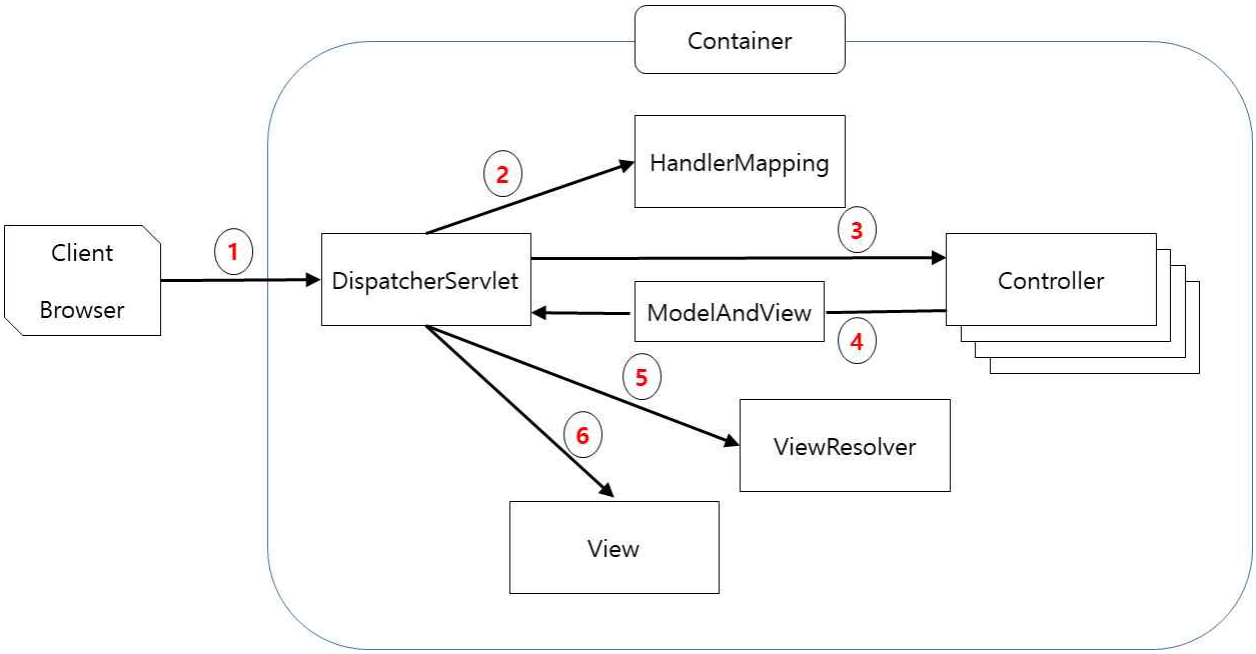
제 11 강 Spring MVC 구조

1. Spring MVC의 수행 흐름

우리는 앞에서 Spring MVC와 구조가 같은 Controller를 만들어 게시판 프로그램에 적용했다. 지금부터는 Spring MVC의 구조를 살펴볼 것인데 앞에서 우리가 개발한 MVC 프레임워크와 비교해보면 좀 더 쉽게 이해할 수 있을 것이다.



① 클라이언트로부터의 모든 "*.do" 요청을 DispatcherServlet이 받는다.
② DispatcherServlet은 HandlerMapping을 통해 요청을 처리할 Controller를 검색한다.
③ DispatcherServlet은 검색된 Controller를 실행하여 클라이언트의 요청을 처리한다.
④ Controller는 비즈니스 로직의 수행결과로 얻어낸 Model 정보와 Model을 보여줄 View 정보를 ModelAndView 객체에 저장하여 리턴한다.
⑤ DipspatcherServlet은 ModelAndView로부터 View 정보를 추출하고 ViewResolver를 이용하여 응답으로 사용할 View를 얻어낸다.
⑥ DispatcherServlet은 ViewResolver를 통해 찾아낸 View를 실행하여 응답을 전송한다.

※ Spring MVC 구조 흐름을 보면 앞에서 적용했던 MVC 프레임워크와 비슷하다는 것을 느낄 것이다. 유일하게 다른 점은 Controller의 리턴타입이 String이 아니라 ModelAndView로 바뀐 것이다.

