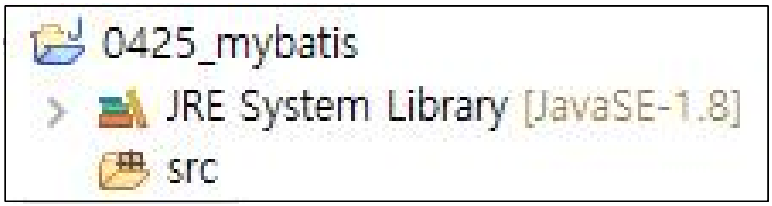
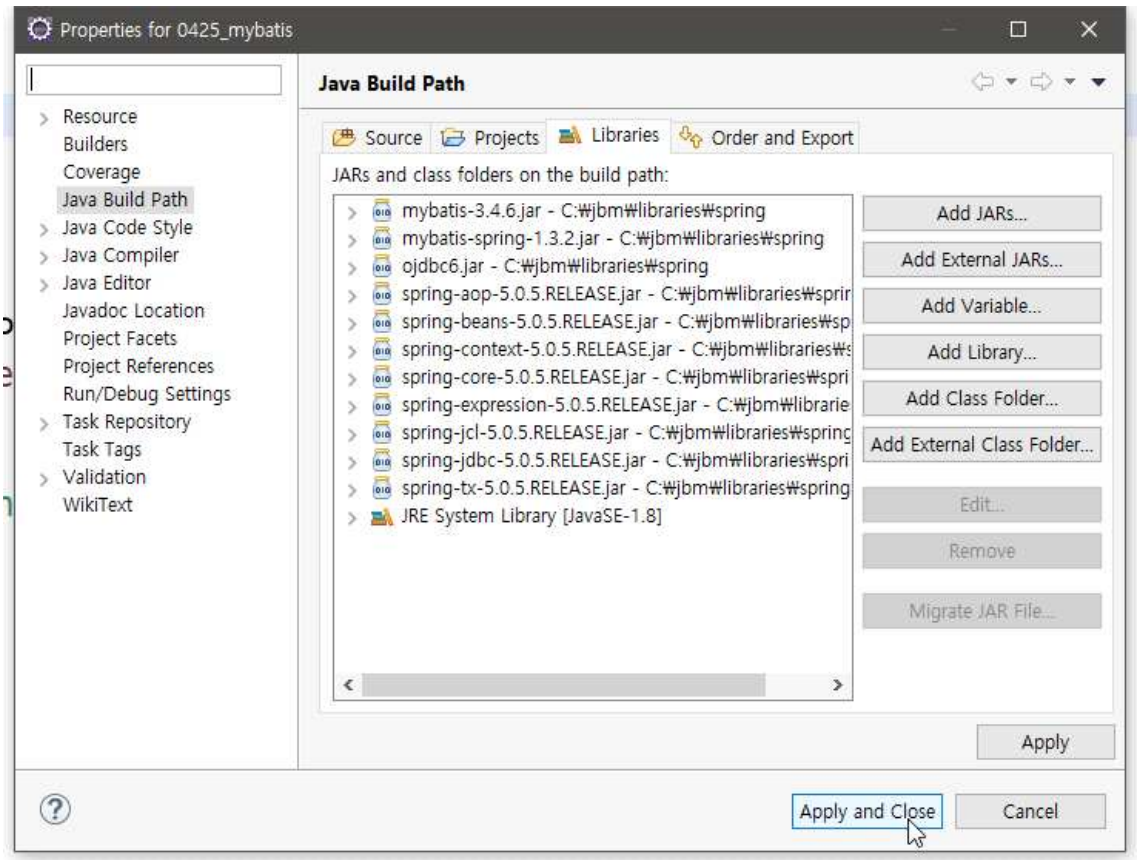
■ Spring과 mybatis 연동

