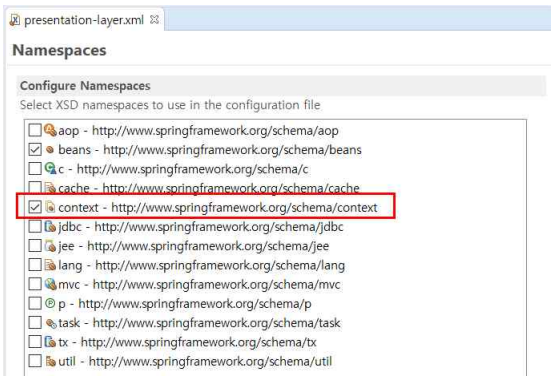
제 12 강 Spring MVC - 어노테이션 기반 개발

스프링은 어노테이션 기반 설정을 제공함으로써 과도한 XML 설정으로 인한 문제를 해결한다. 앞서 살펴보았듯이 스프링 설정파일에 HandlerMapping, Controller, ViewResolver 같은 여러 클래스를 등록해야 하므로 어노테이션 설정을 활용하여 XML 설정을 최소화할 필요가 있다.

1. 어노테이션 관련 설정

① <context:component-sacn> 엘리먼트

▢ 스프링 MVC 어노테이션을 사용하려면 먼저 <bean> 루트 엘리먼트에 context 네임스페이스를 추가한다.



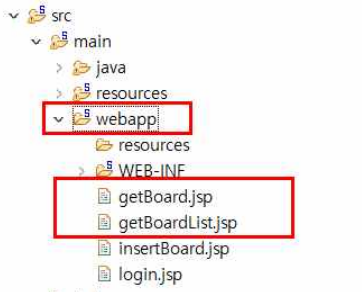
▢ 기존에 등록한 모든 <bean>을 삭제하고 <context:component-scan> 엘리먼트로 대체한다.

```
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          xsi:schemaLocation="http://www.springframework.org/schema/beans
6     http://www.springframework.org/schema/beans/spring-beans.xsd
7                   http://www.springframework.org/schema/context
8                   http://www.springframework.org/schema/context/spring-context-4.3.xsd">

9          <context:component-scan base-package="tommy.spring.web">
10         </context:component-scan>
11 </beans>
```

▢ 위 설정을 통하여 tommy.spring.web 패키지 및 하위 패키지가 모두 스캔대상에 포함되었다.

▢ 앞으로 어노테이션 활용법을 실습하게 될 것이다. 어노테이션에 집중하기 위하여 ViewResolver 설정도 삭제하였다. 이제 getBoardList.jsp와 getBoard.jsp 파일도 원래대로 돌려놓자.



② @Controller 사용하기

▢ 클래스 선언부에 @Controller 어노테이션을 설정하면 <context:component-scan>으로 스프링 컨테이너가 컨트롤러 객체를 자동으로 생성한다. 단순히 객체를 생성하는 것에 그치지 않고 DispatcherServlet이 인식하는 Controller 객체로 만들어준다.

▢ 이는 @Component를 상속한 @Controller, @Service, @Repository 등도 마찬가지이다.

■ InsertBoardController 클래스를 @Controller 어노테이션을 이용하여 스프링 프레임워크가 지향하는 POJO 스타일의 클래스로 구현해 보자.

```
1  package tommy.spring.web.board;
2  import javax.servlet.http.HttpServletRequest;
3  import org.springframework.stereotype.Controller;
4  import tommy.spring.web.board.impl.BoardDAO;
5  @Controller
6  public class InsertBoardController {
7          public void insertBoard(HttpServletRequest request) {
8                  System.out.println("글 등록 처리");
9                  // 1. 사용자 입력 정보 추출
10                 // request.setCharacterEncoding("UTF-8");
11                 String title = request.getParameter("title");
12                 String writer = request.getParameter("writer");
13                 String content = request.getParameter("content");
14                 // 2. 데이터베이스 연동 처리
15                 BoardVO vo = new BoardVO();
16                 vo.setTitle(title);
17                 vo.setWriter(writer);
18                 vo.setContent(content);
19                 BoardDAO boardDAO = new BoardDAO();
20                 boardDAO.insertBoard(vo);
21         }
22 }
```

▢ 코드에서 InsertBoardController 클래스 객체는 스프링 컨테이너가 자동으로 생성하고 Controller 객체로 인식한다. 중요한 것은 InsertBoardController가 POJO 클래스로 변경되었으므로 메서드이름을 insertBoard, 리턴타입을 void, 매개변수를 HttpServletRequest로 변경할 수 있다.

③ @RequestMapping 사용하기

▢ Controller 어노테이션을 이용하여 Controller 객체로 인식하게 할 수 있지만 클라이언트의 요청에 insertBoard() 메서드가 실행되게 할 수는 없다.

▢ 클라이언트의 insertBoard.do 요청에 대해서 insertBoard() 메서드가 수행되게 하려고 앞에서는 HandlerMapping을 이용하였다.

▢ 스프링에서는 @RequestMapping을 이용하여 HandlerMapping을 대체한다.

■ InsertBoardController 수정

```
1  package tommy.spring.web.board;
2  import javax.servlet.http.HttpServletRequest;
```

```
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import tommy.spring.web.board.impl.BoardDAO;
6  @Controller
7  public class InsertBoardController {
8          @RequestMapping(value = "/insertBoard.do")
9          public void insertBoard(HttpServletRequest request) {
10                 System.out.println("글 등록 처리");
```

▢ 참고 @RequestMaping의 value 속성은 생략할 수 있으면 생략한다. 위의 설정은 아래의 XML 설정과 동일한 결과를 가진다.

```
1  <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
2          <property name="mappings">
3                  <props>
4                          <prop key="/insertBoard.do">insertBoard</prop>
5                  </props>
6          </property>
7  </bean>
8  <bean id="insertBoard" class="tommy.spring.web.board.InsertBoardController"></bean>
```

④ 클라이언트의 요청 처리

▢ 대부분 Controller는 사용자의 입력 정보를 추출하여 VO 객체에 저장한다. 그리고 비즈니스 컴포넌트의 메서드를 호출할 때 VO 객체로 전달한다.

▢ 사용자의 입력정보는 HttpServletRequest 객체의 getParameter() 메서드를 이용하는데 이때 입력정보의 양이 많아지면 코드가 길어지는 불편함이 발생한다.

▢ 이러한 문제를 Command 객체를 이용하면 쉽게 해결할 수 있다.

■ InsertBoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import tommy.spring.web.board.impl.BoardDAO;
5  @Controller
6  public class InsertBoardController {
7          @RequestMapping(value = "/insertBoard.do")
8          public void insertBoard(BoardVO vo) {
9                  System.out.println("글 등록 처리");
10                 // 1. 사용자 입력 정보 추출 : 커맨드 객체가 자동으로 처리해 줌.
11                 // 2. 데이터베이스 연동 처리
12                 BoardDAO boardDAO = new BoardDAO();
13                 boardDAO.insertBoard(vo);
14         }
15 }
```

▢ 스프링 컨테이너가 사용자 입력 값을 Command 객체에 자동으로 설정하는 과정을 이해하기 위해 아래의 HTML 폼과 Command 객체의 관계를 살펴보자.

```
게시글 쓰기 입력 폼:
<form method="post">
   <input type="hidden" name="parentId" value="0" />
   제목: <input type="text" name="title" /><br/>
   내용: <textarea name="content"></textarea><br/>
   <input type="submit" />
</form>
```

입력항목의 이름과 일치하는 프로퍼티에 값이 저장

```
public class NewArticleCommand {
   private String title;
   private String content;
   private int parentId;
   public void setTitle(String title) {
      this.title = title;
   }
   public void setContent(String content) {
      this.content = content;
   }
   public void setParentId(int parentId) {
      this.parentId = parentId;
   }
   // getter 메서드
}
```

2. 어노테이션으로 게시판 프로그램 구현하기

① 글 등록 기능 구현하기

■ InsertBoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import tommy.spring.web.board.impl.BoardDAO;
5  @Controller
6  public class InsertBoardController {
7          @RequestMapping(value = "/insertBoard.do")
8          public String insertBoard(BoardVO vo, BoardDAO boardDAO) {
9                  System.out.println("글 등록 처리");
10                 boardDAO.insertBoard(vo);
11                 return "getBoardList.do";
12         }
13 }
```

▢ 글 등록 처리가 성공한 후에는 글 목록을 출력해야 함으로 GetBoardListController를 실행시키기 위해 리턴타입을 String으로 수정한 후 “getBoardList.do”를 리턴하였다. 사용자의 입력 등을 처리하기 위하여 BoardVO와 BoardDAO를 매개변수로 선언하였다. DAO 객체 역시 Command 객체와 마

찬가지로 매개변수로 선언하면 스프링 컨테이너가 해당 객체를 생성하여 전달해 준다.

▢ Controller 메서드가 실행되고 View 경로를 리턴하면 기본적으로 포워딩 방식으로 처리한다. 만약 리다이렉트 방식을 원할 때는 "redirect:"라는 접두어를 붙여야 한다.

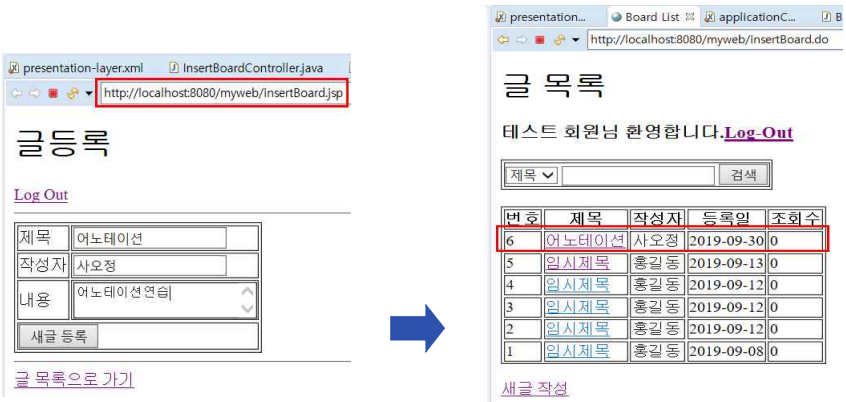
② 글 목록 검색 구현하기

■ GetBoardListController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import org.springframework.web.servlet.ModelAndView;
5  import tommy.spring.web.board.impl.BoardDAO;
6  @Controller
7  public class GetBoardListController{
8          @RequestMapping("/getBoardList.do")
9          public ModelAndView getBoardList(BoardVO vo, BoardDAO boardDAO,
                                                ModelAndView mav) {
10                 System.out.println("글 목록 검색 처리");
11                 mav.addObject("boardList", boardDAO.getBoardList(vo)); // Model 정보저당
12                 mav.setViewName("getBoardList.jsp"); // View 정보저장
13                 return mav;
14         }
15 }
```

▢ 실행 결과 및 테스트

- BoardDAOSpring과 BoardServiceImple을 전체 주석으로 처리 후 아래와 같이 실행한다.



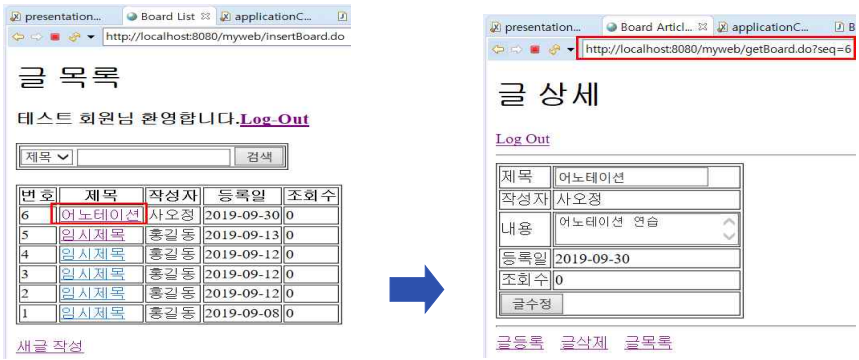
③ 글 상세 보기 구현

■ GetBoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import org.springframework.web.servlet.ModelAndView;
5  import tommy.spring.web.board.impl.BoardDAO;
6  @Controller
```

```
7  public class GetBoardController{
8          @RequestMapping("/getBoard.do")
9          public ModelAndView getBoard(BoardVO vo, BoardDAO boardDAO,
                                                    ModelAndView mav) {
10                 System.out.println("글 상세 보기 처리");
11                 mav.addObject("board", boardDAO.getBoard(vo));
12                 mav.setViewName("getBoard.jsp");
13                 return mav;
14         }
15 }
```