2. DispatcherServlet 등록 및 스프링 컨테이너 구동

① DispatcherServlet 등록

☐ Spring MVC에서 가장 먼저 해야 할 일은 WEB-INF/web.xml 파일에 DispatcherServlet을 등록하는 것이다.

```
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         <servlet>
7             <servlet-name>action</servlet-name>
8             <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
9         </servlet>
10        <servlet-mapping>
11            <servlet-name>action</servlet-name>
12            <url-pattern>*.do</url-pattern>
13        </servlet-mapping>
14    </web-app>
```

☐ 위에 설정한 의미를 분석해 보면 사용자가 "*.do"라고 요청을 서버에 전달하면 컨테이너는 action 이라는 이름으로 등록된 DispatcherServlet 클래스 객체를 생성한다.

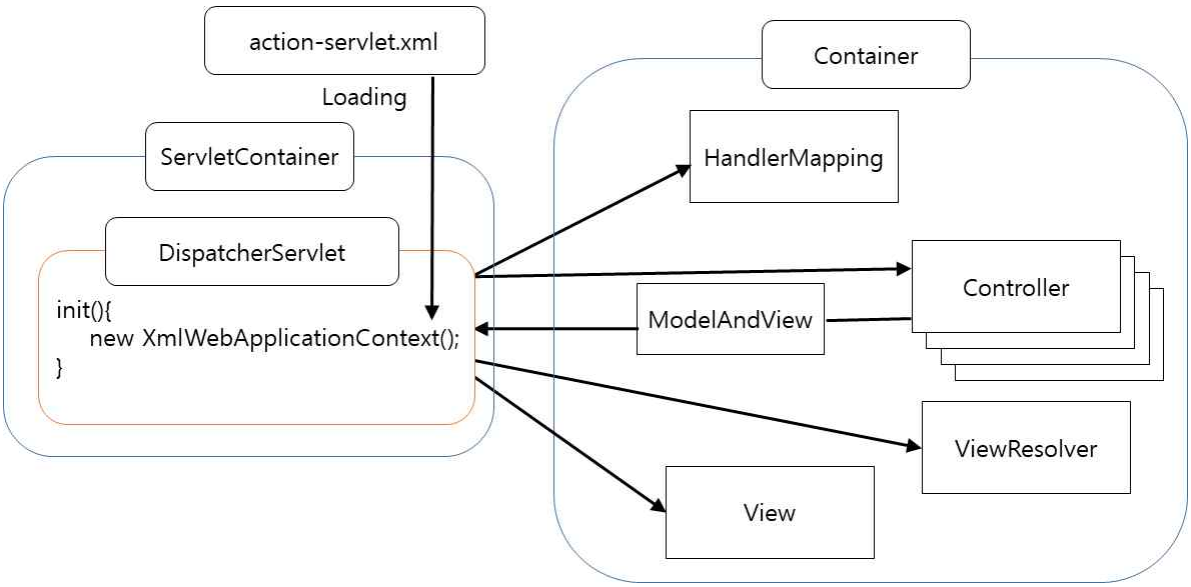
☐ DispatcherServlet 클래스에 재정의 된 init() 메서드가 자동으로 실행되어 XmlWebApplicationContext라는 스프링 컨테이너가 구동된다.

☐ XmlWebApplicationContext는 ApplicationContext를 구현한 클래스 중 하나이다.

☐ XmlWebApplicationContext는 우리가 직접 생성하는 것이 아니라 DispatcherServlet이 생성한다.

☐ DispatcherServlet 객체 혼자서는 클라이언트의 요청을 처리할 수 없고 반드시 HandlerMapping, Controller, ViewResolver 객체들과 상호작용해야 한다. 이 객체들을 메모리에 생성하기 위하여 DispatcherServlet은 스프링 컨테이너를 구동하는 것이다.

② 스프링 컨테이너의 구동과정 : XmlWebApplicationContext



☐ 서블릿 컨테이너가 DispatcherServlet 객체를 생성하고 나면 재정의 된 init() 메서드가 자동으로 실행된다. 그러면 init() 메서드는 스프링 설정 파일(action-servlet.xml)을 로딩하여 XmlWebApplicationContext를 생성한다.

☐ 스프링 설정파일(action-servlet.xml)에 DispatcherServlet이 사용할 HandlerMapping, Controller, ViewResolver 클래스를 <bean> 등록하면 스프링 컨테이너가 해당 객체들을 생성해준다.

