public class CommonUtils {
6
7           public static String getRandomString(){
8               return UUID.randomUUID().toString().replaceAll("-", "");
9           }
10      }
```

위에서 DB를 생성할 때 서버에 저장될 파일명은 32글자로 한다고 이야기를 했었다.
getRandomString() 메서드는 32글자의 랜덤한 문자열(숫자포함)을 만들어서 반환해주는 기능을 한다.

다음은 **FileUtils.java**에 다음의 내용을 작성하자.

```
1  package first.common.util;
2
3  import java.io.File;
4  import java.util.ArrayList;
5  import java.util.HashMap;
6  import java.util.Iterator;
7  import java.util.List;
8  import java.util.Map;
9
10import javax.servlet.http.HttpServletRequest;
11
12import org.springframework.stereotype.Component;
13import org.springframework.web.multipart.MultipartFile;
14import org.springframework.web.multipart.MultipartHttpServletRequest;
15
16@Component("fileUtils")
17public class FileUtils {
18    private static final String filePath = "C:₩₩spring₩₩dev₩₩file₩₩";
19
20    public    List<Map<String,Object>>    parseInsertFileInfo(Map<String,Object>    map,
```

```
21HttpServletRequest request) throws Exception{
22        MultipartHttpServletRequest                multipartHttpServletRequest                =
23(MultipartHttpServletRequest)request;
24        Iterator<String> iterator = multipartHttpServletRequest.getFileNames();
25
26        MultipartFile multipartFile = null;
27        String originalFileName = null;
28        String originalFileExtension = null;
29        String storedFileName = null;
30
31        List<Map<String,Object>> list = new ArrayList<Map<String,Object>>();
32        Map<String, Object> listMap = null;
33
34        String boardIdx = (String)map.get("IDX");
35
36        File file = new File(filePath);
37        if(file.exists() == false){
38            file.mkdirs();
39        }
40
41        while(iterator.hasNext()){
42            multipartFile = multipartHttpServletRequest.getFile(iterator.next());
43            if(multipartFile.isEmpty() == false){
44                originalFileName = multipartFile.getOriginalFilename();
45                originalFileExtension                                                    =
46originalFileName.substring(originalFileName.lastIndexOf("."));
47                storedFileName = CommonUtils.getRandomString() + originalFileExtension;
48
49                file = new File(filePath + storedFileName);
50                multipartFile.transferTo(file);
51
52                listMap = new HashMap<String,Object>();
53                listMap.put("BOARD_IDX", boardIdx);
54                listMap.put("ORIGINAL_FILE_NAME", originalFileName);
55                listMap.put("STORED_FILE_NAME", storedFileName);
56                listMap.put("FILE_SIZE", multipartFile.getSize());
57                list.add(listMap);
58            }
59        }
```

```
        return list;
    }
}
```

아까 위에서 설명을 했던 내용을 기반으로 파일을 특정 폴더에 저장하고 DB에 입력될 정보를 반환하도록 구성하였다.

소스를 살펴보자.
먼저 16번째 줄의 **@Component("fileUtils")** 부분이다.
@Component 어노테이션을 이용하여 이 객체의 관리를 스프링이 담당하도록 할 계획이다.

그 다음 18번째 줄 **private static final String filePath = "C:₩₩dev₩₩file₩₩;** 부분은 파일이 저장될 위치를 선언해주었다.
지금은 로컬에서만 하기 때문에 저장 위치를 소스에 명시하였는데, 나중에 이 부분은 properties를 이용하여 로컬과 서버의 저장위치를 따로따로 관리할 예정이다.

그 다음으로 31번째 줄
List<Map<String,Object>> list = new ArrayList<Map<String,Object>>(); 부분은 클라이언트에서 전송된 파일 정보를 담아서 반환해줄 List이다. 여태까지는 단 하나의 파일만 전송을 하였지만, 다중파일전송을 하도록 수정할 계획이기 때문에 미리 그에 맞도록 구성하였다.

34번째 줄은 ServiceImpl 영역에서 전달해준 map에서 신규 생성되는 게시글의 번호를 받아오도록 하였다. 이 부분의 ServiceImpl 구현은 잠시후에 살펴보자.

37~39번째 줄은 파일을 저장할 경로에 해당폴더가 없으면 폴더를 생성하도록 하였다.

다음 44~47번째 줄은 파일의 정보를 받아서 새로운 이름으로 변경하는 부분이다.
먼저 파일의 원본이름을 받아와서 해당 파일의 확장자를 알아낸 후, 아까 CommonUtils 클래스에 만들었던 getRandomString() 메서드를 이용하여 32자리의 랜덤한 파일이름을 생성하고 원본파일의 확장자를 다시 붙여주었다.

그리고 49~50번째 줄이 서버에 실제 파일을 저장하는 부분이다.
multipartFile.transferTo() 메서드를 이용하여 지정된 경로에 파일을 생성하는것을 알 수 있다.

그리고 52~57번째 줄은 위에서 만든 정보를 list에 추가하는 부분이다.
이렇게 FileUtils를 구현하였다. 이제 ServiceImpl 영역과 쿼리를 살펴보자.

13.3.4). ServieImpl 영역

먼저 SampleServiceImpl 영역을 다음과 같이 수정하자.

