

## Servlet& JSP 시작하기

### Servlet과 JSP의 탄생 이야기

#### 1990년대 후반, 웹 애플리케이션의 초창기

웹이 점점 대중화되면서, 기업들도 웹을 통해 정보를 제공하거나 사용자와 상호작용하기 시작했는데 이때 주로 사용된 기술이 CGI (Common Gateway Interface)였다.

#### 문제점 발생 - CGI의 한계

CGI는 서버에서 하나의 요청마다 새로운 프로세스를 생성해서 처리한다.

**하지만** 사용자가 많아지면?

**서버에 과부하 발생! -> 매 요청마다 프로세스 생성 → 느려짐 → 자원 낭비 → 비효율적!**

이 문제를 해결하고자 **Java** 진영에서 **새로운 해법**을 제시

#### Servlet의 등장 (Java EE 표준 기술)

Java는 객체지향, 멀티스레드 등 안정적인 특징을 갖고 있었기 때문에, 웹 개발에서도 Java를 활용하자는 움직임이 생기면서 Servlet 탄생

- ✓ Java로 작성한 서버 측 프로그램
- ✓ 클라이언트 요청마다 새로운 **스레드**로 처리 (프로세스보다 훨씬 가볍고 빠름)
- ✓ 재사용 가능, 유지보수 용이, 확장성 좋음

**Servlet은 웹의 동적 처리를 효율적으로 처리할** 수 있었고, CGI의 대안으로 자리 잡았다.

**그런데 또 다른 문제 등장...**

Servlet은 자바 코드 안에 HTML을 문자열로 써야 하는 어려움

```
out.println("<html>");  
out.println("<body>");  
out.println("<h1>Welcome</h1>");  
out.println("</body>");  
out.println("</html>");
```

HTML 코드가 Java 코드에 뒤섞여서 너무 복잡하고 비효율적!

## 그래서 등장한 해법

### JSP (JavaServer Pages)의 등장

Java 코드와 HTML을 역전시킨 기술이다.

- ✓ HTML 안에 Java 코드를 삽입
- ✓ 디자이너와 개발자가 협업하기 쉬움
- ✓ 유지보수성 높음

```
<html>  
<body>  
<%  
    String name = "Grace";  
%>  
<h1>Welcome, <%= name %></h1>  
</body>  
</html>
```

이 JSP는 내부적으로 **Servlet**으로 변환되어 실행된다. 즉, **JSP는 서블릿을 더 쉽게 작성하게** 해준다.

## 웹 개발이 점점 커지고 복잡 해지던 시절

개발자 A가 혼자 웹 페이지를 만들었는데 HTML에 Java 코드 섞고, DB까지 연결하고, 요청도 받

고 다 한 파일에서 처리했다. 예전에는 이렇게 해도 페이지가 몇 개 안되어 괜찮았지만,

**2010년대** 오면서 온라인시장이 활성화되고 웹프로젝트 규모가 커지면서 역할이 섞여 협업과 유지보수가 너무 힘들어 졌다.

#### 서비스가 커지면서...

- "이거 디자인 바꾸려면 Java 코드도 수정해야 돼?"
- "비즈니스 로직 수정하려 했는데 HTML 까지 깨짐..."
- 기능 하나 수정하려고 했는데, 엉뚱한 화면이 깨지고 디자인 수정하다가 로직에 버그 생기고 테스트도 힘들고... 결국 프로젝트는 점점 혼돈 속으로...

코드가 너무 뒤섞여 있어서

**프론트 개발자, 백엔드 개발자, 디자이너가 서로 충돌이 생겼다.**

그래서 나온 구조가 **MVC 패턴!**

**역할을 분리해서 각자 책임만 딱!** 지도록 만든 구조가 바로 **MVC 패턴**

**M (Model)** - 데이터를 다루는 DB 처리, 비즈니스로직 담당

**V (View)** - HTML, 디자인, 데이터만 출력 담당

**C (Controller)** - 요청을 받으면 어떤 로직(Model) 실행할지 결정하고, 결과를 어떤 화면(View)에 보여줄지 선택

#### MVC구조 개념!!! -> Model 2방식

3tier Architecture(Presentation Layer, Business Layer, Data Access=Persistence Layer)

1) Model :JAVA(BusinessLogic, Data Access Object, DTO...)

2) View : HTML, CSS, JS, JSP

3) Controller : Servlet

(Model과 View사에서 클라이언트의 요청(request)을 받아

Model쪽으로 보내고 다시 그 결과 응답(response)을 브라우저로 전송 )

## Servlet이란?

Servlet은 Java EE(지금은 Jakarta EE)의 표준 기술 중 하나로,

📌 **Java 언어로 작성된 서버 측 프로그램으로,**

📌 **웹 브라우저의 요청(Request)을 받아서 처리하고, 그에 대한 응답(Response)을 생성하는 역할**을 한다. 즉, 동적인 웹 페이지를 만들기 위한 Java 기반의 기술이다.

## Servlet의 특징

특징	설명
동적 처리	사용자의 요청에 따라 동적으로 데이터를 처리하고 결과를 생성
자바 기반	Java 언어로 작성되어 안정성과 확장성이 높음
웹 컨테이너 필요	Tomcat 같은 웹 서버(컨테이너)에서 실행됨
HTTP 처리	주로 <code>HttpServlet</code> 을 상속받아 HTTP 요청/응답 처리

