■ 웹개발 방식

1) 웹개발방식은 스크립트 방식 / MODEL1 / MODEL2 방식이 존재

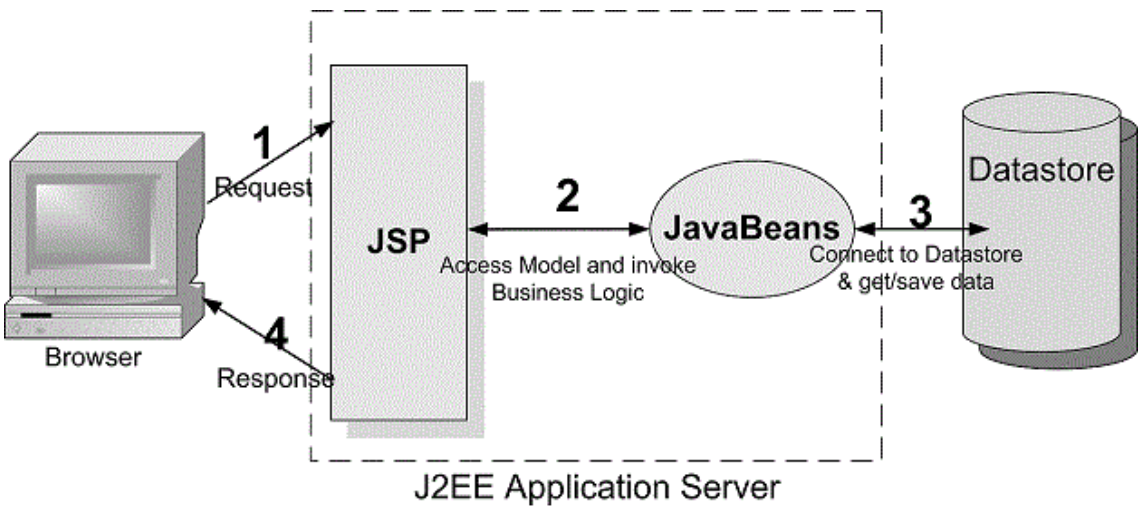
2) 스크립트 방식은 모든 코드를 jsp에

- 중복된 코드가 많아짐
- 자바코드와 HTML이 혼재되어 유지보수가 어려움

> 상품 전체를 리스트로 불러오는 자바코드가 존재한다.
> 만약 이 코드를 관리자페이지(admin.jsp), 메인페이지(index.jsp), 상품 상세페이지(productList.jsp)등에서 사용한다고 한다면, 같은 자바코드를 각 페이지에 모두 똑같이 써줘야 한다.

3) MODEL1방식은 jsp와 Java파일들로(웹에서는 자바빈이라고 부름)

- 디자인코드(HTML)와 자바코드(비즈니스 로직)를 구분하지 않고 하나의 jsp파일내에 함께 기술해서 웹 프로그램을 제작하는 방식
- 개발하기 쉽고 배우기 쉽다.
- 스크립트 방식보다 중복된 코드가 현저하게 줄어듦(코드의 재사용성)
- 여전히 중복된 코드가 존재함
- 디자인(HTML)과 비즈니스 로직의 구분이 명확하지 않음



글을 삭제하는 delete.jsp를 유저가 주소창에 입력하면 접근할 수 있다. 만약 로그인 되어있음을 확인하지 않는다면 삭제할 수도 있다.

■ MODEL2(MVC패턴)

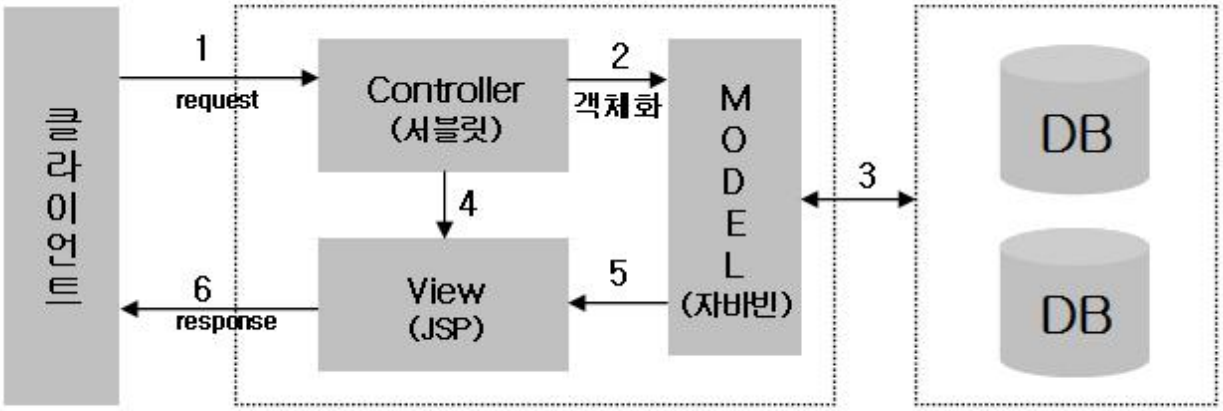
1) Model2방식은 웹 어플리케이션을 개발할 때 MVC패턴을 적용하여 웹 어플리케이션의 개발이 가능하도록 구현한 것

2) MVC는 Model-View-Controller로 각각의 역할을 나누어서 개발하는 하는 방식을 말함

- Model은 데이터 그 자체
- View는 client가 직접 사용하는 부분이며, Model에서 생성된 Data를 client에게 보여주는 역할을 담당한다. 웹에서는 JSP가 담당함
- Controller는 사용자의 요청을 받아서 요청에 해당하는 비즈니스로직을 수행하도록 하고, 응답을 client에 보내는 역할을 함. 웹에서는 서블릿이 담당

3) 단점 : 초기 설계에 많은 시간이 소요된다.
개발자에게 MVC패턴에 대한 개념이 필수적으로 요구된다.

4) 장점 : 디자인코드와 비즈니스 로직이 분리된다.
비즈니스 로직의 재사용성이 높아진다.
비즈니스로직 계층의 확장성이 용이하다.
유지보수하기 편하다.



Model2 방식의 웹 개발 방법

■ Controller의 역할

사용자의 요청을 받는다.

사용자의 요청을 분석한다.

사용자의 요청을 처리할 자바 빈의 생성하고, 비즈니스 로직이 구현된 메소드를 실행한다.

비즈니스 로직 수행 후 사용자의 요청을 JSP페이지나 혹은 특정 URL로 이동시킨다.

■ Model의 역할

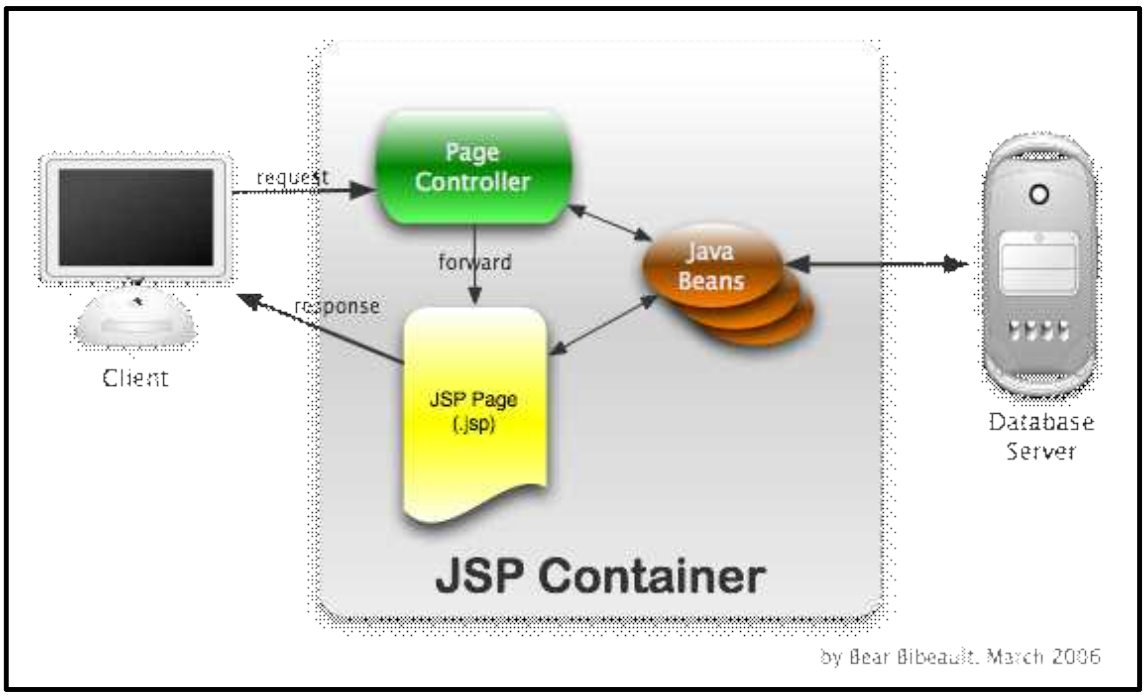
데이터 그 자체

■ View의 역할

클라이언트에게 최종적으로 보여지는 영역이다.

웹에서는 JSP를 이용해서 구현한다.

■ Front-Controller 패턴



1) 클라이언트의 모든 요청을 하나의 컨트롤러(서블릿)에서 처리

2) 로그인 처리나 input처리등을 공통된 방법으로 처리

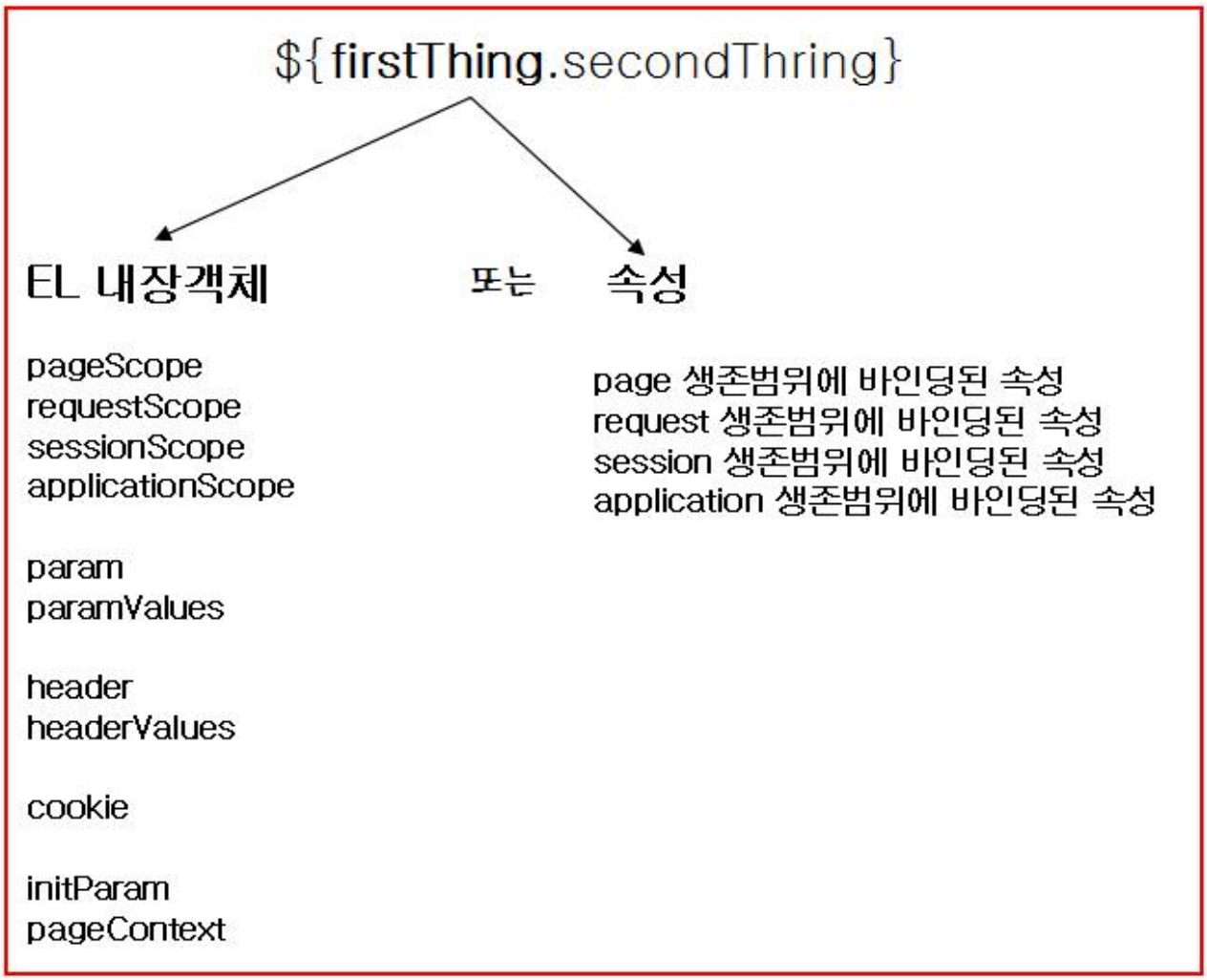
□ EL(Expression Language)

■ EL은 JSP 스펙 2.0부터 JSP 표현식 언어로 추가되었다.

■ EL은 Attribute, request의 파라미터, ServletContext의 초기화 파라미터등에 접근할 수 있다.