```
1  package first.sample.service;
2
3  import java.util.List;
4  import java.util.Map;
5
6  import javax.annotation.Resource;
7  import javax.servlet.http.HttpServletRequest;
8
9  import org.apache.log4j.Logger;
10import org.springframework.stereotype.Service;
11
12import first.common.util.FileUtils;
13import first.sample.dao.SampleDAO;
14
15@Service("sampleService")
16public class SampleServiceImpl implements SampleService{
17    Logger log = Logger.getLogger(this.getClass());
18
19    @Resource(name="fileUtils")
20    private FileUtils fileUtils;
21
22    @Resource(name="sampleDAO")
23    private SampleDAO sampleDAO;
24
25    @Override
26    public List<Map<String, Object>> selectBoardList(Map<String, Object> map) throws
27Exception {
28        return sampleDAO.selectBoardList(map);
29
30    }
31
32    @Override
33    public void insertBoard(Map<String, Object> map, HttpServletRequest request) throws
34Exception {
35        sampleDAO.insertBoard(map);
36
37        List<Map<String,Object>> list = fileUtils.parseInsertFileInfo(map, request);
38        for(int i=0, size=list.size(); i<size; i++){
```

```
39              sampleDAO.insertFile(list.get(i));
40          }
41      }
42
43      @Override
44      public Map<String, Object> selectBoardDetail(Map<String, Object> map) throws
45Exception {
46          sampleDAO.updateHitCnt(map);
47          Map<String, Object> resultMap = sampleDAO.selectBoardDetail(map);
48          return resultMap;
49      }
50
51      @Override
52      public void updateBoard(Map<String, Object> map) throws Exception{
53          sampleDAO.updateBoard(map);
54      }
55
56      @Override
57      public void deleteBoard(Map<String, Object> map) throws Exception {
58          sampleDAO.deleteBoard(map);
        }

    }
```

위의 소스는 앞서 작성해왔던 SampleServiceImpl을 수정한 것이다.
먼저 살펴봐야 하는것은 19~20번째 줄이다.
앞에서 FileUtils 클래스를 만들 때, @Component 어노테이션을 이용하여 객체의 관리를 스프링에 맡긴다고 이야기를 하였다.
따라서 클래스를 사용할때 new 를 사용하여 객체를 만드는것이 아니라, 위에서 보는것과 같이 @Resource 어노테이션을 이용하여 객체의 선언만 해주면 된다.
그렇게 되면 객체의 생성 및 정리는 스프링에서 알아서 처리를 해준다.

그 다음으로 37~39번째 줄이 FIleUtils 클래스를 이용하여 파일을 저장하고 그 데이터를 가져온 후, DB에 저장하는 부분이다.
현재 insertFile() 메서드는 아직 구현하지 않았기 때문에 에러가 날 것이다.
나중에 MyBatis에서 Batch 기능을 제공하기 때문에 저렇게 for문을 이용하여 하나씩 insert를 할 필요는 없다.
다만 여기서는 아직까지 batch설정을 하지 않았기 때문에, 임시로 하나씩 insert를 하도록 하였다.

13.3.5). DAO 영역

SampleDAO에 다음의 메서드를 추가하자.

```
1       public void insertFile(Map<String, Object> map) throws Exception{
2           insert("sample.insertFile", map);
3       }
```

그동안 작성해왔던 insert와 다른점이 없다.

13.3.6). SQL

먼저 파일정보를 저장하는 sql을 작성하자.

Sample_SQL.xml에 다음의 쿼리를 추가하자.

