

# 스프링(Spring) 개발

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

## (15) 다중 파일 업로드 & 다운로드

### 15.1. 첨부파일 다중 업로드

단일 첨부파일 업로드를 수정해서 여러개의 첨부파일을 등록하도록 수정하자.

#### 15.1.1). JSP

먼저 boardWrite.jsp를 다음과 같이 수정하자.

```
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" class="wdp_90">
20          </input></td>
21          </tr>
22          <tr>
23            <td colspan="2" class="view_text">
24              <textarea rows="20" cols="100" title="내용" id="CONTENTS"
25                name="CONTENTS"></textarea>
```

```

26         </td>
27     </tr>
28 </tbody>
29 </table>
30 <div id="fileDiv">
31     <p>
32         <input type="file" id="file" name="file_0">
33         <a href="#this" class="btn" id="delete" name="delete">삭제</a>
34     </p>
35 </div>
36
37 <br/><br/>
38 <a href="#this" class="btn" id="addFile">파일 추가</a>
39 <a href="#this" class="btn" id="write">작성하기</a>
40 <a href="#this" class="btn" id="list">목록으로</a>
41 </form>
42
43 <%@ include file="/WEB-INF/include/include-body.jspf" %>
44 <script type="text/javascript">
45     var gfv_count = 1;
46
47     $(document).ready(function(){
48         $("#list").on("click", function(e){ //목록으로 버튼
49             e.preventDefault();
50             fn_openBoardList();
51         });
52
53         $("#write").on("click", function(e){ //작성하기 버튼
54             e.preventDefault();
55             fn_insertBoard();
56         });
57
58         $("#addFile").on("click", function(e){ //파일 추가 버튼
59             e.preventDefault();
60             fn_addFile();
61         });
62
63         $("a[name='delete']").on("click", function(e){ //삭제 버튼
64             e.preventDefault();

```

```

65         fn_deleteFile($(this));
66     });
67 });
68
69     function fn_openBoardList(){
70         var comSubmit = new ComSubmit();
71         comSubmit.setUrl("<c:url value='/sample/openBoardList.do' />");
72         comSubmit.submit();
73     }
74
75     function fn_insertBoard(){
76         var comSubmit = new ComSubmit("frm");
77         comSubmit.setUrl("<c:url value='/sample/insertBoard.do' />");
78         comSubmit.submit();
79     }
80
81     function fn_addFile(){
82         var str = "<p><input type='file' name='file_'+(gfv_count++)+' '><a
83 href='#this' class='btn' name='delete'>삭제</a></p>";
84         $("#fileDiv").append(str);
85         $("a[name='delete']").on("click", function(e){ //삭제 버튼
86             e.preventDefault();
87             fn_deleteFile($(this));
88         });
89     }
90
91     function fn_deleteFile(obj){
92         obj.parent().remove();
93     }
94 </script>
95 </body>
96 </html>

```

파일 추가 버튼과 삭제버튼이 추가 되었다.

먼저 파일 추가버튼을 누르면 fileDiv라는 파일 영역의 마지막에 새로운 파일 태그 및 삭제버튼을 추가하도록 하였다.

그 후 추가된 삭제버튼에도 삭제 기능을 위한 클릭이벤트를 바인딩 하였다.

여기서 살펴볼 것은 <input type='file'> 태그의 name이 동일할 경우, 서버에는 단 하나의 파일만

전송되는 문제가 발생한다. 따라서 gfv\_count 라는 전역변수를 선언하고, 태그가 추가될때마다 그 값을 1씩 증가시켜서 name값이 계속 바뀌도록 하였다.

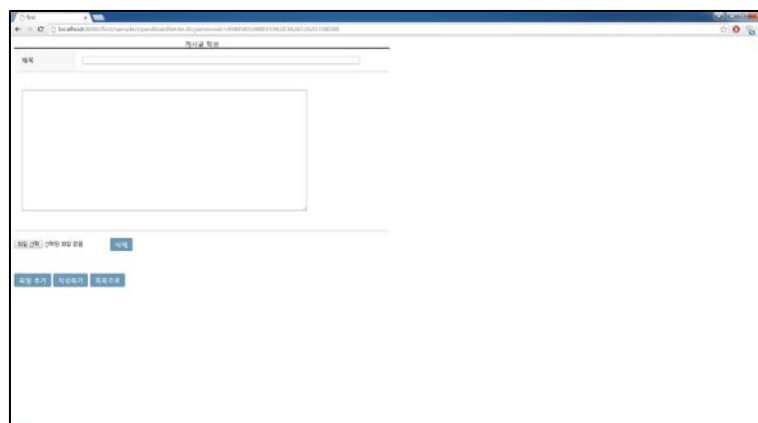
사실 여기서는 핵심적인 부분만 설명하기 위해서 간략히 작성을 하였는데, 실제로는 좀 더 복잡한 처리가 여러가지 필요하다. 예를 들어, 파일의 크기나 유효성 검사도 하지 않았으며, 추가할 수 있는 파일의 개수도 제한하지 않았다. 또한 파일의 전송에 따라서 파일의 순서가 바뀔수도 있기 때문에, 첨부파일의 순서를 지정하는 작업도 필요하다.