■ EL 표현식은 항상 중괄호로 묶고 제일 앞에 달러 기호($)를 붙인다.

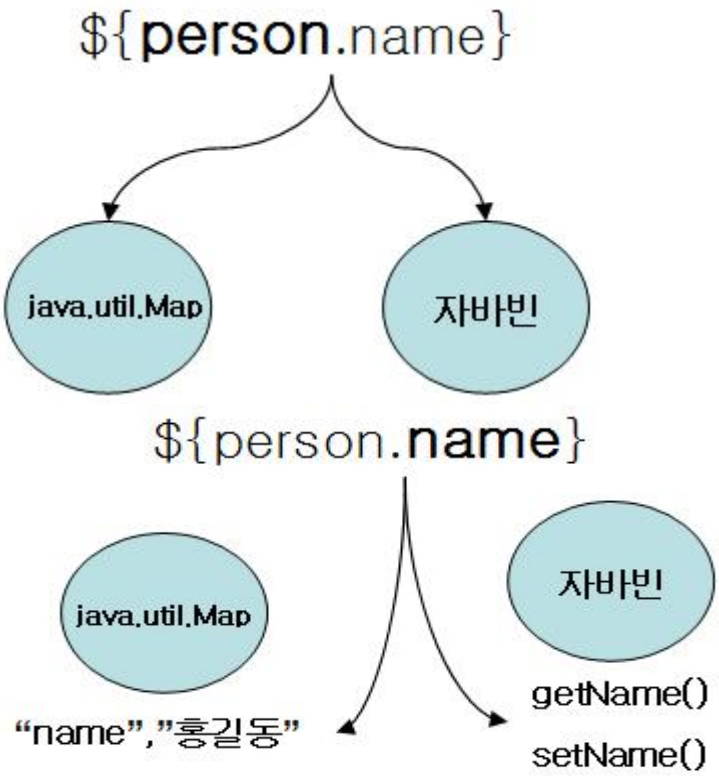
▮ 형식 : ${person.name}



■ 도트 연산자 : Attribute에 저장된 맵이나 자바 빈의 값을 표현할 수 있다.

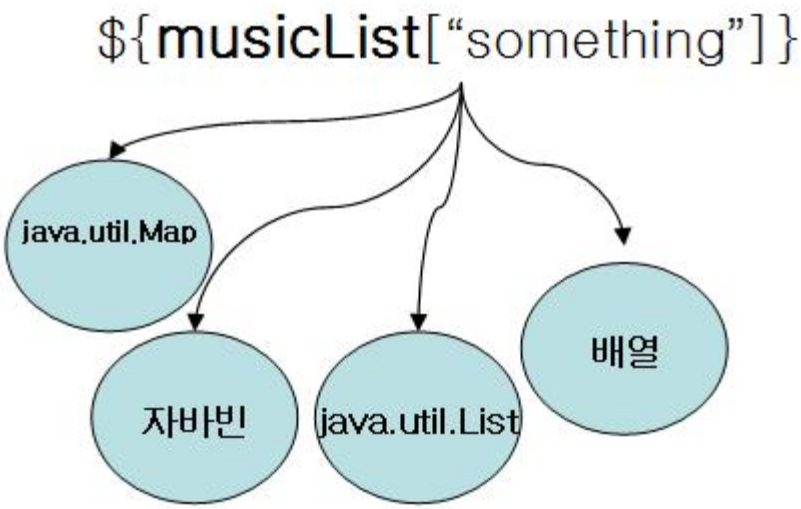
▮ 표현식에서 도트연산자 왼쪽은 반드시 맵 또는 빈이어야 한다.

▮ 표현식에서 도트 연산자 오른쪽은 반드시 맵의 키이거나 빈의 프로퍼티여야 한다.

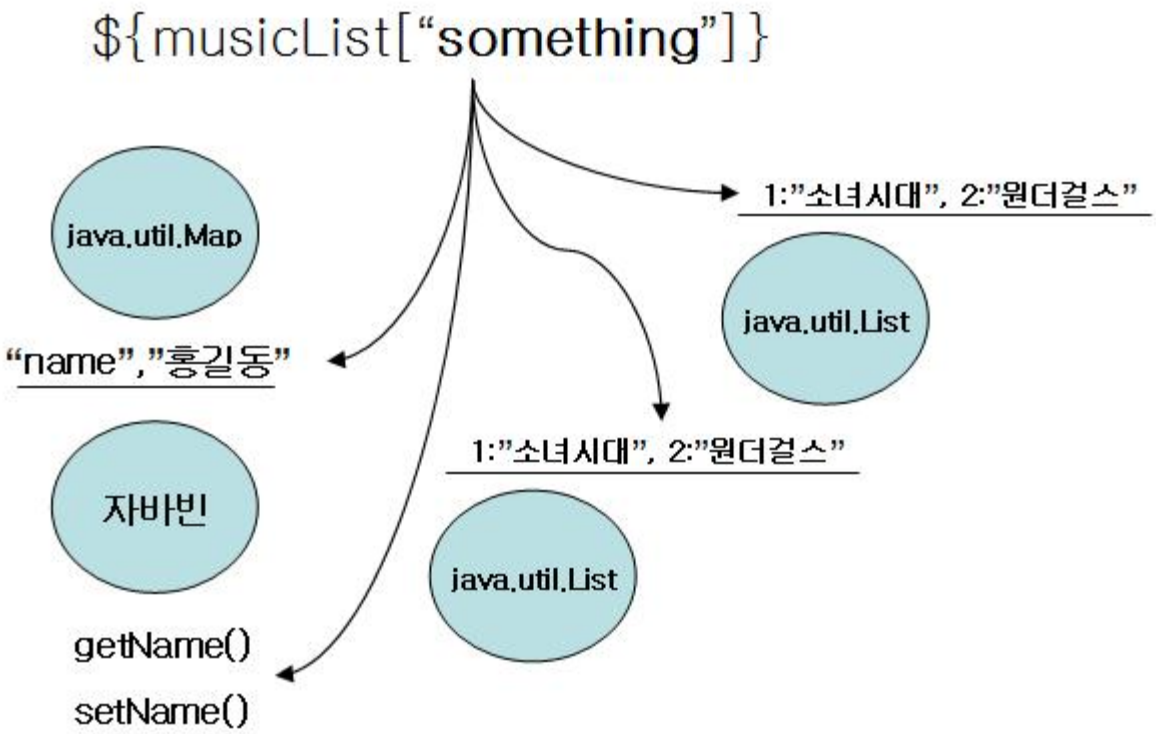


■ [] 연산자 : List나 배열 객체의 값을 표현할 수 있다.

▮ [] 연산자의 왼편에는 맵, 빈, 배열, 리스트 변수가 올 수 있다.



▮ [] 연산자 안의 값이 문자열이라면, 맵의 키이거나 빈의 프로퍼티가 될수 있고, 배열이나 리스트인 경우에는 인덱스가 될 수 있다.



■ 실행예제

```
<%@page import="vo.Movie"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.util.List"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
        pageContext.setAttribute("name", "김연아");

        request.setAttribute("height", 170);

        Movie movie = new Movie();

        movie.setName("인터스텔라");
        movie.setDirector("크리스토퍼 놀란");

        //controller에서 model을 등록하는 건 이런 거
        request.setAttribute("movie", movie);
%>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="UTF-8">
<title>EL공부</title>
</head>
<body>
        <h1>이름 : ${name }</h1>
```

```
        <h2>키 : ${height }</h2>
        <h2>영화이름 : ${movie.name }</h2>
        <h2>영화감독 : ${movie.director }</h2>
        <h2>설국열차의 감독 : ${movies[2].director }</h2>

<%--
        EL은 그냥 쓸 수 있습니다.

        Expression Language로 표현식을 더 편리하게 사용합니다.

        ${}

        EL은

        1) attribute를 출력합니다.
        2) request parameter
        3) request header

        Attribute의 종류

        1) pageContext : 한 페이지 내부에서만(안씀)
        2) request : 요청객체가 살아있는동안(포워드 방식)

           (Model은 사실 request의 attribute로)

        3) HttpSession : 세션이 살아있는 동안(브라우저 켜있을때)
        4) ServletContext : 서버는 켜있는 동안(잘 안씀)

 --%>
</body>
</html>
```

■ EL 연산자와 예약어

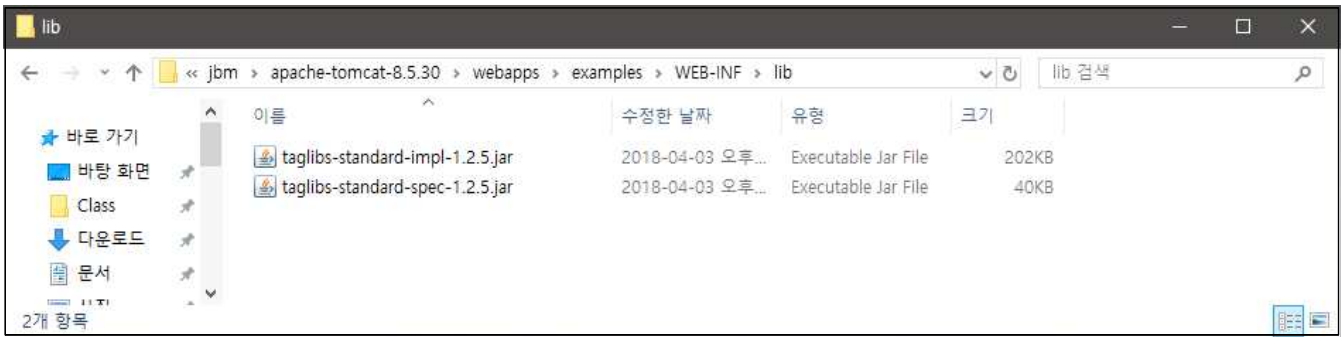
산술 연산자	
더하기	+
빼기	-
곱하기	*
나누기	/ 혹은 div
나머지	% 혹은 mod
논리 연산자	
AND	&& 혹은 and
OR	|| 혹은 or

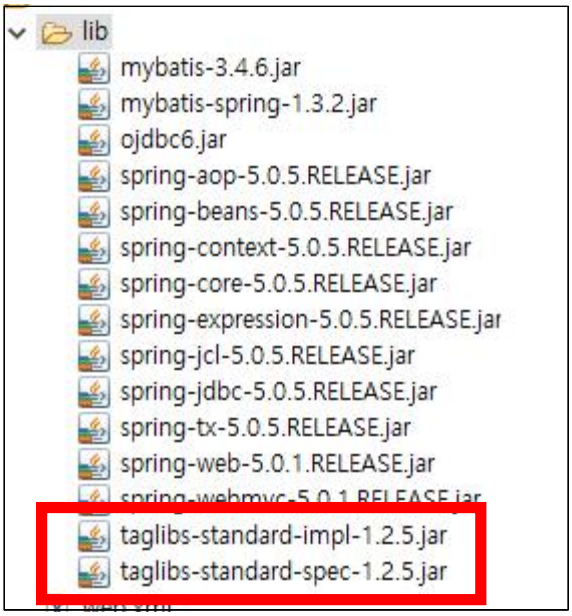
NOT	! 혹은 not
관계 연산자	
등호	== 혹은 eq
부등호	!= 혹은 ne
~보다 작다	< 혹은 lt
~보다 크다	> 혹은 gt
~보다 작거나 같다	<= 혹은 le
~보다 크거나 같다	>= 혹은 ge
예약어	
true	
false	
null	
instanceof	
empty	null이거나 비어있는지 체크하기 위한 연산자. null이거나 비어있으면 true를 반환한다.

□ JSTL(JSP Standard Tag Library) : JSP 표준 태그 라이브러리

■ JSTL은 JSP에서 스크립틀릿(자바코드)을 사용하지 않고 반복 작업을 하거나 조건문을 실행할 수 있는 태그를 제공한다.

■ JSTL은 라이브러리가 필요함





■ JSTL은 Core, XML, I18N, SQL, Function 5개의 태그라이브러리를 정의하고 있다.

■ JSTL의 Core 태그 라이브러리는 변수선언, 흐름제어, URL제어 등의 기능을 수행할 수 있는 태그들을 정의하고 있다.

■ JSTL의 Core 태그

기능	태그
변수 선언	set remove
흐름제어	choose when otherwise

	forEach forTokens if
URL 제어	import redirect url param
기타	catch out

▮ forEach 태그 : 배열과 콜렉션 데이터를 루프로 돌리는 작업을 처리할 수 있다.

▮ if 태그 : 분기문 처리를 할 수 있다.

형식 : <c:if test="${조건식}"> 조건식이 참 일때 수행</c:if>▮ <c:choose>, <c:when test="">, <c:othoerwise>

<c:if> 태그에는 else 처리를 할 수 있는 태그가 존재하지않는다.

