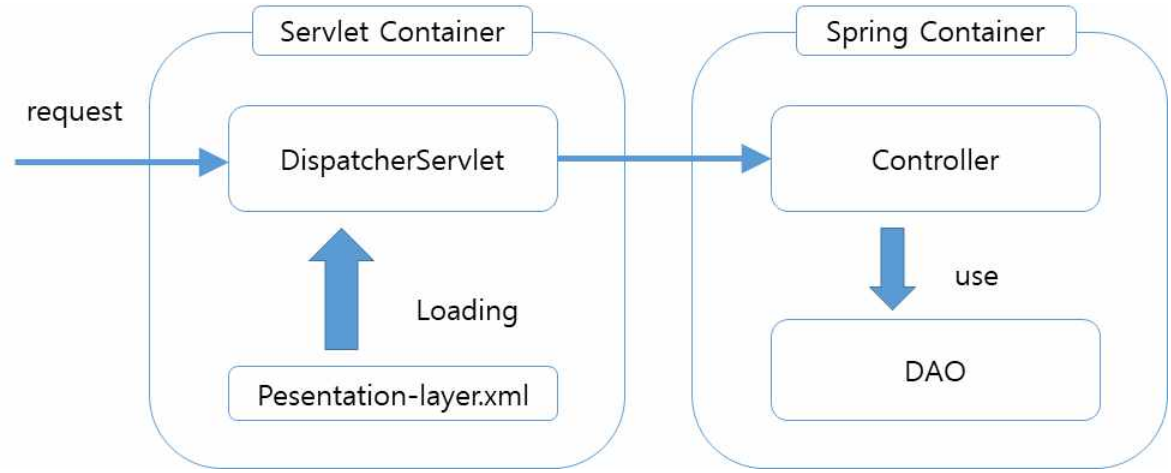
제 13 강 스프링 MVC - Spring MVC 활용

1. 프리젠테이션 레이어와 비즈니스 레이어 통합

▢ 지금까지 Spring MVC를 기반으로 개발한 게시판의 구조와 실행 순서를 살펴보면 아래 그림과 같다.



▢ 위의 구조에서 눈여겨 볼 것은 Controller의 모든 메서드가 사용자의 요청을 처리할 때 DAO 객체를 직접 이용하고 있다는 점이다. 물론 이렇게 한다고 해서 프로그램이 실행되지 않거나 심각한 문제가 발생하는 것은 아니다. 그러나 Controller가 DAO 객체를 직접 이용해서는 안 되며 반드시 비즈니스 컴포넌트를 이용해야 한다.

▢ 비즈니스 컴포넌트 사용

: Controller는 비즈니스 컴포넌트를 이용해서 사용자의 요청을 처리해야 하며 이때 컴포넌트가 제공하는 Service 인터페이스를 이용해야 한다. 지금처럼 Controller가 직접 DAO 객체의 메서드를 호출하도록 하면 안 되며 이는 여러 가지 문제를 일으킨다.

① DAO 클래스 교체하기

▢ Controller 메서드에서 DAO의 메서드를 호출하면 안 되는 첫 번째 이유는 유지보수 과정에서 DAO 클래스를 다른 클래스로 쉽게 교체하기 위해서이다. 현재 우리는 BoardController의 모든 메서드가 BoardDAO 객체를 매개변수로 받아서 DB연동을 처리하고 있다. 만약 DAO 클래스를 앞에서 우리가 작성했던 BoardDAOSpring으로 변경하거나 앞으로 추가할 다른 DAO 클래스로 변경하고자 한다면 BoardController의 모든 메서드를 수정해야 한다.

▢ 이렇게 비즈니스 컴포넌트가 변경되거나 새로운 요소가 추가될 때마다 이를 사용하는 Controller의 소스코드를 매번 수정한다면 유지보수는 어려울 수밖에 없다.

▢ 비즈니스 컴포넌트가 수정되더라도 이를 사용하는 Controller는 수정되지 않아도 되게 하려면 어떻게 해야 할까? 비즈니스 컴포넌트 입장에서 자신을 사용해주는 클라이언트는 Controller이다.

▢ 클라이언트가 인터페이스를 통해 비즈니스 컴포넌트를 이용하면 컴포넌트의 구현 클래스를 수정하거나 다른 클래스로 대체해도 이를 사용하는 클라이언트는 수정하지 않아도 된다. 이것이 다형성의 장점이자 객체지향 언어의 중요한 특성이다.

■ BoardController의 모든 메서드를 BoardService 컴포넌트의 인터페이스를 이용하도록 수정하자.

```
1  package tommy.spring.web.board;
2  import java.util.HashMap;
3  import java.util.Map;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.stereotype.Controller;
6  import org.springframework.ui.Model;
7  import org.springframework.web.bind.annotation.ModelAttribute;
8  import org.springframework.web.bind.annotation.RequestMapping;
9  import org.springframework.web.bind.annotation.SessionAttributes;
10 @Controller
11 @SessionAttributes("board")
12 public class BoardController {
13         @Autowired
14         private BoardService boardService;
15         @ModelAttribute("conditionMap")
16         public Map<String, String> searchConditionMap() {
17                 Map<String, String> conditionMap = new HashMap<String, String>();
18                 conditionMap.put("제목", "TITLE");
19                 conditionMap.put("내용", "CONTENT");
20                 return conditionMap;
21         }
22         @RequestMapping("/insertBoard.do")
23         public String insertBoard(BoardVO vo) {
24                 System.out.println("글 등록 처리");
25                 boardService.insertBoard(vo);
26                 return "getBoardList.do";
27         }
28         @RequestMapping("/updateBoard.do")
29         public String updateBoard(@ModelAttribute("board") BoardVO vo) {
30                 System.out.println("글 수정 기능 처리");
31                 boardService.updateBoard(vo);
32                 return "getBoardList.do";
33         }
34         @RequestMapping("/deleteBoard.do")
35         public String deleteBoard(BoardVO vo) {
36                 System.out.println("글 삭제 처리");
37                 boardService.deleteBoard(vo);
38                 return "getBoardList.do";
39         }
40         @RequestMapping("/getBoard.do")
41         public String getBoard(BoardVO vo, Model model) {
42                 System.out.println("글 상세 보기 처리");
43                 model.addAttribute("board", boardService.getBoard(vo));
44                 return "getBoard.jsp";
45         }
46         @RequestMapping("/getBoardList.do")
47         public String getBoardList(BoardVO vo, Model model) {
```

```
48                      System.out.println("글 목록 검색 처리");
49                      System.out.println("검색 조건 : " + vo.getSearchCondition());
50                      System.out.println("검색 단어 : " + vo.getSearchKeyword());
51                      model.addAttribute("boardList", boardService.getBoardList(vo));
52                      return "getBoardList.jsp";
53              }
54      }
```

▢ 위와 같이 수정하면 BoardServiceImpl 클래스의 멤버변수로 선언된 BoardDAO를 다른 DAO 클래스로 얼마든지 변경할 수 있다.

■ BoardServiceImpl 클래스를 열어서 기존의 BoardDAOSpring 타입의 멤버변수를 다음과 같이 수정하자.

```
1   package tommy.spring.web.board.impl;
2   import java.util.List;
3   import org.springframework.beans.factory.annotation.Autowired;
4   import org.springframework.stereotype.Service;
5   import tommy.spring.web.board.BoardService;
6   import tommy.spring.web.board.BoardVO;
7   @Service("boardService")
8   public class BoardServiceImpl implements BoardService {
9           @Autowired
10          private BoardDAO boardDAO;
11  <!-- 하단 부분 생략 -->
```

▢ 여기서 중요한 점은 이렇게 BoardServiceImpl 클래스에서 사용하는 DAO 클래스가 변경되어도 클라이언트에 해당하는 BoardController는 수정할 필요가 없다는 점이다.

② AOP 설정 적용하기

▢ Controller 메서드에서 DAO의 메서드를 호출하면 안 되는 두 번째 이유는 AOP 적용 때문이다. 횡단관심에 해당하는 어드바이스가 동작하려면 반드시 Service 구현 클래스의 비즈니스 메서드가 실행되어야 한다.

▢ 아래는 앞에서(4강 참조) 우리가 설정했던 포인트 컷이다.

Annotation 설정	@Aspect
	public class PointcutCommon {
	@Pointcut("execution(* tommy.spring.web..*impl.*(..))“
	public void allPointcut(){}
	}

XML 설정	<aop:pointcut id="allPointcut" expression="execution(* tommy.spring.web..*impl.*(..)) />

▢ 위 내용을 보면 우리가 포인트 컷을 설정할 때 DAO 클래스의 메서드에 지정한 것이 아니라 Service 구현 클래스의 메서드에 설정했다는 것이다. 따라서 모든 어드바이스는 비즈니스 클래스

의 메서드가 호출될 때 동작한다.