▢ 실행 결과 및 테스트

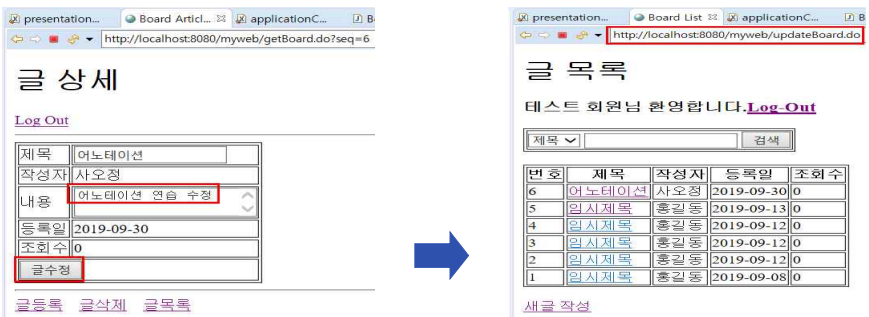


④ 글 수정 기능 구현하기

■ UpdateBoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import tommy.spring.web.board.impl.BoardDAO;
5  @Controller
6  public class UpdateBoardController {
7          @RequestMapping("/updateBoard.do")
8          public String updateBoard(BoardVO vo, BoardDAO boardDAO) {
9                  System.out.println("글 수정 기능 처리");
10                 boardDAO.updateBoard(vo);
11                 return "getBoardList.do";
12         }
13 }
```

▢ 실행 결과 및 테스트

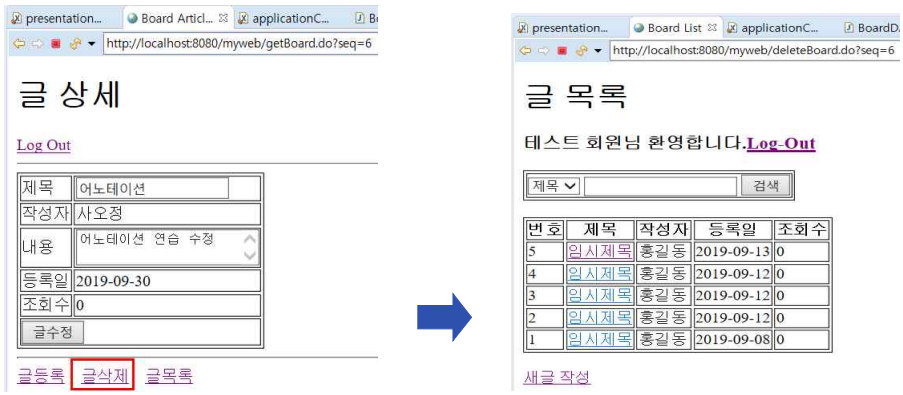


⑤ 글 삭제 기능 구현하기

■ DeleteBoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import tommy.spring.web.board.impl.BoardDAO;
5  @Controller
6  public class DeleteBoardController{
7          @RequestMapping("/deleteBoard.do")
8          public String deleteBoard(BoardVO vo, BoardDAO boardDAO) {
9                  System.out.println("글 삭제 처리");
10                 boardDAO.deleteBoard(vo);
11                 return "getBoardList.do";
12         }
13 }
```

▢ 실행 결과 및 테스트



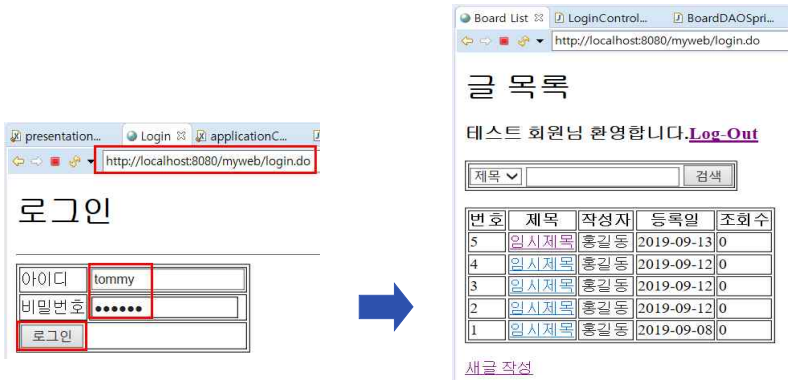
⑥ 로그인 기능 구현하기

■ LoginController 수정

```
1  package tommy.spring.web.user;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import tommy.spring.web.user.impl.UserDAO;
5  @Controller
6  public class LoginController {
7          @RequestMapping("/login.do")
8          public String login(UserVO vo, UserDAO userDAO) {
9                  System.out.println("로그인 처리");
10                 if (userDAO.getUser(vo) != null) {
11                         return "getBoardList.do";
12                 } else {
13                         return "login.jsp";
                   }
```

```
        }
}
```

▢ 실행 결과 및 테스트



⑦ 로그아웃 기능 구현하기

■ LogoutController 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  @Controller
6  public class LogoutController{
7          @RequestMapping("/logout.do")
8          public String logout(HttpSession session) {
9                  System.out.println("로그아웃 처리");
10                 session.invalidate();
11                 return "login.jsp";
12         }
13 }
```

▢ 실행 결과 및 테스트

⑧ 컨트롤러 통합하기

▢ 어노테이션을 이용하면 대부분의 Controller 클래스를 4~5줄 내외로 간단하게 구현할 수 있다. 이렇게 간단하게 구현되는 Controller의 경우 하나의 클래스로 묶어서 처리하면 관리가 편리할 것이다. 단 무조건 통합을 추천하는 것은 아니고 기능의 이 같은 경우 또 통합처리를 하여도 복잡성이 증가하지 않는 경우에 통합을 추천한다.

■ BoardController 구현

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import org.springframework.web.servlet.ModelAndView;
5  import tommy.spring.web.board.impl.BoardDAO;
6  @Controller
7  public class BoardController {
8          @RequestMapping("/insertBoard.do")
9          public String insertBoard(BoardVO vo, BoardDAO boardDAO) {
10                 System.out.println("글 등록 처리");
11                 boardDAO.insertBoard(vo);
12                 return "getBoardList.do";
13         }
14         @RequestMapping("/updateBoard.do")
15         public String updateBoard(BoardVO vo, BoardDAO boardDAO) {
16                 System.out.println("글 수정 기능 처리");
17                 boardDAO.updateBoard(vo);
18                 return "getBoardList.do";
19         }
20         @RequestMapping("/deleteBoard.do")
21         public String deleteBoard(BoardVO vo, BoardDAO boardDAO) {
22                 System.out.println("글 삭제 처리");
23                 boardDAO.deleteBoard(vo);
24                 return "getBoardList.do";
25         }
26         @RequestMapping("/getBoard.do")
27         public ModelAndView getBoard(BoardVO vo, BoardDAO boardDAO,
                                                    ModelAndView mav) {
28                 System.out.println("글 상세 보기 처리");
29                 mav.addObject("board", boardDAO.getBoard(vo));
30                 mav.setViewName("getBoard.jsp");
31                 return mav;
32         }
33         @RequestMapping("/getBoardList.do")
34         public ModelAndView getBoardList(BoardVO vo, BoardDAO boardDAO,
                                                    ModelAndView mav) {
35                 System.out.println("글 목록 검색 처리");
36                 mav.addObject("boardList", boardDAO.getBoardList(vo)); // Model 정보저장
37                 mav.setViewName("getBoardList.jsp"); // View 정보저장
38                 return mav;
39         }
```

```
40 }
```

■ BoardController 작성이 완료되면 나머지 Board 관련된 Controller 들을 삭제한다.
InsertBoardController, UpdateBoardController, DeleteBoardController, GetBoardController, GetBoardListController 이상 5개

▢ 여기까지 작업을 했으면 전체적으로 수행이 잘되는지 테스트 해 보자.