1) java project 생성



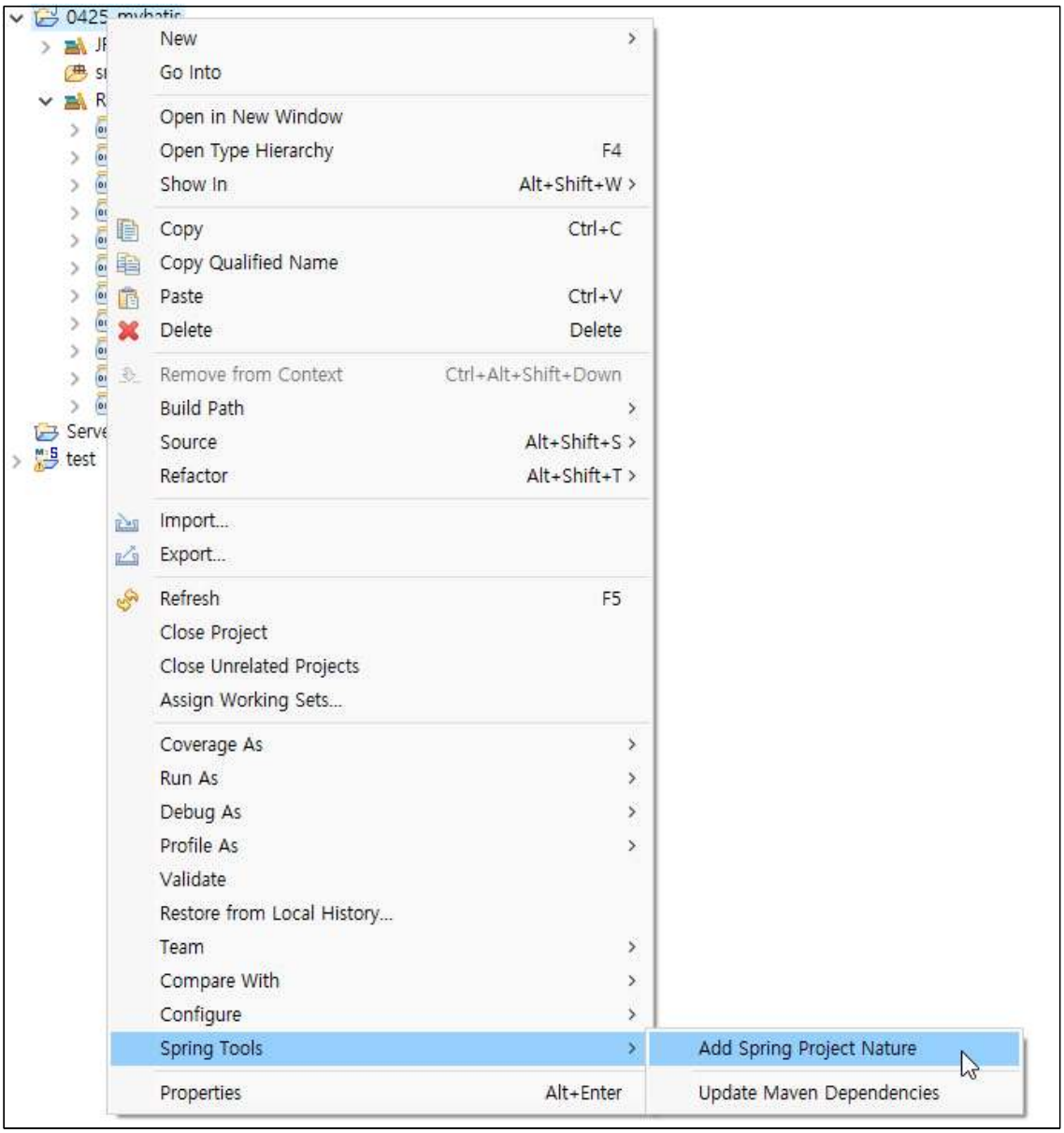
2) build path에 spring library 추가



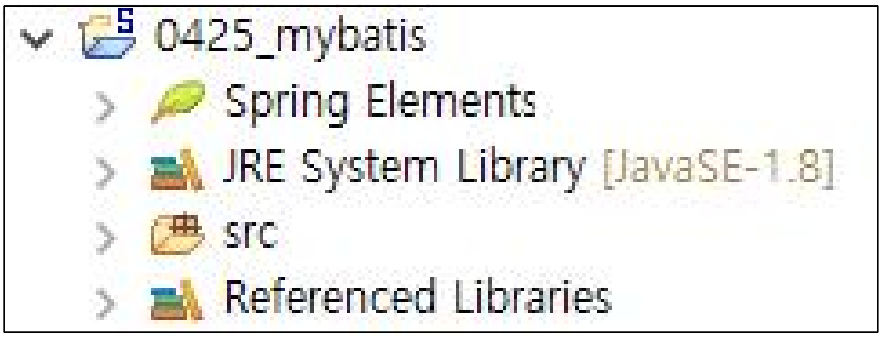
- mybatis : 마이바티스 프레임워크
- mybatis-spring : 마이바티스와 스프링 연동 라이브러리
- ojdbc6 : 오라클 드라이버
- spring-aop : 스프링 aop 라이브러리
- spring-bean : 빈팩토리 라이브러리
- spring-context : 빈팩토리를 확장한 라이브러리
- spring-core : 핵심 라이브러리
- spring-expression : 스프링 내부의 표현언어 라이브러리
- spring-jcl : 자카르타 커먼 로깅 / 로깅 라이브러리

- spring-jdbc : 스프링과 jdbc와 연동 라이브러리
- spring-tx : 트랜잭션관련 라이브러리

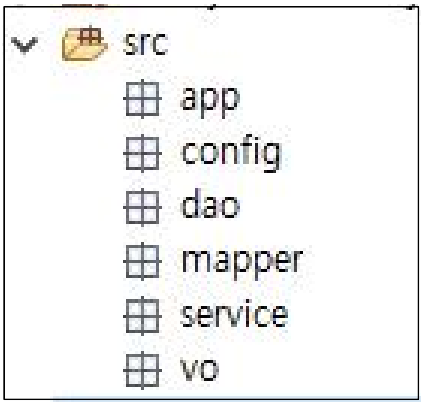
3) 프로젝트에 spring nature 추가



- 프로젝트에 'S'가 붙음



4) app / config / dao / mapper / service / vo 패키지 추가



- service 패키지 : Service 객체들이 들어감

- Service객체가 하는 일 : 트랜잭션 및 비지니스 로직 처리

5) VO / mapper 설정은 동일함

vo.Genre

```
package vo;

public class Genre {

	private int no;
	private String name;

	public Genre() {
	}

	public int getNo() {
		return no;
	}

	public void setNo(int no) {
		this.no = no;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

mapper/genres.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC
"-//mybatis.org/DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="genres">

	<select id="selectList" resultType="vo.Genre">
		SELECT no,name
		FROM genres
	</select>

</mapper>
```