▢ 그러면 어떻게 해야 AOP 설정이 처리되어 어드바이스를 동작할 수 있을까?
대부분의 비즈니스 컴포넌트는 인터페이스를 가지고 있으며 이 인터페이스만 컴포넌트를 사용하는 클라이언트에 노출된다. 따라서 비즈니스 컴포넌트를 사용하는 클라이언트는 인터페이스의 추상 메서드를 호출하여 인터페이스를 구현한 Service 구현 객체의 메서드를 실행할 수 있다.

▢ 따라서 Controller 클래스는 비즈니스 컴포넌트의 인터페이스 타입의 멤버변수를 가지고 있어야 하며 이 변수에 비즈니스 객체를 의존성 주입해야 한다. 그러면 Controller 메서드에서는 인터페이스를 통하여 비즈니스 객체의 메서드를 호출할 수 있고 결국 AOP로 설정한 어드바이스들이 동작한다.

③ 비즈니스 컴포넌트 의존성 주입하기

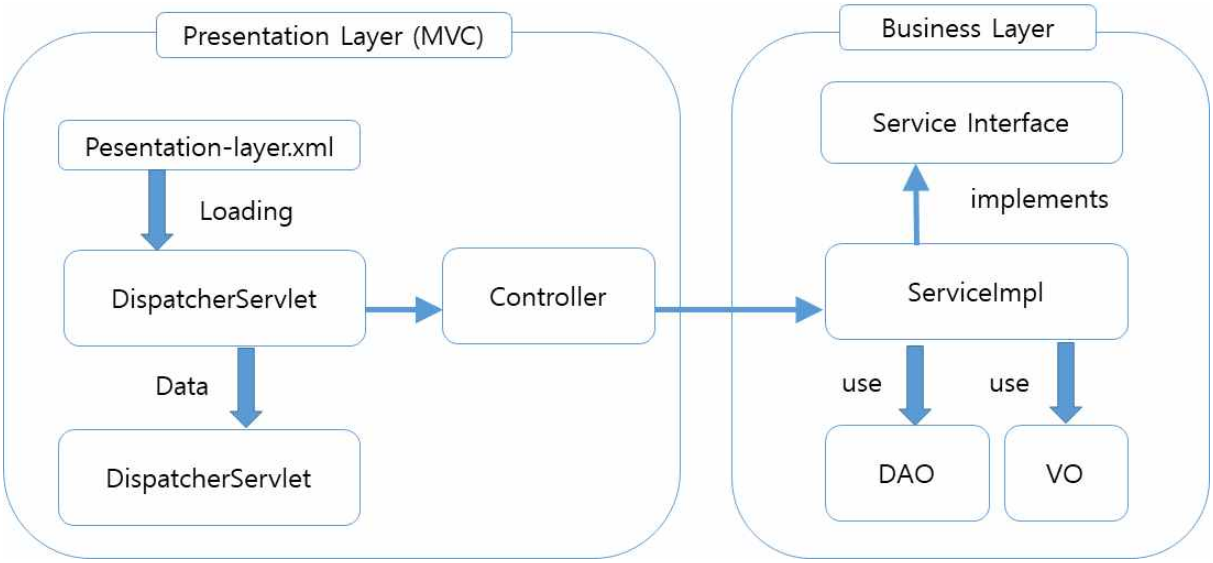
▢ 현재까지 저장하고 실행을 하면 에러 메시지가 나오는 것을 볼 수 있다.
이것은 BoardController가 @Autowired로 의존성 주입을 처리하려고 하는데 BoardService 타입의 객체가 메모리에 없어서 의존성 주입을 할 수 없다는 에러 메시지이다.

▢ @Autowired 어노테이션을 사용하려면 의존성 주입 대상이 되는 객체가 반드시 메모리에 올라가야 한다. 결론적으로 BoardController 보다 의존성 주입될 BoardServiceImpl 객체가 먼저 생성되어야 한다. 하지만 persentation-layer.xml 파일에는 Controller 객체만 컴포넌트 스캔하도록 설정했기 때문에 BoardServiceImpl 객체는 생성되지 않는다.

▢ 결국 Controller보다 의존성 주입 대상이 되는 비즈니스 컴포넌트를 먼저 생성하려면 비즈니스 컴포넌트를 먼저 생성하는 또 다른 스프링 컨테이너가 필요하다. 그리고 이 컨테이너를 Controller를 메모리에 생성하는 컨테이너보다 먼저 구동하면 된다.

④ 비즈니스 컴포넌트 로딩 : 2-Layered 아키텍처

▢ 일반적으로 프레임워크 기반의 웹 프로젝트를 보면 아래 그림처럼 두 개의 레이어로 나누어 시스템을 개발하는데 이를 2-Layered 아키텍처 스타일이라고 한다.



▢ 우리는 이미 이러한 구조로 게시판 프로그램을 개발하였다. src/main/resource 폴더에는 비즈니스 레이어에 해당하는 설정파일인 applicationContext.xml이 있으며 /WEB-INF/config 폴더에는 프리젠테이션 레이어에 해당하는 설정파일인 pesentation-layer.xml 파일이 있다.

▢ DispatcherServlet이 생성되어 presentation-layer.xml 파일을 읽어서 스프링 컨테이너를 구동하면 Controller 객체들이 메모리에 생성된다. 하지만 Controller 객체들이 생성되기 전에 누군가 먼저 src/main/resources 폴더에 있는 applicationContext.xml 파일을 읽어서 비즈니스 컴포넌트들을 메모리에 생성해야 한다. 이때 사용하는 클래스가 스프링에서 제공하는 ContextLoaderListener 이다.

⑤ ContextLoaderListener 등록

■ web.xml 파일 수정

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

6  <listener>
7           <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
8  </listener>
9  <!-- 하단 부분 생략 -->
```

▢ 이제 서버를 구동하면 FileNotFoundException이 발생하는데 이는 ContextLoaderListener가 기본적으로 /WEB-INF/applicationContext.xml 파일을 읽어서 스프링 컨테이너를 구동하기 때문이다.

▢ 따라서 src/main/resources 폴더에 있는 applicationContext.xml 파일을 WEB-INF 폴더로 복사하면 스프링 컨테이를 정상적으로 구동할 수 있다.

▢ 하지만 이렇게 하면 비즈니스 레이어에 해당하는 스프링 설정파일이 두 곳에 있어서 나중에 관리상 문제가 발생할 수 있다.

▢ 그렇다면 두 곳에 있는 XML 설정 파일 중 어떤 파일을 사용해야 할까?
일단은 유지보수 과정에서 비즈니스 컴포넌트를 수정하고 테스트를 진행하기 위해서라도 src/main/resources 폴더에 XML 파일을 위치시키는 것이 맞다. 그래야 src/test/java 폴더에 테스트 클라이언트 프로그램도 실행할 수 있기 때문이다.

▢ 그렇다면 src/main/resources 폴더에 위치한 스프링 설정파일을 어떻게 ContextLoaderListener가 읽어 들일까? 정답은 web.xml 파일에 <context-param> 설정을 추가하면 된다.

■ web.xml 파일 수정

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5       https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

6       <context-param>
7           <param-name>contextConfigLocation</param-name>
8           <param-value>classpath:applicationContext.xml</param-value>
9       </context-param>
10      <listener>
11          <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
12      </listener>
13  <!-- 하단 부분 생략 -->
```

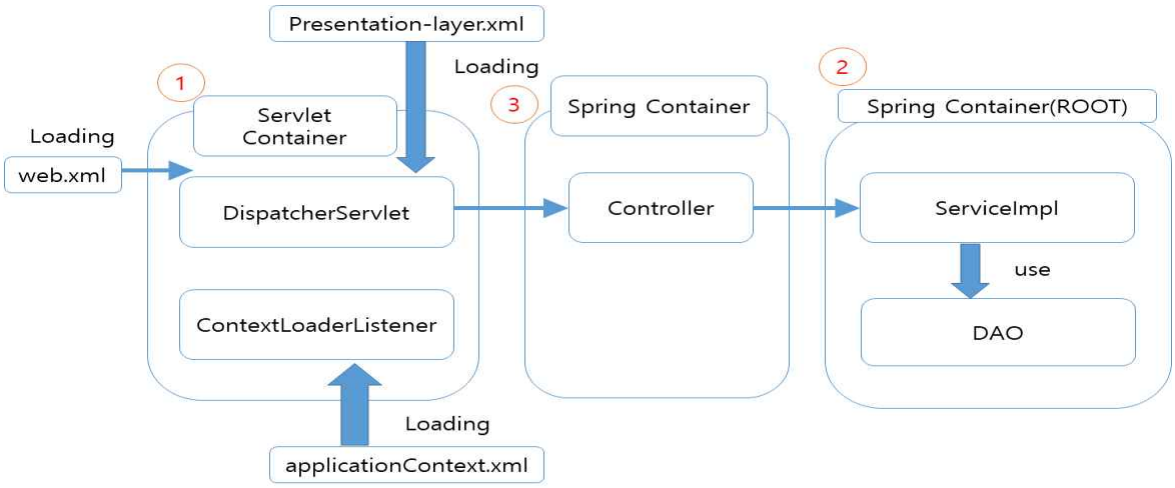
▢ 이제 서버를 구동하면 아래와 같이 ContextLoaderListener 객체가 Pre-Loading 되어 스프링 컨테이너를 구하동하고 이때 비즈니스 컴포넌트 객체들이 생성되는 것을 확인할 수 있다.

```
10월 04, 2019 11:42:09 오전 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring root WebApplicationContext
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization started
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 2413 ms
10월 04, 2019 11:42:12 오전 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
10월 04, 2019 11:42:12 오전 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["ajp-nio-8009"]을(를) 시작합니다.
10월 04, 2019 11:42:12 오전 org.apache.catalina.startup.Catalina start
정보: 서버가 [5,440] 밀리초 내에 시작되었습니다.
```