⑨ 요청방식에 따른 처리

▢ @RequestMapping 어노테이션에 method 속성을 설정하지 않을 경우 GET, POST등 모든 HTTP 전송 방식을 처리하게 된다.

▢ @RequestMapping 이용하면 마치 Service처럼 클라이언트의 요청 방식(GET/POST)에 따라 수행될 메서드를 다르게 처리할 수 있다.

■ LoginController 수정

```
1  package tommy.spring.web.user;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.web.bind.annotation.RequestMapping;
4  import org.springframework.web.bind.annotation.RequestMethod;
5  import tommy.spring.web.user.impl.UserDAO;
6  @Controller
7  public class LoginController {
8          @RequestMapping(value ="/login.do", method = RequestMethod.GET)
9          public String loginView(UserVO vo) {
10                 System.out.println("로그인 화면으로 이동");
11                 vo.setId("test");
12                 vo.setPassword("test");
13                 return "login.jsp";
14         }
15         @RequestMapping(value="/login.do", method = RequestMethod.POST)
16         public String login(UserVO vo, UserDAO userDAO) {
17                 System.out.println("로그인 인증 처리");
18                 if (userDAO.getUser(vo) != null) {
19                         return "getBoardList.do";
20                 } else {
21                         return "login.jsp";
22                 }
23         }
24 }
```

▢ 이제 로그인처리가 잘 되는지 테스트 해 보자.
: GET 방식으로 요청할 경우 로그인 인증처리를 수행하지 않고 바로 화면으로 이동되는 것을 확인할 수 있을 것이다.

■ 참고 : JSP에서 Command 객체사용 : EL 태그 이용

■ src/main/webapp/login.jsp 수정

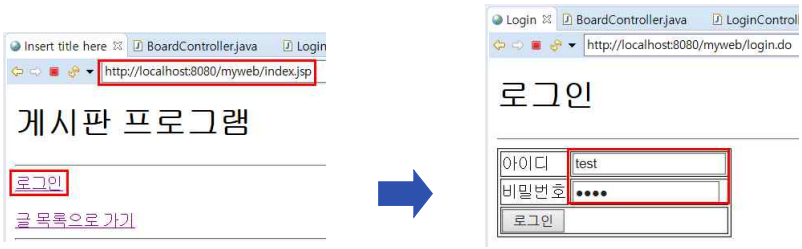
```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8">
6  <title>Login</title>
7  </head>
8  <body>
9  <h1>로그인</h1>
10 <hr>
11 <form action="login.do" method="post">
12 <table border="1">
13 <tr>
14          <td>아이디</td>
15          <td><input type="text" name="id"  value="${userVO.id}"/></td>
16 </tr>
17 <tr>
18          <td>비밀번호</td>
19          <td><input type="password" name="password" value="${userVO.password}"/></td>
20 </tr>
21 <tr>
22          <td colspan="2"><input type="submit" value="로그인" /></td>
23 </tr>
24 </table>
25 </form>
26 </body>
27 </html>
```

■ 위와 같이 login.jsp를 수정하였다면 테스트를 위하여 index.jsp를 작성한다.

■ src/main/webapp/index.jsp

```
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>Insert title here</title>
7  </head>
8  <body>
9  <h1>게시판 프로그램</h1>
10 <hr/><br/>
11 <a href="login.do">로그인</a><br></br>
12 <a href="getBoardList.do">글 목록으로 가기</a><br></br>
13 <hr/>
14 </body>
15 </html>
```

▢ 실행 결과



⑩ @ModelAttribute 사용

▢ 스프링 컨테이너가 생성하는 Command 객체의 이름은 클래스이름의 첫 글자를 소문자로 변경한 이름이 자동으로 설정된다. 이 이름을 변경하고자 할 때 @ModelAttribute를 사용할 수 있다.

	사용 예

LoginController	@RequestMapping(value ="/login.do", method = RequestMethod.GET)
	public String loginView(@ModelAttribute("user") UserVO vo) {
	System.out.println("로그인 화면으로 이동");
	vo.setId("test");
	vo.setPassword("test");
	return "login.jsp";
	}

login.jsp	<tr> <td>아이디</td> <td><input type="text" name="id" value="${user.id}"/></td> </tr> <tr> <td>비밀번호</td> <td><input type="password" name="password" value="${user.password}"/></td> </tr>

⑪ Servlet API 사용

▢ Controller 메서드에서 사용자가 입력한 정보를 추출하기 위해서 HttpServletRequest 대신 Command 객체를 사용하였다. 하지만 HttpServletRequest 객체가 사용자 입력 값을 추출할 때만 사용되는 것은 아니다. 즉 여러 경우에 HttpServletRequest를 직접 받아야 하는 경우가 생긴다.

▢ 스프링 MVC에서는 Controller 메서드의 매개변수로 다양한 Servlet API를 사용할 수 있도록 지원하고 있다.

▢ @RequestMapping 어노테이션이 적용된 메서드는 아래의 다섯 가지 타입의 파라미터를 전달 받을 수 있다.

▷ javax.servlet.http.HttpServletRequest / javax.servlet.ServletRequest

▷ javax.servlet.http.HttpServletResponse / javax.servlet.ServletResponse

▷ javax.servlet.http.HttpSession

▢ 스프링 MVC가 제공하는 어노테이션을 이용해서 헤더, 쿠키, 세션 등의 정보에 접근할 수 있기 때

문에 서블릿 API를 직접 사용해야 하는 경우는 드물지만 아래의 경우에는 서블릿 API를 사용하는 것이 유리하다.

