

# 5장. EL & JSTL



*EL / JSTL*



# EL(Expression Language)

## EL 언어

- MVC 패턴에 따라 뷰(view) 역할을 수행하는 JSP를 더욱 효과적으로 만들려는 목적으로 개발(개발 배경 : 코드와 태그가 섞여서 복잡해짐)
- JSP 페이지에서 자바 코드를 직접 쓰지 않고, 간단한 표현식으로 데이터(변수, 객체, 속성 등)를 접근할 수 있게 해주는 기능을 한다.
- JSP 2.0부터 도입
- **`${...}` 구문을 사용**
- JSP 내에서 자바 코드(<% ... %>)를 최소화하고, 보기 쉽게 데이터를 표현하는 목적
- JSP 내장 객체(request, session, application 등)에 담긴 속성 값을 쉽게 꺼낼 수 있음



# EL(Expression Language)

## EL 언어

- MVC 패턴에 따라 뷰(view) 역할을 수행하는 JSP를 더욱 효과적으로 만들려는 목적으로 개발(개발 배경 : 코드와 태그가 섞여서 복잡해짐)
- JSP의 데이터를 표현할 때 스크립트릿 <% %>이나 표현식 <%= %>을 대체하기 위해 사용되는 언어이다.

```
<jsp:useBean id="test" class="TestBean" />  
<%= test.getName()>
```



```
${test.name}
```

- ① 표현 언어는 \$로 시작한다.
- ② 모든 내용은 '{표현식}'과 같이 표기한다.
- ③ 표현식에는 기본적으로 변수 이름, 혹은 '**객체\_이름**.멤버 변수\_이름' 구조
- ④ 표현식에는 기본적인 연산을 할 수 있다.



## 표현 언어 사용 연산자

### 산술 연산자

연산자	기능	연산자	기능
+	더하기	-	빼기
*	곱하기	/	몫
%	나머지		

### 비교 / 조건 연산자

연산자	기능	연산자	기능
== 또는 eq	같다	!= 또는 ne	같지 않다.
< 혹은 lt	좌변이 우변보다 작다.	> 혹은 gt	좌변이 우변보다 크다
<= 혹은 le	좌변이 우변보다 같거나 작다.	>= 혹은 ge	좌변이 우변보다 크거나 같다
a ? x : y	a가 참이면 x, 거짓이면 y를 반환한다.		

## 표현 언어 사용 연산자

### 논리 연산자

연산자	기 능
&& 또는 and	AND 연산
또는 or	OR 연산
! 또는 not	NOT 연산

### empty 연산자

연산자	기 능
empty <값>	<값> 이 null 이거나 빈 문자열이면 true 를 반환 \${ <b>empty</b> param.n}

# EL 언어

## 표현 언어 실습

```
<h3>문자, 숫자 데이터 표현</h3>
```

```
${300}<br>
```

```
${"감사합니다."}<br>
```

```
${12 * 2}
```

```
${12 / 2}
```

el/el01.jsp

```
<h3>비교, 논리 연산</h3>
```

```
${10 == 11}<br>
```

```
${10 eq 11}<br>
```

```
${10 != 11}<br>
```

```
${10 < 11}<br>
```

```
${10 lt 11}<br>
```

```
${10 > 11}<br>
```

```
${10 gt 11}<br><br>
```

```
${(10 > 11) and (10 != 11)}<br>
```

```
${(10 > 11) or (10 != 11)}<br><br>
```

### 문자, 숫자 데이터 표현

300

감사합니다.

24 6.0

### 비교, 논리 연산

false

false

true

true

true

false

false

false

true



# JSTL(JSP 표준태그 라이브러리)

## JSTL(Jsp Standard Tag Library)

- JSTL은 'JSP Standard Tag Library'의 약자로, JSP에서 자바 코드 대신 사용할 수 있는 표준 태그 라이브러리입니다.
- JSTL을 사용하면 스크립틀릿(<% %>)과 같은 자바 코드를 JSP 페이지에서 제거하여 가독성을 높이고, 반복문, 조건문, 데이터베이스 연동 등 일반적인 작업을 태그 형태로 간편하게 처리할 수 있습니다.
- 핵심(Core), XML, I18N(국제화), 데이터베이스(SQL), 함수(Function) 라이브러리로 구성됨.
- 일반적으로 EL(Expression Language)과 함께 사용함