▢ 위 그림에서 보면 아직 DispatcherServlet은 생성되지 않은 상태이다. 이제 login.do를 요청하면 아래와 같이 DispatcherServlet이 생성되는 것을 확인할 수 있다.

```
정보: Initializing Spring root WebApplicationContext
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization started
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 2413 ms
10월 04, 2019 11:42:12 오전 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
10월 04, 2019 11:42:12 오전 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["ajp-nio-8009"]을(를) 시작합니다.
10월 04, 2019 11:42:12 오전 org.apache.catalina.startup.Catalina start
정보: 서버가 [5,440] 밀리초 내에 시작되었습니다.
10월 04, 2019 11:43:03 오전 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring DispatcherServlet 'action'
INFO : org.springframework.web.servlet.DispatcherServlet - Initializing Servlet 'action'
INFO : org.springframework.web.servlet.DispatcherServlet - Completed initialization in 455 ms
로그인 화면으로 이동
```

▢ 스프링 컨테이너의 관계



▢ 톰캣서버를 처음 구동하면 ① web.xml 파일을 로딩하여 서블릿 컨테이너가 구동된다. 그리고 ② 서블릿 컨테이너는 web.xml 파일에 등록된 ContextLoaderListener 객체를 생성(Pre-Loading)한다. 이때 ContextLoaderListener 객체는 src/main/resources 폴더에 있는 applicationContext.xml 파일을 로딩하여 스프링 컨테이너를 구동하는데 이를 ROOT 컨테이너라고 한다.

▢ 그리고 이때 Service 구현 클래스나 DAO 객체들이 메모리에 생성된다. 그리고 사용자가 로그인 버튼을 클릭하여 *.do 요청을 서버에 전달하면 서블릿 컨테이너는 DiapatcherServlet 객체를 생성하고 ③ DispatcherServlet 객체는 /WEB-INF/config 폴더에 있는 presentation-layer.xml 파일을 로딩하여 두 번째 스프링 컨테이너를 구동한다. 이 두 번재 스프링 컨테이너가 Controller 객체를 메모리에 생성한다.

▢ ContextLoaderListener가 생성하는 스프링 컨테이너를 ROOT 컨테이너라고 하며 쉽게 부모 컨테이너라고 생각하면 된다. 그리고 DispatcherServlet이 생성한 컨테이너는 ROOT 컨테이너가 생성한 객체를 이용하는 자식 컨테이너가 된다. 따라서 부모 컨테이너가 생성한 비즈니스 객체를 자식 컨테이너가 생성한 Controller에서 참조하여 사용할 수 있다.

2. 검색기능 추가 구현

① 검색정보 추출

▢ 먼저 검색기능을 구현하기 위한 화면이 필요한데 이것은 이미 getBoardList.jsp 파일에 구현해 놓았다. 검색 화면에서 검색 조건과 검색 키워드에 해당하는 파라미터 이름은 searchCondition과 searchKeyword 이다. searchCondition 파라미터 값은 사용자가 검색 조건을 제목으로 선택하면 “TITLE”이 내용으로 선택했으면 “CONTENT”가 설정된다. 검색 키워드는 텍스트 필드이므로 사용자가 직접 입력하면 된다.

② Command 객체 수정

▢ searchCondition과 searchKeyword를 BoardVO 객체의 멤버변수로 설정하고 getter/setter를 추가해야 하는데 이는 이미 우리가 앞에서 작업을 해 놓았다.

③ Controller 구현

■ BoardController 수정

```
 1  package tommy.spring.web.board;
 2  import java.util.HashMap;
 3  import java.util.Map;
 4  import org.springframework.beans.factory.annotation.Autowired;
 5  import org.springframework.stereotype.Controller;
 6  import org.springframework.ui.Model;
 7  import org.springframework.web.bind.annotation.ModelAttribute;
 8  import org.springframework.web.bind.annotation.RequestMapping;
 9  import org.springframework.web.bind.annotation.SessionAttributes;
10  @Controller
11  @SessionAttributes("board")
12  public class BoardController {
13          @Autowired
14          private BoardService boardService;
            <!-- 중간 부분 생략 -->
```

```
15          @RequestMapping("/getBoardList.do")
16          public String getBoardList(BoardVO vo, Model model) {
17                  System.out.println("글 목록 검색 처리");
18                  // null 체크
                    if(vo.getSearchCondition() == null) vo.setSearchCondition("TITLE");
19                  if(vo.getSearchKeyword() == null) vo.setSearchKeyword("");
20                  model.addAttribute("boardList", boardService.getBoardList(vo));
21                  return "getBoardList.jsp";
22          }
23  }
```

▢ 클라이언트가 입력한 검색 조건과 검색 키워드 정보는 BoardVO 객체에 자동으로 설정되어 전달된다. 그런데 문제는 검색 조건과 검색 키워드가 전달되지 않을 때 이다.

▢ 예를 들면 로그인에 성공 하고나서 “getBoardList.do” 요청이 전달되거나 상세화면에서 글 목록 링크를 클릭하여 “getBoardList.do” 요청이 서버에 전달되면 검색조건과 검색 키워드 정보는 전달되지 않는다.

▢ 이때 BoardVO 객체의 searchCondition과 searchKeyword 변수에는 null이 설정된다. 따라서 위 코드처럼 null값에 대한 로직을 추가하여야 한다.

④ BoardDAO 클래스 수정

■ JDBC 기반의 BoardDAO 클래스 수정

```
1   package tommy.spring.web.board.impl;
2   import java.sql.Connection;
3   import java.sql.PreparedStatement;
4   import java.sql.ResultSet;
5   import java.util.ArrayList;
6   import java.util.List;
7   import org.springframework.stereotype.Repository;
8   import tommy.spring.web.board.BoardVO;
9   import tommy.spring.web.common.JDBCUtil;
10  @Repository("boardDAO")
11  public class BoardDAO {
12          private Connection conn = null;
13          private PreparedStatement pstmt = null;
14          private ResultSet rs = null;
            private final String BOARD_INSERT = "insert into myboard(seq, title, writer, content) "
15                          + "values((select nvl(max(seq), 0)+1 from myboard), ?, ?, ?)";
            private final String BOARD_UPDATE = "update myboard set title=?, " +
16                                                  "content=? where seq=?";
17          private final String BOARD_DELETE = "delete myboard where seq=?";
18          private final String BOARD_GET = "select * from myboard where seq=?";
            private final String BOARD_LIST_T =
19                  "select * from myboard where title like '%'||?||'%' order by seq desc";
            private final String BOARD_LIST_C =
                    "select * from myboard where content like '%'||?||'%' order by seq desc";
```

```
20          <!-- 중간 부분 생략 -->
21          public List<BoardVO> getBoardList(BoardVO vo) {
22                  System.out.println("JDBC로 getBoardList() 기능 처리");
23                  List<BoardVO> boardList = new ArrayList<BoardVO>();
24                  try {
25                          conn = JDBCUtil.getConnection();
26                          if (vo.getSearchCondition().equals("TITLE")) {
27                                  pstmt = conn.prepareStatement(BOARD_LIST_T);
28                          } else if (vo.getSearchCondition().equals("CONTENT")) {
29                                  pstmt = conn.prepareStatement(BOARD_LIST_C);
30                          }
31                          pstmt.setString(1, vo.getSearchKeyword());
32                          rs = pstmt.executeQuery();
            <!-- 하단 부분 생략 -->
```