```
1           <insert id="insertFile" parameterType="hashmap">
2               <![CDATA[
3                   INSERT INTO TB_FILE
4                   (
5                       IDX,
6                       BOARD_IDX,
7                       ORIGINAL_FILE_NAME,
8                       STORED_FILE_NAME,
9                       FILE_SIZE,
10                      CREA_ID
11                  )
12                  VALUES
13                  (
14                      SEQ_TB_FILE_IDX.NEXTVAL,
15                      #{BOARD_IDX},
16                      #{ORIGINAL_FILE_NAME},
17                      #{STORED_FILE_NAME},
18                      #{FILE_SIZE},
19                      'Admin'
20                  )
21              ]]>
22          </insert>
```

아까 만든 TB_FILE에 데이터를 저장하는 쿼리이다. 특별한 부분은 없다.

그런데, 여기까지 작성을 하게되면 끝일 것 같은데 아직 문제점이 남아있다.

아직 해결 안된 문제는 BOARD_IDX 부분이다.

아까 FileUtils를 만들 때, 게시글의 번호를 받아오는 부분이 있었다.

String boardIdx = (String)map.get("IDX");
그래서 게시글 번호를 DB에 저장하도록 했는데, 문제는 저 IDX를 받아오는 부분이 없다는 것이다. SampleServiceImpl을 다시 한번보면 sampleDAO.insertBoard(map); 를 통해서 단순히 insert만 하기 때문에, 신규 등록된 게시글의 번호를 알 수가 없다.
이를 해결하기 위해서 Sample_SQL.xml의 insertBoard 쿼리를 다음과 같이 수정하자.

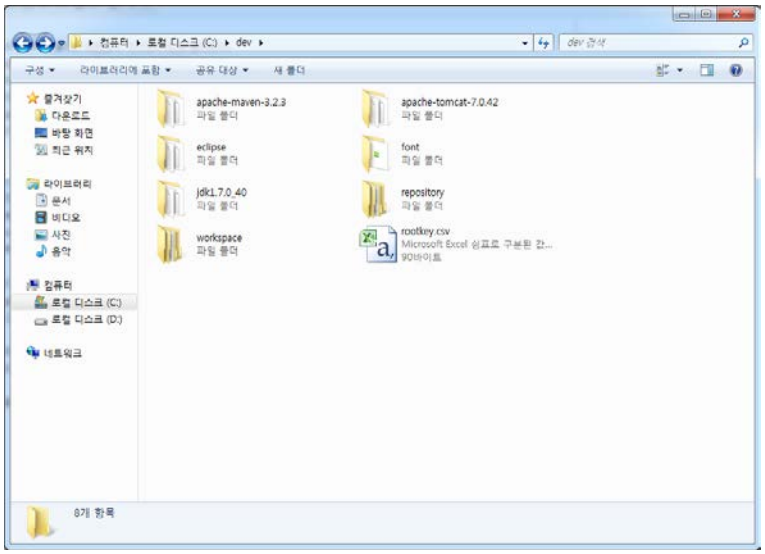
```
1 <insert id="insertBoard" parameterType="hashmap" useGeneratedKeys="true"
2 keyProperty="IDX">
3     <selectKey keyProperty="IDX" resultType="string" order="BEFORE">
4         SELECT SEQ_TB_BOARD_IDX.NEXTVAL FROM DUAL
5     </selectKey>
6     <![CDATA[
7         INSERT INTO TB_BOARD
8         (
9             IDX,
10            TITLE,
11            CONTENTS,
12            HIT_CNT,
13            DEL_GB,
14            CREA_DTM,
15            CREA_ID
16        )
17        VALUES
18        (
19            #{IDX},
20            #{TITLE},
21            #{CONTENTS},
22            0,
23            'N',
24            SYSDATE,
25            'Admin'
26        )
27    ]]>
28</insert>
```

기존글에서 작성한 insertBoard 쿼리와 비교하였을 때, 바뀐점이 무엇인지 한번 살펴보자.
먼저 <insert> 태그에서 useGeneratedKeys, keyProperty라는 속성이 추가되었다.
먼저 useGeneratedKeys는 MyBatis에서 제공하는 기능으로 DB에 입력(insert, update)시 데이터베이스에서 자동적으로 증가되는 키를 받는 JDBC의 getGeneratedKeys() 메서드를 사용하도록 하는

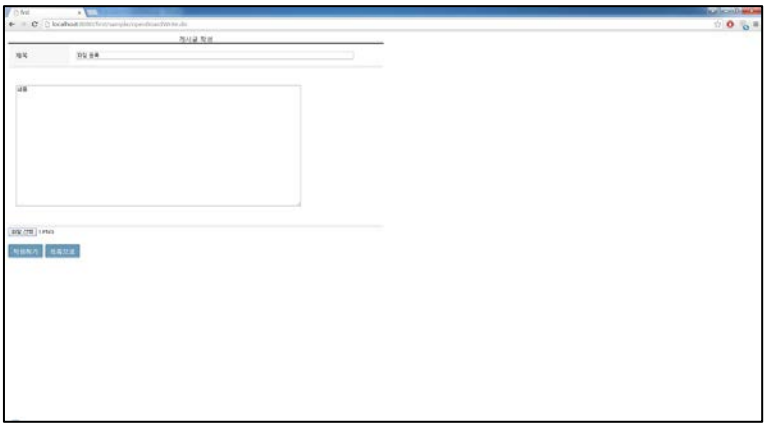
것이다. 여기서 자동적으로 증가가 되는 키는 MySql이나 MSSql의 Auto Increment가 설정된 컬럼을 의미한다.
그 다음으로 keyProperty는 getGeneratedKeys() 메서드나 insert구문의 selectKey에 의해 선택된 키를 셋팅하는 속성이다.