6) DAO는 interface로 선언후 구현클래스 생성

- 왜? : 확장성과 유연성 때문에 무조건 interface로

dao.GenresDAO 인터페이스

```
package dao;

import java.util.List;

import vo.Genre;

public interface GenresDAO {

	public List<Genre> selectList();

}
```

dao.GenresDAOImpl (구현)클래스

```
package dao;

import java.util.List;

import org.apache.ibatis.session.SqlSession;

import vo.Genre;

public class GenresDAOImpl implements GenresDAO {

	//의존성을 setter주입받으려고
	private SqlSession session;

	public void setSession(SqlSession session) {
		this.session = session;
	}


	@Override
	public List<Genre> selectList() {
		return session.selectList("genres.selectList");
	}

}
```

- GenresDAO를 구현

- SqlSession에 의존적이기 때문에 setter주입을 위해 멤버필드 / setter 필요

- try~ catch~ finally 구문과 close()메서드 호출이 필요없음

- 다 알아서 spring이 해줌

- 싱글턴 패턴도 구현할 필요가 없음(스프링의 빈은 기본이 싱글턴)

7) mybatis-config.xml은 JDBC관련 설정이 사라짐

config/mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org/DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

<typeAliases>
        <typeAlias type="vo.Genre" alias="Genre"/>
</typeAliases>

<mappers>
        <mapper resource="mapper/genres.xml"/>
</mappers>

</configuration>
```

8) 대략 한개의 테이블당 한개의 Service 인터페이스와 구현 ServiceImpl

service.GenresService 인터페이스

```
package service;

import java.util.List;

import vo.Genre;

public interface GenresService {

        //SQL구문을 모르는 개발자도 호출할 수 있도록
        //상식적인 이름으로 메서드를 선언
        public List<Genre> getGenres();


}
```

service.GenresServiceImpl 클래스

```
package service;

import java.util.List;

import dao.GenresDAO;
import vo.Genre;

public class GenresServiceImpl implements GenresService {
	//서비스객체는 DAO에 의존적 setter주입
	private GenresDAO dao;

	public void setDao(GenresDAO dao) {
		this.dao = dao;
	}

	@Override
	public List<Genre> getGenres() {
		return dao.selectList();
	}
}
```

9) 이 모든 설정을 applicationContext.xml에서

config/applicationContext.xml

```
<?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">

<!-- DataSource(커넥션풀) 설정 -->
<bean id="dataSource"
p:driverClassName="oracle.jdbc.OracleDriver"
p:username="test"
p:password="1111"
p:url="jdbc:oracle:thin:@localhost:1521:xe"
class="org.springframework.jdbc.datasource.DriverManagerDataSource"/>

<!--  SqlSessionFactory 펙토리 -->
<bean id="sqlSessionFactory"
p:configLocation="config/mybatis-config.xml"
p:dataSource-ref="dataSource"
```

```
class="org.mybatis.spring.SqlSessionFactoryBean"/>
<!-- p:mapperLocations="mapper/*.xml" -->

<!-- SqlSessionTemplate(편리한 기능 : 이게 SqlSession) -->
<bean id="sqlSession"
class="org.mybatis.spring.SqlSessionTemplate">
<constructor-arg ref="sqlSessionFactory"/>
</bean>

<!-- DAO -->
<bean class="dao.GenresDAOImpl"
id="genresDAO"
p:session-ref="sqlSession"
/>

<!-- Service -->
<bean class="service.GenresServiceImpl"
p:dao-ref="genresDAO"
id="genresService"/>

</beans>
```

10) GenreApp에서 실행

app.GenreApp

```
package app;

public class GenresApp {

	public static void main(String[] args) {

		ApplicationContext context =
	new ClassPathXmlApplicationContext("config/applicationContext.xml");

		GenresService service =
				(GenresService)context.getBean("genresService");

		List<Genre> list = service.getGenres();