## 예시)

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello Servlet!</h1>");
    }
}
```

## Servlet API

<https://jakarta.ee/specifications/servlet/6.0/apidocs/jakarta.servlet/module-summary.html>

## Servlet 작성법

1) `HttpServlet`을 상속받는다.

- 2) 반드시 public class로 선언한다.
- 3) 필요한 메소드 오버라이딩 한다.
- 4) 작성한 servlet문서가 브라우저에서 실행되기 위해서 등록(생성) 과정이 필요하다.

**등록 방법은 두 가지 있다.**

☞ web.xml문서에 등록하는 방법

web.xml문서는 웹 애플리케이션의 설정을 정의하는 구성파일  
- 동작방식, 서블릿, 필터, 리스너등 설정

☞ @annotation 방법 (JavaEE 5 이후 부터 설정기반이 아닌 어노테이션으로 가능)

## Servlet 생명주기(Life Cycle) 관련 주요 메서드

1. **init()** → 최초 1회 초기화

2. **service()** → 요청이 올 때마다 실행

↳ **doGet(), doPost()** 등으로 분기

3. **destroy()** → 종료 직전 실행

## init() 메서드

### 📌 역할

- Servlet 객체가 생성될 때 **딱 한 번만 호출**된다.
- **초기화 작업**을 수행하는 데 사용된다.

### 🧠 예시

DB 연결, 설정 파일 읽기, 리소스 준비 등

### 🔍 특징

- 오버라이드할 때 @Override를 붙이고 throws ServletException 필요

- return 타입: void

### service() 메서드

#### 역할

- 클라이언트 요청이 들어올 때마다 호출된다.
- **요청 방식에 따라 내부적으로 doGet(), doPost() 등으로 분기 처리한다.**

#### 특징

- 대부분 개발자가 직접 작성하지 않아도 되고, HttpServlet이 알아서 호출해 준다.
- 필요하면 오버라이딩도 가능하지만 일반적으로 doGet()이나 doPost()를 사용한다.

### doGet() 메서드

#### 역할

- **HTTP GET 방식의 요청을 처리한다.**  
(예: 브라우저 주소창 입력, <a href>, <form method="get">)

#### 특징

- 요청 파라미터는 URL 뒤에 붙음 (?name=value)
- URL에 노출되므로 민감한 데이터는 지양

### doPost() 메서드

#### 역할

- **HTTP POST 방식의 요청을 처리한다.**  
(예: <form method="post">로 데이터 전송)

#### 특징

- 요청 데이터가 본문(Body)에 담겨서 전송되므로 보안에 더 유리
- 주로 로그인, 회원가입 등 폼 전송에 사용됨

### destroy() 메서드

#### 역할

- Servlet이 메모리에서 내려가기 직전에 호출된다.

- 자원 정리, 메모리 해제 등을 수행할 수 있다.

#### 특징

- 서버가 종료되거나 서블릿이 언로드될 때 호출
- init()과 마찬가지로 **딱 한 번만 호출**

#### 예시)Servlet 작성법 -> TestServlet.java

```
public class TestServlet extends HttpServlet{
    /기능 => 재정의해서 사용
}
```

#### web.xml문서에 설정기반

```
<!-- Servlet 등록
testServlet = new TestServlet();

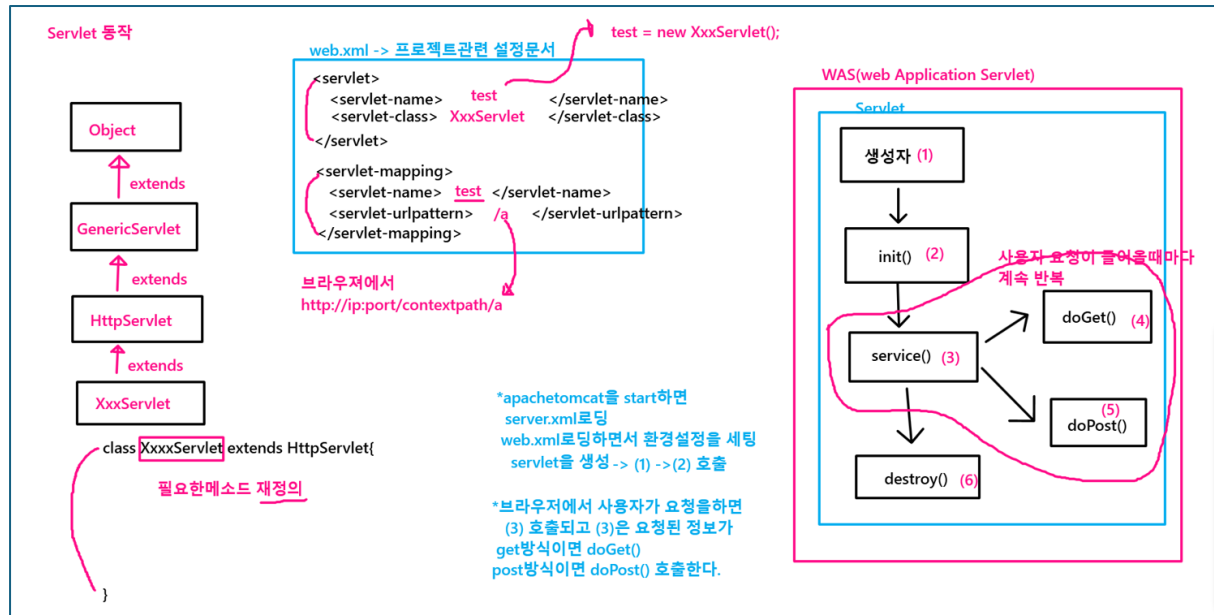
    load-on-startup옵션은 was가 start될때 미리 메모리에 생성한다
-->
<servlet>
<servlet-name>testServlet</servlet-name>
<servlet-class>kosta.servlet.TestServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
<servlet-name>testServlet</servlet-name>
<url-pattern>/test</url-pattern>
</servlet-mapping>
```

#### 브라우저 실행