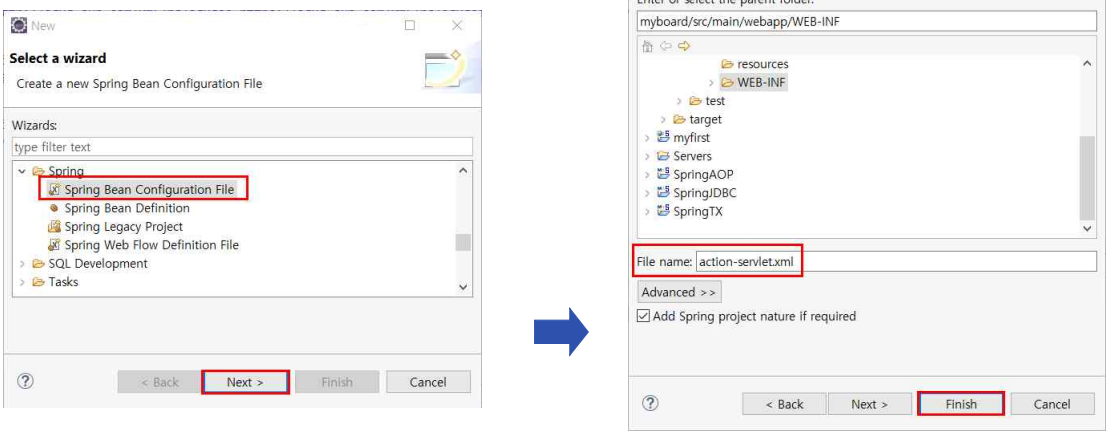
③ 스프링 설정 파일 등록

☐ 지금까지의 상태에서는 DispatcherServlet이 스프링 컨테이너를 구동할 때 무조건 /WEB-INF/action-servlet.xml 파일을 찾아서 로딩 한다. 그 이유는 DispatcherServlet은 Spring 컨테이너를 구동할 때 [서블릿이름-servlet.xml] 파일을 찾기 때문이다.

☐ DispatcherServlet이 스프링 컨테이너를 구동할 때 로딩 할 스프링 설정파일을 추가하자.

☐ WEB-INF 폴더를 선택 후 우 클릭하여 [New]-[Other]를 선택하고 [Spring]-[Spring Bean Configuration File]을 선택하고 [Next]를 클릭한다.

☐ action-servlet.xml 파일명을 입력하고 [Finish]를 클릭한다.



④ 스프링 설정 파일 변경

☐ DispatcherServlet은 자신이 사용할 객체들을 생성하기 위해서 스프링 컨테이너를 구동한다. 이때 스프링 컨테이너를 위한 설정 파일의 이름과 위치는 서블릿 이름을 기준으로 자동으로 결정된다.

☐ 하지만 서블릿 초기화 파라미터를 이용하면 설정파일의 이름이나 위치를 변경할 수 있다.

☐ /WEB-INF/config 폴더를 생성하고 action-servlet.xml 파일을 persentation-layer.xml 변경한다.

■ web.xml 수정

```
1  <!-- 상단 부분 생략 -->
2          <servlet>
3                  <servlet-name>action</servlet-name>
4                  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
5                  <init-param>
6                          <param-name>contextConfigLocation</param-name>
7                          <param-value>/WEB-INF/config/presentation-layer.xml</param-value>
8                  </init-param>
9          </servlet>
10         <servlet-mapping>
11                 <servlet-name>action</servlet-name>
12                 <url-pattern>*.do</url-pattern>
13         </servlet-mapping>
14 </web-app>
```

☐ 위와 같이 DispatcherServlet을 설정하면 스프링 컨테이너가 DispatcherServlet 객체를 설정한 후

아래 샘플코드와 같이 init()를 호출한다. 그리고 contextConfigLocation이라는 파라미터로 설정한 정보를 추출하여 스프링 컨테이너를 구동할 때 사용한다.

```
1  public class DispatcherServlet extends HttpServlet{
2          private String contextConfigLocation;
3          public void init(ServletConfig config) throws ServletException{
4                  contextConfigLocation = config.getInitParameter("contextConfigLocation");
5                  new XmlWebApplicationContext(contextConfigLocation);
6          }
7  }
```

⑤ 인코딩 설정

▢ 요청 파라미터의 캐릭터 인코딩이 ISO-8859-1이 아닌 경우 setCharacterEncoding() 메서드를 사용해서 인코딩을 해주어야 한다.

```
request.setCharacterEncoding("UTF-8");
```