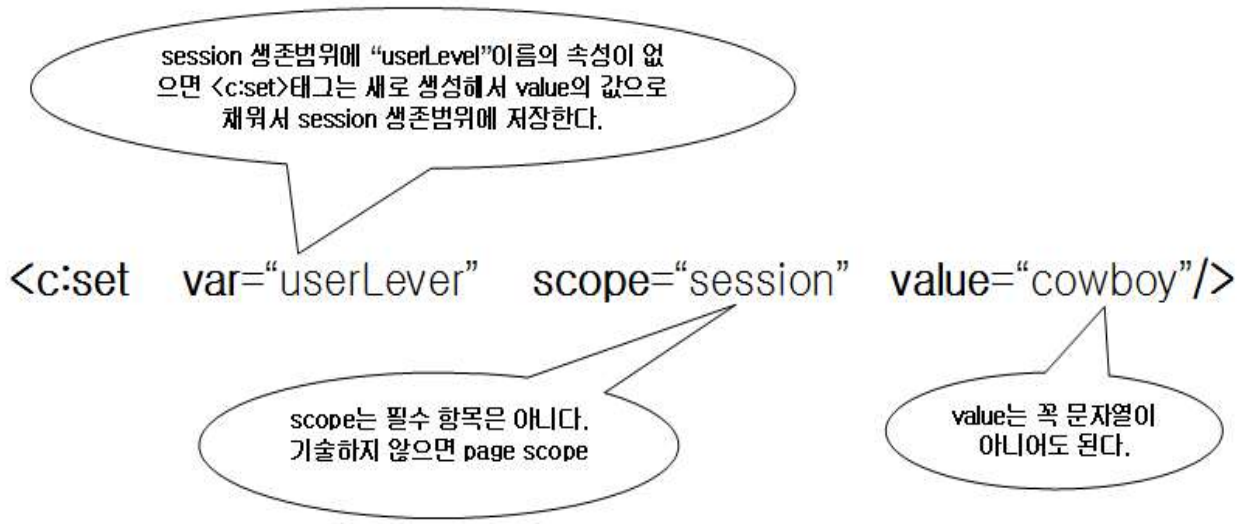
<c:choose>, <c:when>, <c:otherwise> 태그를 사용해서

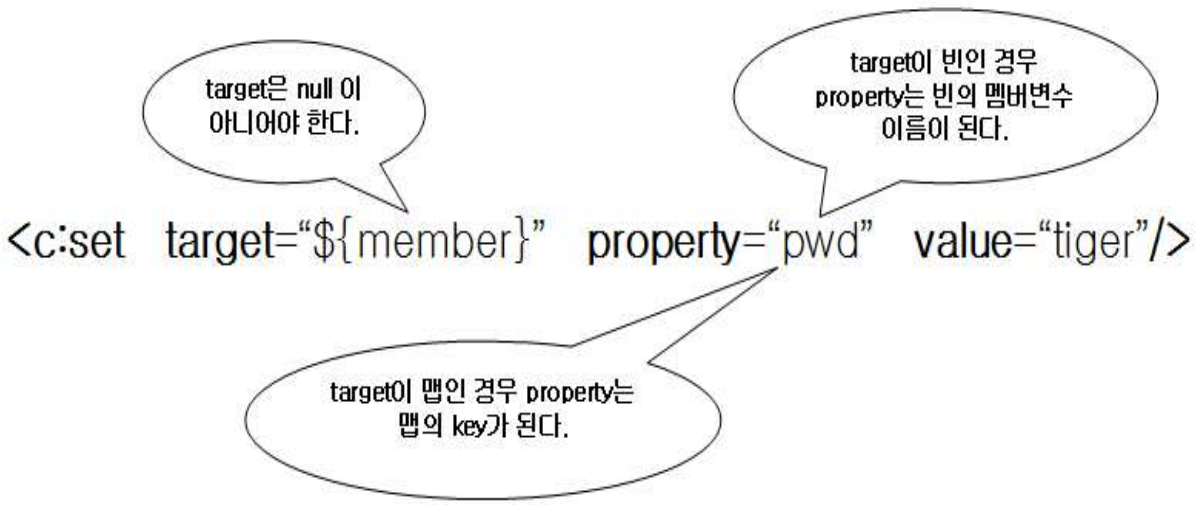
if ~ else if ~ else if ~ else 형태의 분기문 처리를 할 수 있다.

```
<c:choose>
	<c:when test="{조건식1}"
		조건식1이 참인 경우 수행된다.
	</c:when>
	<c:when test="{조건식2}"
		조건식2가 참인 경우 수행된다.
	</c:when>
	<c:otherwise>
		조건식1과 조건식2를 모두 만족하지 못한 경우 수행된다.
	</c:otherwise>
</c:choose>
```

※ <c:otherwise>는 필수 사항은 아님

▮ <c:set> 태그 : 특정 scope에 속성으로 빈이나 맵을 추가하거나 삭제할 수 있다.





※ <c:set> 태그 사용시 주의사항

. <c:set>태그에 var와 target을 동시에 사용할 수 없다.

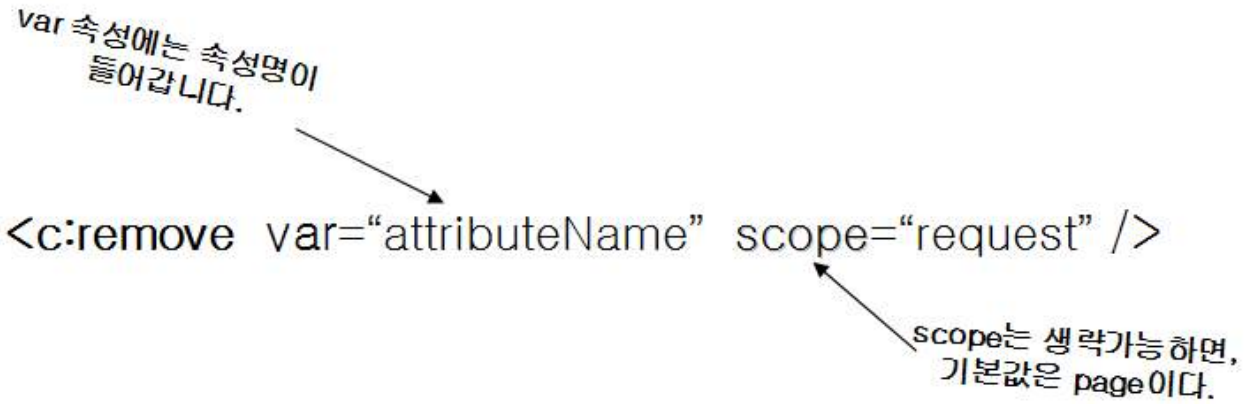
. value가 null인 경우, var의 속성은 사라진다.

. var에 지정된 이름이 속성이 존재하지 않는 경우, 자동으로 만든다.

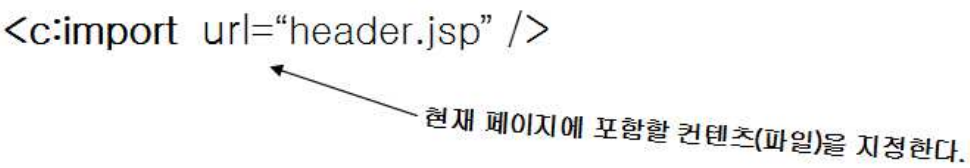
. target에는 실제 개체를 표현하는 표현식이 들어있어야 한다.

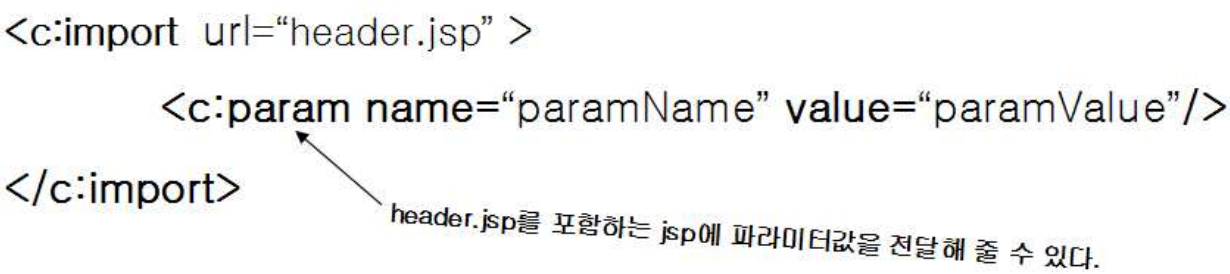
. target에 명시된 표현식으로 찾아진 객체는 반드시 bean이거나 맵이어야 한다.

▮ <c:remove> 태그 : 특정 scope에 저장된 속성을 삭제할 수 있다.



▮ <c:import> 태그 : 요청이 들어오는 시점에, url 속성에 명시한 파일을 현재 컨텐츠에 포함한다. <c:import>는 외부 자원도 포함할 수 있다.





■ 실행예제

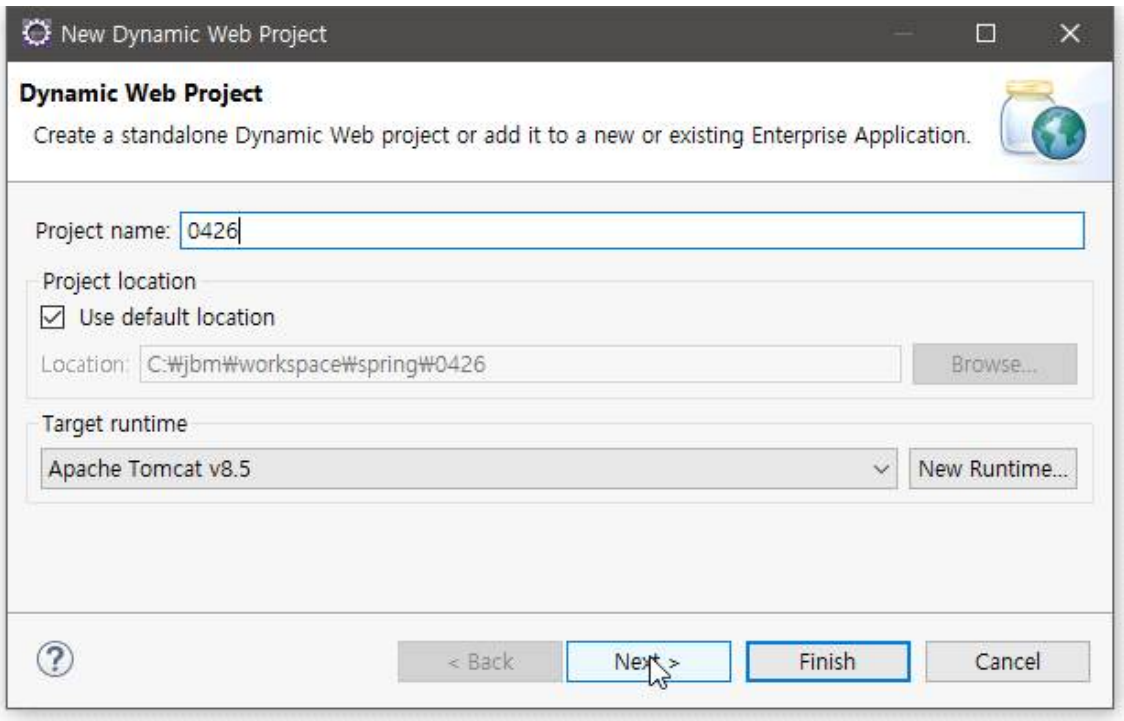
```
<%@page import="java.util.ArrayList"%>
<%@page import="vo.Movie"%>
<%@page import="java.util.List"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
        //여기 쓰는건 controller에서 했던 일

        //로그인 되어있다는 뜻
        session.setAttribute("loginUser", "xxxx");
```

```
        List<Movie> movies = new ArrayList();

        movies.add(new Movie("마션","리들리 스콧"));
        movies.add(new Movie("인셉션","크리스토퍼 놀란"));
        movies.add(new Movie("설국열차","봉준호"));
        movies.add(new Movie("주토피아","백감독"));

        //요청에 attribute로 등록
        request.setAttribute("movies", movies);

%>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="UTF-8">
<title>JSTL의 문법</title>
</head>
<body>
        <%-- 단순 if --%>
        <c:if test="${3<6}">
                <h1>True네요.</h1>
        </c:if>

        <%--
                if~else

                <c:choose>
                        <c:when test="">
                                참일때
                        </c:when>
                        <c:otherwise>
                                거짓일때
                        </c:otherwise>
                </c:choose>

        --%>

        <%--로그인 되었을때 --%>

        <c:choose>
        <c:when test="${loginUser!=null }">
                <h1>XXX님 환영</h1>
                <a href="logout.html">로그아웃</a>
        </c:when>
        <c:otherwise>
                <form action="login.html" method="post">
                        <fieldset>
```