```
http://ip:port/contextPath/urlpattern
```

## 동작방식



## Servlet관리

### 1. Servlet 관리 주체

- 과거(Java EE 시절) :** Java EE(Enterprise Edition)는 Oracle이 주도하고 JCP(Java Community Process)에서 관리.
- 현재(Jakarta EE 시절) :** 2017년 Oracle이 Java EE를 Eclipse Foundation에 이관하면서, 사양 이름도 **Jakarta EE로 변경**.
  - 현재 Jakarta Servlet을 포함한 모든 Jakarta EE 스펙은 Eclipse Foundation 산하의 Jakarta EE Working Group에서 관리됨

### 2. Tomcat, Servlet, Java 버전 정리 (2025년 9월 기준)

Tomcat 버전	지원 Servlet 버전	Jakarta EE 버전	최소 Java 버전
9.0.x	Servlet 4.0 (javax)	Java EE 8	Java 8 이상
10.0.x	Servlet 5.0 (jakarta)	Jakarta EE 9	Java 8 이상
10.1.x	Servlet 6.0 (jakarta)	Jakarta EE 10	Java 11 이상
11.0.x	Servlet 6.1 (jakarta)	Jakarta EE 11	Java 17 이상



<https://tomcat.apache.org/whichversion.html>

### 3. Java EE → Jakarta EE 전환 배경

#### 1. 상표권 문제

- "Java"와 javax.\* 네임스페이스는 Oracle의 상표.
- Eclipse Foundation이 자유롭게 유지·발전시키기 위해 jakarta.\* 네임스페이스로 변경 필요.
- Jakarta EE 9(2020)부터 본격적으로 javax.\* → jakarta.\*로 전환.

#### 2. 거버넌스 변화

- 과거: JCP(Java Community Process) 중심, Oracle이 강한 영향력.
- 현재: **Eclipse Foundation Specification Process (EFSP)** 기반.
  - 개방형, 벤더 독립적.
  - 모든 사양은 반드시 **\*\*TCK(호환성 테스트 키트)\*\***를 제공해야 함.
  - 릴리스 주기도 더 짧아지고 커뮤니티 중심 개발.

#### 3. 현재와 과거의 차이

- **패키지 변경**: javax.servlet.\* → **jakarta.servlet.\***
- **도구 지원**: Tomcat 등에서 제공하는 Migration Tool for Jakarta EE로 기존 코드를 자동 변환 가능.
- **발전 속도**: Java EE 시절보다 Jakarta EE에서 릴리스 속도가 빨라지고, 클라우드/마이크로서비스 친화적 기능 추가 중.
- **개방성**: 특정 기업(Oracle) 중심이 아닌, Eclipse Foundation 주도의 다수 벤더 참여 구조.

## 개발환경

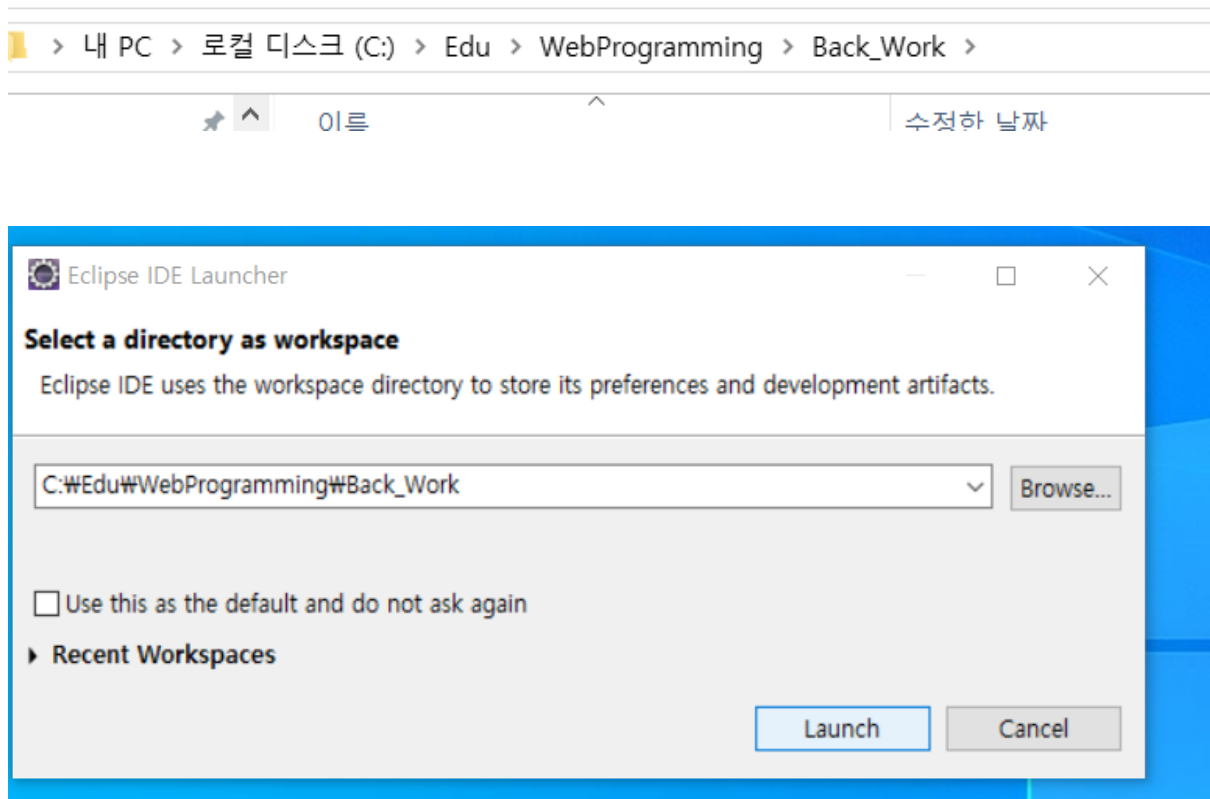
### 1) JDK설치 17

### 2) WAS - ServletContainer 역할

- **Apache tomcat 10.x**준비(웹서버 + WAS 기능을 모두 갖춘 경량 WAS)
- tomcat.apache.org 접속

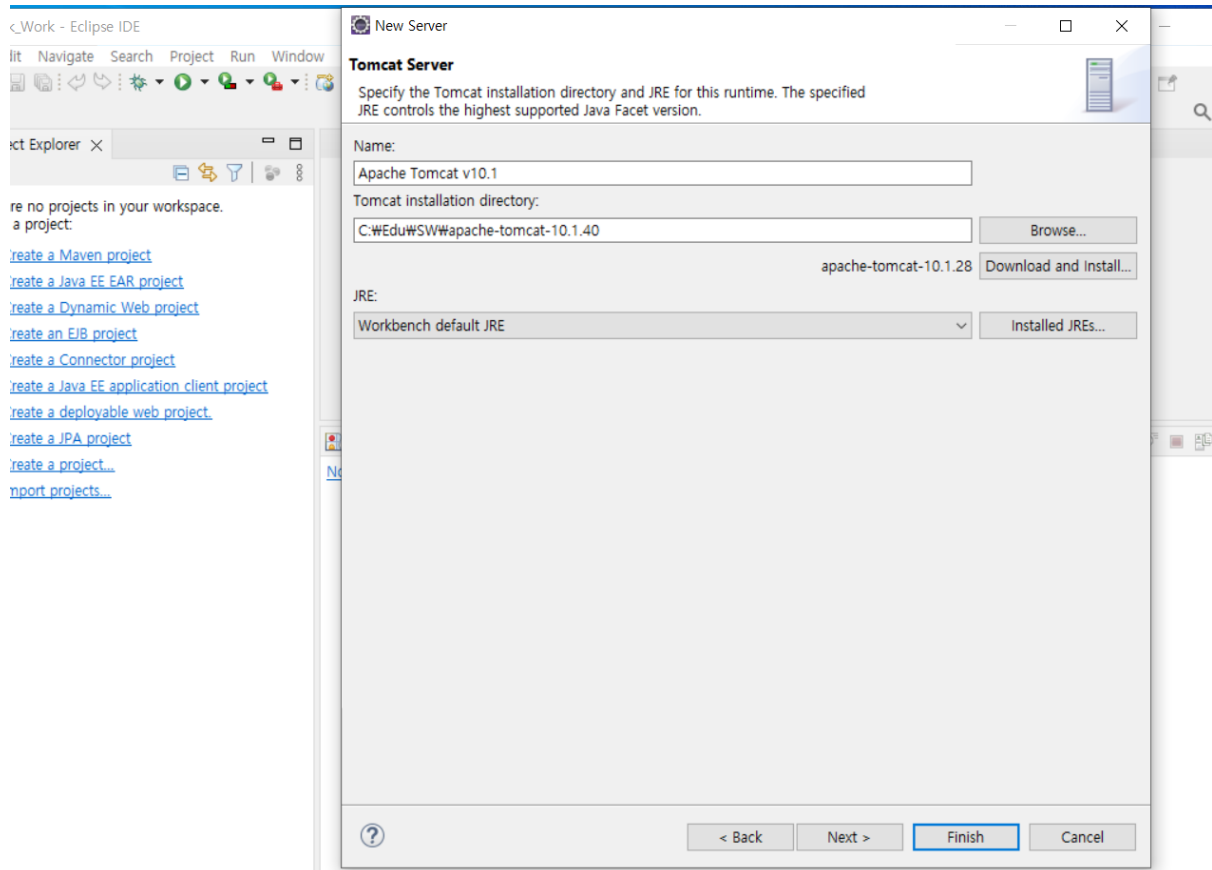
### 3) IDE – eclipse

: workspace 생성 한 후 이클립스를 open한다.




### 4) tomcat Server 연결하기


: server tab을 클릭해서 server등록한다.



## 5) Dynamic web Project 생성하기

 New Dynamic Web Project

**Dynamic Web Project**

Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application. 

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v10.1 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership


☐ Add project to an EAR

EAR project name:

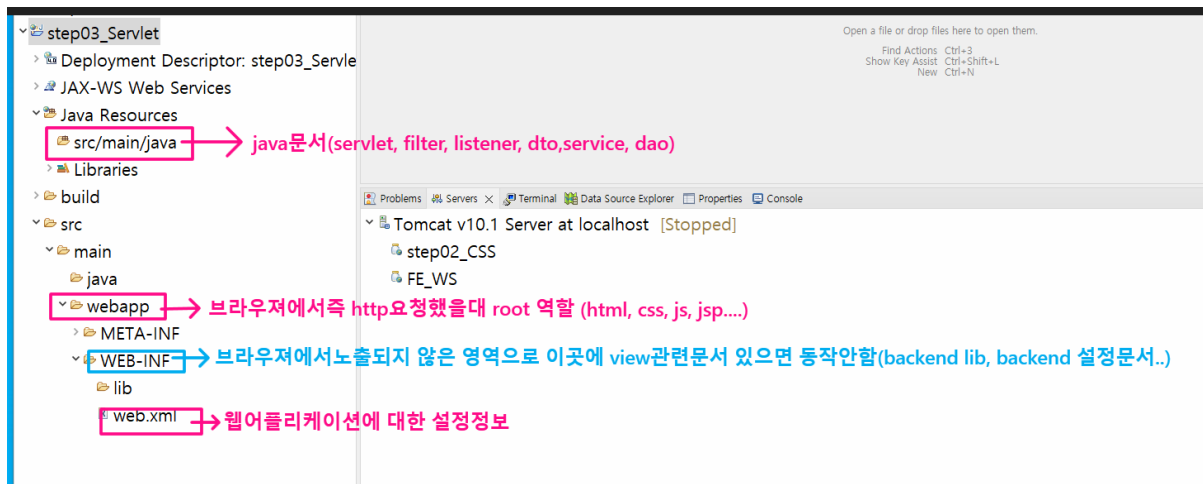
Working sets

☐ Add project to working sets

Working sets:



## 6) Project 생성 후 Directory 구조



## 응답코드 = status

200 : 성공

201 : create 성공

301 : 영구적인 리다이렉션(이전 페이지의 대한 정보등을 보관하지 않고 이동)

302 : 일시적으로 리다이렉션(이전 페이지에 대한 정보를 가지고 있을 수 있다)

404 : FileNotFoundException

405 : 요청방식을 틀렸을 때

400 : bad Request ( parameter로 전송되는 요청에 오류)

403 : 인증은 하였으나 권한부족

500 : 소스오류....

## 이동방식

### Forward vs Redirect 비교

구분	Forward (포워드)	Redirect (리다이렉트)
요청 방식	서버 내부에서 페이지 이동	클라이언트가 새로운 요청을 보내도록 유도
주소(URL)	변경되지 않음 (브라우저 주소 그대로 유지)	변경됨 (새로운 URL로 바뀜)
요청/응답	동일한 요청/응답 객체 사용 (한 사이클)	새로운 요청/응답 생성 (두 사이클)
속도	빠름 (서버 내부 이동)	느림 (클라이언트에 새 요청 지시)
사용 예	게시글 보기, 내부 JSP 이동 등	로그인 후 메인 페이지로 이동 등

### ◆ Forward 방식

#### ✓ 특징

- 서버 내에서 요청을 다른 리소스(JSP/Servlet 등)로 전달
- 클라이언트는 이동한 사실을 **모르며**, **브라우저 주소도 변하지 않음**
- 주로 데이터를 JSP에 넘길 때 사용
- **request, response가 유지된다.**

예시)회원가입 폼 → 처리 → 결과 화면