그 다음으로 삭제버튼이다.

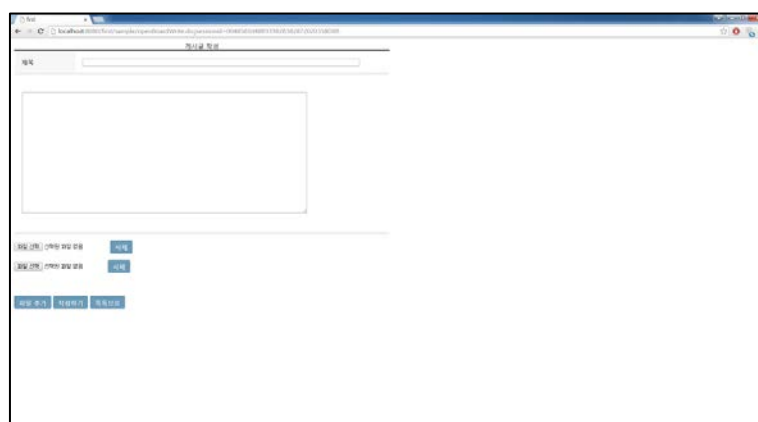
삭제버튼을 누르면 해당 버튼이 위치한 <p>태그 자체를 삭제하도록 구성하였다.

그럼 실행을 시켜보자. 서버 및 쿼리는 지난글에서 다중 파일 업로드를 고려해서 작성을 했었기 때문에, 다시 수정할 건 없다.

최초로 실행시키면 다음과 같은 화면을 볼 수 있다.



여기서 파일 추가를 클릭하면 파일을 입력할 수 있는 새로운 태그가 생성된다.



여러개의 파일을 추가하고 삭제해보고 게시글을 등록하고 확인해보자.



3개의 파일 태그가 존재하고 1.PNG, 2.PNG라는 2개의 파일을 첨부하였다.  
이 상태로 작성하기를 누르면 게시글이 저장되고 목록으로 이동이 될 것이다. 바로 상세화면에서  
파일이 제대로 올라갔는지 확인을 해 보자.



위와 같이 두개의 파일이 정상적으로 등록이 된 것을 확인할 수 있다.  
(기존의 boardDetail.jsp 에서 첨부파일의 목록을 보여주던 <c:forEach>문 사이에 <p>태그를 추가  
하였다.)

다음으로는 이클립스의 로그를 확인해보자.

```

2015-08-03 20:33:00,154 DEBUG [first.common.logger.LoggerInterceptor] ===== START
2015-08-03 20:33:00,154 DEBUG [first.common.logger.LoggerInterceptor] Request URI : /first/sample/insertBoard.do
2015-08-03 20:33:00,154 DEBUG [first.common.dao.AbstractDAO] QueryId : sample.insertBoard
2015-08-03 20:33:00,155 INFO SQL : SELECT SEQ_TB_BOARD_IDX.NEXTVAL FROM DUAL
2015-08-03 20:33:00,171 INFO [doc.resultsettable] -----
2015-08-03 20:33:00,171 INFO [doc.resultsettable] NEXTVAL :
2015-08-03 20:33:00,172 INFO [doc.resultsettable] -----
2015-08-03 20:33:00,172 INFO [doc.resultsettable] 8
2015-08-03 20:33:00,172 INFO [doc.resultsettable] -----
2015-08-03 20:33:00,180 INFO SQL : INSERT INTO TB_BOARD
(
    IDX,
    TITLE,
    CONTENTS,
    HIT_CNT,
    DEL_GB,
    CREA_DTW,
    CREA_ID
)
VALUES
(
    '8',
    'CJ중미일 특별',
    '내용',
    0,
    'N',
    SYSDATE,
    Admin
)
2015-08-03 20:33:00,186 DEBUG [first.common.dao.AbstractDAO] QueryId : sample.insertFile
2015-08-03 20:33:00,187 INFO SQL : INSERT INTO TB_FILE
(
    IDX,
    BOARD_IDX,
    ORIGINAL_FILE_NAME,
    STORED_FILE_NAME,
    FILE_SIZE,
    CREA_ID
)
VALUES
(
    SEQ_TB_FILE_IDX.NEXTVAL,
    '8',
    '1.png',
    '02662098a2d4477969f0999cc89442.PNG',
    418530,
    Admin
)
2015-08-03 20:33:00,188 DEBUG [first.common.dao.AbstractDAO] QueryId : sample.insertFile
2015-08-03 20:33:00,189 INFO SQL : INSERT INTO TB_FILE
(
    IDX,
    BOARD_IDX,
    ORIGINAL_FILE_NAME,
    STORED_FILE_NAME,
    FILE_SIZE,
    CREA_ID
)
VALUES
(
    SEQ_TB_FILE_IDX.NEXTVAL,
    '9',
    '2.png',
    '08d0d4a811f4414916974e0c4e314e0.PNG',
    78657,
    Admin
)