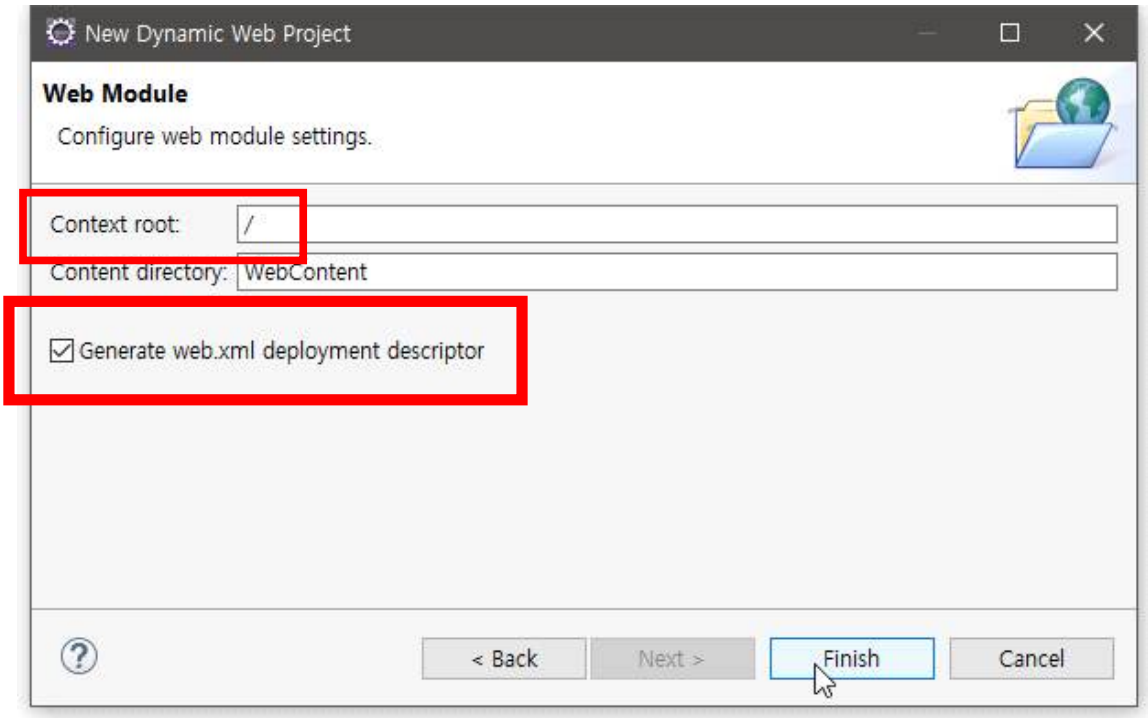
```
                                <legend>로그인폼</legend>
                                <input type="text" id="id" name="id" />
                                <input type="password" id="password"
                                name="password" />
                                <button>로그인</button>
                        </fieldset>
                </form>
        </c:otherwise>
        </c:choose>

        <table border="1">
                <thead>
                        <tr>
                                <th>이름</th>
                                <th>감독</th>
                        </tr>
                </thead>
                <tbody>
                <c:forEach items="${movies }" var="movie">
                        <tr>
                                <td>${movie.name }</td>
                                <td>${movie.director }</td>
                        </tr>
                </c:forEach>
                </tbody>
        </table>
</body>
</html>
```

■ Spring MVC 시작

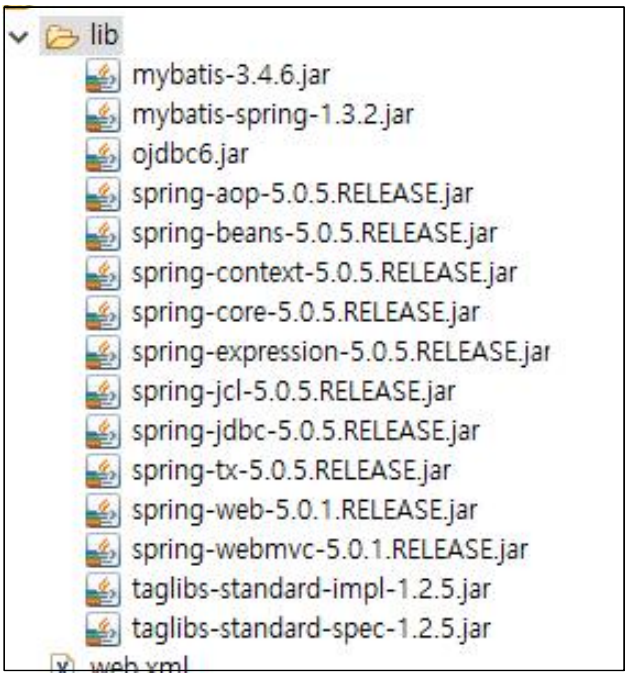
1) 프로젝트 생성



- context root를 '/'로
- web.xml 생성



2) WEB-INF의 lib폴더에 스프링 프레임워크 jar 복사



3) web.xml에 post방식의 한글필터 설정

```
<filter>
	<filter-name>encoding</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>encoding</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

4) web.xml에 applicationContext.xml 읽어들이는 리스너 설정

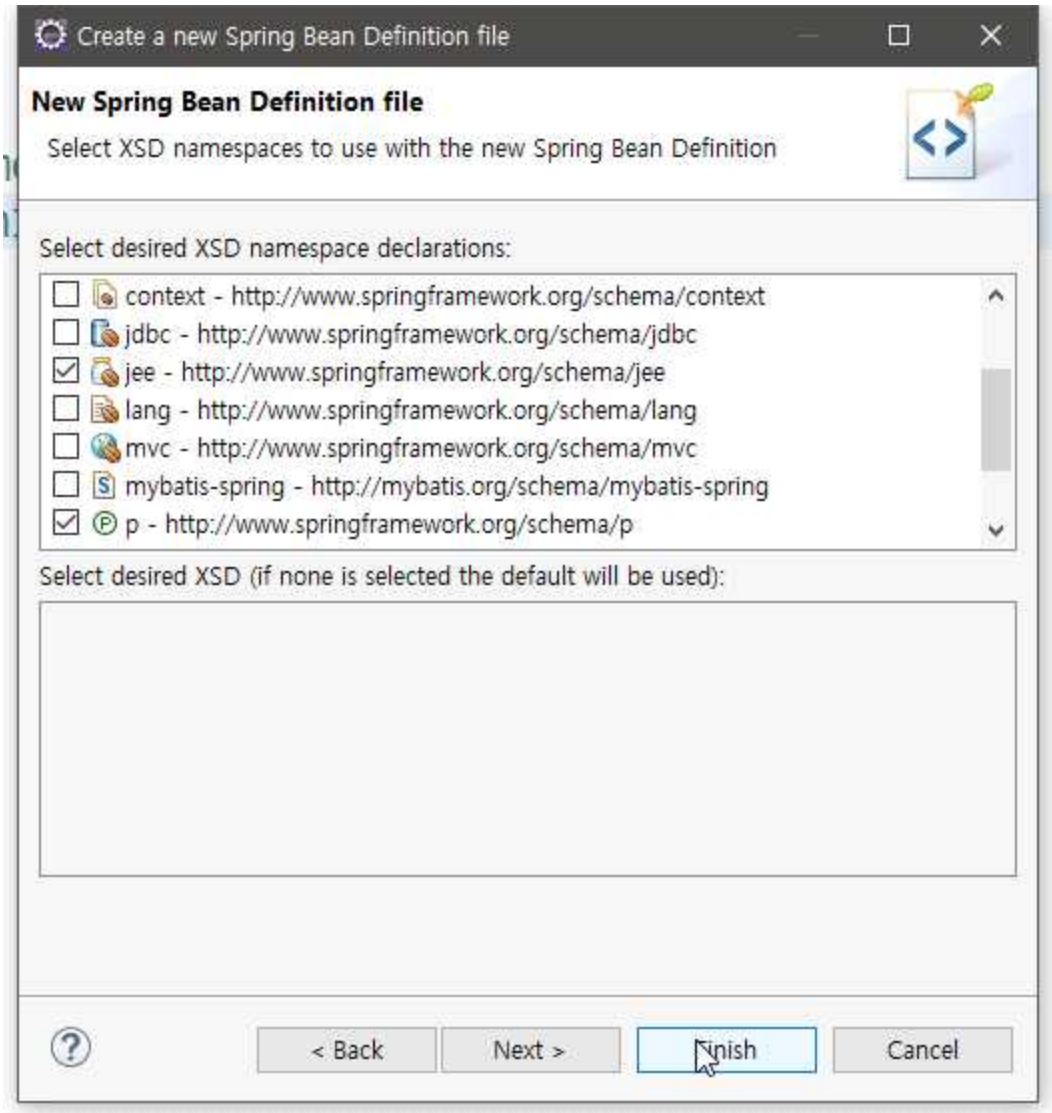
```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

5) web.xml에 DispatcherServlet 설정

```
<servlet>
        <servlet-name>start</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>start</servlet-name>
        <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

6) WEB-INF폴더에 applicationContext.xml 생성

- 스프링 프레임워크 자체에 대한 설정(jee / p 체크)

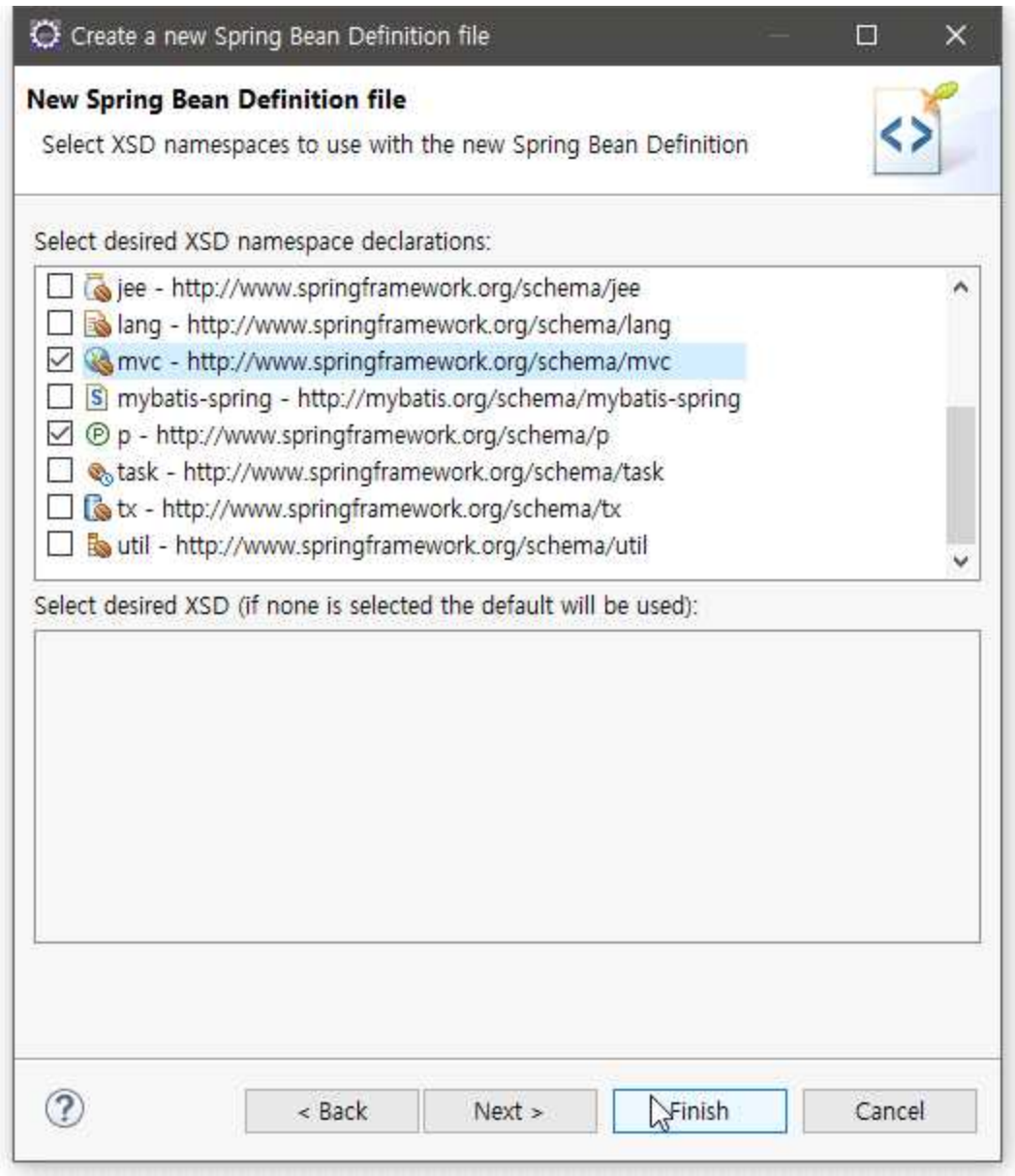


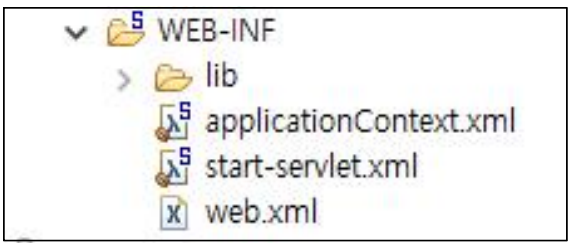
- jee : 커넥션풀 설정용

- p : 프로터티 설정(setter를 'p:' 으로)

7) WEB-INF폴더에 start-servlet.xml 생성

- Spring WEB MVC에 대한 설정

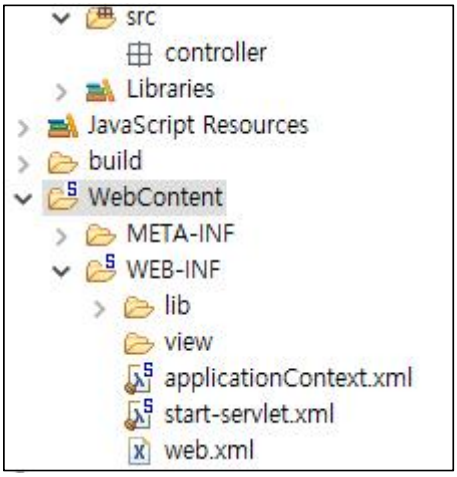




- 추후에 mybatis-config.xml도 WEB-INF에 위치할 예정

8) src에 controller 패키지를 생성 / WEB-INF폴더에 view폴더 생성

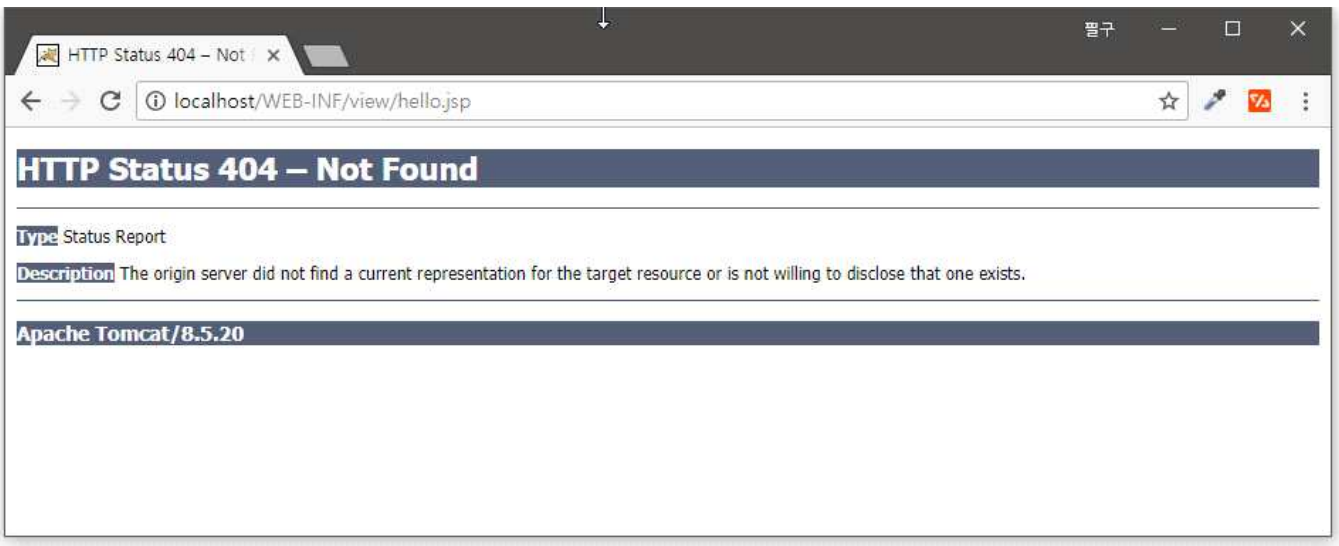
- controller패키지에 controller 클래스를 생성
- view폴더에 jsp(뷰)를 생성



9) view폴더에 hello.jsp 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello Spring</title>
</head>
<body>
        <h1>Hello Spring!</h1>
</body>
</html>
```