		for(Genre genre : list) {
			System.out.println(genre.getName());
		}

	}
}
```

■ 웹개발 방식

1) 웹개발방식은 스크립트 방식 / MODEL1 / MODEL2 방식이 존재

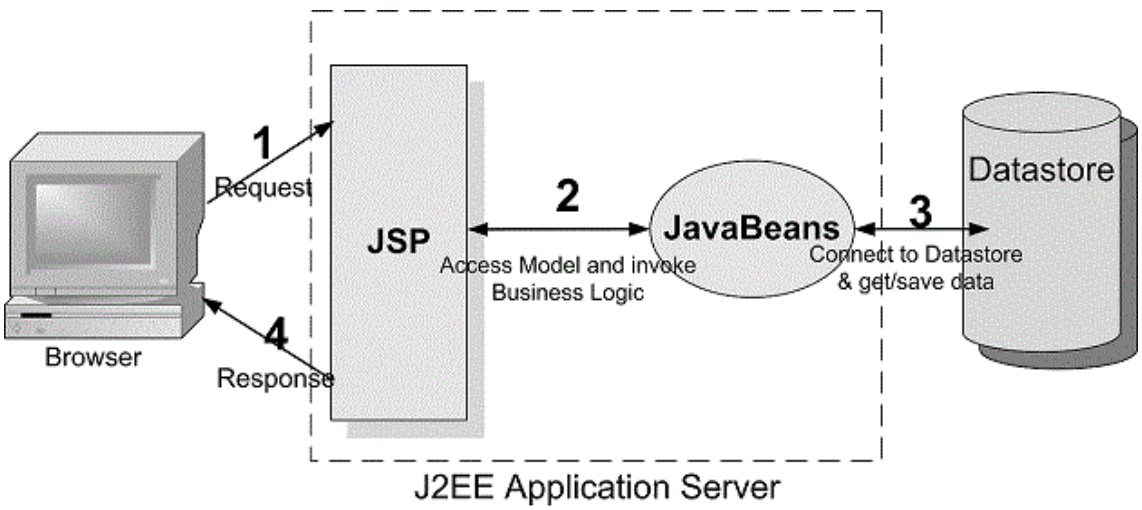
2) 스크립트 방식은 모든 코드를 jsp에

- 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)

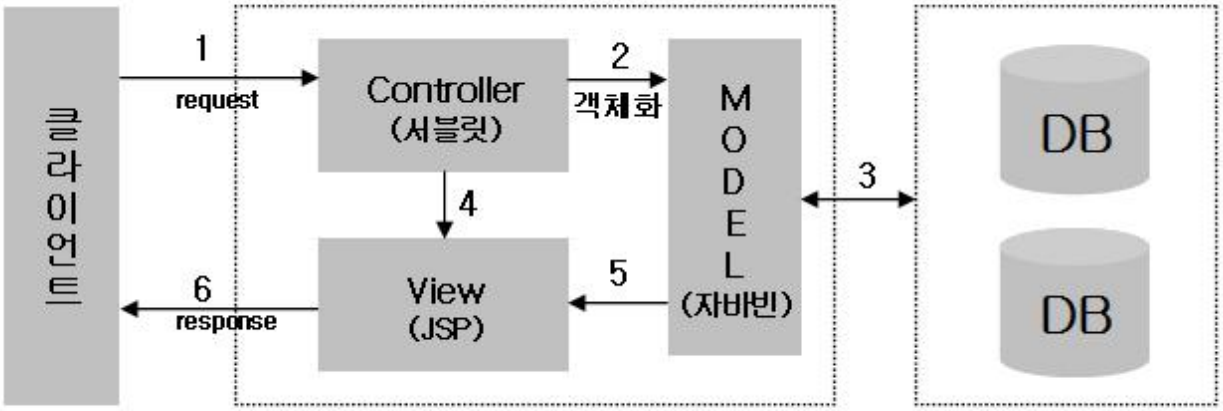
1) Model2방식은 웹 어플리케이션을 개발할 때 MVC패턴을 적용하여 웹 어플리케이션의 개발이 가능하도록 구현한 것

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