```

먼저 게시글을 등록한 다음, 첨부파일의 정보를 순차적으로 저장한 것을 확인할 수 있다.

## 15.2. 첨부파일 수정

프로그램을 작성하기에 앞서 첨부파일의 수정을 어떻게 처리할 것인지 고민을 해야한다.

게시글을 수정할 때 해당되는 첨부파일의 수정은 등록과 다르게 좀 복잡한 프로세스를 가진다.

다음의 경우를 생각해보자.

- 1) 게시글의 내용만 수정을 하고, 첨부파일은 수정하지 않는다.
- 2) 첨부파일을 수정할 때 기존에 등록한 파일을 변경한다.
- 3) 기존에 등록한 파일은 놔두고, 새로운 파일을 추가한다.
- 4) 기존에 등록한 파일을 모두 삭제하고, 새로운 파일을 등록한다.
- 5) 기존에 등록한 파일의 일부를 삭제하고, 새로운 파일을 등록한다.

첨부파일의 수정은 여러가지 경우의 수가 있기 때문에, 그 처리에 있어서 고민을 해야한다.

여기서는 게시글 및 해당 첨부파일을 수정할 때, 먼저 해당 게시글의 첨부파일 목록을 모두 삭제 처리한다. 여기서 삭제처리의 의미는 실제 파일을 삭제하는것이 아니라, DEL\_GB의 값을 모두 'Y'로 변경하는 것을 의미한다.

그 다음으로 FileUtils 클래스에서 파일정보를 list로 변경할 때, 기존에 첨부가 되어있던 파일의 정보와 신규로 입력된 파일 정보를 구분한다.

그 후, 기존에 첨부가 되어있던 파일은 DEL\_GB값을 다시 N으로 변경(update)하고, 신규 추가된

파일 정보는 입력(insert)한다.

이렇게 하면 DB에는 기존에 등록했던 파일 정보까지 모두 남아있지만, 삭제된 파일과 현재 사용 중인 파일을 DEL\_GB값을 이용해서 구분할 수 있다.

단, 실제파일을 삭제하는것은 아니기 때문에 서버에는 모든 파일이 남아있게 된다.

이 방식은 각각 장,단점이 존재한다.

만약 실제 파일을 삭제하는것은 HDD를 사용하기 때문에, 서버의 부담이 커지고 당연히 속도도 저하될 수 밖에 없다. 그렇지만 서버에는 꼭 필요한 파일만 남아있기 때문에 서버의 용량관리에는 좀 더 유리하다.

반대로 여기서는 파일을 삭제하지 않기 때문에 서버속도는 좀 더 빠르지만, 삭제처리된 파일을 그대로 가지고 있기 때문에 용량은 좀 더 많이 차지할 수밖에 없다.

따라서, 개발하는 시스템의 성격 및 개발 방식, 하드웨어의 상황에 따라서 유연하게 구성해야함을 잊지말자.

이제, 실제로 구현을 해 보자.

## 1). Controller

앞서 sampleService.selectBoardDetail 의 리턴값을 변경하였었다.

기존에는 selectBoardDetail을 호출하면 게시글의 내용만 변경하였는데, 첨부파일을 추가하면서 게시글의 내용과 첨부파일 목록 2가지를 반환하도록 변경하였다.

```
1  @RequestMapping(value="/sample/openBoardUpdate.do")
2  public ModelAndView openBoardUpdate(CommandMap commandMap) throws
3  Exception{
4      ModelAndView mv = new ModelAndView("/sample/boardUpdate");
5
6      Map<String,Object> map = sampleService.selectBoardDetail(commandMap.getMap());
7      mv.addObject("map", map.get("map"));
8      mv.addObject("list", map.get("list"));
9
10     return mv;
11 }
```