▢ 모든 컨트롤러에 위와 같이 설정하기는 매우 번거롭다. 따라서 아래와 같이 web.xml 파일에 CharacterEncodingFilter 클래스를 설정함으로써 한 번에 처리할 수 있다.

```
<filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
                <param-name>encoding</param-name>
                <param-value>UTF-8</param-value>
        </init-param>
</filter>
<filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Spring MVC 적용

① 준비작업

▢ Spring MVC를 적용하기에 앞서 직접 개발했던 Controller 관련 파일들을 모두 삭제한다. (tommy.spring.web.controller 패키지 삭제)

▢ tommy.spring.web.controller 패키지를 삭제하면 tommy.spring.web 패키지로 시작하는 Controller 클래스들이 모두 컴파일 되지 않는다. 이제 LoginController 클래스를 비롯하여 기존에 사용했던 모든 Controller 클래스를 들을 스프링이 제공하는 Controller 인터페이스로 구현해야 한다.

■ LoginController 수정

```
1  package tommy.spring.web.user;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import org.springframework.web.servlet.ModelAndView;
5  import org.springframework.web.servlet.mvc.Controller;
6  import tommy.spring.web.user.impl.UserDAO;
7  public class LoginController implements Controller {
8          @Override
9          public ModelAndView handleRequest(HttpServletRequest request,
                                                 HttpServletResponse response) {
10                 System.out.println("로그인 처리");
                   <!-- 중간 부분 생략 -->
                   // 3. 화면 네비게이션
11                 ModelAndView mav = new ModelAndView();
12                 if (user != null) {
13                         mav.setViewName("getBoardList.do");
14                 } else {
15                         mav.setViewName("login.jsp");
16                 }
17                 return mav;
18         }
19 }
```

▢ 수정된 LoginController가 클라이언트의 “login.do” 요청에 동작하게 하려면 스프링 설정파일에 HandlerMapping과 LoginController를 <bean> 등록해야 한다.

■ presentation-layer.xml 수정

```
1  <!-- 상단 부분 생략 -->
2          <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
3                  <property name="mappings">
4                          <props>
5                                  <prop key="/login.do">login</prop>
6                          </props>
7                  </property>
8          </bean>
9          <bean id="login" class="tommy.spring.web.user.LoginController"></bean>
10 </beans>
```

▢ login.jsp 파일을 실행하여 등록되지 않은 아이디로 로그인을 시도하면 로그인 화면으로 돌아온다. 하지만 로그인에 성공하게 되면 뷰를 getBoardList.do 요청이 서버에 전달되지만 SimplUrlHanderMapping에는 그러한 정보가 없으므로 404 에러 화면이 출력되게 된다.

③ 글 목록 검색기능 구현

■ GetBoardListController 구현

```
1  package tommy.spring.web.board;
2  import java.util.List;
3  import javax.servlet.http.HttpServletRequest;
4  import javax.servlet.http.HttpServletResponse;
5  import org.springframework.web.servlet.ModelAndView;
6  import org.springframework.web.servlet.mvc.Controller;
7  import tommy.spring.web.board.impl.BoardDAO;
8  public class GetBoardListController implements Controller {
9          @Override
10         public ModelAndView handleRequest(HttpServletRequest request,
                                              HttpServletResponse response) {
                   <!-- 중간 부분 생략 -->
11                 // 3. 응답 화면 구성
12                 ModelAndView mav = new ModelAndView();
13                 mav.addObject("boardList", boardList); // Model 정보저당
14                 mav.setViewName("getBoardList.jsp"); // View 정보저장
15                 return mav;
16         }
17
18 }
```

▢ 위 코드에서 중요한 점은 검색 결과를 세션이 아닌 ModelAndView 객체에 저장하고 있다는 점이다. 세션이라는 객체는 클라이언트의 브라우저 하나당 하나씩 서버 메모리에 생성되어 클라이언트의 상태정보를 유지하기 위해서 사용된다. 따라서 세션에 많은 정보가 저장되면 서버에 부담을 줄 수 밖에 없다.

▢ 따라서 검색 결과는 세션이 아닌 HttpServletRequest 객체에 저장하는 것이 맞다. HttpServletRequest는 클라이언트의 요청으로 생성되었다가 응답 메시지가 클라이언트로 전송되면 자동으로 삭제되는 일회성 객체이다.

▢ ModelAndView는 클래스 이름에서 알 수 있듯이 Model과 View정보를 모두 저장하여 리턴할 때 사용한다.

■ HandlerMapping 등록