# JSTL(JSP 표준태그 라이브러리)

## JSTL 라이브러리별 URI 및 prefix

라이브러리	URI	prefix
Core(핵심)	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	c
l18N(데이터포맷)	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	fmt
데이터베이스	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	sql
함수	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	fn





# JSTL(JSP 표준태그 라이브러리)

## JSTL이 제공하는 태그의 종류

태그	기능
Core 태그	<ul style="list-style-type: none"><li>- 변수 선언, 삭제 등 변수와 관련된 작업</li><li>- if문 for문과 같은 제어기능</li><li>- URL 처리로 페이지 이동 기능</li></ul>
Formatting 태그	<ul style="list-style-type: none"><li>- 문자열이나 컬렉션을 처리하는 함수 태그</li><li>- 숫자, 날짜, 시간 등을 형식화하는 기능</li><li>- 다국어 지원 기능을 제공</li></ul>
Functions 태그	<ul style="list-style-type: none"><li>- 문자열을 처리하는 함수 제공</li></ul>



# JSTL(JSP 표준태그 라이브러리)

## JSTL(Jakarta Standart Tag Library) 설치

- Maven Repository : <https://mvnrepository.com/>



### 4. Jakarta Standard Tag Library API

[jakarta.servlet.jsp.jstl](#) » [jakarta.servlet.jsp.jstl-api](#)

Jakarta Standard Tag Library API

Last Release on Aug 22, 2024

Maven

Gradle

SBT

Mill

Ivy

Grape

Leiningen

Buildr

Scope: **Compile** ▼

```
<!-- https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api -->
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>3.0.0</version>
</dependency>
```



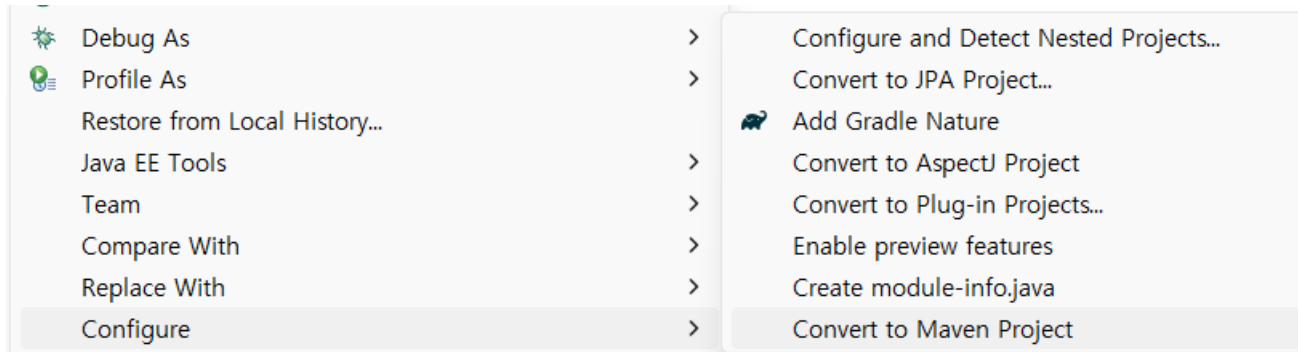
# Maven(메이븐)

## Maven 이란?

- 복잡한 프로젝트의 소스를 체계적으로 컴파일하고 관련 라이브러리의 버전이나 종속성 관리를 쉽게 도와주는 tool(도구)이다.
- 2004년 아파치 프로젝트로 Maven(메이븐)이 출시되어 다수가 사용하는 자바 빌드 도구가 되었으며 특히 스프링 프레임워크 개발에서 기본 빌드 도구로 활용되고 있다.
- 2012년 Gradle(그라이들)이 출시되어 안드로이드 앱 개발의 기본 빌드 도구가 되었고, 스프링 프레임워크에서도 사용되고 있다.