## 2). JSP

boardUpdate.jsp를 다음과 같이 수정하자.

```
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="35%"/>
14                 <col width="15%"/>
15                 <col width="35%"/>
16             </colgroup>
17             <caption>게시글 상세</caption>
18             <tbody>
19                 <tr>
20                     <th scope="row">글 번호</th>
21                     <td>
22                         ${map.IDX }
23                         <input type="hidden" id="IDX" name="IDX" value="${map.IDX }">
24                     </td>
25                     <th scope="row">조회수</th>
26                     <td>${map.HIT_CNT }</td>
27                 </tr>
28                 <tr>
29                     <th scope="row">작성자</th>
30                     <td>${map.CREA_ID }</td>
31                     <th scope="row">작성시간</th>
32                     <td>${map.CREA_DTM }</td>
33                 </tr>
34                 <tr>
35                     <th scope="row">제목</th>
36                     <td colspan="3">
37                         <input type="text" id="TITLE" name="TITLE" class="wdp_90"
38 value="${map.TITLE }"/>
39                     </td>
40                 </tr>
41                 <tr>

```



```

42         <td colspan="4" class="view_text">
43             <textarea rows="20" cols="100" title="내용" id="CONTENTS"
44 name="CONTENTS">${map.CONTENTS }</textarea>
45         </td>
46     </tr>
47     <tr>
48         <th scope="row">첨부파일</th>
49         <td colspan="3">
50             <div id="fileDiv">
51                 <c:forEach var="row" items="${list }" varStatus="var">
52                     <p>
53                         <input type="hidden" id="IDX" name="IDX_${var.index }"
54 value="${row.IDX }">
55                         <a href="#this" id="name_${var.index }"
56 name="name_${var.index }">${row.ORIGINAL_FILE_NAME }</a>
57                         <input type="file" id="file_${var.index }"
58 name="file_${var.index }">
59                         (${row.FILE_SIZE }kb)
60                         <a href="#this" class="btn" id="delete_${var.index }"
61 name="delete_${var.index }">삭제</a>
62                     </p>
63                 </c:forEach>
64             </div>
65         </td>
66     </tr>
67 </tbody>
68 </table>
69 </form>
70
71 <a href="#this" class="btn" id="addFile">파일 추가</a>
72 <a href="#this" class="btn" id="list">목록으로</a>
73 <a href="#this" class="btn" id="update">저장하기</a>
74 <a href="#this" class="btn" id="delete">삭제하기</a>
75
76 <%@ include file="/WEB-INF/include/include-body.jspf" %>
77 <script type="text/javascript">
78     var gfv_count = '${fn:length(list)+1}';
79     $(document).ready(function(){
80         $("#list").on("click", function(e){ //목록으로 버튼

```

```
81         e.preventDefault();
82         fn_openBoardList();
83     });
84
85     $("#update").on("click", function(e){ //저장하기 버튼
86         e.preventDefault();
87         fn_updateBoard();
88     });
89
90     $("#delete").on("click", function(e){ //삭제하기 버튼
91         e.preventDefault();
92         fn_deleteBoard();
93     });
94
95     $("#addFile").on("click", function(e){ //파일 추가 버튼
96         e.preventDefault();
97         fn_addFile();
98     });
99
100    $("a[name^='delete']").on("click", function(e){ //삭제 버튼
101        e.preventDefault();
102        fn_deleteFile($(this));
103    });
104 });
105
106 function fn_openBoardList(){
107     var comSubmit = new ComSubmit();
108     comSubmit.setUrl("<c:url value='/sample/openBoardList.do' />");
109     comSubmit.submit();
110 }
111
112 function fn_updateBoard(){
113     var comSubmit = new ComSubmit("frm");
114     comSubmit.setUrl("<c:url value='/sample/updateBoard.do' />");
115     comSubmit.submit();
116 }
117
118 function fn_deleteBoard(){
119     var comSubmit = new ComSubmit();
```

```

120     comSubmit.setUrl("<c:url value='/sample/deleteBoard.do' />");
121     comSubmit.addParam("IDX", $("#IDX").val());
122     comSubmit.submit();
123 }
124
125 function fn_addFile(){
126     var str = "<p>" +
127         "<input                type='file'                id='file_'+(gfv_count)+'"
128name='file_'+(gfv_count)+'">" +
129         "<a                href='#this'                class='btn'                id='delete_'+(gfv_count)+'"
130name='delete_'+(gfv_count)+'">삭제</a>" +
131         "</p>";
132     $("#fileDiv").append(str);
133     $("#delete_"+(gfv_count++)).on("click", function(e){ //삭제 버튼
134         e.preventDefault();
135         fn_deleteFile($(this));
136     });
137 }
138
139 function fn_deleteFile(obj){
140     obj.parent().remove();
141 }
142 </script>
143</body>
144</html>

```

먼저 실행된 화면을 확인한 후에, 하나씩 확인을 하자.

게시글 상세에서 수정하기 버튼을 눌러서 게시글 수정 화면을 보면 다음과 같다.

기존에 저장이 된 파일명과 해당 파일을 수정할 수 있는 파일 선택 버튼, 그리고 삭제버튼으로 구성하였다. 파일추가 버튼은 boardWrite.jsp에서 만든 기능과 동일하게 첨부파일을 하나 추가하도록 구성하였다.