```
   <!-- 상단 부분 생략 -->
1      <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
2              <property name="mappings">
3                      <props>
4                              <prop key="/login.do">login</prop>
5                              <prop key="/getBoardList.do">getBoardList</prop>
6                      </props>
7              </property>
8      </bean>
9      <bean id="login" class="tommy.spring.web.user.LoginController"></bean>
10     <bean id="getBoardList" class="tommy.spring.web.board.GetBoardListController"></bean>
11 </beans>
```

▢ 이제 서버를 구동하여 로그인이 성공했을 때 글 목록화면이 정상적으로 출력되는지 실행해 보자.

※ 지금까지의 처리 과정을 요약하면 아래와 같다.

☐ 클라이언트로부터 "/getBoardList.do" 요청을 전송하면 DispatcherServlet이 요청을 받고

☐ SimpleUrlHandlerMapping을 통해 요청을 처리할 GetBoardListController를 검색한다.

☐ DispatcherServlet은 검색된 GetBoardListController를 실행하여 요청을 처리한다.

☐ GetBoardListController는 검색결과인 List<BoardVO>와 getBoardList.jsp 이름을 ModelAndView 객체에 저장하여 리턴한다.

☐ DispatcherServlet은 ModelAndView로부터 View정보를 추출하고 ViewResolver를 이용하며 응답으로 사용할 getBoardList.jsp를 검색한다.

☐ DispatcherServlet은 getBoardList.jsp를 실행하여 글 목록 화면을 전송한다.

④ 글 상세 조회 기능 구현

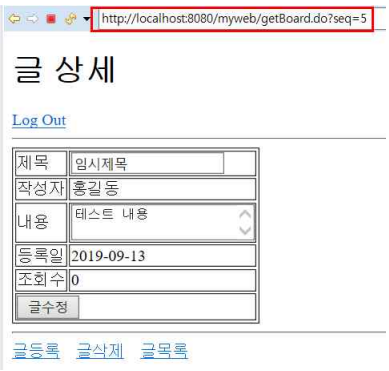
■ GetBoardController 구현

```
1  package tommy.spring.web.board;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import org.springframework.web.servlet.ModelAndView;
5  import org.springframework.web.servlet.mvc.Controller;
6  import tommy.spring.web.board.impl.BoardDAO;
7  public class GetBoardController implements Controller {
8          @Override
9          public ModelAndView handleRequest(
                                  HttpServletRequest request, HttpServletResponse response) {
                   <!-- 중간 부분 생략 -->
10                 // 3. 응답 화면 구현
11                 ModelAndView mav = new ModelAndView();
12                 mav.addObject("board", board);
13                 mav.setViewName("getBoard.jsp");
14                 return mav;
15         }
16
17 }
```

■ HandlerMapping 등록

```
<!-- 상단 부분 생략 -->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
                <props>
                        <prop key="/login.do">login</prop>
                        <prop key="/getBoardList.do">getBoardList</prop>
                        <prop key="/getBoard.do">getBoard</prop>
                </props>
        </property>
</bean>
<bean id="login" class="tommy.spring.web.user.LoginController"></bean>
<bean id="getBoardList" class="tommy.spring.web.board.GetBoardListController"></bean>
<bean id="getBoard" class="tommy.spring.web.board.GetBoardController"></bean>
```

▢ 실행 및 결과확인 : 글 목록 화면에서 제목을 클릭하여 상세화면이 출력되는지 확인해 보자.



⑤ 글 등록, 수정, 삭제, 로그아웃 기능 구현하기

■ InsertBoardController 구현

```
package tommy.spring.web.board;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import tommy.spring.web.board.impl.BoardDAO;
public class InsertBoardController implements Controller {
        @Override
        public ModelAndView handleRequest(
                        HttpServletRequest request, HttpServletResponse response) {
                <!-- 중간 부분 생략 -->
                // 3. 화면 네비게이션
                ModelAndView mav = new ModelAndView();
                mav.setViewName("getBoardList.do");
                return mav;
        }
}
```

■ UpdateBoardController 구현

```
1  package tommy.spring.web.board;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import org.springframework.web.servlet.ModelAndView;
5  import org.springframework.web.servlet.mvc.Controller;
6  import tommy.spring.web.board.impl.BoardDAO;
7  public class UpdateBoardController implements Controller {
8          @Override
9          public ModelAndView handleRequest(
                                  HttpServletRequest request, HttpServletResponse response) {
                   <!-- 중간 부분 생략 -->
10                 // 3. 화면 네비게이션
11                 ModelAndView mav = new ModelAndView();
12                 mav.setViewName("getBoardList.do");
13                 return mav;
14         }
15 }
```

■ DeleteBoardController 구현