- hello.jsp는 접근이 불가능(WEB-INF / META-INF는 접근 불가)

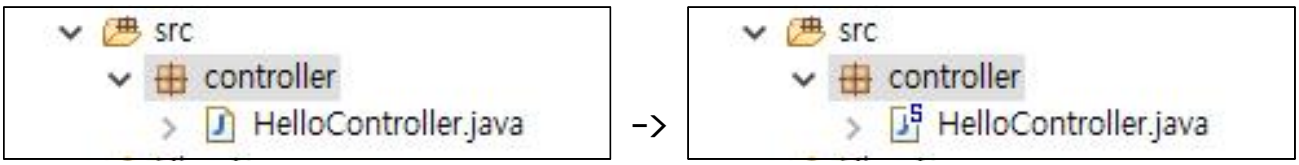


10) HelloController의 작성

```
package controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {

	@RequestMapping("/hello.html")
	public void hello() {
		System.out.println("여기 호출!");
	}
}
```

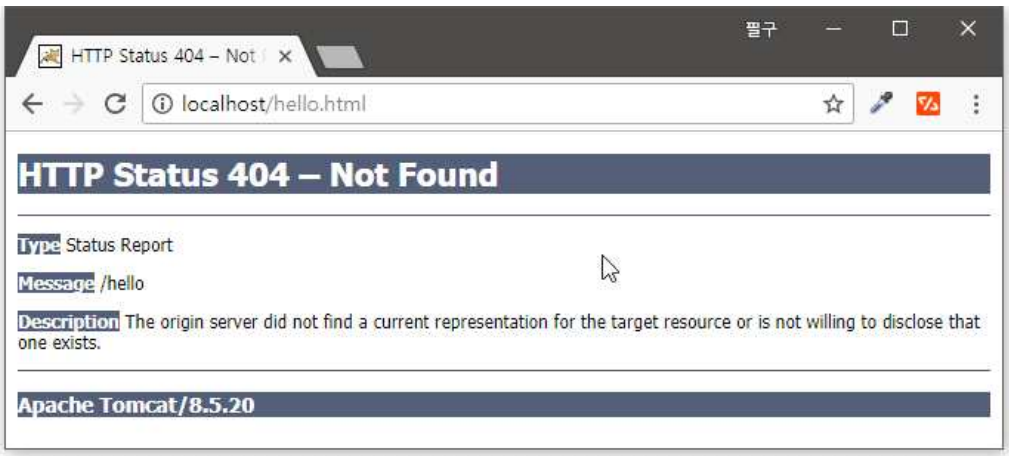
- 어노테이션으로 설정(컨트롤러)
- RequestMapping으로 메서드를 주소와 맵핑
- start-servlet.xml에 HelloController를 빈으로 등록

```
<bean class="controller.HelloController"/>
```



- 빈으로 등록해야 Spring의 자원이 됨

(자동 설정도 가능하나 회사마다 다름)



11) controller에서 view를 선택하는 방법

① 리턴자료형 : ModelAndView

```
@RequestMapping("/hello.html")
public ModelAndView hello() {

        return new ModelAndView("/WEB-INF/view/hello.jsp");

}
```

② 리턴자료형 : View

```
@RequestMapping("/hello.html")
public View hello() {

        return new InternalResourceView("/WEB-INF/view/hello.jsp");

}
```

- 대표적인 View종류

1) InternalResourceView : 일반적인 JSP

2) RedirectView : 리다이렉트

3) JacksonJsonView : json

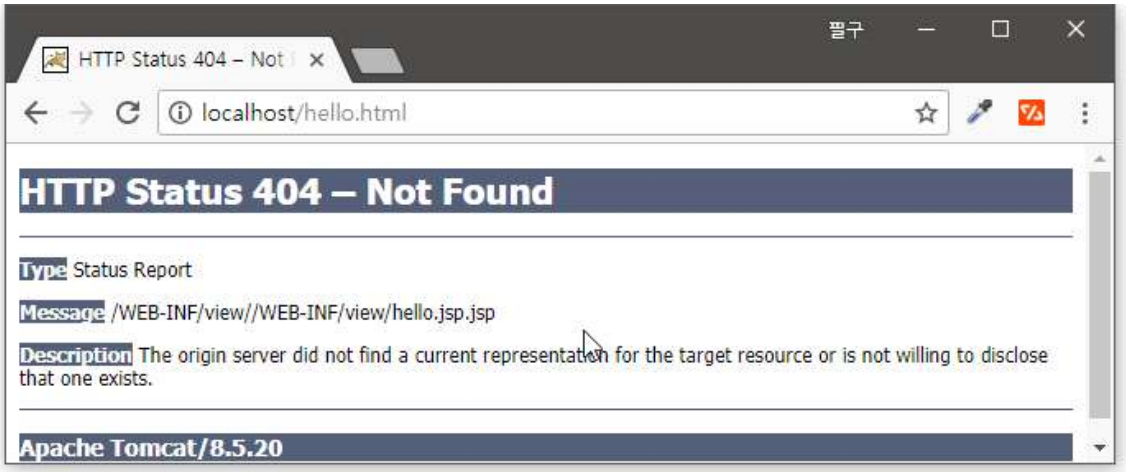
③ 리턴자료형 : String

```
@RequestMapping("/hello.html")
public String hello() {

        return "/WEB-INF/view/hello.jsp";

}
```

- 중복된 'WEB-INF/view/' 와 '.jsp'를 생략가능하게 하는 설정

(start-servlet.xml에)

```
<mvc:view-resolvers>
        <mvc:jsp prefix="/WEB-INF/view/" suffix=".jsp"/>
</mvc:view-resolvers>
```




```
@RequestMapping("/hello.html")
public String hello() {
        return "hello";
}
```

④ 리턴자료형 : void

- 만약 리퀘스트맵핑의 확장자를 제외한 이름과 jsp의 이름이 동일하다면 리턴값이 필요없음(프레임워크가 찾음)

```
@RequestMapping("/hello.html")
public void tttttt() {

}
```

■ MyBatis 설정

1) 패키지의 설정

2) WEB-INF에 mybatis-config.xml 생성

3) META-INF에 context.xml 생성

4) map에 movies.xml 생성

5) VO 생성

6) DAO 인터페이스 / DAO 구현 클래스

7) Service 인터페이스 / Service 구현 클래스

8) applicationContext에 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jee="http://www.springframework.org/schema/jee"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
               http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-4.2.xsd">

<!-- WAS가 가지고 있는 커넥션풀(DataSource)
         JNDI로 얻어오기 -->
         <jee:jndi-lookup id="dataSource"
         jndi-name="java:comp/env/oraclexe"/>

         <!-- SqlSessionFactory 설정 -->
        <bean id="sqlSessionFactory"
        p:configLocation="WEB-INF/mybatis-config.xml"
        p:mapperLocations="classpath:map/*.xml"
        p:dataSource-ref="dataSource"
         class="org.mybatis.spring.SqlSessionFactoryBean"/>

        <!-- SqlSessionTemplate 설정 -->
        <bean id="sqlSession"
         class="org.mybatis.spring.SqlSessionTemplate">
```

```
                <constructor-arg ref="sqlSessionFactory"/>
        </bean>

        <!-- DAO 설정 -->
        <bean id="moviesDAO"
        p:session-ref="sqlSession"
         class="dao.MoviesDAOImpl"/>

        <bean id="usersDAO"
         p:session-ref="sqlSession"
         class="dao.UsersDAOImpl"/>

        <!-- Sevice 설정 -->
        <bean id="usersService"
        p:dao-ref="usersDAO"
        class="service.UsersServiceImpl"/>

        <bean id="moviesService"
        p:dao-ref="moviesDAO"
        class="service.MoviesServiceImpl"/>

</beans>
```

9) Controller에 Service 주입

```
package controller;

import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import service.MoviesService;
import vo.Movie;

@Controller
public class MoviesController {

        private MoviesService service;

        //주입을 위한 세터
        public void setService(MoviesService service) {
                this.service = service;
        }

        @RequestMapping("index.html")
```

```
        public void index(Model model) {

                //View에서 출력할 영화정보를 담은 list를
                //얻어왔습니다.
                List<Movie> list = service.getMovies();

                //이제 model에 list를 넣어주면 됩니다.
                model.addAttribute("movieList", list);

        }

}
```

```
<bean p:service-ref="moviesService"
        class="controller.MoviesController"/>
```

6) index.jsp 뷰

```
<%@page import="vo.Movie"%>
<%@page import="java.util.List"%>
<%@page import="dao.MoviesDAO"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%--
        MODEL2의 View에서는 <% %>, <%=%> 즉,
        자바코드가 한 줄도 안들어갑니다.

        그러면 <% %> <-- 대신에 jstl을
        (Java Standard Tag Library)

        <%= %> <- el을 사용합니다.
        (Expression Language)

        1) jstl라이브러리도 lib폴더에 추가

        2) jstl를 사용하기 위해서 taglib지시어를 선언합니다.