여기서 하나 자세히 봐야하는 것은 첨부파일 목록부분에서 만든 `<input type="hidden" id="IDX" name="IDX_${var.index}">` 태그이다. 이 태그의 name 속성이 IDX\_숫자 로 구성이 되어있는 것을 기억해야 한다.

앞서 파일수정 프로세스를 다룰 때 "FileUtils 클래스에서 파일정보를 list로 변경할 때, 기존에 첨부가 되어있던 파일의 정보와 신규로 입력된 파일 정보를 구분한다." 라는 했었다.

기존에 저장된 파일에서는 해당 파일번호인 IDX 값이 존재하는데, 이를 이용해서 신규 파일정보와 아닌것을 구분하려고 한다.

### 3). Java

#### 1. Controller

SampleController.java의 updateBoard.do를 다음과 같이 변경한다.

```
1 @RequestMapping(value="/sample/updateBoard.do")
2 public ModelAndView updateBoard(CommandMap commandMap, HttpServletRequest
3 request) throws Exception{
4     ModelAndView mv = new ModelAndView("redirect:/sample/openBoardDetail.do");
5
6     sampleService.updateBoard(commandMap.getMap(), request);
7
8     mv.addObject("IDX", commandMap.get("IDX"));
9     return mv;
10 }
```

기존의 updateBoard.do에서 첨부파일 정보를 포함한 HttpServletRequest를 추가하였다.

#### 2. Service 및 ServiceImpl

먼저, SampleService.java의 updateBoard()를 다음과 같이 변경한다.

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

다음으로 중요한것은 ServiceImpl 영역이다.

먼저 SampleServiceImpl.java의 updateBoard를 다음과 같이 변경하자.

```

1  @Override
2  public void updateBoard(Map<String, Object> map, HttpServletRequest request) throws
3  Exception{
4      sampleDAO.updateBoard(map);
5      sampleDAO.deleteFileList(map);
6      List<Map<String, Object>> list = fileUtils.parseUpdateFileInfo(map, request);
7      Map<String, Object> tempMap = null;
8      for(int i=0, size=list.size(); i<size; i++){
9          tempMap = list.get(i);
10         if(tempMap.get("IS_NEW").equals("Y")){
11             sampleDAO.insertFile(tempMap);
12         }
13         else{
14             sampleDAO.updateFile(tempMap);
15         }
16     }
17 }

```

기존의 updateBoard에서는 sampleDAO.updateBoard(map) 한줄만 있었는데, 추가된 내용이 조금 있다.

일단 이렇게 작성을 하게되면

5번째 줄 sampleDAO.deleteFileList(map);

6번째 줄 fileUtils.parseUpdateFileInfo(map, request);

14번째 줄 sampleDAO.updateFile(tempMap);

에서 에러가 발생할 것이다.

일단 에러는 무시하고, 간단히 설명을 하도록 하겠다.

여기서는 위에서 첨부파일의 수정 프로세스를 염두해 두고서 생각하자.

먼저, 게시글의 내용을 수정하는 sampleDAO.updateBoard는 기존과 같다.

그 다음으로 sampleDAO.deleteFileList(map)을 호출하는데, 이는 해당 게시글에 해당하는 첨부파일을 전부 삭제처리(DEL\_GB = 'Y')를 하는 역할을 한다.

이렇게 해서 해당 게시글의 첨부파일은 전부 삭제가 된 상황이다.

그 다음으로, fileUtils의 parseUpdateFileInfo 메서드를 이용해서 request에 담겨있는 파일 정보를 list로 변환한다. 이때, 기존에 저장된 파일 중에서 삭제되지 않은 파일정보도 함께 반환할 것이다.

그 다음으로는 파일을 하나씩 입력(insert) 또는 수정(update)을 할 차례인데, 이는 list에 담긴 파일정보중에서 IS\_NEW라는 값을 이용해서 판별할 계획이다.

IS\_NEW라는 값이 "Y"인 경우는 신규 저장될 파일이라는 의미이고, "Y"가 아니면 기존에 저장되어

있던 파일이라는 의미이다.

그래서 신규저장은 sampleDAO.insertFile을 이용하여 파일정보를 저장하고, 기존에 저장된 파일정보는 다시 삭제처리를 바꿔주기만 할 계획이다.