```
RequestDispatcher dispatcher = request.getRequestDispatcher("result.jsp");
dispatcher.forward(request, response); // 포워드 (데이터 전달 유지)
```

### ◆ Redirect 방식

#### ✓ 특징

- 클라이언트에게 응답으로 새로운 URL을 알려주고, 클라이언트가 다시 요청을 보냄
- **브라우저 주소가 변경됨**
- **새 요청이므로 request 속성 공유 안 됨**
- 주로 POST 이후 중복 방지(P-R-G 패턴 : Post - Redirect - Get)나 외부 페이지 이동 등에 사용

예시)로그인 처리 후 메인 페이지로 이동

```
// 로그인 성공 후
```

```
response.sendRedirect("main.jsp"); // 리다이렉트 (URL 변경)
```

### 핵심요약

포워드(Forward)	리다이렉트(Redirect)
서버 안에서 이동	브라우저가 다시 요청함
주소 그대로 유지	주소가 새로 바뀜
요청 정보 유지됨	요청 정보 유지 안 됨
빠르고 가볍다	느리지만 유용한 경우 존재

### PRG (Post → Redirect → Get) 패턴이란?

사용자가 **POST** 요청을 보낸 뒤, 서버가 **바로 응답 페이지를 보여주지 않고** 다시 **GET** 요청으로 리디렉션시키는 패턴이다.

### 📌 왜 PRG가 필요할까?

예를 들어, 사용자가 게시글을 작성하고 아래처럼 동작하면?

1. POST /write 요청 → 글 등록
2. 서버가 바로 writeSuccess.jsp를 보여줌
3. 그런데, 브라우저에서 새로고침(F5) 하면...?

❌ 같은 POST 요청이 다시 실행돼서 글이 중복으로 등록됨!

### ✅ PRG 패턴의 흐름

1. POST → 서버에서 데이터 처리

2. **Redirect** → `response.sendRedirect("success.jsp");`

3. **GET** → 클라이언트가 새 페이지를 GET으로 요청

새로고침해도 GET 요청만 반복되기 때문에, **중복 처리 방지!**

## ✅ 2. 멍등성(Idempotency)이란?

어떤 연산을 한 번 수행하든 여러 번 수행하든 결과가 같음을 의미한다.

### 📌 HTTP 메서드와 멍등성

메서드	멍등성 있음?	설명
GET	✅ 있음	같은 GET을 여러 번 해도 결과 안 바뀜
POST	❌ 없음	같은 POST를 여러 번 하면 DB에 중복 데이터 생길 수 있음
PUT	✅ 있음	같은 리소스를 업데이트 → 결과 동일
DELETE	✅ 있음	여러 번 삭제 요청해도 결국 리소스는 사라짐

## ✅ PRG 패턴과 멍등성의 관계

- POST는 멍등하지 **않음** → 새로고침(F5)하면 **중복 처리 문제** 발생
- 그래서 **PRG 패턴으로 POST 이후에 GET으로 바꿔주면**,  
사용자는 **안전하게 새로고침**할 수 있다.

## ✅ 정리



개념	설명
PRG 패턴	POST 요청 처리 후, redirect로 GET 요청을 유도해 중복 처리 방지
멱등성	같은 요청을 여러 번 해도 결과가 같은 성질 (GET, PUT 등)
관계	POST는 멱등하지 않기 때문에, PRG 패턴으로 안정성을 확보

## JSP란?

JSP (JavaServer Pages)는

HTML 코드 안에 Java 코드를 삽입해서 동적인 웹 페이지를 생성할 수 있도록 도와주는 서버 측 기술이다.

즉, 정적인 HTML 페이지 안에 동적인 기능(예: DB 데이터 출력, 조건문, 반복문 등)을 Java 코드로 넣을 수 있게 해주는 기술이다.

JSP는 HTML에 Java 코드를 섞어 동적 웹 페이지를 생성하는 기술이며, 실행 시 내부적으로 Servlet으로 변환되어 동작한다.

예시)