 --%>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="UTF-8">
<title>^-^v</title>
```

```
</head>
<body>
	<h1>영화정보</h1>
	<table border="1">
		<caption>영화목록</caption>
		<thead>
			<tr>
				<th>번호</th>
				<th>이름</th>
				<th>감독</th>
				<th>장르</th>
				<th>개봉일</th>
			</tr>
		</thead>
		<tbody>
		<c:forEach items="${movieList}" var="movie">
			<tr>
				<td>${movie.no }</td>
				<td>${movie.name }</td>
				<td>${movie.director }</td>
				<td>${movie.genre }</td>
				<td>${movie.releaseDate }</td>
			</tr>
		</c:forEach>
		</tbody>
	</table>
</body>
</html>
```

■ 우리가 Spring MVC에서 고민할 것들

1) 어떻게 url과 controller를 맵핑시킬까?(HandlerMapping이 하는 일)

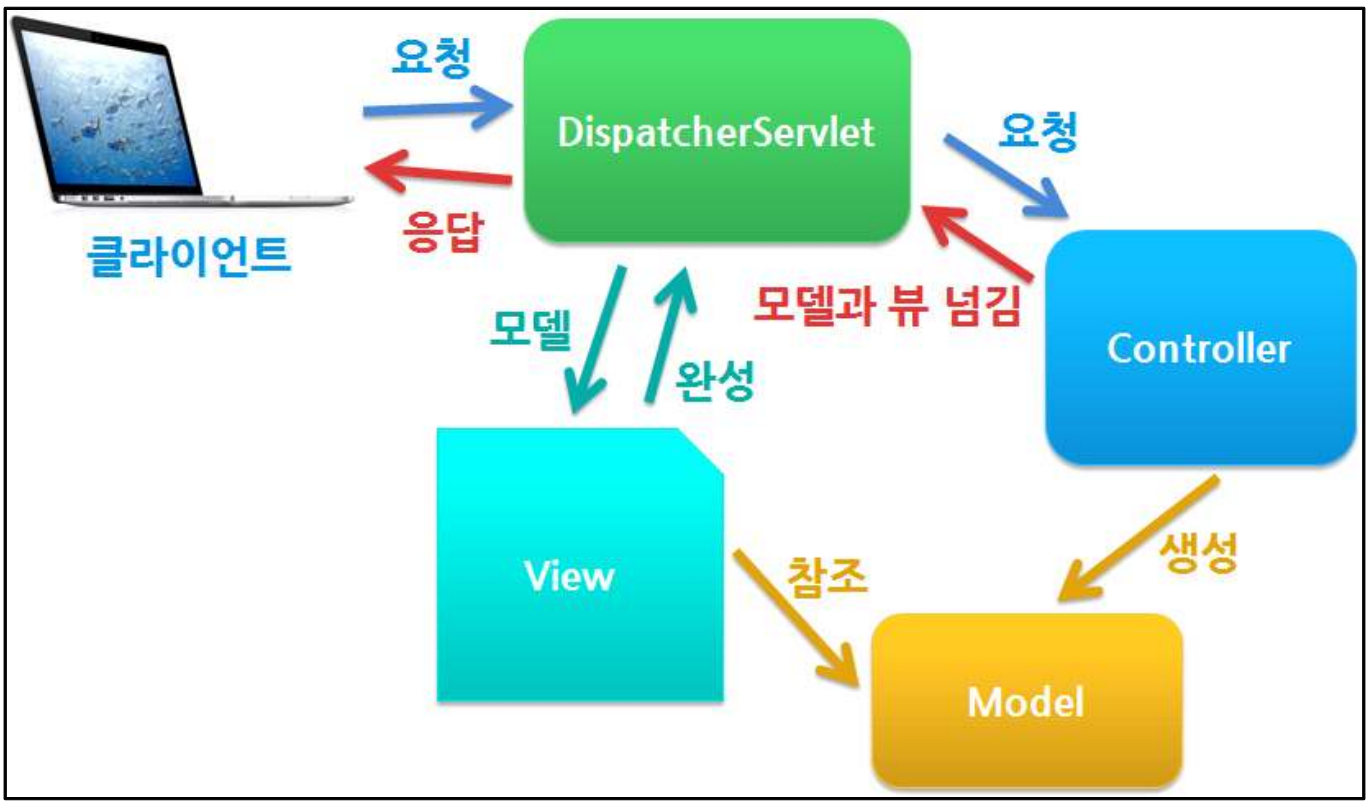
2) 그 controller안에 있는 메서드를 어떻게 선택할까?(HandlerAdapter가 하는 일)

3) 우리는 메서드에서 어떤 것들을 인자로 받을 수 있을까?(DI 주입)

4) 우리는 메서드의 리턴으로 어떤 것들을 보낼 수 있을까?(View의 선택)

5) View로 데이터들을 어떻게 보낼 수 있을까?(Model의 이용)

■ Spring MVC 흐름도



■ Spring MVC의 등장인(?)물

1) DispatcherServlet

- 가장 중요한 객체
- 기본적인 흐름을 담당
- 등장구역 : web.xml

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

<servlet-mapping>
   <servlet-name>kpg</servlet-name>
   <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

- 빨간색 이름으로 : kpg-servlet.xml을 생성해야 함(MVC설정용)
  변경할 경우 init-param으로 변경

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

2) HandlerMapping

- 한개가 아님
- 기본 설정으로 2개가 작동
- url과 요청정보를 기준으로 컨트롤러를 선정

HandlerMapping	특 징
BeanNameUrlHandlerMapping	bean설정시 name 속성으로 (기본)
RequestMappingHandlerMapping	컨트롤러에 어노테이션을 부여하여 맵핑 (기본)
SimpleUrlHandlerMapping	설정파일에 한번에 모아서 설정

ControllerClassNameHandlerMapping	클래스이름과 메서드명으로 맵핑
ControllerBeanNameHandlerMapping	빈의 이름으로 맵핑

- BeanNameUrlHandlerMapping을 이용

mvc설정 xml(예:kpg-servlet.xml)

```
<bean name="/hello.html" class="controller.FirstController" />
```

Controller

```
@Controller
public class FirstController {

	@RequestMapping
	public void test() {

	}

}
```

- RequestMappingHandlerMapping을 이용

mvc설정 xml(예:kpg-servlet.xml)

```
<bean class="controller.FirstController" />
```

Controller

```
@Controller
public class FirstController {
	@RequestMapping("hello.html")
	public void test() {

	}
}
```

3) HandlerAdapter

- 컨트롤러 안의 메서드를 호출할 때 사용
- 기본 설정으로 3개가 작동

HandlerAdapter	특 징
HttpRequestHandlerAdapter	원격옵션을 지원(잘안씀)
SimpleControllerHandlerAdapter	Controller 인터페이스 구현시 httpRequest()호출
RequestMappingHanlderAdapter	우리가 사용하는 어노테이션기반 메서드 호출

- SimpleControllerHandlerAdapter 이용예제

mvc설정 xml(예:kpg-servlet.xml)

```
<bean name="/hello.html" class="controller.FirstController" />
```

Controller

```
public class FirstController  implements Controller{
	@Override
	public ModelAndView handleRequest(HttpServletRequest arg0,
			HttpServletResponse arg1) throws Exception {
		return new ModelAndView("hello");
	}
}
```

- RequestMappingHanlderAdapter 이용예제

mvc설정 xml(예:kpg-servlet.xml)

```
<bean class="controller.FirstController" />
```

Controller

```
@Controller
public class FirstController  {
	@RequestMapping("hello.html")
	public void test() {

	}
}
```

※ 빈와이어링(Bean Wiring)

■ xml스키마를 이용

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd">
```

■ 빈의 생성 및 사용

- 컨텍스트 파일의 루트 요소는 <beans>

- 빈을 static 메서드를 이용해 객체를 생성해야 하는 경우

```
        <bean id="daoIbatis"
class="di.dao.UserDaoIbatisImpl" factory-method="getInstance"/>
```

- null값을 설정할 수 있음

■ 의존 관계 설정

- 생성자 방식(<constructor-arg>태그 사용)

```
       <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
        <constructor-arg><value>1</value></constructor-arg>
        </bean>
```

- 다른 빈을 참조하는 경우 <ref>태그나 <constructor-arg>의 ref속성을 이용

```
        <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
         <constructor-arg><ref bean="articleDao" /></constructor-arg>
        </bean>
```

■ Setter를 이용한 방식(<property>태그 사용)

- 기본자료형과 String의 경우는 <value>태그나 <property>의 value속성을 이용

```
<bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
    <property><value>1</value></property>
</bean>
```

- 다른 빈을 참조하는 경우 <ref>태그나 <constructor-arg>의 ref속성을 이용

```
<bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
    <property><ref bean="articleDao" /></property>
</bean>
```

- xml 네임스페이스를 이용한 프로퍼티 설정(접두어 : p 사용)

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

    <bean id="steampackMarine" class="starcraft.Marine"
        p:no="2"
        p:hp="40"
        p:command-ref="steampackAttack"
        init-method="init"
    />
</beans>
```

p:프로퍼티이름(빈의 멤버필드), p:프로퍼티이름-ref를 사용하면 편리함

■ 빈 객체의 범위 지정

- <bean> 태그의 scope 속성 값

이 름	설명
singleton	스프링 컨테이너에 한 개의 빈 객체만 존재함(기본값)
prototype	빈을 사용할 때 마다 객체를 생성한다.
request	http 요청마다 빈 객체를 생성한다.(WebApplicationContext에서만 가능)
session	http 세션마다 빈 객체를 생성한다.(WebApplicationContext에서만 가능)
global-session	global http 세션에 빈 객체를 생성한다.(포틀릿을 지원하는 context에서만 가능)

■ Collection Wiring

- List 타입

```
<bean id="listBean" class="example.ListBean">
  <property name="listPro">
    <list><ref bean="someBean"/></list>
  </property>
</bean>
```

- Set 타입

```
<bean id="setBean" class="example.SetBean">
  <property name="setPro">
    <set><ref bean="someBean"/></set>
  </property>
</bean>
```

- Map 타입

```
<bean id="mapBean" class="example.MapBean">
    <property name="mapPro">
        <map>
       <entry key="key1"><value>value1</value></entry>
       <entry key="key2"><ref bean="someBean"/></entry>
        </map>
    </property>
</bean>
```

■ 의존관계 자동 설정

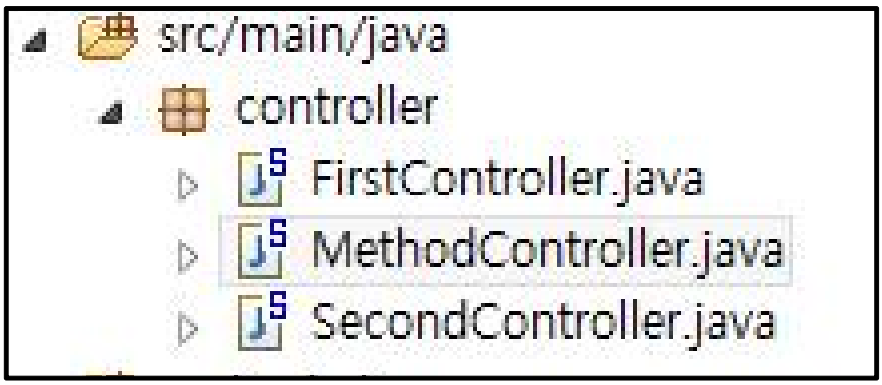
- <bean> 태그의 autowire 속성으로 설정

- autowire 타입

타입종류	설 명
byName	프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정한다.
byType	프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정한다.
constructor	생성자 파라미터 타입과 같은 갖는 빈 객체를 생성자에 전달한다.
autodetect	construtor 방식을 먼저 적용하고, byType방식을 이용하여 적용한다.

- 특별한 경우가 아니라면 사용하지 않아야 함

■ 컨트롤러 메서드의 인자들




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

	<bean name="/index.html" class="controller.FirstController"></bean>
	<bean class="controller.SecondController"></bean>
	<bean class="controller.MethodController"></bean>
</beans>
```

1) HttpServletRequest(비추천)

2) HttpSession

3) @RequestParam

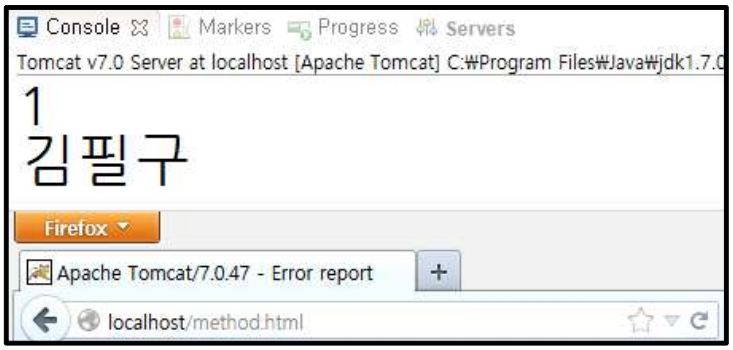
- 파라미터를 받을 수 있는 어노테이션

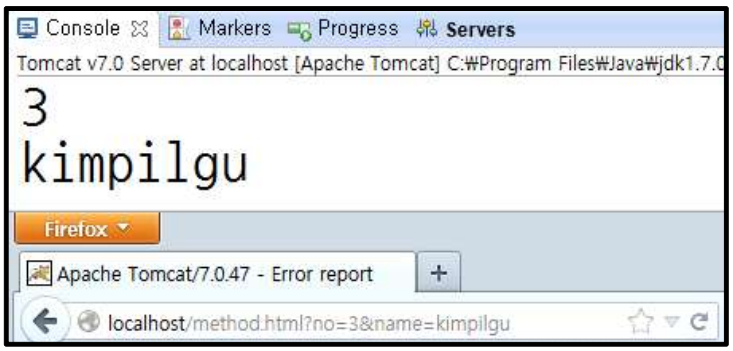
```
package controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@RequestMapping
public class MethodController {

	@RequestMapping("/method.html")
	public void test(@RequestParam int no, @RequestParam String name) {
```

```
                System.out.println(no);
                System.out.println(name);
        }
}
```

- 기본값 지정 / 필요한지 지정 가능

```
@RequestMapping("/method.html")
public void test(@RequestParam(value="no",
                               required=false,
                               defaultValue="1") int no,
                @RequestParam(value="name",
                              required=true,
                              defaultValue="김필구") String name) {
        System.out.println(no);
        System.out.println(name);
}
```





4) @ModelAttribute

second.jsp를 변경

```
	<h1>second.jsp</h1>
	<form action="join.html" method="post">
	<fieldset>
		<legend>회원가입폼</legend>
		<p>
		<label for="id">아이디</label>
		<input type="text" id="id" name="id" />
		</p>
		<p>
		<label for="pwd">비밀번호</label>
		<input type="text" id="pwd" name="password" />
		</p>
		<p>
		<label for="name">이름</label>
		<input type="text" id="name" name="name" />
		</p>
		<p>
		<button>회원가입</button>
		</p>
	</fieldset>
	</form>
```