# Maven 기반 프로젝트

프로젝트 > 우클릭 > configure > Convert to Maven Project



pom.xml 생성됨



# Maven 기반 프로젝트

1. pom.xml 에 <dependencies> 의존성 관리 (2개 추가)
2. 프로젝트 > 우클릭 > Maven > update project

```
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>3.0.1</version>
</dependency>
```



# JSTL(JSP 표준태그 라이브러리)

## ■ Core 태그의 종류

태그	설명
<b>&lt;c:set&gt;</b>	사용할 변수를 설정
<b>&lt;c:if&gt; ~ &lt;/c:if&gt;</b>	조건문 처리
<b>&lt;c:forEach&gt;</b>	반복문 처리
<b>&lt;c:choose&gt;   &lt;c:when&gt;   &lt;c:otherwise&gt;   &lt;/c:choose&gt;</b>	다중 조건문 처리 <c:choose>의 서브 태그로 조건문이 참일 때 수행 <c:choose>의 서브 태그로 조건문이 거짓일 때 수행
<b>&lt;c:out&gt;</b>	출력에 사용

# JSTL(JSP 표준태그 라이브러리)

- JSTL 태그 사용하기

```
<%@ taglib prefix="태그 식별이름" uri="태그 지원 URL" %>
```



```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
```

```
<c:if test="${조건식}">출력문</c:if> // 조건문
```

```
<c:choose> //다중 조건문
```

```
<c:when test="${조건식1}">본문 내용1</c:when>
```

```
<c:when test="${조건식2}">본문 내용2</c:when>
```

```
.....
```

```
<c:otherwise> 본문내용 <c:/otherwise>
```

```
</c:choose>
```



# JSTL(JSP 표준태그 라이브러리)

## ▪ JTSL 실습 예제

**set, out**

product1: 삼성 갤럭시

product2: 애플 아이폰

**if**

주문 제품: 애플 아이폰

삼성 갤럭시 이미 추가됨

**forEach**

- 20
- 40
- 10
- 30



# JSTL(JSP 표준태그 라이브러리)

## ▪ JTSL 실습 예제

```
<h2>JTSL 실습 예제</h2>
<h3>set, out</h3>
<c:set var="product1" value="삼성 갤럭시"></c:set>
<c:set var="product2" value="애플 아이폰"></c:set>

<p>product1: <c:out value="${product1}"></c:out></p>
<p>product2: ${product2}</p>

<h3>if</h3>
<c:set var="checkout" value="true" />
<c:if test="${checkout}">
    <p>주문 제품: ${product2}</p>
</c:if>

<c:if test="${!checkout}"> <!-- value="false" 이면 실행 -->
    <p>주문 제품이 아님</p>
</c:if>
```



# JSTL(JSP 표준태그 라이브러리)

## ▪ JTSL 실습 예제

```
<c:if test="${not empty product1}"> <!-- empty이면 출력안됨 -->
    <p>${product1} 이미 추가됨
</c:if>

<h3>forEach</h3>
<!-- value="${[10, 20, 30, 40]}"도 가능 -->
<c:set var="intArray" value="${{10, 20, 30, 40}}" />
<ul>
    <c:forEach var="num" items="${intArray}">
        <li>${num}</li>
    </c:forEach>
</ul>
```



# JSTL(JSP 표준태그 라이브러리)

## ■ 짝수/홀수 판정

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>jstl - core</title>
</head>
<body>
  <!-- 변수 num에 12를 저장 -->
  <c:set var="num" value="12" />

  <!-- 1. c:if문 사용 -->
  <c:if test="${num % 2 == 0}">
    <c:out value="${num}는(은) 짝수입니다." />
  </c:if>
  <c:if test="${num % 2 == 1}">
    <c:out value="${num}는(은) 홀수입니다." />
  </c:if>
</body>
</html>
```



# JSTL(JSP 표준태그 라이브러리)

- 짝수/홀수 판정

```
<!-- c:choose문 사용 -->
<c:choose>
  <c:when test="${num % 2 == 0 }">
    ${num}는(은) 짝수입니다.
  </c:when>
  <c:otherwise>
    ${num}는(은) 홀수입니다.
  </c:otherwise>
</c:choose>
</body>
```

# JSTL(JSP 표준태그 라이브러리)

## ■ 학점 계산