▷ HttpSession의 생성을 직접 제어해야 하는 경우

▷ 컨트롤러에서 쿠키를 생성해야 하는 경우

▷ 개발자가 서블릿 API 사용을 선호하는 경우

☐ HttpSession사용 시 주의 : HttpSession타입의 파라미터를 가질 경우 세션이 생성됨

```
1  public ModelAndView process(HttpServletRequest request, ...) {
2          if(someCondition){
3                  HttpSession session = request.getSession(false);
4                  //세션이 없으면 생성하지 않음
5          }
6          // 중략 ...
7  }
8  @RequestMapping("/someURL")
9  public ModelAndView process(HttpSession session, ...) {
10         if(session != null){//항상 true
11         // 중략 ...
12 }
```

■ LoginController 수정

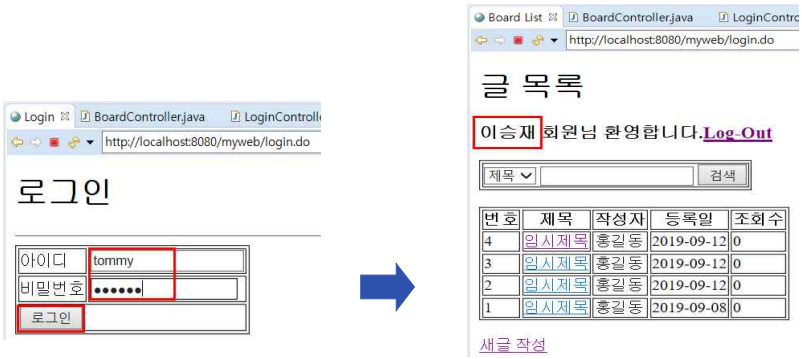
```
        <!-- 상단 부분 생략 -->
1       @RequestMapping(value = "/login.do", method = RequestMethod.POST)
2       public String login(UserVO vo, UserDAO userDAO, HttpSession session) {
3               System.out.println("로그인 인증 처리");
4               UserVO user = userDAO.getUser(vo);
5               if (user != null) {
6                       session.setAttribute("userName", user.getName());
7                       return "getBoardList.do";
8               } else {
9                       return "login.jsp";
10              }
11      }
12 }
```

■ getBoardList.jsp 파일 수정

```
1  <!-- 상단 부분 생략 -->
2  <html>
3  <head>
4  <meta charset="UTF-8">
5  <title>Board List</title>
6  </head>
7  <body>
8  <h1>글 목록</h1>
9  <h3>${userName} 회원님 환영합니다.<a href="logout.do">Log-Out</a></h3>
10 <!-- 검색 시작 -->
```

11	<!-- 하단 부분 수정 -->

▢ 자 이제 로그인을 수행하여 잘 처리되는지 확인해 보자.



⑫ Controller의 리턴타입

▢ Controller에 메서드를 정의할 때 리턴타입은 개발자 마음대로 결정할 수 있으며 리턴타입을 String으로 설정하면 완벽한 View 이름을 문자열로 리턴 하겠다는 것이고 ModelAndView를 설정하면 검색된 Model 데이터와 View 이름을 모두 저장하여 리턴 하겠다는 의미이다.

리턴 타입	소스코드 비교

ModelAndView	public ModelAndView login(UserVO vo, UserDAO userDAO,
	ModelAndView mav) {
	System.out.println("로그인 인증 처리");
	UserVO user = userDAO.getUser(vo);
	if (user != null) {
	mav.setViewName("getBoardList.jsp");
	} else {
	mav.setViewName("login.jsp");
	}
	return mav;
	}

String	public String login(UserVO vo, UserDAO userDAO){
	System.out.println("로그인 인증 처리");
	UserVO user = userDAO.getUser(vo);
	if(user != null){
	return "getBoardList.jsp";
	}else{
	return "login.jsp";
	}
	}

▢ 위 소스코드의 실행 결과는 같지만 대부분 프로젝트는 일관성 있는 코드를 중요하게 생각한다. 따라서 메서드 마다 다르게 설정하는 것보다 하나로 통일하여 사용하는 것을 추천한다.

■ 실습 : BoardController의 getBoard()와 getBoardList() 메서드의 리턴타입을 String으로 수정

```
    package tommy.spring.web.board;
1   import org.springframework.stereotype.Controller;
2   import org.springframework.ui.Model;
3   import org.springframework.web.bind.annotation.RequestMapping;
4   import tommy.spring.web.board.impl.BoardDAO;
5   @Controller
6   public class BoardController {
7           <!-- 중간 생략 -->
8           @RequestMapping("/getBoard.do")
9           public String getBoard(BoardVO vo, BoardDAO boardDAO, Model model) {
10                  System.out.println("글 상세 보기 처리");
11                  model.addAttribute("board", boardDAO.getBoard(vo));
12                  return "getBoard.jsp";
13          }
14          @RequestMapping("/getBoardList.do")
15          public String getBoardList(BoardVO vo, BoardDAO boardDAO, Model model) {
16                  System.out.println("글 목록 검색 처리");
17                  model.addAttribute("boardList", boardDAO.getBoardList(vo));
18                  return "getBoardList.jsp";
19          }
20  }
```

▢ 참고 : Controller 메서드의 리턴타입

리턴타입	설 명
ModelAndView	뷰 정보 및 모델 정보를 담고 있는 ModelAndView 객체
Model	뷰에 전달할 객체 정보를 담고 있는 Model을 리턴 한다. 이때 뷰 이름은 요청 URL로부터 결정된다. RequestToViewNameTranslator를 통해 뷰 결정.
Map	뷰에 전달할 객체 정보를 담고 있는 Map을 리턴 한다. 이때 뷰 이름은 요청 URL로부터 결정된다. RequestToViewNameTranslator를 통해 뷰 결정.
String	뷰 이름을 리턴 한다.
View 객체	View 객체를 직접 리턴. 해당 View 객체를 이용해서 뷰를 생성
void	메서드가 ServletResponse나 HttpServletResponse 타입의 파라미터를 갖는 경우 메서드가 직접 응답을 처리한다고 가정한다. 그렇지 않을 경우 요청 URL로부터 결정된 뷰를 보여준다. RequestToViewNameTranslator를 통해 뷰 결정.
@ResponseBody 어노테이션 적용	메서드에서 @RespnseBody 어노테이션이 적용된 경우 리턴 객체를 HTTP 응답으로 전송한다. HttpMessageConverter를 이용해서 객체를 HTTP 응답 스트림으로 변환한다.

3. 기타 어노테이션

① @RequestParam 사용하기

▢ Command 객체를 이용하면 클라이언트에서 넘겨준 요청 파라미터 정보를 받아낼 수 있다. 이를 위해서는 Command 객체에 Setter 메서드가 반드시 정의 되어 있어야 한다.

▢ 그런데 Command 객체에 없는 파라미터는 어떻게 전달 받을까?

스프링 MVC에서는 HTTP 요청 파라미터를 정보를 추출하기 위한 @RequestParam을 제공한다.