```
jsp

<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head><title>JSP 예제</title></head>
<body>
    <h1>현재 시간은?</h1>
    <%
        java.util.Date now = new java.util.Date();
        out.println(now);
    %>
</body>
</html>
```

- <% %> : JSP에서 **Java 코드**를 작성하는 영역
- out.println() : 브라우저에 출력
- HTML과 Java가 섞여 있음

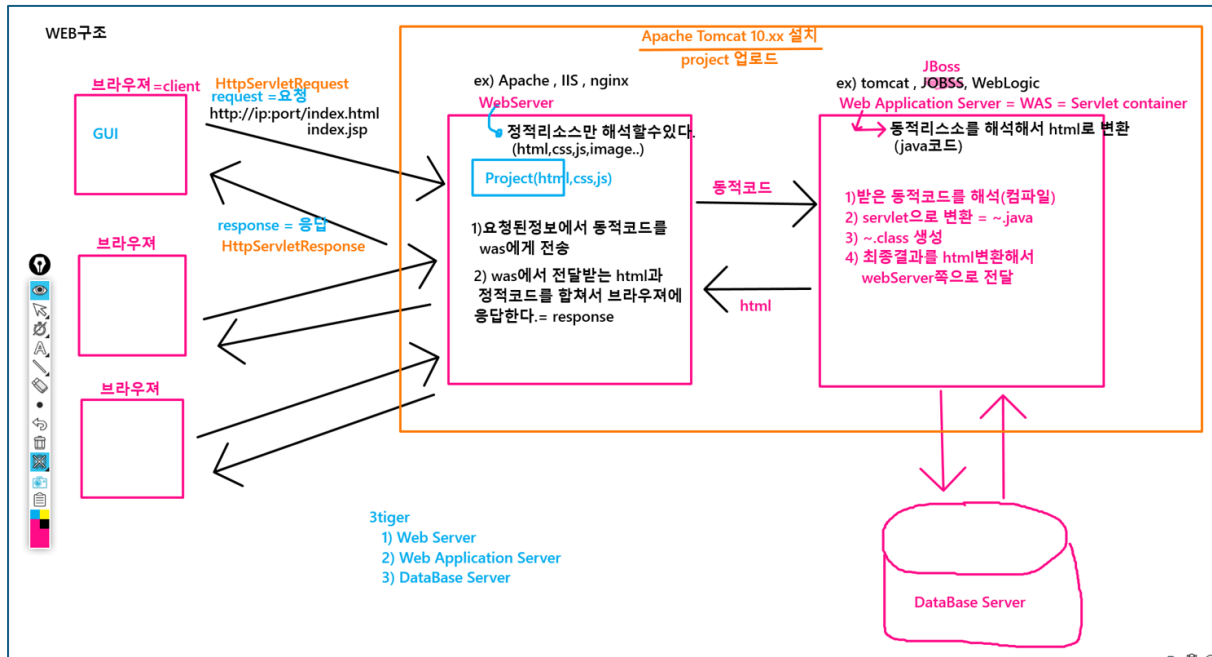
## JSP의 동작 원리

1. 클라이언트가 .jsp 파일을 요청
2. **JSP 엔진이 해당 파일을 .java Servlet 코드로 변환**
3. .java 파일을 컴파일하여 .class 파일 생성
4. 이 Servlet이 실행되어 HTML을 응답함

**결국 JSP는 Servlet으로 변환되어 실행되므로, 내부적으로는 Servlet 기술이다**

## **jsp문서의 Servlet문서 생성되는 위치**

C:\Edu\WebProgramming\Back\_Work\metadata\plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\step01\_Servlet\org\apache\jsp



## JSP의 주요 구성 요소

구분	문법	설명
스크립트릿	<% Java 코드 %>	자바 코드 작성 영역
표현식	<%= 값 %>	값을 바로 출력
선언부	<%! 선언문 %>	변수나 메서드 선언
지시어	<%@ page ... %>	JSP 속성 지정
액션 태그	<jsp:include>, <jsp:forward> 등	재사용, 이동 등 기능

## JSP의 장점

- HTML 기반이라 **웹 디자이너와 협업이 쉬움**
- **빠른 화면 구성 가능**
- 자바와 완전 호환
- 웹 컨테이너(Tomcat 등)에서 쉽게 실행

### ❌ JSP의 단점

- 로직과 뷰가 섞여 있어 유지보수가 어려움
- 대규모 프로젝트에선 MVC 구조로 분리 필요  
→ 그래서 **Servlet + JSP + JavaBeans** 또는 **Spring MVC**로 발전하게 됨

### ✅ JSP 내장객체란?

JSP 페이지에서 별도의 선언 없이 바로 사용할 수 있도록 제공되는 객체이다.

웹 애플리케이션에서 자주 사용하는 기능(요청, 응답, 출력, 세션 등)을 편리하게 쓸 수 있게 해준다.

### request - jakarta.servlet.http.HttpServletRequest

#### 📌 역할

- 클라이언트의 요청 정보를 담고 있는 객체
- 폼 데이터, 요청 방식, 파라미터 등을 얻을 수 있음

#### 🔍 주요 메서드

```
request.getParameter("id");           // 단일 파라미터 값
request.getParameterValues("hobby"); // 다중 선택 값 (배열)
request.getMethod();                  // GET or POST
request.getRequestURI();              // 요청 URI
```