```
<c:set var="score" value="89" />
<h2>시험 점수: ${score}</h2>
<c:choose>
  <c:when test="${score}>=90 && score<=100}">
    <h3>A학점</h3>
  </c:when>
  <c:when test="${score}>=80}">
    <h3>B학점</h3>
  </c:when>
  <c:when test="${score}>=70}">
    <h3>C학점</h3>
  </c:when>
  <c:when test="${score}>=60}">
    <h3>D학점</h3>
  </c:when>
  <c:otherwise>
    <h3>F학점</h3>
  </c:otherwise>
</c:choose>
```

시험 점수: 89

B학점



# JSTL(JSP 표준태그 라이브러리)

- 학점 계산 - 사용자 입력 폼



```
<h2>학점 계산기</h2>
<form action="scorePoint.jsp" method="get" name="form1">
  <p>점수 입력 <input type="text" name="score">
    <button type="button" onclick="checkScore()">학점 출력</button></p>
</form>
```

# JSTL(JSP 표준태그 라이브러리)

## ■ 학점 계산 - 사용자 입력 폼

```
<c:set var="score" value="${param.score}" />
<h2>시험 점수 : ${score}</h2>
<c:choose>
  <c:when test="${score}>=90 && score<=100}">
    <h3>A학점입니다.</h3>
  </c:when>
  <c:when test="${score}>=80 && score<90}">
    <h3>B학점입니다.</h3>
  </c:when>
  <c:when test="${score}>=70 && score<80}">
    <h3>C학점입니다.</h3>
  </c:when>
  <c:when test="${score}>=60 && score<70}">
    <h3>D학점입니다.</h3>
  </c:when>
  <c:otherwise>
    <h3>F학점입니다.</h3>
  </c:otherwise>
</c:choose>
```

scorePoint.jsp



# JSTL(JSP 표준태그 라이브러리)

## ▪ 학점 계산 – 유효성 검사(validation)

```
<script>
    function checkScore(){
        //alert("test");
        let form = document.form1;
        let score = form.score.value.trim(); //공백 제거
        console.log(score);
        console.log(typeof(score)); //string

        if(score == "" || isNaN(score)){
            alert("점수를 입력하세요(숫자만).");
            form.score.focus(); //커서 위치
            return false;
        }
        if(score < 0 || score > 100){
            alert("0~100 사이의 점수를 입력하세요.");
            form.score.focus(); //커서 위치
            return false;
        }

        form.submit(); //서버로 전송
    }
</script>
```





# JSTL(JSP 표준태그 라이브러리)

- 반복문

```
<c:forEach var=변수이름" items="반복할 객체이름"  
    begin="시작값" end=" 마지막값" step="증가값">  
</c:forEach>
```

```
<%@ taglib uri="jakarta.tags.core" prefix="c" %>
```

```
<c:forEach var="i" begin="1" end="4" step="1">  
        ${i}번 반복<br>  
</c:forEach>
```

forEach.jsp

1번 반복  
2번 반복  
3번 반복  
4번 반복



# JSTL(JSP 표준태그 라이브러리)

## ■ 반복문

```
<%  
    String[] names = {"홍길동", "이순신", "유관순"};  
    request.setAttribute("names", names);  
%>  
  
<!-- varStatus 속성  
    status.index → 0부터 시작하는 인덱스  
    status.count → 1부터 시작하는 반복 횟수 -->  
  
<c:forEach var="name" items="${names}" varStatus="status">  
    ${status.count} : ${name}<br>  
</c:forEach>
```

1 : 홍길동  
2 : 이순신  
3 : 유관순



# JSTL(JSP 표준태그 라이브러리)

- 반복문

```
<h2>구구단(5단)</h2>
<c:set var="dan" value="5" />
<c:forEach var="i" begin="1" end="9">
    ${dan} x ${i} = ${dan*i}<br>
</c:forEach>
<p>-----</p>

<h2>구구단 전체</h2>
<c:forEach var="i" begin="2" end="9">
    [${i}] 단<br>
    <c:forEach var="j" begin="1" end="9">
        ${i} x ${j} = ${i*j}<br>
    </c:forEach>
<br>
</c:forEach>
```

gugudan.jsp



# fmt 태그를 이용한 날짜/시간 표기

**formatDate** 태그 - 날짜 정보를 담고 있는 객체를 형식화하여 출력함