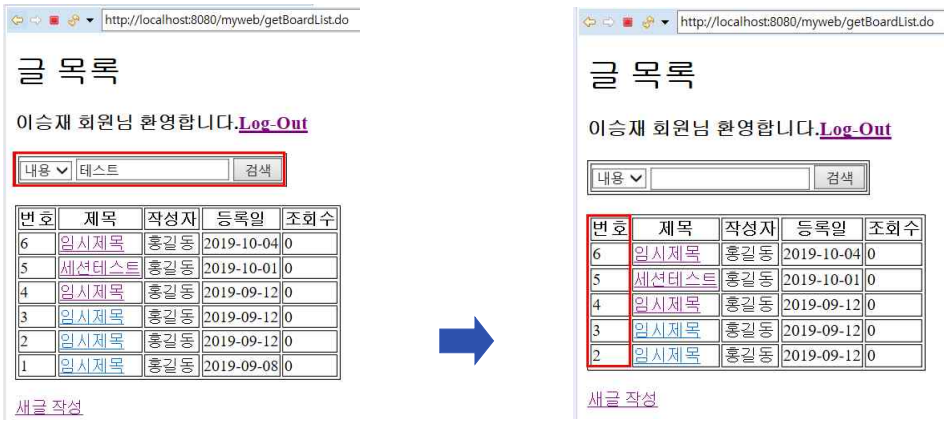
■ Spring JDBC 기반의 BoardDAOSpring 클래스 수정

```
1   package tommy.spring.web.board.impl;
2   import java.util.List;
3   import org.springframework.beans.factory.annotation.Autowired;
4   import org.springframework.jdbc.core.JdbcTemplate;
5   import org.springframework.stereotype.Repository;
6   import tommy.spring.web.board.BoardVO;
7   @Repository
8   public class BoardDAOSpring {
9           @Autowired
10          private JdbcTemplate jdbcTemplate;
11          private final String BOARD_INSERT = "insert into myboard(seq, title, writer, content) "
                        + "values((select nvl(max(seq), 0)+1 from myboard), ?, ?, ?)";
12          private final String BOARD_UPDATE = "update myboard set title=?, " +
                                                        "content=? where seq=?";
13          private final String BOARD_DELETE = "delete myboard where seq=?";
14          private final String BOARD_GET = "select * from myboard where seq=?";
15          private final String BOARD_LIST_T =
                    "select * from myboard where title like '%'||?||'%' order by seq desc";
16          private final String BOARD_LIST_C =
                    "select * from myboard where content like '%'||?||'%' order by seq desc";
            <!-- 중간 부분 생략 -->
17          public List<BoardVO> getBoardList(BoardVO vo) {
18                  System.out.println("Spring JDBC로 getBoardList() 기능 처리");
19                  Object[] args = { vo.getSearchKeyword() };
20                  if (vo.getSearchCondition().equals("TITLE")) {
21                          return jdbcTemplate.query(BOARD_LIST_T, args,
                                                        new BoardRowMapper());
22                  } else if (vo.getSearchCondition().equals("CONTENT")) {
23                          return jdbcTemplate.query(BOARD_LIST_C, args,
                                                        new BoardRowMapper());
24                  }
25                  return null;
```

```
26          }
27  }
```

▢ 기본적인 코드 구성은 앞에서 살펴본 BoardVO와 같지만 검색 키워드를 설정하기 위하여 Object 배열을 사용한다는 것이 기존 코드와의 차이점이라고 할 수 있다.

▢ 이제 실행하여 잘 수행되는지 결과를 확인해 보자



3. 파일 업로드

① 파일 업로드 처리

▢ 게시판 프로그램에 파일 업로드 기능을 추가하기 위하여 글 등록 화면을 수정한다.

■ insertBoard.jsp 수정

```
1   <!-- 상단 부분 생략 -->
2   <body>
3   <h1>글등록</h1>
4   <a href="logout.do">Log Out</a><hr>
5   <form action="insertBoard.do" method="post" enctype="multipart/form-data">
6   <table border="1">
7   <!-- 중간 부분 생략 -->
8   <tr>
9           <td>업로드</td>
10          <td><input type="file" name="uploadFile"/></td>
11  </tr>
12  <tr>
13          <td colspan="2"><input type="submit" value="새글 등록"/></td>
14  </tr>
15  </table>
16  </form><hr>
17  <a href="getBoardList.do">글 목록으로 가기</a>
18  </body>
19  </html>
```

② Command 객체 수정

▢ BoardVO 객체에 업로드 관련 변수를 추가해야 한다.

■ BoardVO 수정

```
1  package tommy.spring.web.board;
2  import java.sql.Date;
3  import org.springframework.web.multipart.MultipartFile;
4  public class BoardVO {
5          private int seq;
6          private String title;
7          private String writer;
8          private String content;
9          private Date regDate;
10         private int cnt;
11         private String searchCondition;
12         private String searchKeyword;
13         private MultipartFile uploadFile;
14         public MultipartFile getUploadFile() {
15                 return uploadFile;
16         }
17         public void setUploadFile(MultipartFile uploadFile) {
18                 this.uploadFile = uploadFile;
19         }
   <!-- 하단 부분 생략 -->
```

③ FileUpload 라이브러리 추가

▢ http://www.mvnrepository.com에 접속하여 commons-fileupload를 검색하여 pom.xml에 아래와 같이 파일업로드 라이브러리를 추가하자.

```
1  <!-- 상단 부분 생략 -->
2          <dependencies>
3                  <!-- 파일업로드 -->
4                  <dependency>
5                          <groupId>commons-fileupload</groupId>
6                          <artifactId>commons-fileupload</artifactId>
7                          <version>1.4</version>
8                  </dependency>
9  <!-- 하단 부분 생략 -->
```

▢ 위 결과로 commons-fileupload 라이브러리와 의존관계에 있는 commons-io 라이브러리 2개가 추가되는 것을 확인할 수 있다.

④ MultipartResolver 설정

▢ 스프링 설정파일에 CommonsMultipartResolver를 <bean>으로 등록하자.

■ presentation-layer.xml

```
1  <!-- 상단 부분 생략 -->
2  <context:component-scan base-package="tommy.spring.web"></context:component-scan>
3
4  <!-- 파일 업로드 설정 -->
5  <bean id="multipartResolver"
```

```
6           class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
7           <property name="maxUploadSize" value="1000000" />
8   </bean>
9   <!-- 하단 부분 생략 -->
```

▢ 위 설정에서 중요한 점은 CommonsMultipartResolver 클래스의 id나 name값을 변경하면 안된다는 점이다. 그 이유는 DispatcherServlet이 특정 이름으로 등록된 CommonsMultipartResolver 객체만 인식하도록 되어있기 때문이다.

▢ 참고 : Spring 설정파일에 등록되는 클래스 중에서 이름이 "Resolver"로 끝나는 클래스들은 대부분 아이디가 정해져 있다.

▢ MultipartFile 인터페이스가 제공하는 주요 메서드 '

메서드	설 명
String getOriginalFileName()	업로드 한 파일명을 문자열로 리턴
void transferTo(File destFile)	업로드 한 파일을 destFile에 저장
boolean isEmpty()	업로드 한 파일 존재 여부를 리턴(없으면 true 리턴)

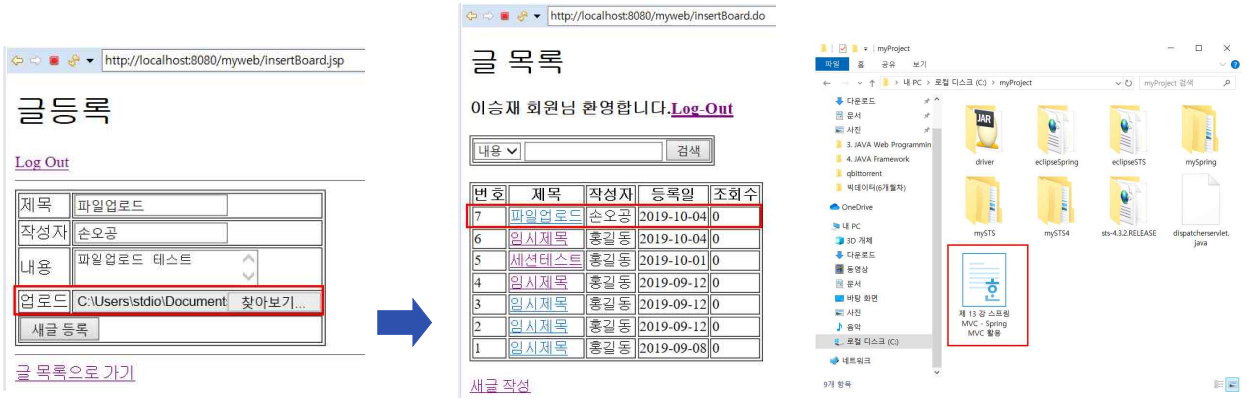
⑤ 파일 업로드 처리

■ BoardController 수정

```
1   package tommy.spring.web.board;
2   import java.io.File;
3   import java.io.IOException;
4   import java.util.HashMap;
5   import java.util.Map;
6   import org.springframework.beans.factory.annotation.Autowired;
7   import org.springframework.stereotype.Controller;
8   import org.springframework.ui.Model;
9   import org.springframework.web.bind.annotation.ModelAttribute;
10  import org.springframework.web.bind.annotation.RequestMapping;
11  import org.springframework.web.bind.annotation.SessionAttributes;
12  import org.springframework.web.multipart.MultipartFile;
13  @Controller
14  @SessionAttributes("board")
15  public class BoardController {
16          @Autowired
17          private BoardService boardService;
18          <!-- 중간 부분 생략 -->
19          @RequestMapping("/insertBoard.do")
20          public String insertBoard(BoardVO vo)  throws IOException{
21                  System.out.println("글 등록 처리");
22                  MultipartFile uploadFile = vo.getUploadFile();
23                  if(!uploadFile.isEmpty()) {
24                          String fileName = uploadFile.getOriginalFilename();
```

```
25                              uploadFile.transferTo(new File("C:/myProject/" + fileName));
26                      }
27                      boardService.insertBoard(vo);
28                      return "getBoardList.do";
29              }
30  <!-- 하단 부분 생략 -->
```

▢ 실행하여 파일 업로드가 수행되는지 확인해 보자.