```
1  package tommy.spring.web.board;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import org.springframework.web.servlet.ModelAndView;
5  import org.springframework.web.servlet.mvc.Controller;
6  import tommy.spring.web.board.impl.BoardDAO;
7  public class DeleteBoardController implements Controller {
8          @Override
9          public ModelAndView handleRequest(
                                  HttpServletRequest request, HttpServletResponse response) {
                   <!-- 중간 부분 생략 -->
10                 // 3. 화면 네비게이션
11                 ModelAndView mav = new ModelAndView();
12                 mav.setViewName("getBoardList.do");
13                 return mav;
14         }
15 }
```

■ LogoutController 구현

```
1  package tommy.spring.web.user;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import javax.servlet.http.HttpSession;
5  import org.springframework.web.servlet.ModelAndView;
6  import org.springframework.web.servlet.mvc.Controller;
7  public class LogoutController implements Controller {
8          @Override
```

```
9           public ModelAndView handleRequest(
                                    HttpServletRequest request, HttpServletResponse response) {
                <!-- 중간 부분 생략 -->
10              // 2. 세션 종료 후 메인 화면으로 이동
11              ModelAndView mav = new ModelAndView();
12              mav.setViewName("login.jsp");
13              return mav;
14          }
15  }
```

■ HandlerMapping 등록

```
    <!-- 상단 부분 생략 -->
1   <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
2           <property name="mappings">
3               <props>
4                   <prop key="/login.do">login</prop>
5                   <prop key="/getBoardList.do">getBoardList</prop>
6                   <prop key="/getBoard.do">getBoard</prop>
7                   <prop key="/insertBoard.do">insertBoard</prop>
8                   <prop key="/updateBoard.do">updateBoard</prop>
9                   <prop key="/deleteBoard.do">deleteBoard</prop>
10                  <prop key="/logout.do">logout</prop>
11              </props>
12          </property>
13  </bean>
14  <bean id="login" class="tommy.spring.web.user.LoginController"></bean>
15  <bean id="getBoardList" class="tommy.spring.web.board.GetBoardListController"></bean>
16  <bean id="getBoard" class="tommy.spring.web.board.GetBoardController"></bean>
17  <bean id="insertBoard" class="tommy.spring.web.board.InsertBoardController"></bean>
18  <bean id="updateBoard" class="tommy.spring.web.board.UpdateBoardController"></bean>
19  <bean id="deleteBoard" class="tommy.spring.web.board.DeleteBoardController"></bean>
20  <bean id="logout" class="tommy.spring.web.user.LogoutController"></bean>
21  </beans>
```

▢ 이제 전체적으로 진행하여 실행이 잘 되는지 확인해 보자. 아마 몇 군데서 이상하게 동작하는 곳이 보일 것이다. 그것은 글 목록을 요청할 때 getBoardList.do로 요청해야 되는데 getBoardList.jsp로 링크가 걸려있기 때문이다.

▢ getBoard.jsp 맨 아래 부분과 insertBoard.jsp 아랫부분을 수정하면 된다.

4. ViewResolver 활용하기.

▢ 우리는 스프링 설정파일인 presentation-layer.xml에 HandlerMapping, Controller 클래스들을 <bean> 등록하여 Spring 컨테이너가 객체를 생성하도록 하였다.

▢ 그런데 아직 적용하지 않은 요소가 한 가지 있다. 그것은 바로 ViewResolver이다.

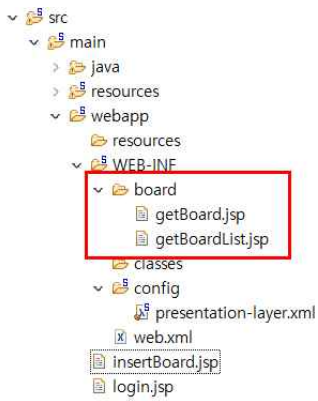
▢ ViewResolver를 이용하면 클라이언트로부터 직접적인 JSP 호출을 차단할 수 있어서 대부분의 웹 프로젝트에서 ViewResolver는 필수적으로 사용한다.

▢ JSP를 View로 이용하는 경우에는 InternalResourceViewResolver를 사용한다.

■ 그렇다면 브라우저에서 직접 JSP를 호출하면 어떠한 문제가 발생할까?
http://localhost:8080/myweb/getBoardList.jsp를 직접 입력하여 실행하여 보자.

▢ 위의 결과를 확인해 보면 JSP를 직접 호출할 경우 오류는 발생하지 않지만 아무런 데이터도 출력되지 않는 것을 확인할 수 있다. 이것은 getBoardList.jsp 파일이 실행되기 전에 GetBoardListController가 실행되지 않기 때문이다.
▢ 따라서 사용자가 getBoardList.jsp를 직접 호출하면 에러를 발생시키고 GetBoardListController부터 실행할 수 있도록 제어해야 하는데 이때 ViewResovler를 이용하면된다.

① ViewResolver 적용
■ /WEB-INF/board 폴더를 생성한다.
■ 기존에 사용했던 목록화면(getBoardList.jsp)와 상세화면(getBoard.jsp)을 이동시킨다.



▢ WEB-INF 폴더는 절대 브라우저에서 접근할 수 없다. 따라서 WEB-INF 폴더로 이동한 JSP 파일들은 클라이언트 브라우저에서 접근할 수 없다.

■ presentation-layer.xml 수정