사실 MySql이나 MsSQL처럼 자동생성키 컬럼을 지원하는 DB에서는 이 두가지만 사용하면 되지만, Oracle과 같이 자동생성키 기능이 없는 경우는 추가적인 작업을 해야 한다.
그것이 <selectkey> 부분이다.
먼저 쿼리를 보면 기존에 insert문에서 바로 사용되던 시퀀스를 따로 빼서 값을 사용하는 것을 볼 수 있다. 그리고 기존의 insert 구문에서는 시퀀스를 바로 사용하는것이 아니라 #{IDX}라고 변수형태로 바뀐것을 볼 수 있다.
<selectKey> 구문을 통해서 다음 게시글 번호를 가져온 후, 그 값은 파라미터에 다시 저장된다.
우리가 DAO에서 쿼리를 호출할 때, insert("sample.insertBoard", map); 이렇게 호출을 했었는데, 여기서 map이 insert를 할때 사용되는 파라미터인데 이 map에 selectKey로 가져온 IDX값이 다시 입력이 되도록 되어있다.

이렇게 되면 insert를 할때 먼저 SELECT SEQ_TB_BOARD_IDX.NEXTVAL FROM DUAL 쿼리가 먼저 실행되고 (order 속성이 BEFORE로 되어있기 때문에) 그 값이 MAP에 IDX라는 키로 저장된다. (keyProperty="KEY")
그 후, insert 쿼리가 실행이 되고 IDX 값이 담긴 map이 그대로 ServiceImpl 영역까지 전달이 된다.
즉, insert를 수행하고 난 후에는 파라미터였던 map에 IDX 값이 담겨있게 된다.
그렇기 때문에 FileUtils 클래스에서 방금 insert가 되었던 IDX값을 사용할 수 있다.

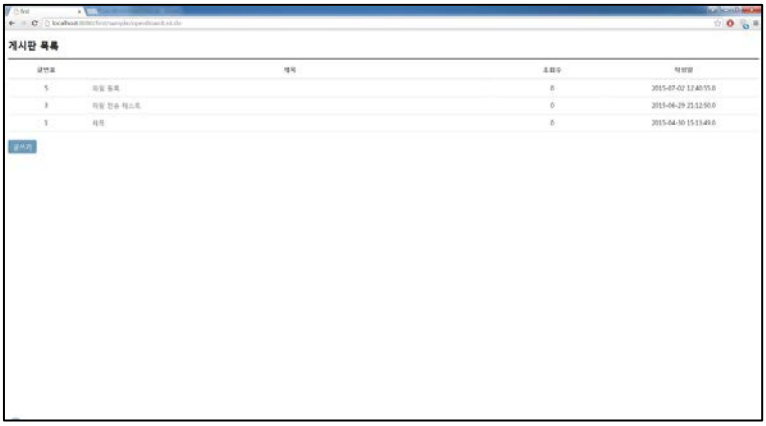
이제 서버를 실행시키고 제대로 동작을 하는지 확인하자.
먼저 C:₩₩spring₩₩dev 폴더에 file 이라는 폴더가 없는것부터 확인하자.



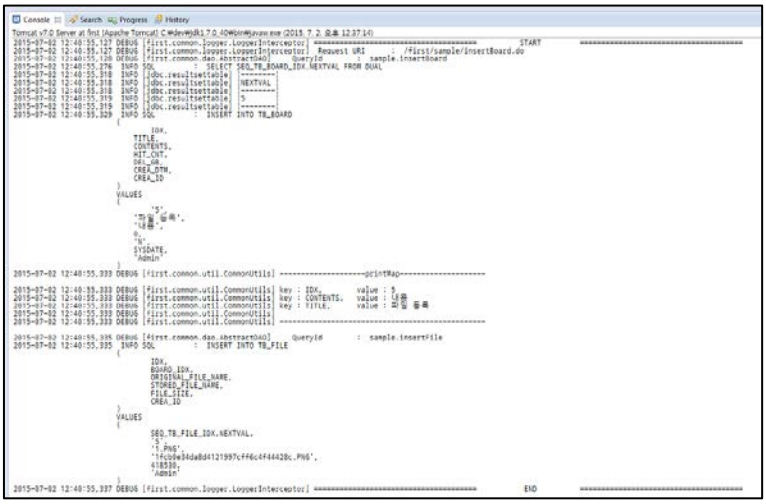
서버를 실행시키고 게시글 작성화면에서 제목, 내용, 파일을 등록하자.



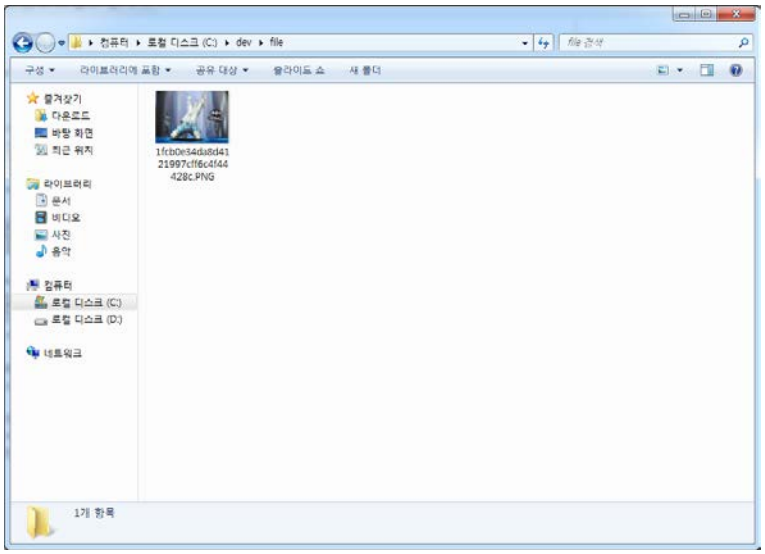
이 후 작성하기 버튼을 누르고 파일이 제대로 전송된 후 폴더에 파일이 생성되고 DB에도 정확히 데이터가 저장되었는지 확인해보자.



먼저 결과 화면부터 보자. 게시글이 등록이 되면서 방금 작성한 파일 등록이라는 글이 정상적으로 저장되었음을 확인할 수 있다.
그럼 이제 세부적으로 하나씩 살펴보자. 먼저 이클립스의 로그부터 살펴보자.

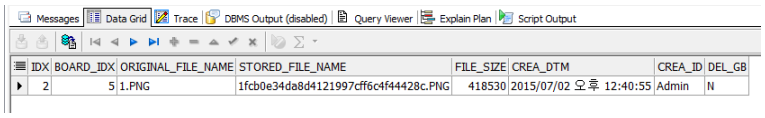


로그를 보면 insertBoard.do를 호출했을 때, 3개의 쿼리가 실행된 것을 볼 수 있다.
첫번째로 SEQ_TB_BOARD_IDX 시퀀스를 호출한 후, insertBoard 쿼리가 호출된 것을 볼 수 있다.
이렇게 게시글이 저장이 된 후, insertFile 쿼리가 실행되었다.
여기서 쿼리를 잘 살펴보자.
위에서 게시글 신규로 생성된 게시글 번호 5가 BOARD_IDX라는 값으로 정확히 들어온 것을 볼 수 있다. 그 다음으로 원본 파일명은 "1.PNG" 이고 실제로 서버에 저장될 때는 "1fcb0e34da8d4121997cff6c4f44428c.PNG" 라는 이름으로 저장되는 것을 볼 수 있다.
그럼 정말로 파일이 저장되었는지 확인해보자.



이미지에서 볼 수 있듯이 c:₩₩spring₩₩dev 폴더밑에 file 이라는 폴더가 생성된 후, 화면에서 전송한 이미지가 "1fcb0e34da8d4121997cff6c4f44428c.PNG" 라는 이름으로 변경되어 저장된 것을 확인할 수 있다.

이제 마지막으로 DB를 확인해보자.


Messages | Data Grid | Trace | DBMS Output (disabled) | Query Viewer | Explain Plan | Script Output

IDX	BOARD_IDX	ORIGINAL_FILE_NAME	STORED_FILE_NAME	FILE_SIZE	CREA_DTM	CREA_ID	DEL_GB
2	5	1.PNG	1fcb0e34da8d4121997cff6c4f44428c.PNG	418530	2015/07/02 오후 12:40:55	Admin	N

TB_FILE 테이블을 조회한 결과이다.
파일번호, 게시글 번호, 파일명 등이 모두 정상적으로 저장된것을 확인할 수 있다.
이것으로 단일 파일 전송은 완료되었다.

first5.zip

(14) 파일 다운로드

14.1. 첨부파일 보여주기

지난글에서는 게시판에 첨부파일을 등록하는 기능을 작성했었다. 이제 해당 게시글에서 첨부파일을 보여주는것을 먼저 시작하자.

14.1.1). SQL

다음의 쿼리를 **Sample_SQL.xml** 파일에 작성하자.