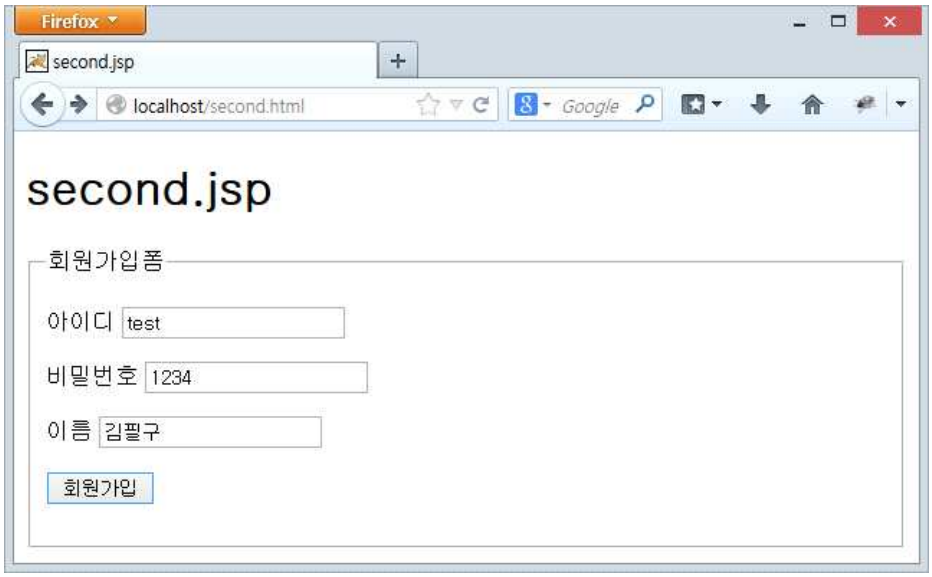
vo패키지에 User VO 생성

```
package vo;

public class User {

	private String id,password,name;
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getPassword() {
		return password;
	}
```

```
	public void setPassword(String password) {
		this.password = password;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

MethodController에 추가

```
@RequestMapping("/join.html")
public void login(@ModelAttribute User user) {
	System.out.println("id : "+user.getId());
	System.out.println("password : "+user.getPassword());
	System.out.println("name : "+user.getName());
}
```



- 생략가능

```
@RequestMapping("/join.html")
public void login(User user) {
	System.out.println("id : "+user.getId());
	System.out.println("password : "+user.getPassword());
	System.out.println("name : "+user.getName());
}
```

5) @RequestHeader

MethodController에 추가

```
@RequestMapping("/header.html")
public void header(@RequestHeader String referer) {
	System.out.println(referer);
}
```

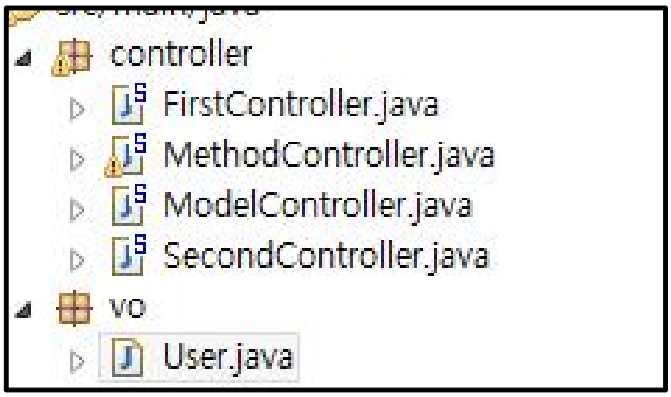
first.jsp에 추가

```
	<h1>first.jsp</h1>
	<a href="third.html">GET으로 third.html요청</a>
	<form action="third.html" method="post">
		<button>POST로 third.html요청</button>
	</form>
	<a href="header.html">header.html으로 이동</a>
</body>
```

6) Map, Model, ModelMap

1) view에 데이터를 보낼 모델

2) ModelController 생성 및 빈 등록




```
        <bean name="/index.html" class="controller.FirstController"></bean>
        <bean class="controller.SecondController"></bean>
        <bean class="controller.MethodController"></bean>
        <bean class="controller.ModelController"></bean>
</beans>
```

```
package controller;

@RequestMapping
public class ModelController {

        @RequestMapping("/list.html")
        public String list(Model model) {

                List<String> names = new ArrayList<String>();

                names.add("김연아");
                names.add("박지성");
                names.add("류현진");
                names.add("추신수");

                model.addAttribute("names", names);

                model.addAttribute("msg", "model이 view로 잘 전달되었군요!");

                return "WEB-INF/view/list.jsp";
```

```
        }
}
```

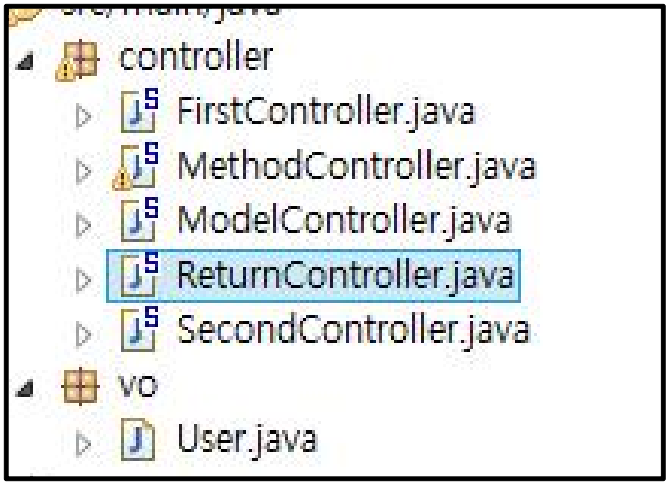
view에 list.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>한국을 빛낸 스포츠 스타</title>
</head>
<body>
        <h1>스포츠 스타</h1>
        <ul>
        <c:choose>
                <c:when test="${names!=null && names.size()!=0 }">
                        <c:forEach items="${names }" var="name">
                                <li>${name }</li>
                        </c:forEach>
                </c:when>
                <c:otherwise>
                        <li>스타가 없습니다.</li>
                </c:otherwise>
        </c:choose>
        </ul>
        <h2>${msg}</h2>
</body>
</html>
```

■ 컨트롤러 메서드의 리턴타입

1) ModelAndView

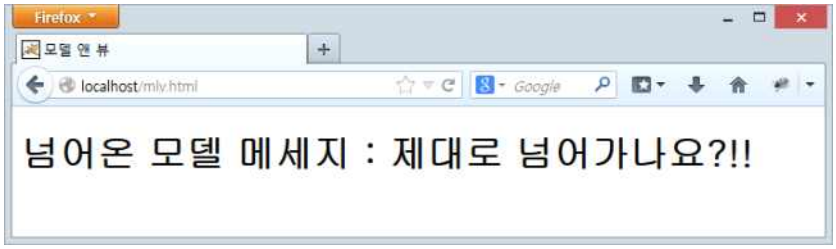
- 우리가 만든 MVC의 ActionForward + request객체와 비슷함
- Forward될 jsp와 Model을 지정함
- 더 편리한 경우가 많이 사용되지 않음



mvc-servlet.xml에 추가

```
<bean name="/index.html" class="controller.FirstController"></bean>
<bean class="controller.SecondController"></bean>
<bean class="controller.MethodController"></bean>
<bean class="controller.ModelController"></bean>
<bean class="controller.ReturnController"></bean>
```

```
package controller;
@Controller
public class ReturnController {
	@RequestMapping("/mlv.html")
	public ModelAndView mlv() {
		ModelAndView mlv = new ModelAndView();
		mlv.setViewName("WEB-INF/view/mlv.jsp");
		mlv.addObject("msg", "제대로 넘어가나요?!!");
		return mlv;
	}
}
```



2) String

- view의 이름

3) void

- 리턴이 없는 경우 InternalResourceViewResolver를 통해 자동으로 view 선택

applicationContext.xml에 추가

```
<bean p:prefix="/WEB-INF/view/" p:suffix=".jsp"
class="org.springframework.web.servlet.view.InternalResourceViewResolver"/>
```

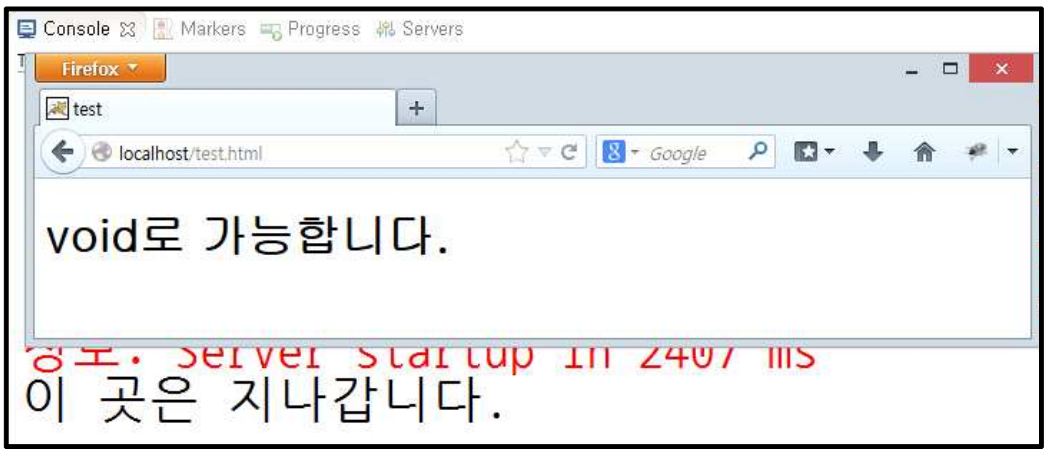
ReturnController에 추가

```
@RequestMapping("/test.html")
public void test() {
	System.out.println("이 곳은 지나갑니다.");
}
```

test.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>test</title>
</head>
<body>
	<h1>void로 가능합니다.</h1>
```

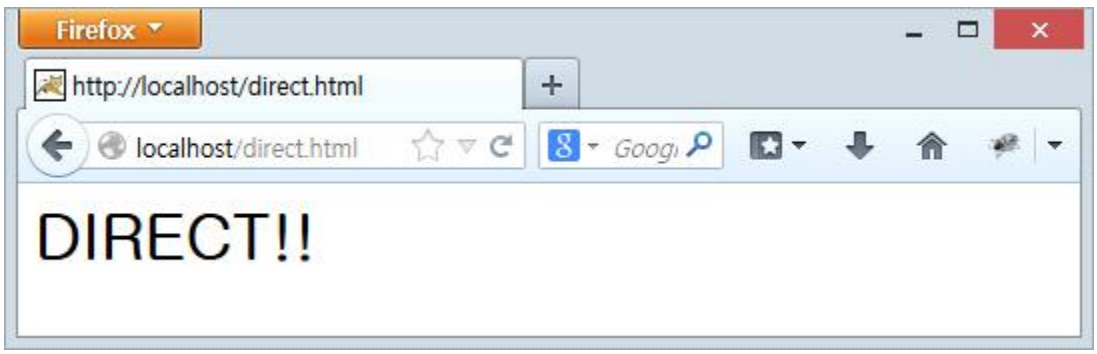
```
</body>
</html>
```



4) @ResponseBody로 직접 응답

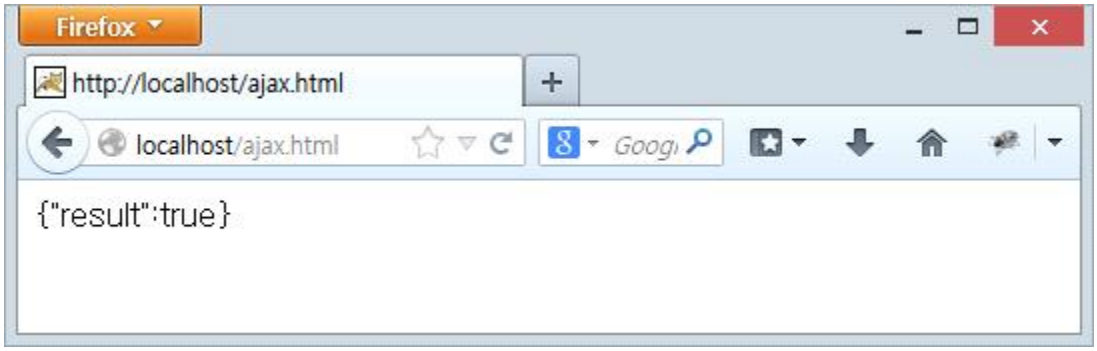
ReturnController에 추가

```
@RequestMapping("/direct.html")
@ResponseBody
public String direct() {
	return "<html><h1>DIRECT!!</h1></html>";
}
```



ReturnController에 추가

```
@RequestMapping("/ajax.html")
@ResponseBody
public String ajax() {
        return "{\"result\":true}";
}
```



5) redirect 지정

ReturnController에 추가

```
@RequestMapping("/redirect.html")
public String redirect() {
        System.out.println("redirect");
        return "redirect:index.html";
}
```

■ post방식의 한글처리를 위한 filter설정