```
<%@ taglib uri="jakarta.tags.fmt" % prefix="fmt" >
```

## 날짜를 다양한 형식으로 표기

현재 날짜 및 시간 : Tue Oct 14 07:29:18 KST 2025

날짜: 2025. 10. 14.

시간: 오전 7:29:18

날짜와 시간: 2025. 10. 14. 오전 7:29:18

## 패턴 지정 예제

yyyy-MM-dd 형식: 2025-10-14

yyyy년 MM월 dd일 a hh:mm:ss 형식: 2025년 10월 14일 오전 07:29:18

## 로케일 지정 예제

영문 형식: Tuesday, October 14, 2025, 7:29:18 AM

한글 형식: 2025년 10월 14일 화요일 오전 7:29:18



# fmt 태그를 이용한 날짜/시간 표기

## formatDate 태그

```
<%@ taglib uri="jakarta.tags.fmt" prefix="fmt" %>
```

```
<h2>날짜를 다양한 형식으로 표기</h2>
```

```
<jsp:useBean id="now" class="java.util.Date" />
```

formatDate.jsp

```
<p>현재 날짜 및 시간 : ${now}</p>
```

```
<!-- fmt:formatDate 사용 -->
```

```
<p>날짜:
```

```
    <fmt:formatDate value="${now}" type="date" />
```

```
</p>
```

```
<p>시간:
```

```
    <fmt:formatDate value="${now}" type="time" />
```

```
</p>
```

```
<p>날짜와 시간:
```

```
    <fmt:formatDate value="${now}" type="both" />
```

```
</p>
```

```
<hr>
```



# fmt 태그를 이용한 날짜/시간 표기

## formatDate 태그

```
<h3>패턴 지정 예제</h3>
<p>yyyy-MM-dd 형식:
    <fmt:formatDate value="${now}" pattern="yyyy-MM-dd" />
</p>
<p>yyyy년 MM월 dd일 a hh:mm:ss 형식:
    <fmt:formatDate value="${now}" pattern="yyyy년 MM월 dd일 a hh:mm:ss" />
</p>

<hr>

<h3>로케일 지정 예제</h3>
<fmt:setLocale value="en_US" />
<p>영문 형식:
    <fmt:formatDate value="${now}" type="both" dateStyle="full" />
</p>

<fmt:setLocale value="ko_KR" />
<p>한글 형식:
    <fmt:formatDate value="${now}" type="both" dateStyle="full" />
</p>
```



# fmt 태그를 이용한 숫자 표기

## formatNumber 태그

사용자의 로케일(Locale)에 따라 숫자를 다양한 형식으로 출력함

### 숫자 표기

가격: 25000

가격: 25,000

가격: 25,000

### 통화 표기

한국 원(KRW): ₩25,000

일본 엔(JPY): ¥ 25,000

미국 달러(USD): \$25,000.00

### 퍼센트 표기

기본 퍼센트 표시: 26%

소수점 1자리까지 표시: 25.7%

# fmt 태그를 이용한 숫자 표기

## formatNumber 태그

```
<h3>숫자 표기</h3>
<:set var="price" value="25000"></:set>
<p>가격: ${price}</p>

<p>가격: <fmt:formatNumber value="25000" />
<p>가격: <fmt:formatNumber value="${price}" />
<hr>

<h3>통화 표기</h3>
<fmt:setLocale value="ko_KR" />
<p>한국 원(KRW):
    <fmt:formatNumber value="${price}" type="currency" />
</p>

<fmt:setLocale value="ja_JP" />
<p>일본 엔(JPY):
    <fmt:formatNumber value="${price}" type="currency" />
</p>
```

formatNumber.jsp





# fmt 태그를 이용한 숫자 표기

## formatNumber 태그

formatNumber.jsp

```
<fmt:setLocale value="en_US" />
<p>미국 달러(USD):
    <fmt:formatNumber value="${price}" type="currency" />
</p>
<hr>