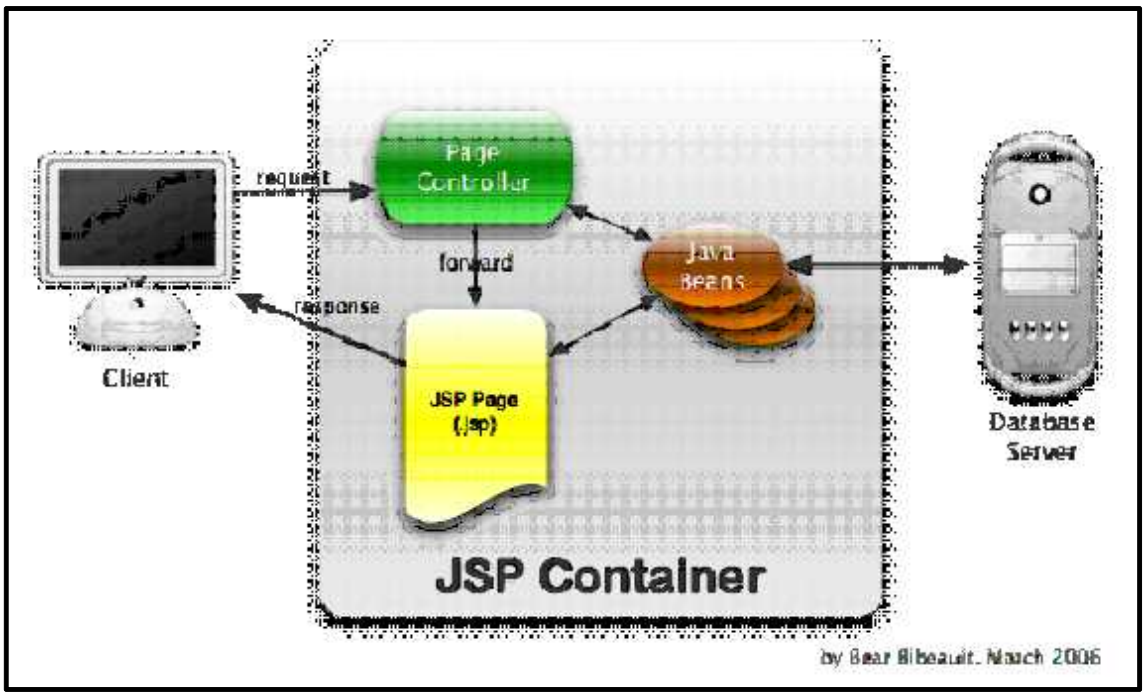
데이터 그 자체

■ 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

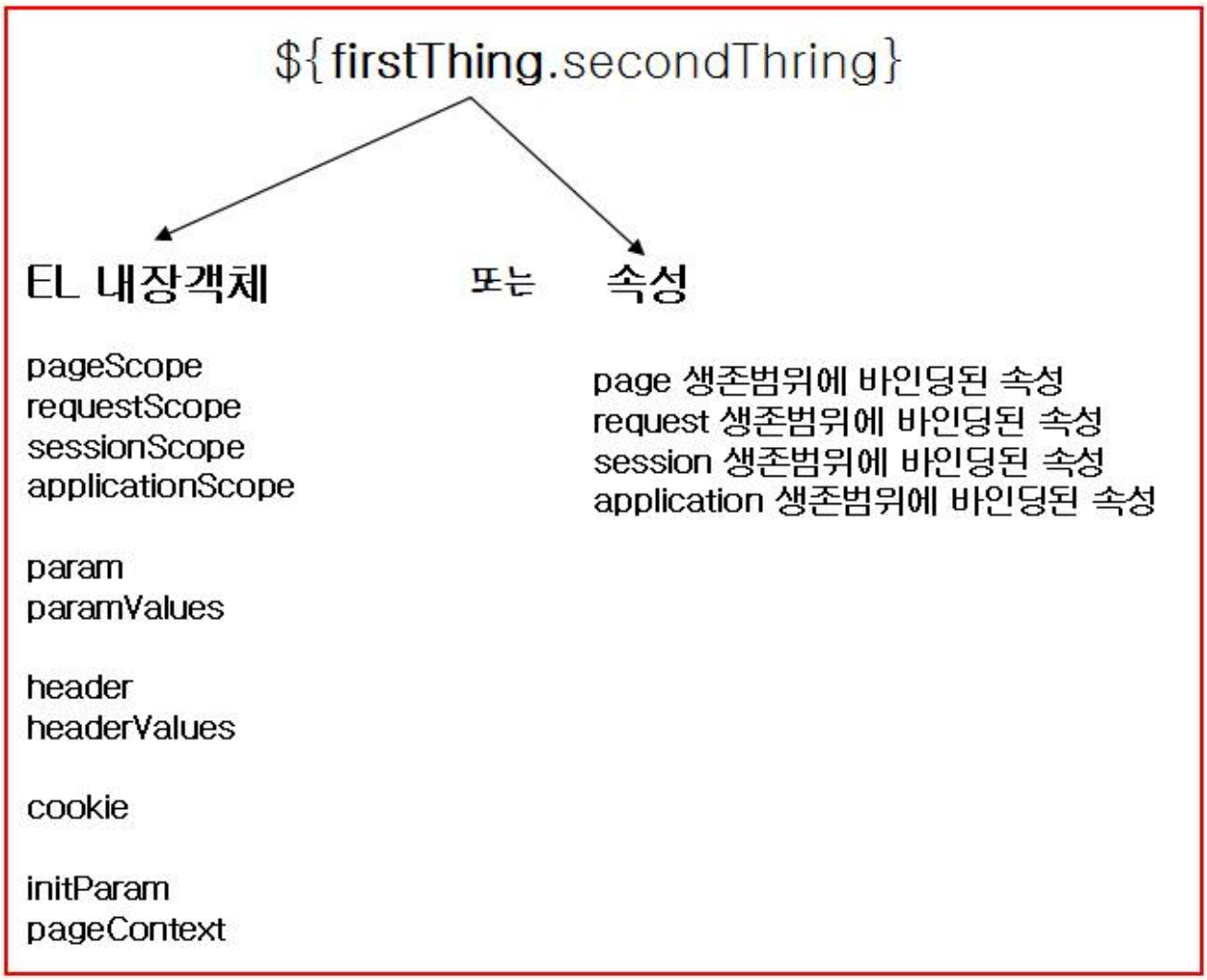
□ EL(Expression Language)

■ 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.

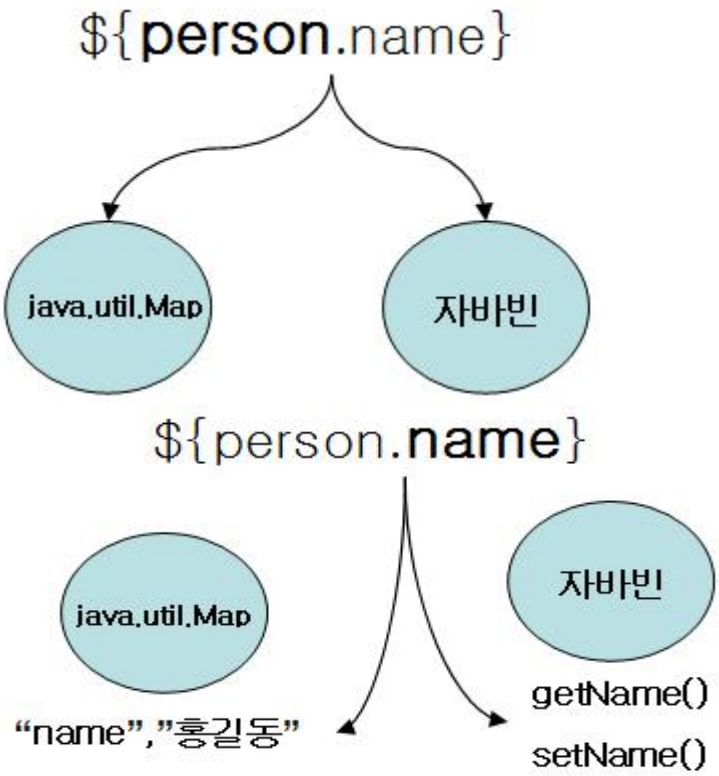
▮ 형식 : ${person.name}



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

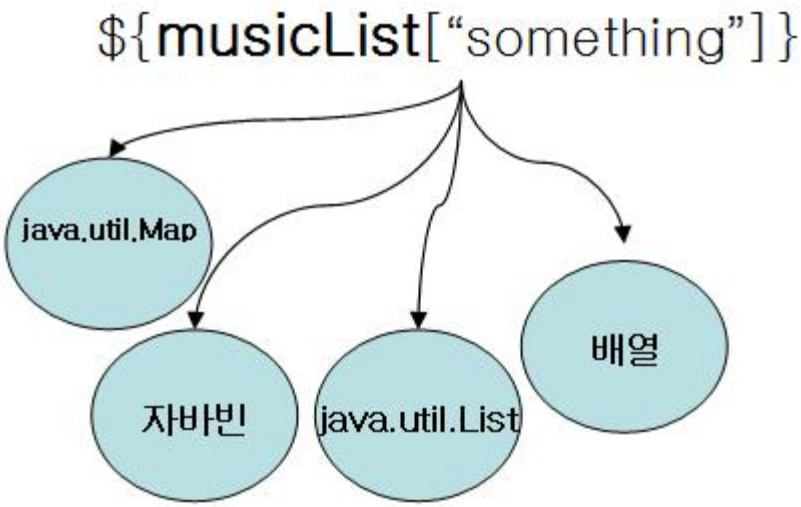