```
<select id="selectFileList" parameterType="hashmap" resultType="hashmap">
    <![CDATA[
        SELECT
            IDX,
            ORIGINAL_FILE_NAME,
            ROUND(FILE_SIZE/1024,1) AS FILE_SIZE
        FROM
            TB_FILE
        WHERE
            BOARD_IDX = #{IDX}
            AND DEL_GB = 'N'
    ]]>
</select>
```

선택된 게시글 번호에 해당하는 첨부파일의 목록을 조회하는 쿼리이다.
첨부파일의 크기를 Kb 단위로 보여주기 위해서 ROUND 함수를 사용하였다.

14.1.2). Java

먼저 기존 소스를 수정할 SampleController와 SampleServiceImp을 살펴보자.

1) SampleController

SampleController.java 파일에서 게시글의 상세정보를 가져오는 openBoardDetail 부분을 다음과 같이 변경하자.

```
@RequestMapping(value="/sample/openBoardDetail.do")
public ModelAndView openBoardDetail(CommandMap commandMap) throws Exception{
    ModelAndView mv = new ModelAndView("/sample/boardDetail");

    Map<String,Object> map = sampleService.selectBoardDetail(commandMap.getMap());
```

```
6       mv.addObject("map", map.get("map"));
7       mv.addObject("list", map.get("list"));
8
9       return mv;
10}
```

살펴봐야 할것은 6,7번째 줄이다.
기존에는 sampleService.selectBoardDetail()의 리턴값을 그대로 map이라는 이름으로 바로 화면으로 전송하였는데, 이번에는 map에서 2가지를 가져온 후, 각각 mv에 넣어주는 것을 볼 수 있다.
6번째 줄의 map.get("map")은 기존의 게시글 상세정보이다.
7번째 줄의 map.get("list")는 첨부파일의 목록을 가지고 있다. 게시글 상세정보와 첨부파일 정보를 각각 보내주는것을 확인하자.
그럼 다음으로 SampleServiceImpl을 수정하자.

2) SampleServiceImpl

SampleServiceImp.java 파일에서 selectBoardDetail 부분을 다음과 같이 변경하자.

```
1  @Override
2  public Map<String, Object> selectBoardDetail(Map<String, Object> map) throws
3  Exception {
4       sampleDAO.updateHitCnt(map);
5       Map<String, Object> resultMap = new HashMap<String,Object>();
6       Map<String, Object> tempMap = sampleDAO.selectBoardDetail(map);
7       resultMap.put("map", tempMap);
8
9       List<Map<String,Object>> list = sampleDAO.selectFileList(map);
10      resultMap.put("list", list);
11
12      return resultMap;
13}
```

먼저 6번째 줄 sampleDAO.selectBoardDetail()을 통해서 게시글의 상세정보를 가져온다.
그리고 그 결과값을 "map" 이라는 이름으로 resultMap에 저장한다.

그 다음으로 sampleDAO.selectFileList()를 통해서 게시글의 첨부파일 목록을 가져온다.
앞에서 각 게시글에는 하나의 첨부파일만 저장할 수 있도록 했었지만, 곧 다중 업로드가 가능하도록 수정할 계획이라서 가능한 소스의 수정을 적게하도록, 미리 첨부파일의 목록을 가져오도록 하였다.
그 다음으로 resultMap에 "list"라는 이름으로 저장하였다.

여기서 resultMap에 "map"과 "list"라는 키를 다시 한번 확인하자.
이 키는 앞의 SampleController에서 map.get("map), map.get("list") 라는 키로 사용되었다.

이제 마지막으로 selectFileList 메서드를 구현하자.

3) SampleDAO

SampleDAO.java에 다음의 소스를 작성하자.