그럼 이러한 기능을 수행할 FileUtils 클래스의 parseUpdateFileInfo 메서드를 살펴보자.

```
1 public List<Map<String, Object>> parseUpdateFileInfo(Map<String, Object> map,  
2 HttpServletRequest request) throws Exception{  
3     MultipartHttpServletRequest multipartHttpServletRequest =  
4 (MultipartHttpServletRequest)request;  
5     Iterator<String> iterator = multipartHttpServletRequest.getFileNames();  
6  
7     MultipartFile multipartFile = null;  
8     String originalFileName = null;  
9     String originalFileExtension = null;  
10    String storedFileName = null;  
11  
12    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();  
13    Map<String, Object> listMap = null;  
14  
15    String boardIdx = (String)map.get("IDX");  
16    String requestName = null;  
17    String idx = null;  
18  
19  
20    while(iterator.hasNext()){  
21        multipartFile = multipartHttpServletRequest.getFile(iterator.next());  
22        if(multipartFile.isEmpty() == false){  
23            originalFileName = multipartFile.getOriginalFilename();  
24            originalFileExtension =  
25 originalFileName.substring(originalFileName.lastIndexOf("."));  
26            storedFileName = CommonUtils.getRandomString() + originalFileExtension;  
27  
28            multipartFile.transferTo(new File(filePath + storedFileName));  
29  
30            listMap = new HashMap<String, Object>();  
31            listMap.put("IS_NEW", "Y");  
32            listMap.put("BOARD_IDX", boardIdx);  
33            listMap.put("ORIGINAL_FILE_NAME", originalFileName);
```

```

34     listMap.put("STORED_FILE_NAME", storedFileName);
35     listMap.put("FILE_SIZE", multipartFile.getSize());
36     list.add(listMap);
37 }
38 else{
39     requestName = multipartFile.getName();
40     idx = "IDX_" + requestName.substring(requestName.indexOf("_")+1);
41     if(map.containsKey(idx) == true && map.get(idx) != null){
42         listMap = new HashMap<String, Object>();
43         listMap.put("IS_NEW", "N");
44         listMap.put("FILE_IDX", map.get(idx));
45         list.add(listMap);
46     }
47 }
48 }
49 return list;
50 }

```

기존에 작성한 `parseInsertFileInfo` 메서드를 기반으로 약간 변경하였다.

먼저 `multipartFile`이 비어있지 않은 경우, 즉 첨부파일이 있는 경우는 기존과 동일하다.

첨부파일이 있다는 것은 해당 파일이 변경됨을 뜻하고 이는 새롭게 저장을 해야한다. 따라서 기존에 `parseInsertFileInfo`에서 한 것과 동일하게 파일을 저장하고 그걸 정보를 `list`에 추가하였다.

여기서 다른점은 `SampleServiceImpl` 에서 사용할 "IS\_NEW"라는 키로 "Y"라는 값을 저장하였다.

그 다음으로 봐야할 `else` 문, 즉 `multipartFile`이 비어있는 경우(`multipartFile.isEmpty() == true`)이다.

단순히 게시글을 작성할 경우에는 이는 무시하면 되는 부분이었다.

그렇지만 수정에서는 첨부파일이 없더라도 해당 파일은 저장이 될 수도 있고 아닐수도 있다는 것을 무시하면 안된다.

게시글에서 파일을 수정하지 않을 경우, 해당 `multipartFile`은 비어있다. 그렇지만 그대로 놔두면 이미 파일은 지워져있기 때문에, 최종적으로는 파일이 없다고 나오게 된다.

따라서, 파일정보가 없는 경우에는 해당 파일정보가 기존에 저장이 되어있던 내용인지 아니면 단순히 빈 파일인지 구분해야한다.

그것을 구분하는게 39, 40번째 줄이다.

먼저 39번째 줄에서는 `requestName = multipartFile.getName()`이라고 되어있는데, 이는 `html` 태그에서 `file` 태그의 `name` 값을 가져오게 된다.

아까 `jsp`에서 파일 태그를 `<input type="file" id="file_숫자" name="file_숫자">` 로 만들었던 것을 다시 한번 확인하자.

이 태그의 `name`값인 "file\_숫자" 값을 가져오는 것이 `multipartFile.getName()` 메서드이다.

이 name에서 뒤에 있는 숫자를 가져오게 되면, map에서 IDX\_숫자 값이 있는지 여부를 판별할 수 있다.

기존에 저장되어있던 파일의 경우, <input type="hidden" id="IDX" name="IDX\_숫자" value="파일번호"> 태그를 생성했었다. 그리고 신규로 생성이 된 파일의 경우, 위의 태그를 만들어주지 않았었다. 따라서, 위 태그의 값이 있을 경우가 기존에 저장된 파일임을 알 수 있다.

그래서 39번째 줄은 "IDX\_" 라는 키 값에 해당 태그의 네임에서 숫자를 가져와서 합쳐준다. 그렇게 되면 IDX\_1, IDX\_2 등의 값을 가지게 되는 것이다.

그 다음 40번째 줄에서는 이제 화면에서 넘어온 값 중에서 IDX\_숫자 키가 있는지를 확인하는 것이다.