```
String value = request.getParameter(String name);
- request 로 넘어오는 name 에 해당하는 value 값 받기
```

```
request.setCharacterEncoding("UTF-8");
- request 로 넘어오는 한글인코딩 변환
```

```
String str [] = request.getParameterValues("hobby");
- name 에 해당하는 value 여러개 일 때 사용함.
```

```
Enumeration<String> e = request.getParameterNames();
```

- request 로 넘어오는 name 에 대한 정보 가져오기

```
String ip = request.getRemoteAddr() ;
```

- 접속한 클라이언트 ip 가져오기

```
Cookie co [] = request.getCookies();
```

- 접속한 클라이언트에 저장된 쿠키정보(클라이언트정보) 가져오기

## response - jakarta.servlet.http.HttpServletResponse

### 역할

- 클라이언트(브라우저)에게 응답을 보낼 때 사용하는 객체
- 응답 타입 설정, 리다이렉트 등 가능

### 주요 메서드

```
response.setContentType("text/html;charset=UTF-8");
response.sendRedirect("main.jsp"); // 리다이렉트
```

```
response.sendRedirect(String url);
```

- 클라이언트의 요청페이지를 URL 로 이동시킴.

```
response.addCookie(Cookie co);
```

- 클라이언트쪽 PC 에 클라이언트의 정보를 저장함.

```
response.setContentType(String encoding);
```

- 클라이언트쪽에 한글인코딩 설정 = ex) "text/html;charset=UTF-8"

```
response.setStatus(int code) ;//
```

- 클라이언트쪽에 상태코드 설정

## session - jakarta.servlet.http.HttpSession

### 역할

- 사용자별로 정보를 저장하고 공유하는 객체

- 로그인 정보 유지, 장바구니 등 사용자 상태 관리에 사용됨
- 기본 세션의 시간은 30 분(1800 초)

#### 주요 메서드

```
session.setAttribute("user", "grace");
String name = (String)session.getAttribute("user");
session.invalidate(); // 세션 초기화
```

```
session.setAttribute(String name, Object value);
```

- 세션의 정보를 저장.

```
Object value = session.getAttribute(String name);
```

- 세션의 정보가져오기

```
session.setMaxInactiveInterval(int interval);
```

- 세션의 유지되는 시간설정(초단위)

```
int interval = session.getMaxInactiveInterval();
```

- 설정된 세션의 시간 가져옴(초단위)

```
String id = session.getId();
```

- 세션이 생성되면 자동으로 만들어지는 세션아이디

```
Enumeration e =session.getAttributeNames();
```

- 세션에저장된 name 가져오기

```
boolean b = session.isNew();
```

- 현재 브라우저창의 세션이 새로운 것 인지 판별  
(true 면 새로운 페이지, false 기존페이지)

```
session.invalidate();
```

- 세션의 모든정보를 지운다.

```
session.removeAttribute(java.lang.String name);
```

- 저장된 세션의 정보중 name 에 해당하는 정보 삭제

```
long time = session.getLastAccessedTime();
```

- 마지막 접속시간

```
long time = session.getCreationTime();
```

- 세션이 시작된 시간.

## application - `jakarta.servlet.ServletContext`

### 역할

- 웹 애플리케이션 전체에서 공유 가능한 객체
- 모든 사용자에게 공통적으로 적용되는 전역 데이터를 저장

### 주요 메서드

```
application.setAttribute("siteName", "KOSTA");
String name = (String)application.getAttribute("siteName");
```

```
application.setAttribute(String name, Object value);
```

- 정보를 저장하는 기능

```
Object value = application.getAttribute(String name);
```

- name 에 해당하는 정보를 가져오는 기능

```
application.removeAttribute(String name);
```

- name 에 해당하는 정보를 삭제하는 기능

```
application.getRealPath(java.lang.String path);
```

- 실행되는 문서의 경로 가져오는 기능

```
Enumeration e = application.getAttributeNames();
```

- 저장된 정보의 name 가져오는 기능

## out - `PrintWriter`

### 역할

- 브라우저에 출력하는 객체

- System.out.println()처럼 서버 콘솔이 아닌, 웹 브라우저 화면에 출력

### 🔍 주요 메서드

```
out.print("출력");
out.println("<h1>환영합니다!</h1>");
```

### exception

#### 📌 역할

- JSP 오류 페이지에서 예외 정보를 담은 객체
- 오류 처리용 JSP 페이지(isErrorPage="true")에서만 사용 가능

#### 💡 예시 (에러 페이지에서 사용 시)

```
<%@ page isErrorPage="true" %>
<%
    out.println("오류 메시지: " + exception.getMessage());
%>
```

### 요약하기

객체 이름	용도 / 설명
request	클라이언트 요청 정보 (폼 데이터 등)
response	서버 응답 처리 (출력, 리다이렉트 등)
session	사용자별 정보 저장 (로그인 상태 등)
application	웹 전체 공유 정보 저장 (사이트 공통 설정 등)
out	브라우저에 데이터 출력
exception	예외 처리용 객체 (isErrorPage="true" 필요)



## JSP와 Servlet에서 사용하는 정보 저장 범위(scope)

데이터를 얼마나 오래, 어디까지 사용할 수 있느냐?

Scope	유효 범위	생존 시간	대표 객체
pageContext	현재 JSP 페이지	페이지 실행 중	pageContext
request	하나의 요청	요청 ~ 응답까지	request
session	하나의 사용자 세션	세션 유지 동안	session
application	전체 웹 애플리케이션	서버 종료 전까지	application

pageContext < request < session < application

### 1. pageContext (페이지 범위)

현재 JSP 페이지 내에서만 유효  
다른 페이지로 넘어가면 사라짐.  
같은 요청이라도 include, forward 되면 공유되지 않는다.

항목	내용
유효 범위	현재 JSP 페이지
생존 시간	페이지 실행 중
사용 용도	잠깐 쓰고 말 데이터를 저장할 때

### 2. request (요청 범위)

하나의 HTTP 요청이 처리되는 동안 유지됨  
클라이언트가 요청하고, 응답을 돌려줄 때까지 살아 있음  
forward 나 include 할 때도 같은 요청이면 유지됨.

항목	내용
유효 범위	요청을 처리하는 동안
생존 시간	요청 시작 → 응답 직전까지
사용 용도	컨트롤러 → JSP로 데이터 전달 시

### 3. session (세션 범위)

클라이언트(브라우저) 하나와의 연결을 유지하는 동안 유효함.

로그인한 사용자마다 각각의 session 이 존재함.

서버가 JSESSIONID 라는 쿠키를 통해 클라이언트와 구분함.

항목	내용
유효 범위	같은 사용자(브라우저)와의 세션
생존 시간	브라우저 닫거나 세션 만료까지
사용 용도	로그인 정보, 장바구니 등

### 4. application (애플리케이션 범위)

서버에 배포된 웹 애플리케이션 전체에서 공유됨.

모든 사용자, 모든 요청이 접근 가능함.

서버가 꺼질 때까지 유지됨.

항목	내용
유효 범위	웹 애플리케이션 전체
생존 시간	서버 종료 or 재배포 시까지
사용 용도	전체 공통 데이터, 통계 등

scope 개념은 MVC 구조에서 데이터를 어디까지 전달할 건지 결정할 때도 중요

- Controller → View 로 값 전달할 땐 request
- 로그인 정보 저장할 땐 session

- 전체 방문자 수 추적할 땐 application

### 공통의 메소드 제공

#### 정보 저장