```
        <!-- 상단 부분 생략 -->
        <!--ViewResovler 등록 -->
1       <bean id="viewResolver"
                class="org.springframework.web.servlet.view.InternalResourceViewResolver">
2               <property name="prefix" value="/WEB-INF/board/"></property>
3               <property name="suffix" value=".jsp"></property>
4       </bean>
5   </beans>
```

② InternalResourceViewResolver 설정 방법

■ InternalResourceViewResolver 클래스는 JSP나 HTML 파일과 같이 웹 어플리케이션의 내부 자원을 이용하여 뷰를 생성하는 AbstractUrlBasedView 타입의 뷰 객체를 리턴 한다. 기본적으로 사용하는 클래스는 InternalResourceView 클래스 이다.

■ InternalResourceViewResolver 클래스 설정 예

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/WEB-INF/viewjsp/" />
        <property name="suffix" value=".jsp" />
</bean>
```

■ 만약 컨트롤러에서 ModelAndView를 "hello"라고 지정했다고 가정하면 아래와 같이 InternalResourceViewResolver가 사용하는 경로가 결정된다.

■ 기본적으로 사용되는 InternalResourceViewResolver 클래스는 단순히 지정한 자원 경로로 요청을 전달한다. 만약 스프링의 국제화 관련 정보를 JSTL에서 사용하고 싶다면 JstlView 클래스를 viewClass 프로퍼티로 지정하면 된다. (위의 예에서 파란색 부분)

③ Controller 수정

▢ ViewResolver를 적용했을 때 ModelAndView 객체에 저장되는 View 이름은 ViewResolver 설정을 고려하여 등록해야 한다. 따라서 앞에서 작성한 Controller 클래스들의 내비게이션 코드를 수정하자.

■ LoginController

```
                <!-- 상단 부분 생략 -->
                // 3. 화면 네비게이션
1               ModelAndView mav = new ModelAndView();
2               if (user != null) {
3                       mav.setViewName("redirect:getBoardList.do");
4               } else {
5                       mav.setViewName("redirect:login.jsp");
6               }
                <!-- 하단 부분 생략 -->
```

▢ 위 코드처럼 View 이름 앞에 redirect: 이 붙여 있다면 ViewResolver가 설정되어 있더라도 이를

무시하고 리다이렉트 한다.

■ GetBoardListController

```
                <!-- 상단 부분 생략 -->
                // 3. 응답 화면 구성
1               ModelAndView mav = new ModelAndView();
2               mav.addObject("boardList", boardList); // Model 정보저당
3               mav.setViewName("getBoardList"); // View 정보저장
4               return mav;
                <!-- 하단 부분 생략 -->
```

■ GetBoardController

```
                <!-- 상단 부분 생략 -->
                // 3. 응답 화면 구현
1               ModelAndView mav = new ModelAndView();
2               mav.addObject("board", board);
3               mav.setViewName("getBoard");
4               return mav;
                <!-- 하단 부분 생략 -->
```

■ InsertBoardController, UpdateBoardController, DeleteBoardController

```
                <!-- 상단 부분 생략 -->
                // 3. 화면 네비게이션
1               ModelAndView mav = new ModelAndView();
2               mav.setViewName("redirect:getBoardList.do");
3               return mav;
                <!-- 하단 부분 생략 -->
```

■ LogoutController

```
                <!-- 상단 부분 생략 -->
                // 2. 세션 종료 후 메인 화면으로 이동
1               ModelAndView mav = new ModelAndView();
2               mav.setViewName("redirect:login.jsp");
3               return mav;
                <!-- 하단 부분 생략 -->
```

▢ 자 이제 서버를 구동하고 전체적으로 실행이 잘 되는지 확인해 보자.