```
@SuppressWarnings("unchecked")
public List<Map<String, Object>> selectFileList(Map<String, Object> map) throws Exception{
    return (List<Map<String, Object>>)selectList("sample.selectFileList", map);
}
```

앞서 만들었던 selectFileList 쿼리를 호출하는 역할을 한다.
쿼리 및 서버부분은 끝났으므로, JSP화면에서 첨부파일 목록을 보여주면 된다.

14.1.3). JSP

boardDetail.jsp를 다음과 같이 변경하자.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="ko">
<head>
<%@ include file="/WEB-INF/include/include-header.jspf" %>
</head>
<body>
    <table class="board_view">
        <colgroup>
            <col width="15%"/>
            <col width="35%"/>
            <col width="15%"/>
            <col width="35%"/>
        </colgroup>
        <caption>게시글 상세</caption>
        <tbody>
            <tr>
                <th scope="row">글 번호</th>
```

```
                <td>${map.IDX }</td>
                <th scope="row">조회수</th>
                <td>${map.HIT_CNT }</td>
            </tr>
            <tr>
                <th scope="row">작성자</th>
                <td>${map.CREA_ID }</td>
                <th scope="row">작성시간</th>
                <td>${map.CREA_DTM }</td>
            </tr>
            <tr>
                <th scope="row">제목</th>
                <td colspan="3">${map.TITLE }</td>
            </tr>
            <tr>
                <td colspan="4">${map.CONTENTS }</td>
            </tr>
            <tr>
                <th scope="row">첨부파일</th>
                <td colspan="3">
                    <c:forEach var="row" items="${list }">
                        <input type="hidden" id="IDX" value="${row.IDX }">
                        <a href="#this" name="file">${row.ORIGINAL_FILE_NAME }</a>
                        (${row.FILE_SIZE }kb<span style="font-family: 돋움; font-size: 9pt; line-height:
1.5;">)</span><br>
<span id="callbacknestaddio3305tistorycom847517" style="width:1px; height:1px; float:right">
<embed allowscriptaccess="always" id="bootstrapperaddio3305tistorycom847517"
src="http://addio3305.tistory.com/plugin/CallBack_bootstrapperSrc?nil_profile=tistory&nil_type=
copied_post" width="1" height="1" wmode="transparent" type="application/x-shockwave-
flash" enablecontextmenu="false"
flashvars="&callbackId=addio3305tistorycom847517&host=http://addio3305.tistory.com&embe
dCodeSrc=http%3A%2F%2Faddio3305.tistory.com%2Fplugin%2FCallBack_bootstrapper%3F%26s
rc%3Dhttp%3A%2F%2Fs1.daumcdn.net%2Fcfs.tistory%2Fresource%2F724%2Fblog%2Fplugins%2
FCallBack%2Fcallback%26id%3D84%26callbackId%3Daddio3305tistorycom847517%26destDocId
%3Dcallbacknestaddio3305tistorycom847517%26host%3Dhttp%3A%2F%2Faddio3305.tistory.co
m%26float%3Dleft" swliveconnect="true"></span>
                    </c:forEach>
                </td>
            </tr>
```

```
59          </tbody>
60      </table>
61      <br/>
62
63      <a href="#this" class="btn" id="list">목록으로</a>
64      <a href="#this" class="btn" id="update">수정하기</a>
65
66      <%@ include file="/WEB-INF/include/include-body.jspf" %>
67      <script type="text/javascript">
68          $(document).ready(function(){
69              $("#list").on("click", function(e){ //목록으로 버튼
70                  e.preventDefault();
71                  fn_openBoardList();
72              });
73
74              $("#update").on("click", function(e){ //수정하기 버튼
75                  e.preventDefault();
76                  fn_openBoardUpdate();
77              });
78
79              $("a[name='file']").on("click", function(e){ //파일 이름
80                  e.preventDefault();
81              });
82          });
83
84          function fn_openBoardList(){
85              var comSubmit = new ComSubmit();
86              comSubmit.setUrl("<c:url value='/sample/openBoardList.do' />");
87              comSubmit.submit();
88          }
89
90          function fn_openBoardUpdate(){
91              var idx = "${map.IDX}";
92              var comSubmit = new ComSubmit();
93              comSubmit.setUrl("<c:url value='/sample/openBoardUpdate.do' />");
94              comSubmit.addParam("IDX", idx);
95              comSubmit.submit();
96          }
97      </script>
```

```
98 </body>
99 </html>
```

기존과 변경된 부분은 37~43번째 줄에 첨부파일을 보여주는 부분이 추가된 것이다.
소스를 살펴보면 기존에 게시판 목록을 보여주는것과 다른점이 없다.
그리고 파일이름을 클릭했을 때 첨부파일을 다운로드 할 수 있도록 파일이름에 클릭 이벤트를 바인딩 해놓은것을 볼 수 있다.(79~81번 줄)

이제 이렇게 작성하고 소스를 실행시켜 보자.

위와같이 앞서 첨부했던 파일의 이름과 파일크기를 정상적으로 보여주는 것을 볼 수 있다.
만약 첨부파일이 없을 경우, "첨부파일이 없습니다." 와 같은 메시지를 보여주고 싶으면 <c:forEach> 구문안에 분기문을 작성해서 따로 처리해주면 된다.

그럼 이제 첨부파일을 다운로드 하는 방법을 살펴보자.

14.2. 첨부파일 다운로드

14.2.1). JSP

먼저 boardDetail.jsp에서 파일 이름을 클릭할 때, 해당 첨부파일을 다운로드 하는 주소로 이동시키도록 수정하자.
boardDetail.jsp의 스크립트 부분을 다음과 같이 수정하자.

```
1  <script type="text/javascript">
2      $(document).ready(function(){
3          $("#list").on("click", function(e){ //목록으로 버튼
4              e.preventDefault();
```

```
5               fn_openBoardList();
6           });
7
8           $("#update").on("click", function(e){ //수정하기 버튼
9               e.preventDefault();
10              fn_openBoardUpdate();
11          });
12
13          $("a[name='file']").on("click", function(e){ //파일 이름
14              e.preventDefault();
15              fn_downloadFile($(this));
16          });
17      });
18
19      function fn_openBoardList(){
20          var comSubmit = new ComSubmit();
21          comSubmit.setUrl("<c:url value='/sample/openBoardList.do' />");
22          comSubmit.submit();
23      }
24
25      function fn_openBoardUpdate(){
26          var idx = "${map.IDX}";
27          var comSubmit = new ComSubmit();
28          comSubmit.setUrl("<c:url value='/sample/openBoardUpdate.do' />");
29          comSubmit.addParam("IDX", idx);
30          comSubmit.submit();
31      }
32
33      function fn_downloadFile(obj){
34          var idx = obj.parent().find("#IDX").val();
35          var comSubmit = new ComSubmit();
36          comSubmit.setUrl("<c:url value='/common/downloadFile.do' />");
37          comSubmit.addParam("IDX", idx);
38          comSubmit.submit();
39      }
40  </script>
```

첨부파일의 파일명을 클릭하면 fn_downloadFile 이라는 함수가 실행되도록 하였다.
fn_downloadFile 함수에서는 해당 파일의 IDX값을 가져와서 /common/downloadFile.do 라는 주소

로 submit을 하는것을 알 수 있다.