▮ 표현식에서 도트연산자 왼쪽은 반드시 맵 또는 빈이어야 한다.

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

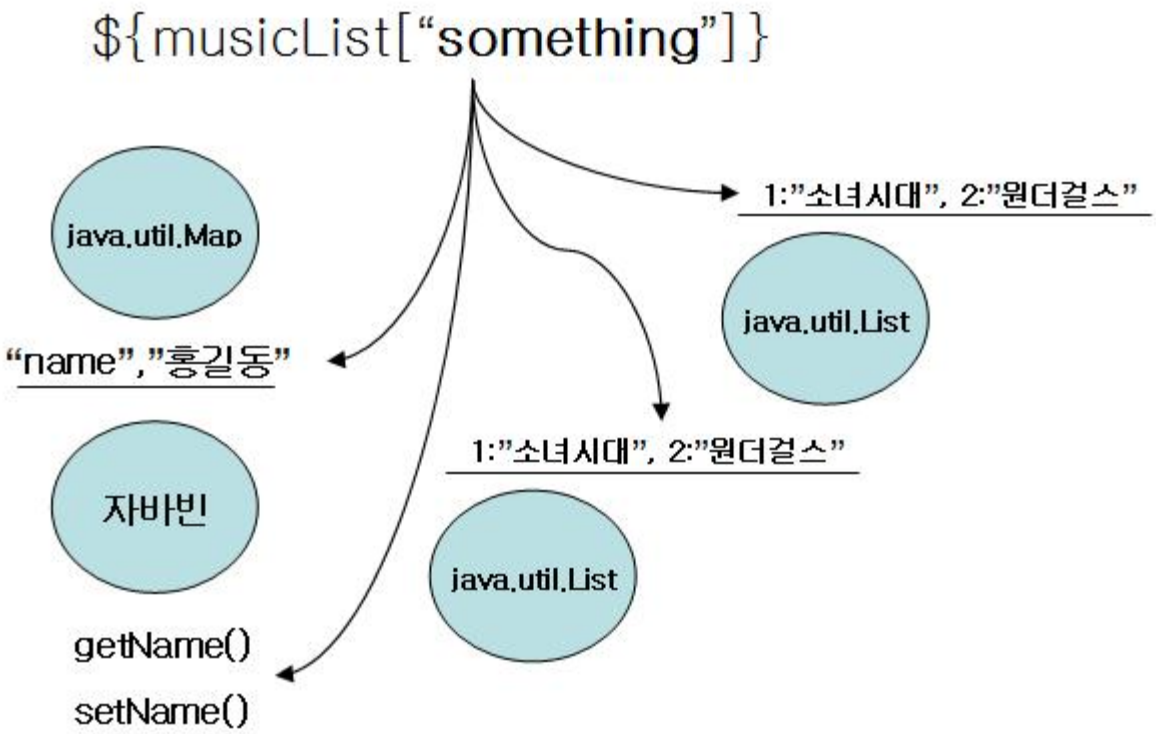


■ [] 연산자 : List나 배열 객체의 값을 표현할 수 있다.

▮ [] 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

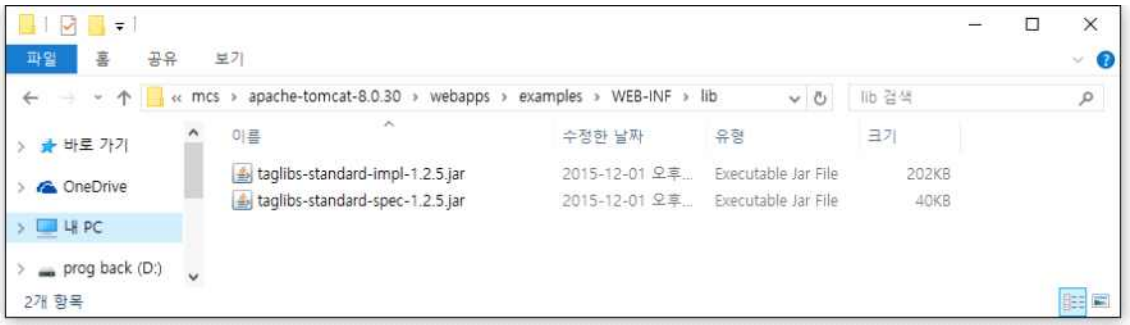
산술 연산자	
더하기	+
빼기	-
곱하기	*
나누기	/ 혹은 div
나머지	% 혹은 mod
논리 연산자	
AND	&& 혹은 and
OR	|| 혹은 or

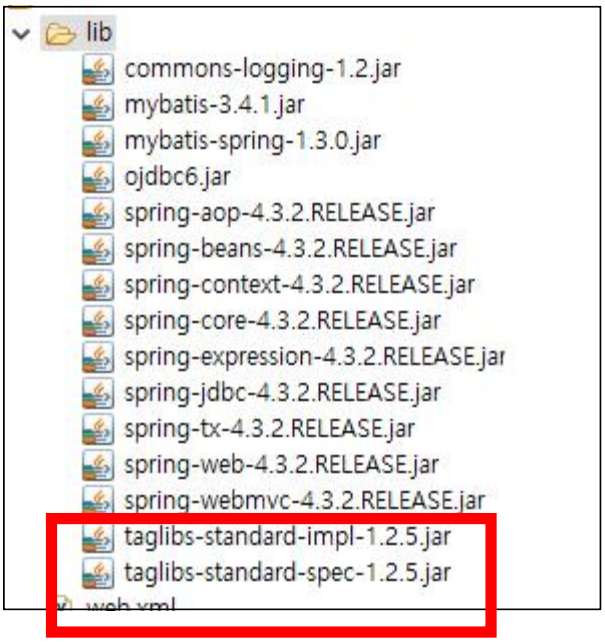
NOT	! 혹은 not
관계 연산자	
등호	== 혹은 eq