```
~.setAttribute(String name, Object obj);
```

#### 정보 조회

```
Object obj = ~.getAttribute(String name);
```

## 쿠키(Cookie)란? - javax.servlet.http.Cookie

클라이언트(브라우저)에 저장되는 작은 데이터 정보로 서버가 클라이언트에게 정보를 기억시키기 위해 사용한다.

### 쿠키의 기본 개념

- 서버 → 클라이언트에게 **응답할 때** 쿠키를 보냄
- 클라이언트는 **쿠키를 저장**
- 이후 클라이언트가 서버에 요청할 때, **해당 쿠키를 자동으로 함께 전송**

### Servlet에서 쿠키 사용법

```
// 쿠키 생성 (이름, 값)
Cookie cookie = new Cookie("userName", "grace");

// 쿠키 유효 시간 설정 (초 단위) - 1시간
cookie.setMaxAge(60 * 60);

// 클라이언트에 전송
response.addCookie(cookie);
```

### 쿠키 읽기 (클라이언트 → 서버로 보냄)

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for (Cookie c : cookies) {
        if (c.getName().equals("userName")) {
            String value = c.getValue();
            out.println("쿠키 값: " + value);
        }
    }
}
```

### 쿠키 삭제

```
Cookie cookie = new Cookie("userName", null);
cookie.setMaxAge(0); // 유효 시간을 0으로 설정
response.addCookie(cookie); // 클라이언트에 다시 전송해서 삭제
```

### 쿠키의 한계와 주의점

항목	설명
용량 제한	하나당 4KB 이하, 사이트당 20개 정도 제한
보안 위험	클라이언트에 저장되므로 민감한 정보 저장 ❌
위변조 가능성	사용자가 쿠키 값을 수정할 수도 있음 → 서버에서 검증 필요

## JSP에서 제공되는 액션태그

### <jsp:include>

다른 JSP 페이지의 출력 결과를 현재 페이지에 포함(include)시켜주는 태그로  
이건 마치 “여기 이 페이지 결과를 꺼 넣어줘~” 하는 느낌이다.  
헤더, 푸터, 네비게이션 바처럼 여러 페이지에서 반복되는 공통 부분을 불러올 때 사용한다.

```
<jsp:include page="다른페이지.jsp" />
```

#### 동작시점

요청(Request)이 들어왔을 때 실행됨 (runtime 포함)  
즉, 동적으로 다른 JSP 의 결과를 포함시킨다.

#### 파라미터 전달

```
<jsp:include page="sub.jsp">
    <jsp:param name="name" value="Grace"/>
</jsp:include>
```

### <jsp:forward>

현재 JSP 페이지의 처리를 멈추고 다른 페이지로 요청을 넘김(포워딩).  
즉, “이 페이지 그만! 저 페이지로 바톤 터치할게” 하는 느낌이다.

```
<jsp:forward page="다른페이지.jsp" />
```

#### 동작시점

실행 중에 만나면 즉시 현재 페이지 실행을 중단하고 해당 페이지로 이동  
forward 된 페이지가 최종 응답을 담당한다.

## 파라미터 전달

```
<jsp:forward page="result.jsp">
    <jsp:param name="name" value="Grace"/>
</jsp:forward>
```

## include vs forward 비교

항목	<jsp:include>	<jsp:forward>
역할	다른 JSP의 결과를 포함	다른 JSP로 완전히 이동
실행 시점	런타임(runtime)	런타임(runtime)
현재 페이지 처리	계속 진행함	중단됨
응답의 결과	현재 페이지 + 포함된 결과	오직 포워딩된 페이지 결과만 출력됨
주요 사용 용도	공통 레이아웃, 반복되는 출력 포함	흐름 분기 (조건 분기, 페이지 이동 등)

## DBCP(DataBase Connection Pool)란?

데이터베이스 연결(Connection)을 미리 만들어서 풀(pool)에 보관해두고, 필요할 때 빌려쓰는 방식이다. 연결이 필요할 때마다 새로 만들고 닫는 게 아니라 **재사용**해서 성능을 높여주는 기술이다.

## 왜 DBCP가 필요할까?

### 기존방식

```
Connection con = DriverManager.getConnection(...); // 매번 DB 연결
```

### 문제점

사용자가 많아지면 DB 연결/해제가 계속 반복됨  
DB 자원을 낭비하고, 응답 속도가 느려짐

**해결 방법 → DBCP**

미리 DB 연결을 여러 개 만들어 놓고, 필요할 때 하나씩 꺼내 쓰고 다시 반납  
효율적이고 빠름

**DBCP 구성요소**

구성 요소	설명
Connection Pool	DB 연결을 미리 만들어 저장해두는 공간
DataSource	Connection을 꺼내주는 객체 (JNDI로 사용)
Resource Ref	web.xml에서 리소스 이름 지정

<https://tomcat.apache.org/tomcat-10.1-doc/jndi-datasource-examples-howto.html>

**DBCP 연동 방법 (Tomcat 기준)**

**Step 1.** context.xml에 리소스 등록

(src/main/webapp/META-INF/context.xml 또는 Tomcat/conf/context.xml)

**Step 2.** web.xml에 ResourceRef 설정 – 생략가능

```
<resource-ref>
  <description>MySQL Connection</description>
  <res-ref-name>jdbc/myDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

**Step 3.** Servlet 또는 DAO에서 DBCP 연결 사용

```
Context initContext = new InitialContext();
Context envContext = (Context)initContext.lookup("java:comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/myDB");

Connection conn = ds.getConnection(); // 풀에서 하나 꺼내기
```

```
// 이후 사용
PreparedStatement ps = conn.prepareStatement("SELECT * FROM member");
ResultSet rs = ps.executeQuery();
...
rs.close();
ps.close();
conn.close(); // 실제로는 풀에 반납됨!
```

## 장점

장점	설명
성능 향상	매번 DB 연결/해제하지 않아 빠름
자원 절약	연결 재사용으로 DB 부하 감소
코드 단순화	JNDI로 쉽게 관리 가능
확장성 ↑	사용자 증가에도 안정적인 대응 가능

DBCP는 데이터베이스 연결을 효율적으로 관리하기 위한 커넥션 풀 기술이며, Tomcat의 context.xml + JNDI(DataSource) 방식으로 연동해서 사용한다.

## Expression Language (EL)이란?

JSP에서 데이터를 출력하거나 조건을 검사할 때 사용하는 간결한 표현 문법이다.

`${}` 문법을 사용해서 스크립틀릿 `<% %>` 없이도 값을 쉽게 출력할 수 있다.

### 왜 EL을 사용할까?

기존 방식 (스크립틀릿):