14.2.2). Java 및 SQL

먼저 src/main/java 밑의 first/common 패키지 밑에 공통기능을 수행할 공통Controller, Service, DAO를 생성하자. **controller, service 패키지를 생성하고 CommonController, CommonService, CommonServiceImpl, CommonDAO를 생성한다.**
그 후, src/main/resources 밑의 mapper 폴더 밑에 공통쿼리를 작성할 Common_SQL 파일을 생성하자. **common 폴더 생성 후 Common_SQL.xml 파일을 생성한다.**

1) CommonController

```
1   package first.common.controller;
2
3   import javax.annotation.Resource;
4
5   import org.apache.log4j.Logger;
6   import org.springframework.stereotype.Controller;
7
8   import first.common.service.CommonService;
9
10  @Controller
11  public class CommonController {
12      Logger log = Logger.getLogger(this.getClass());
13
14      @Resource(name="commonService")
15      private CommonService commonService;
16  }
```

2) CommonService

```
1  package first.common.service;
2
3  public interface CommonService {
4
5  }
```

3) CommonServiceImpl

```
1  package first.common.service;
```

```
2
3   import javax.annotation.Resource;
4
5   import org.apache.log4j.Logger;
6   import org.springframework.stereotype.Service;
7
8   import first.common.dao.CommonDAO;
9
10 @Service("commonService")
11 public class CommonServiceImpl implements CommonService{
12      Logger log = Logger.getLogger(this.getClass());
13
14      @Resource(name="commonDAO")
15      private CommonDAO commonDAO;
16 }
```

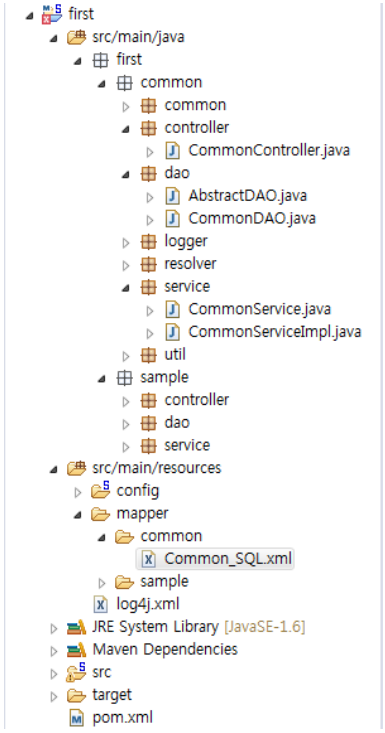
4) CommonDAO

```
1           package first.common.dao;
2
3           import org.springframework.stereotype.Repository;
4
5           @Repository("commonDAO")
6           public class CommonDAO extends AbstractDAO{
7
8           }
```

5) Common_SQL.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5  <mapper namespace="common">
6
   </mapper>
```

여기까지 완료되면 다음과 같은 구조를 가진다.



이제 하나씩 작성을 하자. 먼저 첨부파일을 다운로드 하는 방식을 살펴보자.
여기서는 다음과 같은 과정을 거쳐서 다운로드 기능을 작성할 것이다.

화면에서 특정 첨부파일 다운로드 요청 -> 서버에서 해당 첨부파일 정보 요청 -> DB에서 파일정보 조회 -> 조회된 데이터를 바탕으로 클라이언트에 다운로드가 가능하도록 데이터 전송

다음의 소스를 작성하자.

1) CommonService

```
1  Map<String, Object> selectFileInfo(Map<String, Object> map) throws Exception;
```

2) CommonServiceImpl

```
1  @Override
2  public Map<String, Object> selectFileInfo(Map<String, Object> map) throws Exception {
3      return commonDAO.selectFileInfo(map);
4  }
```

3) CommonDAO

```
1  @SuppressWarnings("unchecked")
2  public Map<String, Object> selectFileInfo(Map<String, Object> map) throws Exception{
```

```
3       return (Map<String, Object>)selectOne("common.selectFileInfo", map);
4   }
```

4) Common_SQL

```
1  <select id="selectFileInfo" parameterType="hashmap" resultType="hashmap">
2      <![CDATA[
3          SELECT
4              STORED_FILE_NAME,
5              ORIGINAL_FILE_NAME
6          FROM
7              TB_FILE
8          WHERE
9              IDX = #{IDX}
10     ]]>
11 </select>
```

쿼리에서는 저장된 파일명과 원본 파일명 두가지를 모두 조회하는것만 살펴보면 된다.

이제 Controller를 작성할 차례이다. Controller에 다음의 소스를 작성하자.

```
1   @RequestMapping(value="/common/downloadFile.do")
2   public void downloadFile(CommandMap commandMap, HttpServletResponse response)
3   throws Exception{
4       Map<String,Object> map = commonService.selectFileInfo(commandMap.getMap());
5       String storedFileName = (String)map.get("STORED_FILE_NAME");
6       String originalFileName = (String)map.get("ORIGINAL_FILE_NAME");
7
8       byte fileByte[] =
9               FileUtils.readFileToByteArray(new File("C:₩₩dev₩₩file₩₩"+storedFileName));
10
11      response.setContentType("application/octet-stream");
12      response.setContentLength(fileByte.length);
13      response.setHeader("Content-Disposition",        "attachment;        fileName=₩""        +
14  URLEncoder.encode(originalFileName,"UTF-8")+"₩";");
15      response.setHeader("Content-Transfer-Encoding", "binary");
16      response.getOutputStream().write(fileByte);
17
18      response.getOutputStream().flush();
```