부등호	!= 혹은 ne
~보다 작다	< 혹은 lt
~보다 크다	> 혹은 gt
~보다 작거나 같다	<= 혹은 le
~보다 크거나 같다	>= 혹은 ge
예약어	
true	
false	
null	
instanceof	
empty	null이거나 비어있는지 체크하기 위한 연산자. null이거나 비어있으면 true를 반환한다.

□ 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.

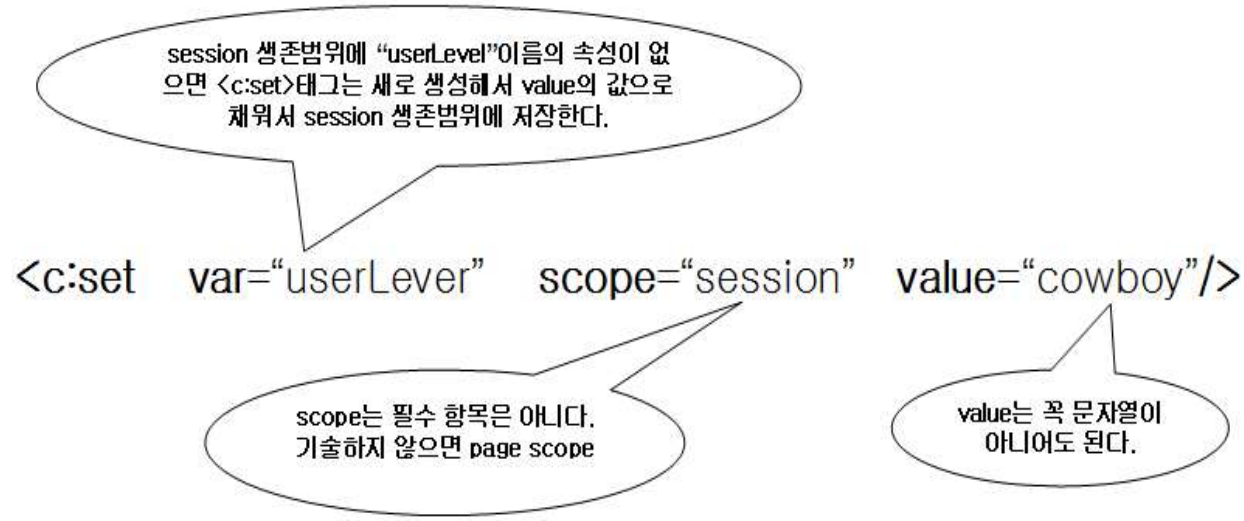
<c:choose>, <c:when>, <c:otherwise> 태그를 사용해서

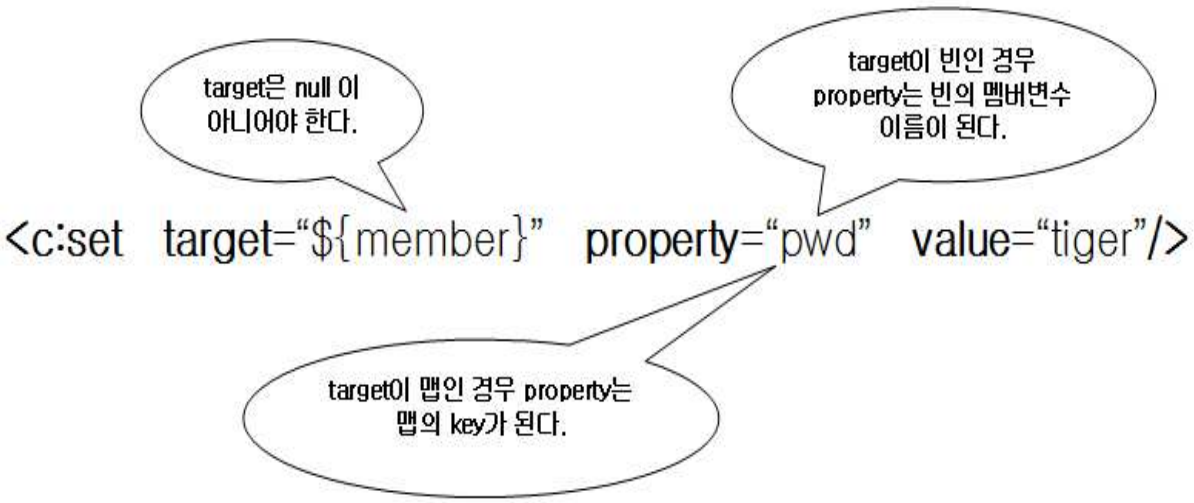
if ~ else if ~ else if ~ else 형태의 분기문 처리를 할 수 있다.