그래서 IDX\_숫자 키가 있다면 그것은 기존에 저장되어있던 파일 정보임을 의미하는 "N" 이라는 값을 "IS\_NEW"키로 저장하게 된다.

#### **4). DAO**

이제 SampleDAO에 미완성된 기능을 추가하면 된다.

```
1 public void deleteFileList(Map<String, Object> map) throws Exception{
2     update("sample.deleteFileList", map);
3 }
4
5 public void updateFile(Map<String, Object> map) throws Exception{
6     update("sample.updateFile", map);
7 }
```

#### **5). SQL**

이제 마지막으로 쿼리를 추가하면 된다. 다음 두 개의 쿼리를 추가하자.

```
1     <update id="deleteFileList" parameterType="hashmap">
2         <![CDATA[
3             UPDATE TB_FILE
4             SET  DEL_GB = 'Y'
5             WHERE BOARD_IDX = #{IDX}
6         ]]>
7     </update>
8
9     <update id="updateFile" parameterType="hashmap">
10        <![CDATA[
11            UPDATE TB_FILE
12            SET  DEL_GB = 'N'
```



```

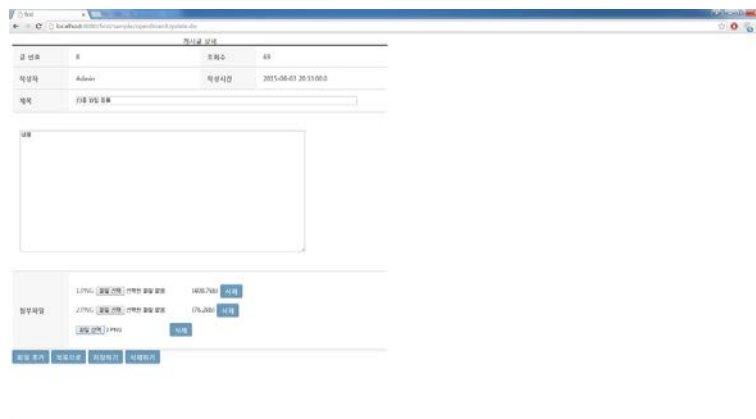
13         WHERE   IDX = #{FILE_IDX}
14     ]]>
15 </update>

```

## 6). 실행 및 결과 확인

이제 서버를 실행시키고 결과를 확인해 볼 차례이다. 몇가지 상황을 가정하고 정확히 동작하는지 확인해보자.

먼저 새로운 파일을 추가하는 경우이다.



The screenshot shows a web browser window with a URL starting with 'localhost:8080'. The page title is '게시글 작성' (Write Post). The form contains the following fields:

- 글 번호 (Post Number): 89
- 작성일 (Write Date): 2015-08-01 20:11:00
- 제목 (Title): 다중 파일 등록 (Multiple File Upload)
- 내용 (Content): (Empty text area)
- 첨가파일 (Attachments): (Empty list)

At the bottom, there are buttons for '등록완료' (Complete Registration), '등록취소' (Cancel Registration), '등록확인' (Check Registration), and '등록취소' (Cancel Registration).

실행화면에서 볼 수 있듯이 기존의 게시글에서 3.PNG라는 파일을 새롭게 추가하였다. 이제 저장하기를 눌러서 결과를 확인해 보자.



The screenshot shows the same web browser window, but the '첨가파일' (Attachments) section now displays the following information:

- 첨가파일명 (Attachment Name): 3.PNG (1838.75K)
- 첨가파일명 (Attachment Name): 3.PNG (176.20K)
- 첨가파일명 (Attachment Name): 3.PNG (186.85K)

At the bottom, there are buttons for '등록완료' (Complete Registration) and '등록확인' (Check Registration).

위에서 확인할 수 있듯이 3개의 파일이 정상적으로 저장된 것을 확인할 수 있다. 그럼 다음으로 이클립스의 로그를 살펴보자.

```

2015-08-04 21:00:40.767 DEBUG [first.common.logger.LoggerInterceptor] ===== START =====
2015-08-04 21:00:40.763 DEBUG [first.common.logger.LoggerInterceptor] Request URI : /first/sample/updateBoard.do
2015-08-04 21:00:40.763 DEBUG [first.common.dao.AbstractDao] QueryId : sample.updateBoard
2015-08-04 21:00:40.764 INFO SQL
SET
TITLE = '다들 다들 놀라'
WHERE
IDX = '8'
2015-08-04 21:00:40.765 INFO SQL
UPDATE TB_FILE SET
DEL_GB = 'Y'
WHERE
BOARD_IDX = '8'
2015-08-04 21:00:40.769 DEBUG [first.common.dao.AbstractDao] QueryId : sample.updateFile
2015-08-04 21:00:40.769 INFO SQL
UPDATE TB_FILE SET
DEL_GB = 'N'
WHERE
IDX = '6'
2015-08-04 21:00:40.770 DEBUG [first.common.dao.AbstractDao] QueryId : sample.updateFile
2015-08-04 21:00:40.771 INFO SQL
UPDATE TB_FILE SET
DEL_GB = 'N'
WHERE
IDX = '7'
2015-08-04 21:00:40.771 DEBUG [first.common.dao.AbstractDao] QueryId : sample.insertFile
2015-08-04 21:00:40.772 INFO SQL
INSERT INTO TB_FILE
(
IDX,
BOARD_IDX,
ORIGINAL_FILE_NAME,
STORED_FILE_NAME,
FILE_TITLE,
CREA_TS
)
VALUES
(
SEQ_TB_FILE.IDX.NEXTVAL,
8,
'3.png',
'd0e7ec111e4130aa3008a04038af3.png',
'Admin',
Admin
)
2015-08-04 21:00:40.773 DEBUG [first.common.logger.LoggerInterceptor] ===== END =====