<h3>퍼센트 표기</h3>
<c:set var="rate" value="0.257" />

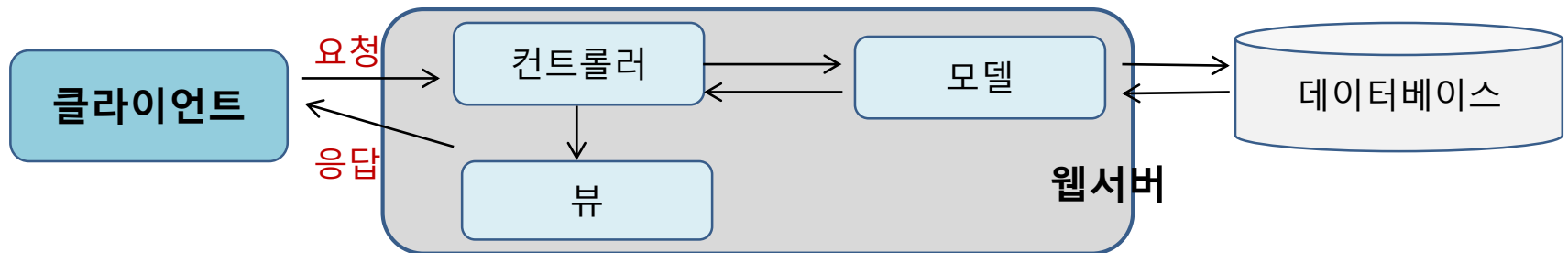
<p>기본 퍼센트 표시:
    <fmt:formatNumber value="${rate}" type="percent" />
</p>
<p>소수점 1자리까지 표시:
    <fmt:formatNumber value="${rate}" type="percent" maxFractionDigits="1" />
</p>
```



# MVC란?

## ● MVC란?

- Model, View, Controller의 약자로 웹 애플리케이션을 비즈니스 로직, 프레젠테이션 로직, 데이터로 분리하는 디자인 패턴이다.
- 클라이언트의 요청 처리, 응답 처리, 로직 처리 부분을 모듈화한 구조이다.



## ● MVC 구성 요소와 기능

### Controller

- 서블릿이 컨트롤러의 역할을 한다.
- 클라이언트의 요청을 분석한다.
- 요청에 대해서 필요한 모델을 호출한다.
- Model에서 처리한 결과를 보여주기 위해 jsp를 선택한다.

### Model

- 데이터베이스 연동과 같은 비즈니스 로직을 수행한다.
- DTO(VO)와 DAO클래스로 이루어져 있다.

### View

- JSP가 화면 기능을 담당한다..
- Model에서 처리한 결과를 화면에 표시한다.



- EL 언어에서 사용할 수 있는 내장 객체(데이터 저장소)

내장 객체	기 능
pageScope	page 영역의 생명 주기에서 사용되는 저장소
<b>requestScope</b>	request 영역의 생명 주기에서 사용되는 저장소
sessionScope	session 영역의 생명 주기에서 사용되는 저장소
applicationScope	application 영역의 생명 주기에서 사용되는 저장소

## ● MVC 실습

### 변수, 배열

현재 계절: 가을

현재 계절 : 가을

전체 계절

봄

여름

가을

겨울

### 리스트(List)

과일: 사과

과일 장바구니: 사과 바나나 딸기

### 맵(Map)

모델명: EV6

연식: 2025

- 컨트롤러(서블릿) - 모델을 만들어 뷰에 보내줌

```
@WebServlet("/model")
public class ModelController extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        //변수
        String season = "가을";

        //배열
        String[] seasons = {"봄", "여름", "가을", "겨울"};

        //"season"을 모델로 뷰(mvc01)에 전달
        request.setAttribute("season", season);
        request.setAttribute("seasons", seasons);
    }
}
```

- 컨트롤러(서블릿) - 모델을 만들어 뷰에 보내줌

```
// List
List<String> fruits = Arrays.asList("사과", "바나나", "딸기");
request.setAttribute("fruits", fruits);

// map
Map<String, Object> cars = new HashMap<>();
cars.put("brand", "EV6");
cars.put("year", 2025);
request.setAttribute("cars", cars);

//포워딩 - /model 경로에서 mvc01.jsp 출력함
RequestDispatcher rd =
    request.getRequestDispatcher("/model/mvc01.jsp");
rd.forward(request, response);
}
```

## ● 뷰(mvc01.jsp)

```
<h3>변수, 배열</h3>
<p>현재 계절: ${season}</p>
<p>현재 계절 : ${seasons[2]}</p>
<p>전체 계절</p>
<c:forEach var="season" items="${seasons}">
    ${season} <br>
</c:forEach>
<hr>