4. 예외처리

▢ 클라이언트의 요청에 따라 Controller의 메서드가 실행되다 보면 예기치 않은 예외가 발생할 수 있다. 이러한 예외 발생 시 사용자에게 적절한 메시지가 담긴 화면을 보여주는 작업을 예외처리하고 할 수 있다.

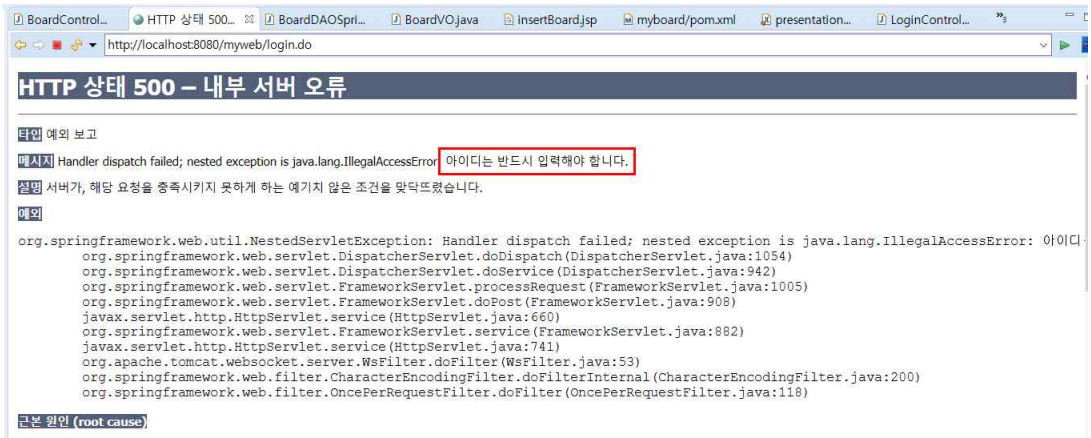
▢ 예를 들어 클라이언트가 로그인 요청을 전송할 때 아이디를 입력하지 않으면 IllegalArguementException이 발생한다고 가정하고 LoginController를 수정하자.

■ LoginController 수정

```
1   package tommy.spring.web.user;
2   import javax.servlet.http.HttpSession;
3   import org.springframework.stereotype.Controller;
4   import org.springframework.web.bind.annotation.RequestMapping;
5   import org.springframework.web.bind.annotation.RequestMethod;
6   import tommy.spring.web.user.impl.UserDAO;
7   @Controller
8   public class LoginController {
9           <!-- 중간 부분 생략 -->
10          @RequestMapping(value = "/login.do", method = RequestMethod.POST)
11          public String login(UserVO vo, UserDAO userDAO, HttpSession session) {
12                  System.out.println("로그인 인증 처리");
13                  if (vo.getId() == null || vo.getId().equals("")) {
14                          throw new IllegalAccessError("아이디는 반드시 입력해야 합니다.");
15                  }
16                  UserVO user = userDAO.getUser(vo);
17                  if (user != null) {
```

```
18                              session.setAttribute("userName", user.getName());
19                              return "getBoardList.do";
20                      } else {
21                              return "login.jsp";
22                      }
23              }
24      }
```

▢ 이제 로그인할 때 아이디를 입력하지 않고 로그인 요청을 하면 아래와 같이 500번 에러가 발생한다.



① 어노테이션 기반의 예외처리

▢ 스프링에서는 @ControllerAdvice와 @ExceptionHandler 어노테이션을 이용하여 컨트롤러의 메서드 수행 중 발생하는 예외를 일괄적으로 처리할 수 있다.

■ presentation-layer.xml 수정

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <beans xmlns="http://www.springframework.org/schema/beans"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xmlns:context="http://www.springframework.org/schema/context"
5           xmlns:mvc="http://www.springframework.org/schema/mvc"
6           xsi:schemaLocation="http://www.springframework.org/schema/mvc
7     http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
8                   http://www.springframework.org/schema/beans
9     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-4.3.xsd">
12
13          <context:component-scan base-package="tommy.spring.web">
14          </context:component-scan>
15
16          <mvc:annotation-driven></mvc:annotation-driven>
17          <!-- 하단 부분 생략 -->
```

■ CommonExceptionHandler 클래스 작성

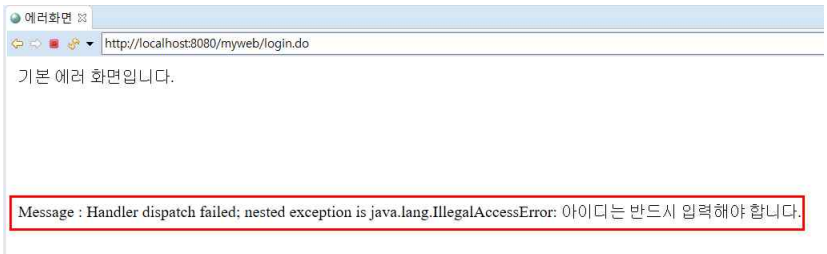
```
1  package tommy.spring.web.common;
2  import org.springframework.web.bind.annotation.ControllerAdvice;
3  import org.springframework.web.bind.annotation.ExceptionHandler;
4  import org.springframework.web.servlet.ModelAndView;
5  @ControllerAdvice("tommy.spring.web")
6  public class CommonExceptionHandler {
7          @ExceptionHandler(ArithmeticException.class)
8          public ModelAndView handlerArithmeticException(Exception e) {
9                  ModelAndView mav = new ModelAndView();
10                 mav.addObject("exception", e);
11                 mav.setViewName("/common/arithmeticError.jsp");
12                 return mav;
13         }
14         @ExceptionHandler(NullPointerException.class)
15         public ModelAndView handlerNullPointerException(Exception e) {
16                 ModelAndView mav = new ModelAndView();
17                 mav.addObject("exception", e);
18                 mav.setViewName("/common/nullPointerError.jsp");
19                 return mav;
20         }
21         @ExceptionHandler(Exception.class)
22         public ModelAndView handlerException(Exception e) {
23                 ModelAndView mav = new ModelAndView();
24                 mav.addObject("exception", e);
25                 mav.setViewName("/common/error.jsp");
26                 return mav;
27         }
28 }
```

■ src/main/webapp/common/error.jsp 파일 작성

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8">
6  <title>에러화면</title>
7  </head>
8  <body>
9  <table>
10 <tr><td>기본 에러 화면입니다.</td></tr>
11 </table>
12 <br><br>
13 <table>
14 <tr>
15 <td>
16 <br><br><br><br>
17 Message : ${exception.message }
18 <br><br><br><br>
```

```
19  </td>
20  </tr>
21  </table>
22  </body>
23  </html>
```

▢ 이제 다시 한 번 아이디 없이 로그인을 시도해 보자.



② XML 기반의 예외 처리

▢ 스프링은 예외처리를 어노테이션이 아닌 XML 설정 방식으로도 지원하는데 이 방법이 조금 더 쉬운 방법이라고 할 수 있다. 어노테이션 설정과 기본 개념은 똑같으나 CommonExceptionHandler처럼 예외 처리 클래스를 별도로 구현하지 않아도 되며 단지 XML 설정만 처리하면 된다.

▢ presentation-layer.xml 파일에 SimpleMappingExceptionResolver 클래스를 <bean>으로 등록하기만 하면 된다.

■ presentation-layer.xml 파일 수정

```
1   <!-- 상단 부분 생략 -->
2   <context:component-scan base-package="tommy.spring.web"></context:component-scan>
3   <!-- 파일 업로드 설정 -->
4   <bean id="multipartResolver"
5           class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
6           <property name="maxUploadSize" value="1000000" />
7   </bean>
8   <!-- 예외처리 설정 -->
9   <bean id="exceptionResolver"
10          class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
11      <property name="exceptionMappings">
12          <props>
13                  <prop key="java.lang.ArithmeticException">common/arithmeticError.jsp</prop>
14                  <prop key="java.lang.NullPointerException">common/nullPointerError.jsp</prop>
15          </props>
16      </property>
17      <property name="defaultErrorView" value="common/error.jsp" />
18  </bean>
19  <!-- 하단 부분 생략 -->
```

▢ 이제 다시 한 번 아이디 없이 로그인을 시도해 보자. : 실행결과 앞과 같음.