```

게시글을 수정하기 위해서 여러개의 쿼리가 실행된 것을 확인할 수 있다.

먼저 sample.updateBoard 쿼리가 수행되면서 게시글 정보가 변경된 것을 확인할 수 있다.

그 다음으로 sample.deleteFileList 쿼리를 이용하여 해당 게시글의 모든 파일을 삭제처리 하였다.

그 다음으로 2번의 sample.updateFile 쿼리와 1번의 sample.insertFile 쿼리가 실행된 것을 확인할 수 있다.

기존에 저장되어 있던 1.PNG, 2.PNG 파일의 IDX는 각각 6,7번 이었다. 따라서 두 개의 파일은 삭제여부만 다시 바뀌주면 되는 정보들이었고, DEL\_GB = 'N' 으로 바뀌는것을 볼 수 있다.

그 후, 새롭게 추가한 3.PNG는 기존에 없었던 파일이었기 때문에 신규로 정보가 저장된 것을 확인할 수 있다.

이렇게 해서 3개의 파일이 정상적으로 저장이 되었다.

그럼 다른 상황을 살펴보자.



이번에는 2.PNG 파일을 4.PNG 파일로 변경하고, 빈 파일 태그를 하나 추가하였다.

이 상태로 저장하기를 누른 후 결과를 확인해보자.



정상적으로 1.PNG, 3.PNG, 4.PNG 파일이 저장된 것을 볼 수 있다.  
마지막으로 파일의 변경 및 삭제가 동시에 일어나는 상황을 살펴보자.



이번에는 1.PNG 파일을 삭제하고, 3.PNG 파일을 2.PNG 파일로 변경, 그리고 빈 태그 하나와 5.PNG 파일을 첨부하였다.  
이제 저장을 해보자.



결과는 정상적으로 2.PNG, 4.PNG, 5.PNG 3개가 저장된 것을 볼 수 있다.  
여기까지 하고 이클립스의 로그를 확인하는 것은 생략하고 이번에는 DB를 잠시 살펴보자.

	IDX	BOARD_IDX	ORIGINAL_FILE_NAME	STORED_FILE_NAME	FILE_SIZE	CREA_DTM	CREA_ID	DEL_GB
▶	16	8	5.PNG	7d5e74cf3e824db9bbace6df2bc4a2b.PNG	754575	2015/08/04 오후 9:11:32	Admin	N
	15	8	2.PNG	d2c48df3a2ab4176a66403adedadffb2.PNG	78057	2015/08/04 오후 9:11:32	Admin	N
	14	8	4.PNG	5f286cf184c54138b6e8fee36b52cdfb.PNG	148833	2015/08/04 오후 9:08:35	Admin	N
	13	8	3.PNG	dde1eec1111e4150aa3008a0b4658af3.PNG	88461	2015/08/04 오후 9:00:40	Admin	Y
	7	8	2.PNG	08d08d4a811f4414916974e0c4e314e0.PNG	78057	2015/08/03 오후 8:33:00	Admin	Y
	6	8	1.PNG	b2b682090a2d4477969fd0993cc09442.PNG	418530	2015/08/03 오후 8:33:00	Admin	Y

여기서 보면 그동안 수정을 했던 파일의 내역이 모두 남아있는 것을 확인할 수 있다. 그리고 현재 사용중인 파일은 DEL\_GB의 값이 N인 파일만 사용중임을 알 수 있다.

이렇게 하나의 테이블에서 파일의 변경내역을 관리를 할 수도 있지만, 로그파일 또는 로그 DB등을 이용하여 별도로 관리할 수도 있다.

그것은 해당 프로젝트마다 다르기 때문에, 프로젝트의 성격에 맞춰서 개발을 하면 된다.

 first7.zip