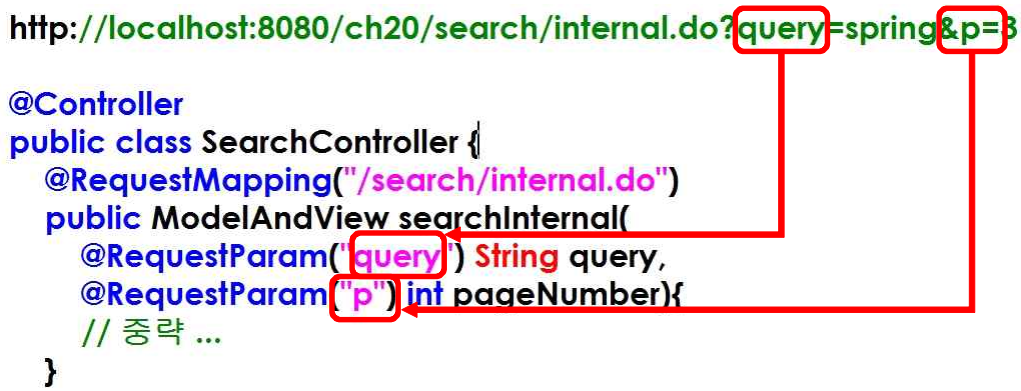
▢ 참고 : 컨트롤러 메서드의 파라미터 타입

■ 컨트롤러의 @RequestMapping 어노테이션이 적용된 메서드는 커맨드 클래스뿐만 아니라 HttpSerlvletRequest, HttpSession, Locale등 웹 어플리케이션과 관련된 다양한 타입의 파라미터를 가질 수 있다.

파라미터 타입	설 명
HttpServletRequest HttpServletResponse HttpSession	서블릿 API
java.util.Locale	현재 요청에 대한 Locale
InputStreram, Reader	요청 컨텐츠에 직접 접근할 때 사용
OutputStream, Writer	응답 컨텐츠를 생성할 때 사용
@PathVariable 어노테이션 적용 파라미터	URI 템플릿 변수에 접근할 때 사용
@RequestParam 어노테이션 적용 파라미터	HTTP 요청 파라미터를 매핑
@RequestHeader 어노테이션 적용 파라미터	HTTP 요청 헤더를 매핑
@CookieValue 어노테이션 적용 파라미터	HTTP 쿠키 매핑
@RequestBody 어노테이션 적용 파라미터	HTTP 요청의 몸체 내용에 접근할 때 사용. HttpMessageConverter를 이용해서 HTTP 요청 데이터를 해당 타입으로 변환한다.
Map, Model, ModelMap	뷰에 전달할 모델 데이터를 설정할 때 사용
커맨드 객체	HTTP 요청 파라미터를 저장한 객체, 기본적으로 클래스이름을 모델명으로 사용. @ModelAttribute 어노테이션을 사용하여 모델명을 설정할 수 있다.
Errors, BindingResult	HTTP 요청 파라미터를 커맨드 객체에 저장한 결과. 커맨드 객체를 위한 파라미터 바로 다음에 위치
SessionStatus	폼 처리를 완료했음을 처리하기 위해 사용. @SessionAttributes 어노테이션을 명시한 session 속성을 제거하도록 이벤트를 발생시킨다.

▢ @RequestParam 어노테이션을 이용한 파라미터 매핑

■ 컨트롤러를 구현하면서 가장 많이 사용되는 어노테이션이 바로 @RequestParam 어노테이션이다. @RequestParam 어노테이션은 HTTP 요청 파라미터를 메서드의 파라미터로 전달받을 때 사용된다.



■ @RequestParam 어노테이션이 적용된 파라미터가 String이 아닐 경우 실제 타입에 따라서 알맞게 타입 변환을 수행한다.

■ 만약 변환에 실패하면 스프링 MVC는 잘못된 요청(Bad Request)를 의미하는 400 응답코드를 웹 브라우저에 전송한다.
ex) http://localhost:8080/ch20/search/internal.do?query=spring&p=a (오류)

■ @RequestParam 어노테이션이 적용된 파라미터는 기본적으로 필수 파라미터이다.
ex) http://localhost:8080/ch20/search/internal.do?query=spring (오류)

■ 필수가 아닌 파라미터인 경우 required 속성 값을 false로 지정하면 된다.
ex) @RequestParam(value="query", required=false) String query

■ 필수가 아닌 요청 파라미터의 값이 존재하지 않을 경우 null값을 할당한다. 만약 null 값을 할당할 수 없는 기본 데이터 타입인 경우는 타입 변환 에러가 발생한다.

■ 따라서 이런 경우 defaultValue 속성을 이용해서 기본 값을 지정하면 된다.
ex) @RequestParam(value="p", defaultValue="1") int pageNumber

▢ @RequestParam 어노테이션 실습

■ getBoardList.jsp 수정

```
1  <!-- 상단 부분 생략 -->
2  <!-- 검색 시작 -->
3  <form action="getBoardList.do" method="post">
4  <table border="1">
5  <tr>
6          <td>
7                  <select name="searchCondition">
8                          <option value="TITLE">제목</option>
9                          <option value="CONTENT">내용</option>
10                 </select>
11                 <input type="text" name="searchKeyword" />
12                 <input type="submit" value="검색" />
13         </td>
14 </tr>
15 </table>
16 </form><br/>
17 <!-- 검색 종료 -->
18 <!-- 하단 부분 생략 -->
```

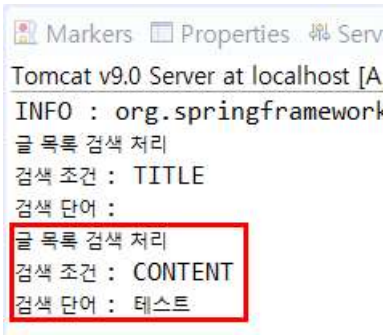
▢ 사용자가 글 목록 화면에서 검색조건과 검색 키워드를 입력하고 검색 버튼을 클릭하면 "/getBoardList.do"와 매핑된 getBoardList() 메서드가 실행된다. 그런데 BoardVO라는 Command 객체에는 searchCondition, searchKeyword라는 변수와 Setter 메서드가 없다.

▢ 따라서 BoardVO를 Command 객체로 사용할 수는 없다. 이러한 경우에 @RequestParam을 사용하면 검색과 관련된 파라미터를 추출할 수 있다. @RequestParam은 HttpServletRequest에서 제공하는 getParameter() 메서드와 같은 기능의 어노테이션이다.

■ BoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.ui.Model;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RequestParam;
6  import tommy.spring.web.board.impl.BoardDAO;
7  @Controller
8  public class BoardController {
9        @RequestMapping("/getBoardList.do")
10       public String getBoardList(
11           @RequestParam(value = "searchCondition", defaultValue = "TITLE", required = false)
                                                                  String condition,
12           @RequestParam(value = "searchKeyword", defaultValue = "", required = false)
                                          String keyword, BoardDAO boardDAO, Model model) {
13           System.out.println("글 목록 검색 처리");
14           System.out.println("검색 조건 : " + condition);
15           System.out.println("검색 단어 : " + keyword);
16           //model.addAttribute("boardList", boardDAO.getBoardList(vo)); // Model 정보저장
17           return "getBoardList.jsp";
18       }
19 }
```

▢ 위와 같이 수정 후 getBoardList.do를 요청하여 테스트 해 보자.



▢ 만약 @RequestParam을 사용하기 싫다면 BoardVO 클래스에 searchCondition, searchKeyword 변수를 추가하고 getter/setter 메서드만 추가하면 간단하게 처리할 수 있다.

■ BoardVO 수정

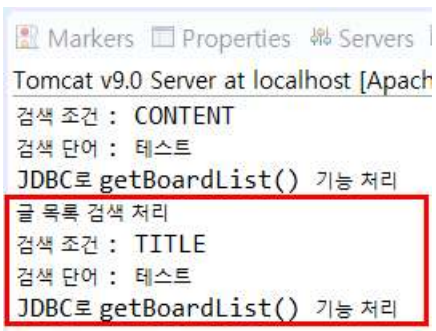
```
1  package tommy.spring.web.board;
2  import java.sql.Date;
3  public class BoardVO {
4          private int seq;
5          private String title;
6          private String writer;
7          private String content;
8          private Date regDate;
9          private int cnt;
10         private String searchCondition;
```

```
11          private String searchKeyword;
12          public String getSearchCondition() {
13                  return searchCondition;
14          }
15          public void setSearchCondition(String searchCondition) {
16                  this.searchCondition = searchCondition;
17          }
18          public String getSearchKeyword() {
19                  return searchKeyword;
20          }
21          public void setSearchKeyword(String searchKeyword) {
22                  this.searchKeyword = searchKeyword;
23          }
    <!-- 하단 부분 생략 -->
```

■ BoardController 수정

```
1  package tommy.spring.web.board;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.ui.Model;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import tommy.spring.web.board.impl.BoardDAO;
6  @Controller
7  public class BoardController {
8          <!-- 중간 부분 생략 -->
9          @RequestMapping("/getBoardList.do")
10         public String getBoardList(BoardVO vo, BoardDAO boardDAO, Model model) {
11                 System.out.println("글 목록 검색 처리");
12                 System.out.println("검색 조건 : " + vo.getSearchCondition());
13                 System.out.println("검색 단어 : " + vo.getSearchKeyword());
14                 model.addAttribute("boardList", boardDAO.getBoardList(vo));
15                 return "getBoardList.jsp";
16         }
17 }
```

▢ 위와 같이 수정 후 getBoardList.do를 요청하여 테스트 해 보자.



② @ModelAttribute 사용하기

▢ 우리는 앞에서 @ModelAttribute를 Controller 메서드의 매개변수로 선언된 Command 객체의 이름을 변경할 때 사용하였다.

▢ @ModelAttribute의 또 다른 기능은 View(JSP)에서 사용할 데이터를 설정하는 용도로 사용할 수 있다.

▢ @ModelAttribute가 설정된 메서드는 @RequestMapping 어노테이션이 적용된 메서드보다 먼저 호출된다. 또한 @ModelAttribute 메서드 실행결과로 리턴된 객체는 자동으로 Model에 저장된다.

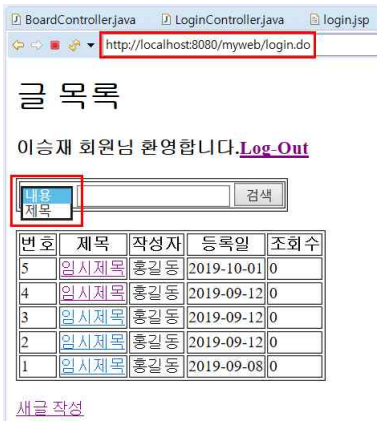
■ BoardController 수정

```
1  package tommy.spring.web.board;
2  import java.util.HashMap;
3  import java.util.Map;
4  import org.springframework.stereotype.Controller;
5  import org.springframework.ui.Model;
6  import org.springframework.web.bind.annotation.ModelAttribute;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import tommy.spring.web.board.impl.BoardDAO;
9  @Controller
10 public class BoardController {
11         @ModelAttribute("conditionMap")
12         public Map<String, String> searchConditionMap() {
13                 Map<String, String> conditionMap = new HashMap<String, String>();
14                 conditionMap.put("제목", "TITLE");
15                 conditionMap.put("내용", "CONTENT");
16                 return conditionMap;
17         }
   <!-- 하단 부분 생략 -->
```

■ getBoardList.jsp 수정

```
   <!-- 상단 부분 생략 -->
1  <!-- 검색 시작 -->
2  <form action="getBoardList.do" method="post">
3  <table border="1">
4  <tr>
5          <td>
6                  <select name="searchCondition">
7                          <c:forEach items="${conditionMap }" var="option">
8                                  <option value="${option.value }">${option.key }</option>
9                          </c:forEach>
10                 </select>
11                 <input type="text" name="searchKeyword" />
12                 <input type="submit" value="검색" />
13         </td>
14 </tr>
15 </table>
16 </form><br/>
17 <!-- 검색 종료 -->
   <!-- 하단 부분 생략 -->
```

▢ 이제 실행하여 로그인 한 후 글 목록 화면으로 이동해 보자.



▢ 결론 : 실행과정을 분석

■ 클라이언트가 "getBoardList.do" 요청을 전송하면

■ @ModelAttribute가 설정된 searchConditionMap() 메서드가 먼저 실행된다.

■ 그러면 @ModelAttribute로 지정한 이름으로 searchConditionMap() 메서드가 리턴 한 값을 Model 객체에 저장한다.

■ 그러고 나서야 클라이언트가 호출한 getBoardList() 메서드가 실행된다.

■ 이때 boardList라는 이름으로 검색 결과를 Model에 저장하면 최종적으로 Model에는 두 개의 컬렉션이 저장되게 된다.

③ @SessionAttributes 사용하기

▢ @SessionAttributes는 수정작업을 처리할 때 유용하게 사용할 수 있는 어노테이션이다. 예를 들어 상세화면에서 게시 글을 수정한다고 가정하면 글 수정 버튼을 클릭하면 사용자가 입력한 수정 내용을 가지고 "/updateBoard.do" 요청을 전송할 것이고 BoardController의 updateBoard() 메서드에서는 사용자가 입력한 정보를 이용해서 글 수정 작업을 처리할 것이다.

▢ 그런데 문제는 사용자가 입력한 정보가 제목과 내용뿐이고 작성자 정보는 전달되지 않았기 때문에 Command 객체인 BoardVO에 writer 정보가 저장되지 않는다.

▢ 물론 현재 상황에서 BoardDAO에 updateBoard()메서드가 제목과 내용 정보만 수정하도록 구현되었기 때문에 문제가 없다. 하지만 만약 아래처럼 SQL문을 수정하여 작성자 정보까지 전달하게 되면 writer 컬럼은 null 값으로 수정된다.