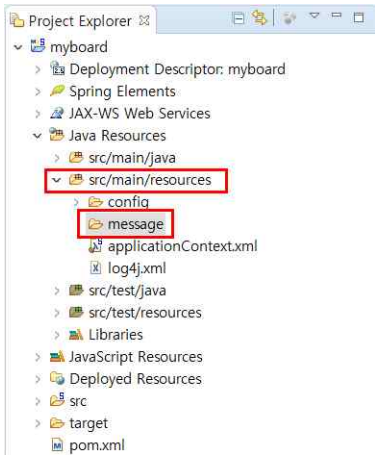
5. 다국어 처리

다국어 지원이란 “국제화”라고도 하며 하나의 JSP 페이지를 다양한 언어로 서비스하는 것을 의미한다. 프레임워크에서 다국어를 지원하기 전에는 언어별로 JSP 파일을 만들어야 했기 때문에 매우 불편했다. 하지만 프레임워크의 다국어 기능을 이용하면 JSP 화면에 출력되는 메시지나 예외 로그 메시지까지 다국어로 처리할 수 있다.

① 메시지 파일 작성하기

▢ 기본적인 메시지 파일의 확장자는 “.properties”이며 파일명은 해당하는 Locale 정보를 결합하여 작성한다.

■ src/main/resources 폴더에 message 패키지 생성



■ 영어 지원용 messageSource_en.properties 파일 작성

```
   # login.jsp
1  message.user.login.title=LOGIN
2  message.user.login.id=ID
3  message.user.login.password=PASSWORD
4  message.user.login.loginBtn=LOG-IN
5  message.user.login.language.en=English
6  message.user.login.language.ko=Korean

   # getBoardList.jsp
7  message.board.list.mainTitle=BOARD_LIST
8  message.board.list.welcomeMsg=! Welcome to my Board
9  message.board.list.search.condition.title=TITLE
10 message.board.list.search.condition.content=CONTENT
11 message.board.list.search.condition.btn=Search
12 message.board.list.table.head.seq=SEQ
13 message.board.list.table.head.title=TITLE
14 message.board.list.table.head.writer=WRITER
15 message.board.list.table.head.regDate=REGDATE
16 message.board.list.table.head.cnt=CNT
17 message.board.list.link.insertBoard=Insert Board
```

■ 한글 지원용 messageSource_ko.properties 파일 작성

▢ properties 파일에 한글을 작성하면 유니코드로 변경되기 때문에 일단 text 파일로 작성한 후에 복사하여 옮기는 것이 편리하다.

```
    # login.jsp
1   message.user.login.title=로그인
2   message.user.login.id=아이디
3   message.user.login.password=비밀번호
4   message.user.login.loginBtn=로그인
5   message.user.login.language.en=영어
6   message.user.login.language.ko=한글

    # getBoardList.jsp
7   message.board.list.mainTitle=게시글 목록
8   message.board.list.welcomeMsg=님! 게시판에 오신걸 환영합니다.
9   message.board.list.search.condition.title=제목
10  message.board.list.search.condition.content=내용
11  message.board.list.search.condition.btn=검색
12  message.board.list.table.head.seq=번호
13  message.board.list.table.head.title=제목
14  message.board.list.table.head.writer=작성자
15  message.board.list.table.head.regDate=등록일
16  message.board.list.table.head.cnt=조회수
17  message.board.list.link.insertBoard=등록
```

② 스프링 설정파일에 MessageSource 등록

■ presentation-layer.xml 수정

```
    <!-- 상단 부분 생략 -->
    <!-- 다국어 설정 : MessageSource 등록 -->
1       <bean id="messageSource"
                class="org.springframework.context.support.ResourceBundleMessageSource">
2           <property name="basenames">
3               <list>
4                   <value>message.messageSource</value>
5               </list>
6           </property>
7       </bean>
    <!-- 하단 부분 생략 -->
```

▢ ResourceBundleMessageSource 클래스를 <bean>으로 등록할 때 주의할 점은 이름이 "messageSource"로 고정되어 있다는 점이다.

▢ 그리고 MessageSource를 등록할 때 메시지 파일의 확장자 ".properties"는 생략한다.

▢ 또한 언어에 해당하는 "_en", "_ko" 역시 생략한다. 이것은 메시지 파일을 좀 더 효율적으로 등록하고 관리하기 위해서 이다.

▢ 일반적으로 화면에서 사용할 메시지를 하나의 파일에 모두 작성하는 경우는 없다. 그렇게 되면 메

시지 파일이 수천 줄이 넘어갈 수 도 있다. 따라서 여러 개의 파일로 나누어서 관리하게 되는데 일반적으로 컴포넌트 하나당 하나의 메시지 파일을 작성한다.

▢ 만약 시스템에 10개의 컴포넌트가 있고 5개의 언어를 지원한다고 하면 산술적으로 50개의 메시지 파일이 필요하다. 유지보수과정에서 새로운 언어가 추가될 수 도 있고 기존에 지원하던 언어를 지원하지 않아도 되는 경우도 생길 수 있다. 그럴 때마다 메시지 파일이 등록된 스프링 설정파일을 수정하는 것은 매우 불편할 것이다.

▢ 따라서 스프링에서는 언어에 해당하는 정보를 메시지 파일명에서 제거하는 방법을 택한 것이다.

③ LocaleResolver 등록

▢ 웹브라우저가 서버에 요청하면 브라우저의 Locale 정보가 HTTP 요청 메시지 헤더에 자동으로 설정되어 전송된다. 이때 스프링은 LocaleResolver를 통해서 클라이언트의 Locale 정보를 추출하고 Locale 정보에 해당하는 언어의 메시지를 선택한다.

▢ 스프링 설정파일에 LocaleResolver가 등록되어 있지 않으면 기본적으로 AcceptHeaderLocaleResolver가 선택된다.

▢ 스프링에서 지원하는 LocaleResolver

LocaleResolver 종류	기능 설명
AcceptHeaderLocaleResolver	브라우저에서 전송된 HTTP 요청 헤더에서 Accept-Language에 설정된 Locale로 메시지를 적용한다.
CookieLocaleResolver	Cookie에 저장된 Locale 정보를 추출하여 메시지를 적용한다.
SessionLocaleResolver	HttpSession에 저장된 Locale 정보를 추출하여 메시지를 적용한다.
FixedLocaleResolver	웹 요청과 무관하게 특정 Locale로 고정한다.

▢ 참고 : 스프링에서는 다양한 LocaleResolver를 지원하지만 세션으로부터 Locale 정보를 추출하고 유지하는 SessionLocaleResolver를 가장 많이 사용한다.

■ presentation-layer.xml 수정

```
    <!-- 상단 부분 생략 -->
    <!-- 다국어 설정 : MessageSource 등록 -->
1       <bean id="messageSource"
                class="org.springframework.context.support.ResourceBundleMessageSource">
2           <property name="basenames">
3               <list>
4                   <value>message.messageSource</value>
5               </list>
6           </property>
7       </bean>
8       <bean id="localeResolver"
                class="org.springframework.web.servlet.i18n.SessionLocaleResolver"></bean>
```

```
<!-- 하단 부분 생략 -->
```

④ Locale 변경하기

▢ 특정 언어로 화면을 보다가 해당 화면의 언어를 변경하고 싶을 때가 있을 수 있다. 이를 위해 스프링에서는 LocaleChangeInterceptor 클래스를 제공한다.

■ LocaleChangeInterceptor 등록

```
<!-- 상단 부분 생략 -->
<!-- LocaleResolver 등록 -->
        <bean id="localeResolver"
                class="org.springframework.web.servlet.i18n.SessionLocaleResolver"></bean>

        <!-- LocaleChangeInterceptor 등록 -->
        <mvc:interceptors>
                <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
                        <property name="paramName" value="lang"/>
                </bean>
        </mvc:interceptors>
<!-- 하단 부분 생략 -->
```

▢ 여기서 중요한 점은 <bean> 루트 엘리먼트에 mvc 네임스페이스를 추가해야 한다.

▢ 클라이언트로부터 “lang”이라는 파라미터로 특정 Locale이 전송되면 해당 Locale로 변경하겠다는 설정이다.

⑤ JSP 파일 작성

▢ JSP 파일에서 메시지 파일에 등록한 메시지로 화면을 구성하려면 스프링이 제공하는 태그 라이브러리를 이용해야 한다. 따라서 taglib 지시자를 반드시 선언해야 한다.