```
<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.

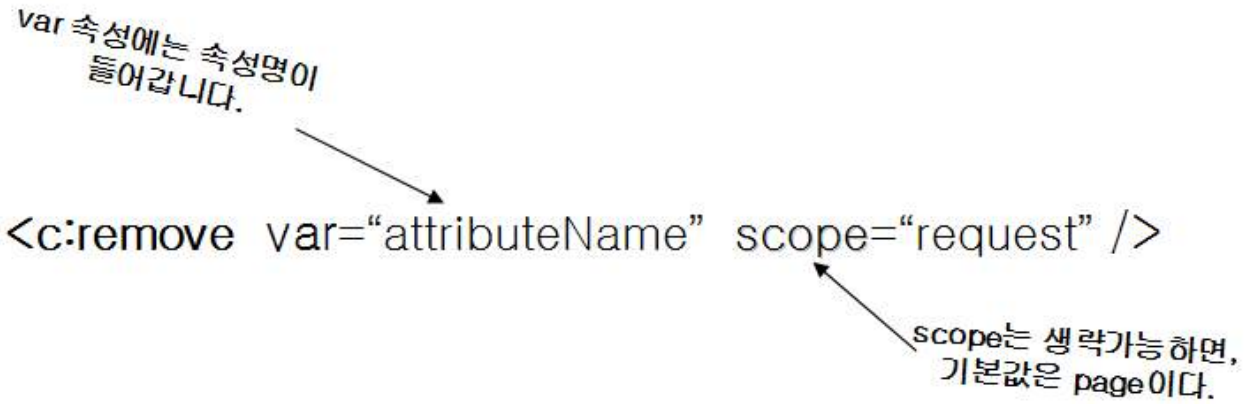
. value가 null인 경우, var의 속성은 사라진다.

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

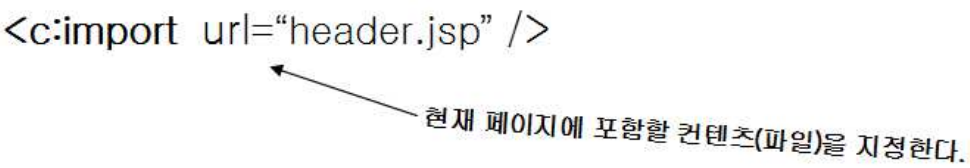
. target에는 실제 개체를 표현하는 표현식이 들어있어야 한다.

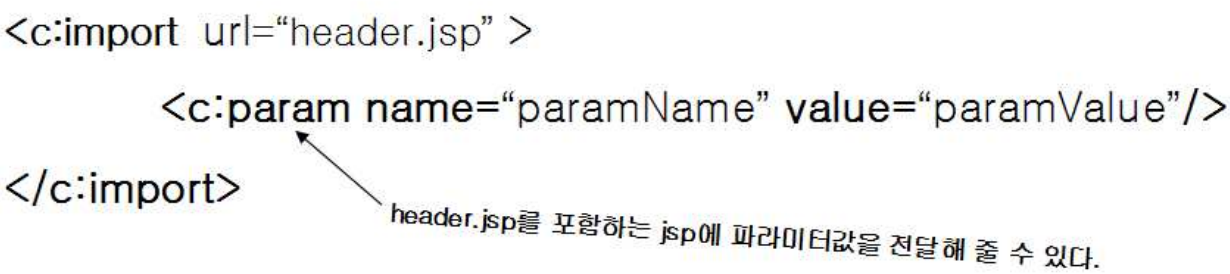
. target에 명시된 표현식으로 찾아진 객체는 반드시 bean이거나 맵이어야 한다.

▮ <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>
```