▢ 지금까지 잘 따라 하였다면 아마 정상적으로 수행이 잘 되는 것을 확인할 수 있을 것이다.
하지만 지금까지 우리가 구현한 방식은 복잡한 Spring MVC의 구조를 좀 더 직관적으로 확인하기 위하여 XML 설정과 Controller 인터페이스를 상속받아서 구현하였다. 하지만 Spring 3.0 부터는 Controller 인터페이스를 사용하는 것 보다. @Controller 어노테이션을 이용하여 Controller를 구현하도록 추천하고 있다.

5. 추가학습 : DispatcherServlet 설정과 ApplicationContext의 관계

■ DispatcherServlet은 클라이언트의 요청을 중앙에서 처리하는 스프링 MVC의 핵심 구성요소이다. web.xml 파일에 한 개 이상의 DispatcherServlet을 설정할 수 있으며 각 DispatcherServlet은 한 개의 WebApplicationContext를 갖게 된다.

■ 또한 각 DispatcherServlet이 공유하는 빈을 설정할 수 도 있다.

① DispatcherServlet 설정

■ DispatcherServlet은 기본적으로 웹 어플리케이션의 /WEB-INF/[서블릿이름]-servlet.xml 파일로부터 스프링 설정 정보를 읽어온다.

```
<servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

■ 한 개 이상의 설정파일을 사용해야 하는 경우나 기본설정이 아닌 다른 이름을 사용하고 싶다면 contextConfigLocation 초기화 파라미터를 설정하면 된다.

■ contextConfigLocation 초기화 파라미터는 설정 파일 목록을 값으로 갖는데 이때 각 설정파일은 콤마(","), 공백문자(" "), 탭("Wt"), 줄바꿈("Wn"), 세미콜론(";")을 이용하여 구분한다.

```
<servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>
                        /WEB-INF/main.xml
                        /Web-INF/board.xml
                </param-value>
        </init-param>
</servlet>
```

② 웹 어플리케이션을 위한 ApplicationContext 설정

■ DispatcherServlet은 그 자체가 서블릿이기 때문에 한 개 이상의 DispatcherServlet을 설정하는 것이 가능하다.

```
<servlet>
        <servlet-name>front</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/front.xml</param-value>
        </init-param>
</servlet>
<servlet>
        <servlet-name>rest</servlet-name>
```

```
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/rest.xml</param-value>
        </init-param>
</servlet>
```

■ 위 경우 두 DispatcherServlet은 각각 별도의 WebApplicationContext를 생성하게 된다.

■ 주의 : front.xml에서는 rest.xml에 설정된 빈 객체를 사용할 수 없다.

■ 만약 공통의 빈의 필요로 하는 경우는 ContextLoaderListener를 사용하여 공통으로 사용될 빈을 설정할 수 있다. 아래와 같이 ContextLoaderListener를 ServletListener로 등록하고 contextConfigLocation 컨텍스트 파라미터를 이용하여 공통으로 사용될 빈 정보를 담고 있는 설정 파일 목록을 지정하면 된다.

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                /WEB-INF/service.xml
                /WEB-INF/persistence.xml
        </param-value>
</context-param>
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
        <servlet-name>front</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet>
        <servlet-name>rest</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

■ ContextLoaderListener가 생성하는 WebApplicationContext는 웹 어플리케이션에서 루트 컨텍스트가 되며 DispatcherServlet이 생성하는 WebApplicationContext는 루트 컨텍스트를 부모로 사용하는 자식 컨텍스트가 된다.

■ ContextLoaderListener는 contextConfigLocation 컨텍스트 파라미터를 명시하지 않으면 /WEB-INF/applicationContext.xml을 설정파일로 사용한다.

■ 만약 클래스 패스에 위치한 파일로부터 설정 정보를 읽어오고 싶다면 "classpath:" 접두어를 사용해서 명시하면 된다.

ex) <param-value>classpath:common.xml</param-value>