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
```

▢ 위와 같이 선언 후 <spring:message /> 태그의 속성 값으로 메시지 파일에 등록된 메시지 키를 등록하면 Locale에 해당하는 메시지를 출력할 수 있다.

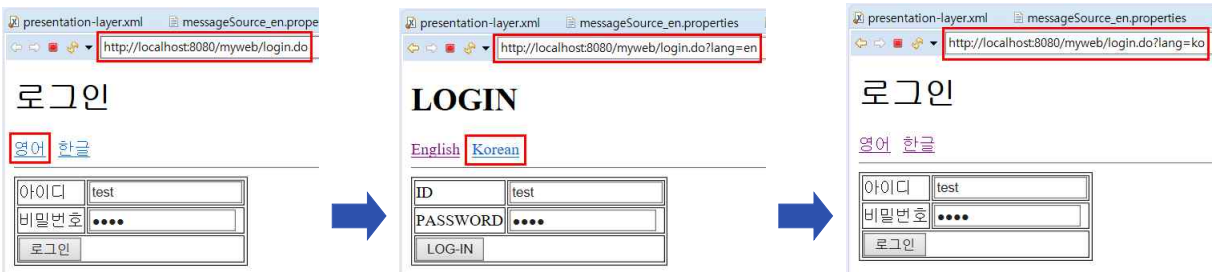
```
<spring:messge code="메시지 킷값" />
```

■ login.jsp 파일 수정

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2  <%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title><spring:message code="message.user.login.title" /></title>
8  </head>
9  <body>
```

```
10  <h1><spring:message code="message.user.login.title" /></h1>
11  <a href="login.do?lang=en">
12          <spring:message code="message.user.login.language.en" /></a>  
13  <a href="login.do?lang=ko"><spring:message code="message.user.login.language.ko" /></a>
14  <hr>
15  <form action="login.do" method="post">
16  <table border="1">
17  <tr>
18          <td><spring:message code="message.user.login.id" /></td>
19          <td><input type="text" name="id"  value="${userVO.id}"/></td>
20  </tr>
21  <tr>
22          <td><spring:message code="message.user.login.password" /></td>
23          <td><input type="password" name="password" value="${userVO.password}"/></td>
24  </tr>
25  <tr>
26          <td colspan="2"><input type="submit"
27                  value="<spring:message code="message.user.login.loginBtn" />" /></td>
28  </tr>
29  </table>
30  </form>
31  </body>
32  </html>
```

▢ 실행 및 결과 확인



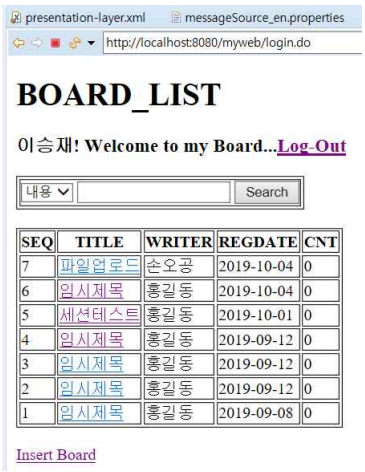
■ getBoardList.jsp 수정

```
1   <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2   <%@ page import="tommy.spring.web.board.BoardVO"%>
3   <%@ page import="java.util.List"%>
4   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
5   <%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
6   <!DOCTYPE html>
7   <html>
8   <head>
9   <meta charset="UTF-8">
10  <title><spring:message code="message.board.list.mainTitle" /></title>
11  </head>
12  <body>
13  <h1><spring:message code="message.board.list.mainTitle" /></h1>
14  <h3>${userName}<spring:message code="message.board.list.welcomeMsg" />...
```

```
15  <a href="logout.do">Log-Out</a></h3>
16  <!-- 검색 시작 -->
17  <form action="getBoardList.do" method="post">
18  <table border="1">
19  <tr>
20          <td>
21                  <select name="searchCondition">
22                          <c:forEach items="${conditionMap }" var="option">
23                                  <option value="${option.value }">${option.key }</option>
24                          </c:forEach>
25                  </select>
26                  <input type="text" name="searchKeyword" />
27                  <input type="submit"
28                  value="<spring:message code="message.board.list.search.condition.btn" />" />
29          </td>
30  </tr>
31  </table>
32  </form><br/>
33  <!-- 검색 종료 -->
34  <table border="1">
35  <tr>
36          <th><spring:message code="message.board.list.table.head.seq" /></th>
37          <th><spring:message code="message.board.list.table.head.title" /></th>
38          <th><spring:message code="message.board.list.table.head.writer" /></th>
39          <th><spring:message code="message.board.list.table.head.regDate" /></th>
40          <th><spring:message code="message.board.list.table.head.cnt" /></th>
41  </tr>
42  <c:forEach var="board" items="${boardList }">
43  <tr>
44          <td>${board.seq }</td>
45          <td><a href="getBoard.do?seq=${board.seq }">${board.title }</a></td>
46          <td>${board.writer }</td>
47          <td>${board.regDate }</td>
48          <td>${board.cnt }</td>
49  </tr>
50  </c:forEach>
51  </table><br/>
52  <a href="insertBoard.jsp"><spring:message code="message.board.list.link.insertBoard" /></a>
53  </body>
54  </html>
```

▢ 실행 및 결과 확인

: 우선 login.do에서 Locale 설정을 영어로 변경한 후 로그인을 해 보자. 아래와 같이 영문으로 된 게시판이 나오면 성공이다.



6. 데이터 변환

▢ 시스템이 복잡해지면서 다른 시스템과 정보를 주고받을 일이 발생하는데, 이때 데이터 교환 포맷으로 주로 JSON과 XML을 사용한다. 검색한 글 목록을 JSON(Javascript Object Notation) 형식의 데이터로 변환해서 브라우저에 출력해 보자.

① Jackson2 라이브러리 내려받기

■ mvnrepository.com에서 jackson-databind를 검색하여 pom.xml에 아래와 같이 의존성을 추가하자.

```
<!-- 상단 부분 생략 -->
        <dependencies>
                <!-- Jackson2 : JSON 처리 -->
                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>2.10.0</version>
                </dependency>
<!-- 하단 부분 생략 -->
```

▢ jackson-databind 파일에 대한 설정만 하면 의존관계에 있는 jackson-annotation, jackson-core 파일도 함께 추가되는 것을 확인할 수 있다.

② HttpMessageConvertter 등록

▢ 일반적으로 브라우저에서 서블릿이나 JSP 파일을 요청하면 서버는 클라이언트가 요청한 서블릿이나 JSP를 찾아 실행한다. 그리고 실행결과를 Http 응답 프로토콜 메시지 바디에 저장하여 브라우저에 전송한다.

▢ 그래서 브라우저는 항상 실행 결과 화면만 표시했던 것이다. 이 응답 결과를 HTML이 아닌 JSON이나 XML로 변환하여 메시지 바디에 저장하려면 스프링에서 제공하는 변환기를 이용해야 한다.

▢ 스프링은 HttpMessageConvertter를 구현한 다양한 변환기를 제공하는데 이 변환기를 이용하면 자바 객체를 다양한 타입으로 변환하여 HTTP 응답 바디에 설정할 수 있다. 이 중에서 자바 객체를 JSON 응답 바디로 변경할 때는 MappingJackson2HttpMessageConvertter를 사용한다.

▢ 우리는 추후에 XML 변환도 시도 할 것이므로 <mvc:annotation-driven>을 설정한다.

■ presentation-layer.xml 수정

```
<!-- 상단 부분 생략 -->
        <context:component-scan base-package="tommy.spring.web">
        </context:component-scan>
        <mvc:annotation-driven></mvc:annotation-driven>
<!-- 하단 부분 생략 -->
```

③ 링크 추가 및 Controller 수정

■ index.jsp 수정

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>게시판 프로그램</h1>
<hr>
<a href="login.do">로그인</a><br></br>
<a href="getBoardList.do">글 목록으로 가기</a><br></br>
<a href="dataTransform.do">글 목록 변환 처리</a>
<hr>
</body>
</html>
```

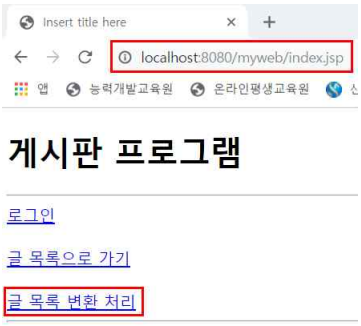
■ BoardController 수정

```
package tommy.spring.web.board;
import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.multipart.MultipartFile;
@Controller
@SessionAttributes("board")
public class BoardController {
        @Autowired
        private BoardService boardService;
```

```
20          @RequestMapping("/dataTransform.do")
21          @ResponseBody
22          public List<BoardVO> dataTransform(BoardVO vo){
23                  vo.setSearchCondition("TITLE");
24                  vo.setSearchKeyword("");
25                  List<BoardVO> boardList = boardService.getBoardList(vo);
26                  return boardList;
27          }
    <!-- 하단 부분 생략 -->
```

▢ 실행 및 결과 확인

: 크롬 브라우저를 실행 하고 index.jsp 요청 후 글 목록 변환 처리 클릭 - 인터넷 익스플로러의 경우 JSON 데이터를 표시할 수 없다. 따라서 결과 확인 시 크롬 브라우저를 이용한다.



▢ 위 실행 결과를 보면 검색 조건, 검색 키워드, 파일 업로드 정보까지 포함되어 있다. 따라서 이러한 정보들은 제외하는 것이 맞다.

■ BoardVO 수정