<h3>리스트(List)</h3>
<p>과일: ${fruits[0]}</p>
<p>과일 장바구니:
<c:forEach var="fruit" items="${fruits}">
    ${fruit}
</c:forEach>
<hr>

<h3>맵(Map)</h3>
<p>모델명: ${cars.brand}</p>
<p>연식: ${cars.year}</p>
```

/model/mvc01.jsp

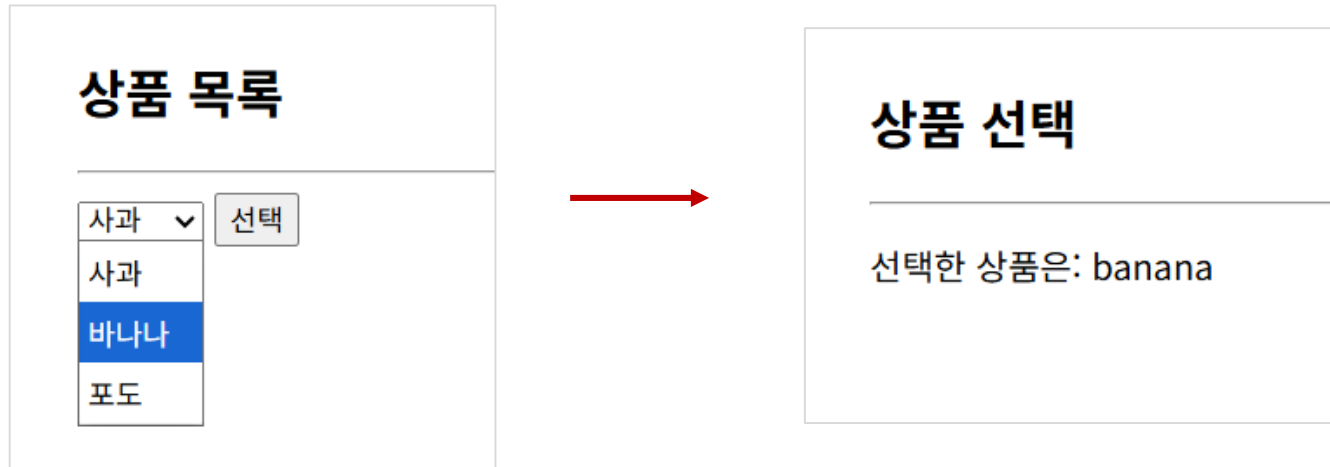


## ■ 상품 출력 프로그램

### 프로그램 소스 목록

파일 이름	역 할
productList.jsp	상품 목록을 출력하기 위한 jsp 파일
selProduct.jsp	productList에서 item을 선택하고 <확인> 버튼을 누르면 호출되는 jsp로 표현언어를 이용해 데이터 출력
ProductServlet.java	컨트롤러 역할을 하는 서블릿으로 선택된 상품을 모델 데이터로 저장하여 뷰(selProduct.jsp)에 보내준다.

## ■ 상품 출력 프로그램



## ■ 상품 출력 프로그램

```
<section id="container">
  <h2>상품 목록</h2>
  <hr>
  <form action="/jweb02/product" method="get">
    <select name="fruit">
      <option value="apple">사과</option>
      <option value="banana">바나나</option>
      <option value="grape">포도</option>
    </select>
    <input type="submit" value="선택">
  </form>
</section>
```

## ■ 상품 출력 프로그램

productServlet.java

```
@WebServlet("/product")
public class ProductServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        //선택한 상품 받기
        String fruit = request.getParameter("fruit");

        //모델로 저장하기
        request.setAttribute("fruit", fruit);

        //뷰로 포워딩하기
        RequestDispatcher rd =
            request.getRequestDispatcher("/product/selProduct.jsp");
        rd.forward(request, response);
    }
}
```

## ■ 상품 출력 프로그램

```
<title>선택 결과</title>
<style>
    #container{width: 80%; margin: 30px auto;}
</style>
</head>
<body>
    <div id="container">
        <h2>상품 선택</h2>
        <hr>
        <p>선택한 상품 : ${fruit}</p>
    </div>
</body>
```

selProduct.jsp