```
 <filter>
        <filter-name>encoding</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>encoding</filter-name>
        <url-pattern>/*</url-pattern>
 </filter-mapping>
```

■ 404에러를 위한 페이지 처리

```
<error-page>
        <error-code>404</error-code>
        <location>/WEB-INF/view/error.jsp</location>
</error-page>
```

- 포워드방식으로 페이지가 보여짐

■ <mvc:view-controller>

1) ParameterizableViewController를 이용한 view 설정을 쉽게

```
<bean name="/join.html"
class="org.springframework.web.servlet.mvc.ParameterizableViewController"/>
```

2) <mvc:view-controller> 이용방법

```
<mvc:view-controller path="/join.html" view-name="join"/>
```

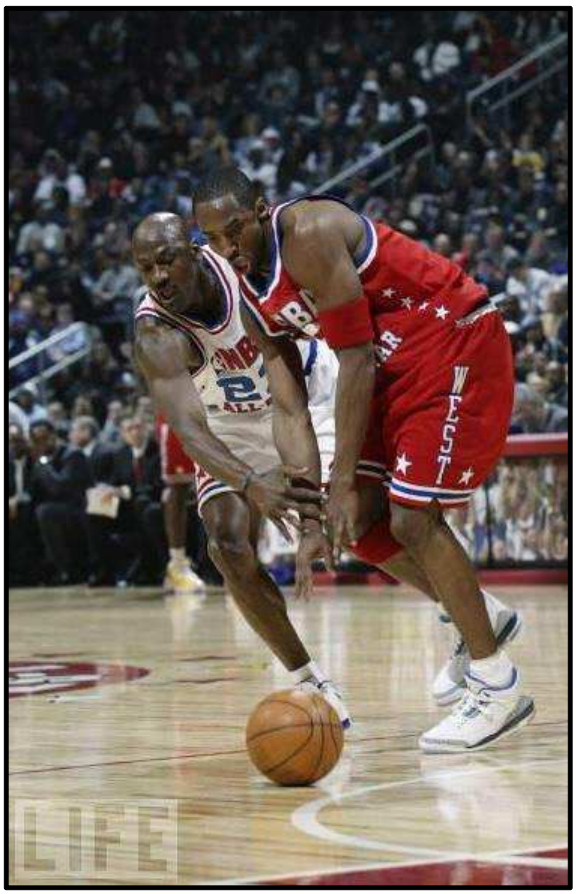
■ <mvc:interceptors>

1) 인터셉터를 사용

```
<mvc:interceptors>
	<mvc:interceptor>
		<mvc:mapping path="/update.html"/>
		<mvc:mapping path="/write.html"/>
		<mvc:mapping path="/delete.html"/>
		<mvc:mapping path="/logout.html"/>
		<bean class="util.LoginCheckInterceptor"/>
	</mvc:interceptor>
</mvc:interceptors>
```

■ 인터셉터란?

1) intercept는 ‘가로채다’라는 뜻으로 농구에서 상대가 가진 볼을 빼앗을때 ‘인터셉트’했다고 표현



2) Spring MVC에서는 요청(예:/update.html)에 대해서 Controller가 작동되기 전에 그 요청을 가로채서 처리하고 컨트롤러로 넘기거나 응답을 할 수 있음

3) 로그인이 필요한 요청에 대한 전처리 가능

■ Spring MVC의 요청 처리 순서

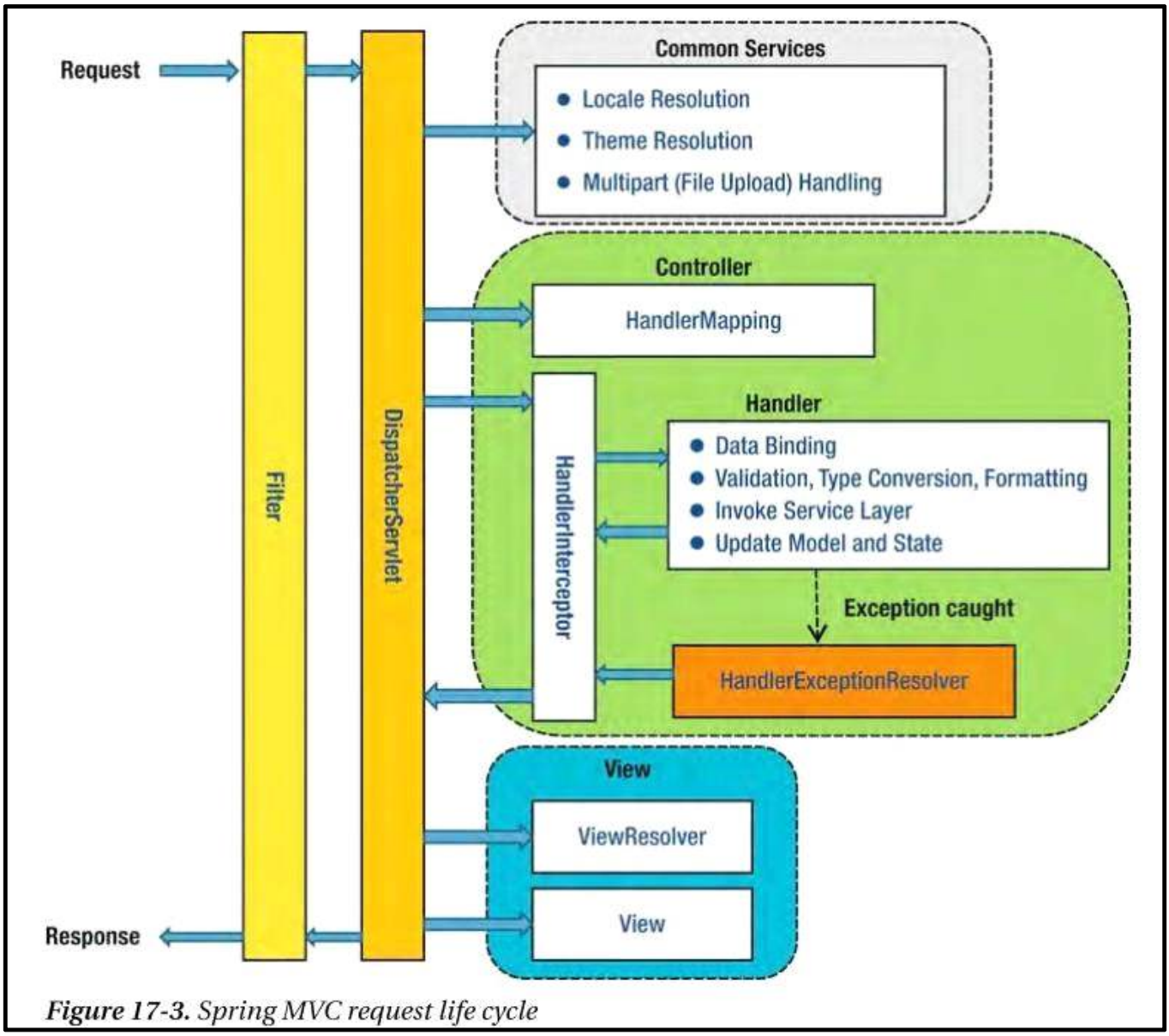


Figure 17-3. Spring MVC request life cycle

2. Interceptor를 이용한 로그인 체크 처리

■ 'update.html', 'write.html', 'delete.html' 등의 요청에는 반드시 로그인이 되어있어야 함

■ 각 메서드에서 로그인 확인을 하면 코드의 중복이 발생

```
@RequestMapping(value="/update.html",method=RequestMethod.GET)
public String updateForm(int no,Model model,HttpSession session) {

	Member login = (Member)session.getAttribute("login");

	if(login==null) {

		 return "redirect:info.html?no="+no;

	}else {
		Board board = boardService.getBoard(no);

		model.addAttribute("board", board);

		return "update";
	}
}
```

■ Interceptor를 이용하면 편리하게 로그인이 되어있는지 확인가능

■ HandlerInterceptor 인터셉터 인터페이스를 이용

메서드명	설명
preHandle()	요청을 컨트롤러에 전달하기 전에 처리
postHandle()	컨트롤러가 요청을 처리한 후에 호출
afterCompletion()	뷰가 클라이언트에 응답된 후에 실행

■ 전부 구현하지 않게 HandlerInterceptorAdapter 상속

```
package util;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import vo.Member;

public class LoginCheckInterceptor extends HandlerInterceptorAdapter {

	@Override
	public boolean preHandle(HttpServletRequest request,
			HttpServletResponse response, Object handler) throws Exception {

		Member login = (Member)request.getSession().getAttribute("login");

		if(login==null) {

			response.sendRedirect("index.html");

			System.out.println("로그인 안되었으므로 컨트롤러로 안가요");
			return false;
		}else {
			System.out.println("로그인 되어있으므로 컨트롤러로 넘김");
			return true;
		}

	}
}
```

■ board-servlet.xml에 설정

```
<mvc:interceptors>
	<mvc:interceptor>
		<mvc:mapping path="/update.html"/>
		<mvc:mapping path="/write.html"/>
		<mvc:mapping path="/delete.html"/>
		<mvc:mapping path="/logout.html"/>
		<bean class="util.LoginCheckInterceptor"/>
	</mvc:interceptor>
</mvc:interceptors>
```

3. Jackson Json View의 사용

■ ajax에서 json으로 응답하는 경우가 필요함

- @ResponseBody를 이용하여 응답 가능
- List의 경우 우리가 직접 json으로 변경하면 어려움
- SpringMVC가 가진 Jackson Json View를 이용

■ ajax.jsp만들기

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>ajax로 불러오기</title>
</head>
<body>
	<h2>글목록</h2>
	<p>
		<button id="requestBtn">글목록 불러오기</button>
	</p>
	<table border="1">
		<caption>글목록</caption>
		<thead>
			<tr>
				<th>no</th>
				<th>title</th>
				<th>writer</th>
				<th>date</th>
				<th>hit</th>
			</tr>
		</thead>
		<tbody>
			<tr>
				<td colspan="5">글이 없습니다.</td>
			</tr>
		</tbody>
	</table>

</body>
</html>
```

■ Controller에서 하는 일은 없음

```
<mvc:view-controller path="/ajax.html" view-name="ajax"/>
```

■ js폴더에 jquery.js를 가져다 놓기

■ jquery에서 list.json을 요청

```
$(function() {
	$("#requestBtn").click(function() {
		$.ajax("list.json",{
			dataType:"json",
			error:function(xhr,error){
				alert(error);
			},
			success:function(json) {
				alert("성공!");
			}
		});
	});
});
```

■ web.xml에 *.json요청도 DispatcherServlet과 맵핑

```
<servlet-mapping>
	<servlet-name>board</servlet-name>
	<url-pattern>*.html</url-pattern>
</servlet-mapping>
<servlet-mapping>
	<servlet-name>board</servlet-name>
	<url-pattern>*.json</url-pattern>
```

```
</servlet-mapping>
```

■ BoardController에 /list.json요청에 대한 메서드 만들기

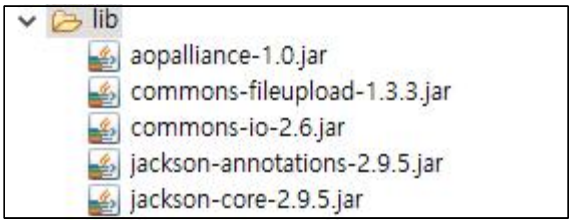
```
@RequestMapping("/list.json")
@ResponseBody
public List<Board> ajax() {

	System.out.println("여기!!");

	return boardService.getList();
}
```

■ 기본적으로 json의 경우 JacksonJson View가 수행됨

- <mvc:annotation-driven> 지정시

파일업로드 처리

■ common-io / common-fileupload 가져다 놓기



■ 리졸버를 설정

```
<bean id="fileRenameUtil" class="com.mcs.board.util.FileRenameUtil" />

<bean id="resizeImageUtil" class="com.mcs.board.util.ResizeImageUtil" />

<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize">
        <value>10000000</value>
    </property>
</bean>
```

■ Controller의 구현

```
@RequestMapping("upload.html")
     public void upload(HttpServletRequest request,
               Model model,
               @RequestParam("image") MultipartFile uploadFile) throws Exception {
          String path = request.getServletContext().getRealPath("/");
          if(!uploadFile.isEmpty()) {

               String name = uploadFile.getOriginalFilename();

               File file = new File(path+File.separator+"upload"+File.separator+name);

               file = renameUtil.rename(file);

               uploadFile.transferTo(file);

               String thumbsPath =
                    path+File.separator+"thumbs"+File.separator+file.getName();
               //thumbs도 만들려면
               resizeImageUtil.resize(file.getAbsolutePath(),thumbsPath, 220);
```

```
            model.addAttribute("uploadImage",file.getName());
        }

    }
```

■ form / 수행결과 페이지

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="UTF-8">
<title>업로드폼</title>
</head>
<body>
     <form action="upload.html" method="post" enctype="multipart/form-data">
     <input type="file" name="image"/>
     <input type="id" name="name"/>
     <button>업로드</button>
     </form>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="UTF-8">
<title>업로드 후</title>
</head>
<body>
<h1>업로드 이미지</h1>
     <img src="thumbs/${uploadImage }" />
<h2>썸즈 이미지</h2>
     <img src="upload/${uploadImage }" />
</body>
</html>
```