```
1   package tommy.spring.web.board;
2   import java.sql.Date;
3   import org.springframework.web.multipart.MultipartFile;
4   import com.fasterxml.jackson.annotation.JsonIgnore;
5   public class BoardVO {
6           // 중간 부분 생략
7           @JsonIgnore
8           public MultipartFile getUploadFile() {
9                   return uploadFile;
10          }
11          @JsonIgnore
12          public String getSearchCondition() {
```

```
13                  return searchCondition;
14          }
15          @JsonIgnore
16          public String getSearchKeyword() {
17                  return searchKeyword;
18          }
19  // 하단 부분 생략
```

☐ BoardVO 클래스의 getter 메서드에 세 개의 @JsonIgnore를 추가하였다. @JsonIgnore는 자바 객체를 JSON으로 변환할 때 특정 변수를 변환에서 제외시킨다. 주의할 점은 getter 메서드 위에 설정해야한다는 점이다.

☐ 실행 및 결과 확인

④ XML로 변환하기

☐ 다른 시스템과 정보를 주고받을 때 JSON을 사용하기도 하지만 XML을 사용하기도 한다. 자바 객체를 XML로 변환하기 위해서는 JAXB2 API가 필요한데 Java 6 버전 이후에 기본적으로 포함되어 있다. 따라서 별도의 라이브러리를 추가할 필요가 없다.

■ BoardVO 수정

```
1   package tommy.spring.web.board;
2   import java.util.Date;
3   import javax.xml.bind.annotation.XmlAccessType;
4   import javax.xml.bind.annotation.XmlAccessorType;
5   import javax.xml.bind.annotation.XmlAttribute;
6   import javax.xml.bind.annotation.XmlTransient;
7   import org.springframework.web.multipart.MultipartFile;
8   import com.fasterxml.jackson.annotation.JsonIgnore;
9   @XmlAccessorType(XmlAccessType.FIELD)
10  public class BoardVO {
11          @XmlAttribute
12          private int seq;
13          private String title;
14          private String writer;
15          private String content;
16          private Date regDate;
17          private int cnt;
18          @XmlTransient
19          private String searchCondition;
20          @XmlTransient
```

```
21          private String searchKeyword;
22          @XmlTransient
23          private MultipartFile uploadFile;

24          @JsonIgnore
25          public MultipartFile getUploadFile() {
25                  return uploadFile;
27          }
    // 하단 부분 생략
```

☐ @XmlAccessorType은 BoardVO 객체를 XML로 변환할 수 있다는 의미이다.
그리고 XmlAccessType.FIELD 때문에 이 객체가 가진 필드 즉 변수들은 자동적으로 자식 엘리먼트로 표현된다. 하지만 이중에서 seq 변수에만 @XmlAttribute가 붙었는데 이는 seq를 속성으로 표현하라는 의미이다.

☐ @XmlTransient는 XML 변환에서 제외하라는 의미다.

☐ 주의할 점은 등록날짜를 지정하는 regDate 변수는 이전까지는 java.sql.Date를 사용하였는데 이 객체는 기본 생성자가 없는 객체이다. 특정 자바 객체를 XML로 변환하려면 반드시 해당 클래스의 기본 생성자가 있어야 한다. 따라서 regDate 변수를 기본 생성자가 있는 java.util.Date 타입의 변수로 변경한 것이다.

■ BoardListVO 추가 작성

☐ XML문서는 반드시 단 하나의 루트 엘리먼트를 가져야 한다. BoardVO는 하나의 게시 글 정보를 저장하려고 사용하는 객체이다. 하지만 우리는 여러 게시 글 목록을 XML로 표현해야 하므로 BoardVO 객체 여러 개를 포함하면서 루트 엘리먼트로 사용할 또 다른 자바 클래스가 필요하다.

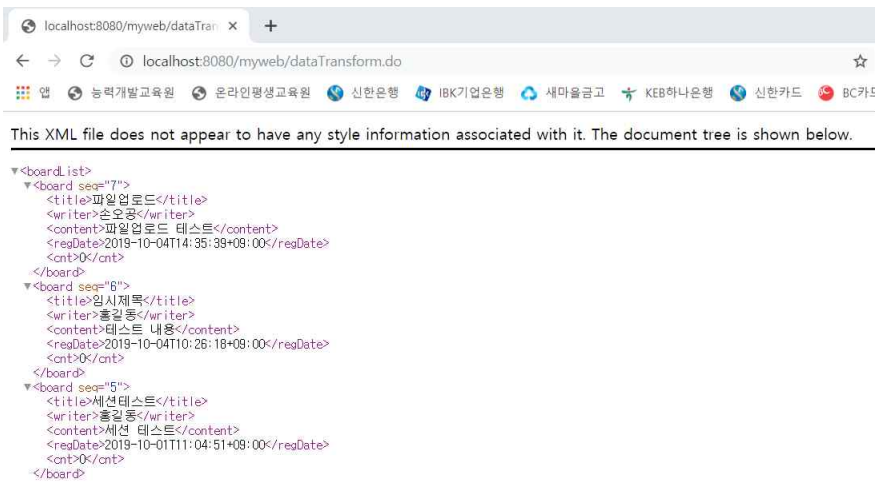
```
1   package tommy.spring.web.board;
2   import java.util.List;
3   import javax.xml.bind.annotation.XmlAccessType;
4   import javax.xml.bind.annotation.XmlAccessorType;
5   import javax.xml.bind.annotation.XmlElement;
6   import javax.xml.bind.annotation.XmlRootElement;
7   @XmlRootElement(name = "boardList")
8   @XmlAccessorType(XmlAccessType.FIELD)
9   public class BoardListVO {
10          @XmlElement(name = "board")
11          private List<BoardVO> boardList;
12          public List<BoardVO> getBoardList() {
13                  return boardList;
14          }
15          public void setBoardList(List<BoardVO> boardList) {
16                  this.boardList = boardList;
17          }
18  }
```

☐ boardList 변수 위에 @XmlElement(name="board")를 추가하였는데 만약 이 설정이 없다면 변수 이름인 boardList가 엘리먼트 이름으로 사용된다.

■ BoardController 수정

```
1  package tommy.spring.web.board;
2  import java.io.File;
3  import java.io.IOException;
4  import java.util.HashMap;
5  import java.util.List;
6  import java.util.Map;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.stereotype.Controller;
9  import org.springframework.ui.Model;
10 import org.springframework.web.bind.annotation.ModelAttribute;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.ResponseBody;
13 import org.springframework.web.bind.annotation.SessionAttributes;
14 import org.springframework.web.multipart.MultipartFile;
15 @Controller
16 @SessionAttributes("board")
17 public class BoardController {
18         @Autowired
19         private BoardService boardService;
20         @RequestMapping("/dataTransform.do")
21         @ResponseBody
22         public BoardListVO dataTransform(BoardVO vo){
23                 vo.setSearchCondition("TITLE");
24                 vo.setSearchKeyword("");
25                 List<BoardVO> boardList = boardService.getBoardList(vo);
25                 BoardListVO boardListVO = new BoardListVO();
27                 boardListVO.setBoardList(boardList);
28                 return boardListVO;
29         }
   // 하단 부분 생략
```

▢ 실행 및 결과 확인



▢ 자바 객체를 JSON으로 변환해주는 MappingJackson2HttpMessageConvertter를 스프링 설정파일에 추가하는 대신 <mvc:annotation-driven> 설정으로 대체하였는데 이 설정은 자바 객체를 XML 응답으로 변환할 때 사용하는 Jaxb2RootElementHttpMessageConvertter 클래스도 같이 등록해 준다.

▢ 실행 결과를 확인해 보면 boardList가 루트 엘리먼트로 사용됐고 여러 개의 board 엘리먼트를 자식으로 가지고 있다. 그리고 seq는 속성으로 사용된 것을 확인할 수 있다.

▢ 실행결과에서 날짜 정보가 이전과 다르게 출력되는데 이는 java.sql.Date를 java.util.Date로 변경했기 때문이다. 따라서 정상적으로 로그인 한 수에 출력되는 글 목록에 날짜 데이터도 아래와 같이 변경된다.

■ getBoardList.jsp 수정

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2  <%@ page import="tommy.spring.web.board.BoardVO"%>
3  <%@ page import="java.util.List"%>
4  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
5  <%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
6  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
7  <!DOCTYPE html>
8  <html>
9  <head>
10 <meta charset="UTF-8">
11 <title><spring:message code="message.board.list.mainTitle" /></title>
12 </head>
13 <body>
14 <!-- 중간 부분 생략 -->
15 <c:forEach var="board" items="${boardList }">
16 <tr>
17         <td>${board.seq }</td>
18         <td><a href="getBoard.do?seq=${board.seq }">${board.title }</a></td>
19         <td>${board.writer }</td>
20         <td><fmt:formatDate value="${board.regDate }" pattern="yyyy-MM-dd"/></td>
21         <td>${board.cnt }</td>
22 </tr>
23 </c:forEach>
24 </table><br/>
25 <a href="insertBoard.jsp"><spring:message code="message.board.list.link.insertBoard" /></a>
25 </body>
27 </html>
```

▢ 이제 실행하여 결과를 확인해 보자.