```
private final String BOARD_UPDATE =
        "update myboard set title=?, writer=?, content=? where seq=?";
```

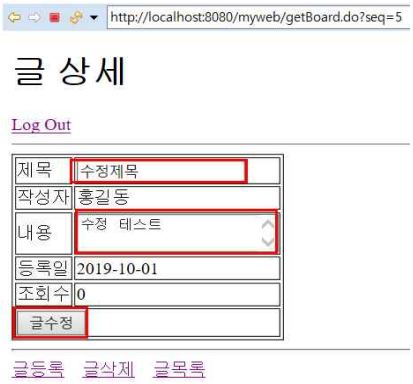
▢ 스프링 MVC에서는 이러한 문제를 방지하기 위해 @SessionAttribute를 제공하고 있다.

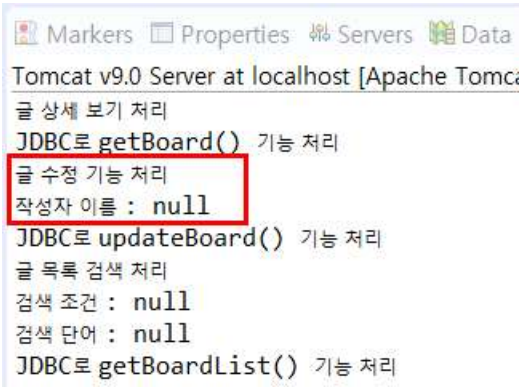
■ BoardController 수정

```
1 package tommy.spring.web.board;
2 import java.util.HashMap;
3 import java.util.Map;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.ModelAttribute;
```

```
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import tommy.spring.web.board.impl.BoardDAO;
9  @Controller
10 public class BoardController {
11         @RequestMapping("/updateBoard.do")
12         public String updateBoard(BoardVO vo, BoardDAO boardDAO) {
13                 System.out.println("글 수정 기능 처리");
14                 System.out.println("작성자 이름 : " + vo.getWriter());
15                 boardDAO.updateBoard(vo);
16                 return "getBoardList.do";
17         }
18 <!-- 하단 부분 생략 -->
```

▢ 수정작업을 수행하여 콘솔창에 나오는 정보를 확인해 보자.



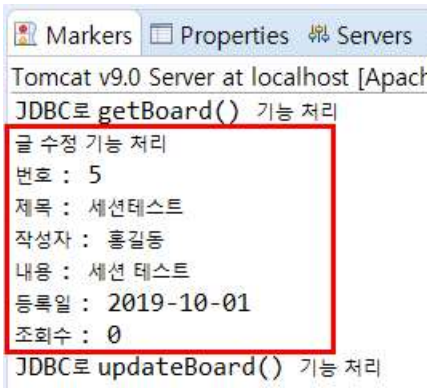


▢ 이제 결과를 확인했으니 @SessionAttributes를 이용하여 writer 컬럼 값이 null로 업데이트 되지 않도록 처리해 보자.

■ BoardController 수정

```
1  package tommy.spring.web.board;
2  import java.util.HashMap;
3  import java.util.Map;
4  import org.springframework.stereotype.Controller;
5  import org.springframework.ui.Model;
6  import org.springframework.web.bind.annotation.ModelAttribute;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import org.springframework.web.bind.annotation.SessionAttributes;
9  import tommy.spring.web.board.impl.BoardDAO;
10 @Controller
11 @SessionAttributes("board")
12 public class BoardController {
13         @RequestMapping("/updateBoard.do")
14         public String updateBoard(@ModelAttribute("board") BoardVO vo,
                                                    BoardDAO boardDAO) {
15                 System.out.println("글 수정 기능 처리");
16                 System.out.println("번호 : " + vo.getSeq());
17                 System.out.println("제목 : " + vo.getTitle());
```

```
18                      System.out.println("작성자 : " + vo.getWriter());
19                      System.out.println("내용 : " + vo.getContent());
20                      System.out.println("등록일 : " + vo.getRegDate());
21                      System.out.println("조회수 : " + vo.getCnt());
22                      boardDAO.updateBoard(vo);
23                      return "getBoardList.do";
24              }
25              @RequestMapping("/getBoard.do")
26              public String getBoard(BoardVO vo, BoardDAO boardDAO, Model model) {
27                      System.out.println("글 상세 보기 처리");
28                      model.addAttribute("board", boardDAO.getBoard(vo));
29                      return "getBoard.jsp";
30              }
<!-- 하단 부분 생략 -->
```



▢ 결론 : 사용자가 상세화면을 요청하면 getBoard() 메서드는 검색결과인 BoardVO 객체를 board라는 이름으로 Model에 저장한다. 이때 BoardController 클래스에 선언된 @SessionAttributes("board") 설정에 의해 Model에 "board"라는 이름으로 저장되는 데이터가 있다면 그 데이터를 세션에도 자동으로 저장하는 것이다.

4. 추가 학습

① @CookieValue 어노테이션을 이용한 쿠키 매핑

☐ @CookieValue 어노테이션을 이용하면 쿠키 값을 파라미터로 전달 받을 수 있다.

☐ 해당 쿠키가 존재하지 않으면 500에러를 발생한다.

☐ 쿠키가 필수가 아닌 경우에는 required 속성의 값을 false로 지정한다.

☐ defaultValue 속성을 이용해서 기본 값을 설정할 수 도 있다.

☐ 사용 예

```
1  @Controller
2  public class CookieController {
3      @RequestMapping("/cookie/make.do")
4      public String make(HttpServletResponse response) {
5              response.addCookie(new Cookie("auth", "1"));
6              return "cookie/made";
7      }
8      @RequestMapping("/cookie/view.do")
9      public String view(@CookieValue(value = "auth", defaultValue = "0") String auth) {
10             System.out.println("auth 쿠키: " + auth);
11             return "cookie/view";
12     }
13 }
```

② @RequestHeader 어노테이션을 이용한 헤더 매핑

☐ @RequestHeader 어노테이션을 이용하면 HTTP 요청 헤더의 값을 메서드의 파라미터로 전달 받을 수 있다.

```
1  @Controller
2  public class HeaderController {
3      @RequestMapping("/header/check.do")
4      public String check(@RequestHeader("Accept-Language") String languageHeader) {
5              System.out.println(languageHeader);
6              return "header/pass";
7      }
8  }
```