```
19      response.getOutputStream().close();
20 }
```

이제 소스를 하나씩 살펴보자.
먼저, 메서드의 파라미터로 **HttpServletResponse response** 라는것이 추가되었다.
기존에 첨부파일을 업로드할 때는 HttpServletRequest request가 추가되었는데, 이번에는 response가 추가되었다.
화면에서 서버로 어떤 요청을 할때는 request가 전송되고, 반대로 서버에서 화면으로 응답을 할때는 response에 응답내용이 담기게 된다.
여기서 다운로드가 가능한 데이터 전송이라는 것은 파일정보를 response에 담아주는것을 의미한다.

4번째 줄의 commonService.selectFileInfo를 통해서 첨부파일의 정보를 가져온다.
그 후, 실제로 파일이 저장된 위치에서 해당 첨부파일을 읽어서 byte[] 형태로 변환을 해야한다.
9번째 줄이 그 역할을 수행하는데, FileUtils 클래스는 기존에 우리가 만들었던 클래스가 아니라 **org.apache.commons.io 패키지의 FileUtils 클래스**이다.
이 라이브러리를 사용하지 않더라도 파일을 읽어와서 byte[] 형식으로 변환할 수는 있지만, 그러면 상당히 복잡한 과정을 거쳐야한다.
읽어오는 파일의 위치는 "C:₩₩spring₩₩dev₩₩file₩₩" 에 저장된 파일이름을 붙이고 있다.
앞에서 파일을 저장하는 위치를 C:₩₩spring₩₩dev₩₩file₩₩로 했었던 것을 기억하자. 거기에 저장된 파일명을 붙여서 가져오도록 하였다.

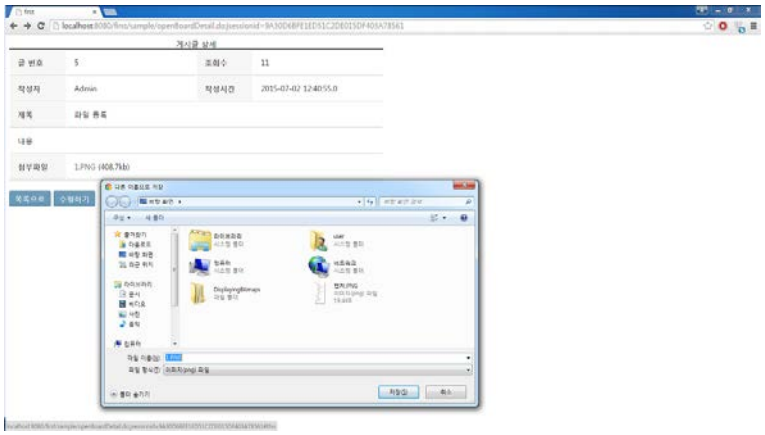
그 다음으로 중요한 부분이 11~16번째 줄이다.
읽어들인 파일정보를 화면에서 다운로드 할 수 있도록 변환을 하는 부분이다.
우리가 인터넷을 통해서 데이터를 전송하면 request나 response에는 전송할 데이터 뿐만이 아니라 여러가지 정보가 담겨있다. 그 정보를 설정해 주는 부분을 11~15번째 줄에서 하는 것이다.

여기서 꼭 확인해야 할것은 13번째 줄이다.
response.setHeader에 "Content-Disposition"이라는 속성을 지정하는 부분이다.
여기서 "attachment"로 설정하고 있다. 이는 첨부파일을 의미한다.
기존에 첨부파일을 전송할 때 패킷을 분석해보면, request의 Content-Disposition 부분은 "multipart-form/data"로 설정이 되어있다.
즉 Content-Disposition 속성을 이용하여 해당 패킷이 어떤 형식의 데이터인지 알 수 있다.

그 다음으로 fileName=₩"" + URLEncoder.encode(originalFileName,"UTF-8")+"₩";" 부분이 있다.
이것은 첨부파일의 이름을 지정해주는 역할을 수행한다.
우리가 파일을 다운로드 받으려고 하면, 파일을 저장할 위치를 선택하는 창이 뜨고 파일의 이름이 지정되어있는데, 이 부분이 그 역할을 수행한다.
여기서 중요한것은 첨부파일을 다운로드 할때, UTF-8로 인코딩 하는 것을 봐야한다.

파일명이 한글인데 UTF-8로 인코딩하지 않으면 파일명이 깨지게 된다.
인터넷에서 첨부파일을 다운받을 때, 아주 가끔 이상한 파일명으로 저장된 기억이 있을것이다. 그게 바로 UTF-8로 인코딩이 되지 않을때이다.
여기서 소스를 작성할 때, 띄어쓰기, 대소문자 구분 등도 주의해야한다.
잘 살펴보면 "attachment; fileName" 사이에 띄어쓰기가 되어있다. 저 띄어쓰기를 안해서 다운로드 기능이 안되기도 한다.
그 외에도 ₩"" 부분도 꼭 붙여줘야지 다운로드가 가능하다.
그 외에 18~19번째 줄에서 보듯이 response를 정리하고 닫아주는 것을 잊지말자.

이제 실행시켜서 파일이 다운로드가 되는지 확인해보자.



먼저, 첨부파일 이름을 클릭하면 위에서 보는것과 같이 파일을 저장할 수 있는 창이 뜬다.
여기서 파일 이름이 1.PNG로 되어있는데, 위에서 Content-Disposition 속성에 fileName값을 지정한 것이 여기에 반영이 된 것이다.
만약 위에서 fileName을 다른 이름으로 한다면 첨부파일은 그 이름으로 저장된다.
그리고 파일을 저장하면 바탕화면에 1.PNG 파일이 새로 생성된다.

이제 이클립스의 로그도 확인해보자.

위에서 보는것과 같이 선택한 파일명의 정보를 정확히 조회하는것을 볼 수 있다.

지금 여기서 설명한 방법은 파일 다운로드의 기초입니다. 실제로 여기서는 예외처리나 보안에 관련된 문제는 하나도 신경쓰지 않았습니다.
또한 파일의 정보를 가져오는것도 프로젝트별로 다를 수 있습니다.

어떤 프로젝트에서는 첨부파일을 서버에 물리적인 파일을 생성하지 않고 DB에 BLOB으로 저장한 경우도 있었고, 물리적인 파일을 파일관리 솔루션을 이용하여 관리한 경우도 있습니다.
따라서 진행하는 프로젝트에서 파일을 어떻게 관리할 것인지에 따라서 세부적인 구현은 달라져야 합니다.

 first6.zip