```
<%= request.getParameter("userName") %>
```

### EL방식

```
${param.userName}
```



훨씬 간단하고, 가독성도 좋아지고, HTML 코드와 잘 어울린다.

## EL 기본 문법

```
${user.name}      // 객체의 속성
${param.id}       // 요청 파라미터
${sessionScope.user} // 세션 속성 접근
```

- 표현언어는 \$시작한다.
- 모든 내용은 {표현식} 으로 구성된다.
- 표현식에는 기본적으로 변수명 혹은 속성명.메소드 구조로 이루어진다.
- 표현식에는 정수형, 실수형, 문자열형, 논리형, null 올수있다.
- 표현식 연산가능함.

## EL에서 사용할 수 있는 주요 객체들 (내장 객체)

객체 이름	설명
<code>param</code>	요청 파라미터 ( <code>request.getParameter()</code> )
<code>paramValues</code>	같은 이름의 여러 파라미터 (배열로)
<code>header</code>	요청 헤더 ( <code>request.getHeader()</code> )
<code>headerValues</code>	요청 헤더 여러 개 값
<code>cookie</code>	쿠키 ( <code>request.getCookies()</code> )
<code>initParam</code>	조기 파라미터 ( <code>web.xml</code> )
<code>pageScope</code>	pageContext 범위
<code>requestScope</code>	request 범위
<code>sessionScope</code>	session 범위
<code>applicationScope</code>	application 범위

**표현언어 연산자****- 산술연산자**

+ , - , \* , /(div) , %(mod)

ex) `${10 div 2}`

```
${10 + 2}    // 12
${price * qty}
```

**-비교 연산자**

==(eq) , !=(ne) , >(gt) , <(lt) , >=(ge) , <=(le)

ex) `${5 gt 2}`

```
${age > 18}    // true
${score == 100}
${name != 'admin'}
```

**- 조건연산자**

조건식 ? 참 : 거짓

```
${score >= 60 ? "합격" : "불합격"}
```

**- 논리연산자**

&&(and), ||(or) , !(not)

```
${param.id != null && param.id != ''}
${not empty userList}
```

₩

**- null 처리 및 empty 검사**

표현식	설명
<code>\${empty user}</code>	user가 null이거나 비어 있으면 true
<code>\${not empty list}</code>	list가 비어있지 않으면 true

**-객체의 속성 접근**

방법	설명
<code>\${user.name}</code>	<code>user.getName()</code> 값을 출력
<code>\${product.price}</code>	getter 메서드 자동 호출
<code>\${list[0].title}</code>	리스트/배열 요소 접근

### 표현언어 내장객체

`pageScope` => `page` 기본객체에 저장된 속성

`requestScope` => `request` 기본객체에 저장된 속성

`sessionScope` => `session` 기본객체에 저장된 속성

`applicationScope` => `application` 기본객체에 저장된 속성

### scope 영역 ( `setAttribute()` / `getAttribute()` )

`page` < `request` < `session` < `application`

`scope`를 명시하지 않으면, **`page` → `request` → `session` → `application`** 순으로 찾음

## JSTL(JSP Standard Tag Library)이란?

JSP에서 자주 사용하는 기능들을 태그로 제공하는 표준 태그 라이브러리이다.

반복문, 조건문, 포매팅, 국제화 등 자주 쓰는 로직을 **Java 코드 없이도** 태그 형태로 쓸 수 있게 해준다. 즉, JSP 안에 `<% %>` 스크립틀릿 없이도 **Java 기능을 HTML 처럼 태그로 표현할 수 있게** 만들어주는 도구이다.

EL(Expression Language)와 함께 쓰면 JSP가 정말 깔끔해지고 유지보수하기 쉬워진다.

### JSTL 주요 태그 종류와 기능

분류	선언 방법	주요 태그 기능
<b>Core</b> (핵심)	<code>c:</code>	조건문, 반복문, 변수, 포워드 등
<b>Format</b>	<code>fmt:</code>	날짜, 숫자, 통화 포매팅
<b>SQL</b>	<code>sql:</code>	데이터베이스 조회 및 처리 (실무에서는 거의 사용 ✕)
<b>XML</b>	<code>x:</code>	XML 데이터 처리
<b>Functions</b>	<code>fn:</code>	문자열 관련 유틸리티 함수

실무에서 가장 많이 쓰는 건 **core**와 **fmt**

### jsp문서에서 JSTL을 사용하기 위한 선언방법

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
```

### JSTL 사용 준비

JSTL 라이브러리 추가

- JSTL 은 별도의 JAR 파일이 필요하다. - <http://tomcat.apache.org/taglibs/> 다운로드
- 일반적으로 다음 두 파일을 WEB-INF/lib 에 추가한다.
  - jstl-1.2.jar
  - standard.jar

### JSTL 사용 선언 (taglib 지시어)

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

여기서 prefix="c"는 core 태그(c:로 시작하는 태그들)를 사용하겠다는 뜻이다.

## Core 태그 주요 예시

### 값을 안전하게 출력

HTML 태그나 특수문자가 포함된 데이터를 출력할 때 스크립팅 공격(XSS)을 방지하고

null 값 처리도 쉽게 해주는 장점

```
<c:out value="${userName}" />
```

```
<c:out value="${email}" default="이메일이 없습니다" />
```

HTML 태그 이스케이프 처리

```
<c:set var="text" value="<b>중요</b>" />
```

```
<p>메시지: <c:out value="${text}" /></p>
```

위 코드 출력 결과

```
메시지: &lt;b&gt;중요&lt;/b&gt;
```

변수 선언 (set)

```
<c:set var="name" value="Grace" />
```

조건문 (if)

```
<c:if test="${score >= 60}">
```

```
<p>합격입니다!</p>
```

```
</c:if>
```

조건 분기 (choose / when / otherwise)

```
<c:choose>
```

```
<c:when test="${score >= 90}">A</c:when>
```

```
<c:when test="${score >= 80}">B</c:when>
```

```
<c:otherwise>F</c:otherwise>
```

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

### 반복문 (forEach)

```
<c:forEach var="item" items="${productList}">
  <p>${item.name} - ${item.price}원</p>
</c:forEach>
```

### 페이지 이동 (redirect)

```
<c:redirect url="main.jsp" />
```

### Format 태그 주요 예시

#### 선언 추가

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

#### 날짜 포맷

```
<fmt:formatDate value="${now}" pattern="yyyy-MM-dd HH:mm:ss" />
```

