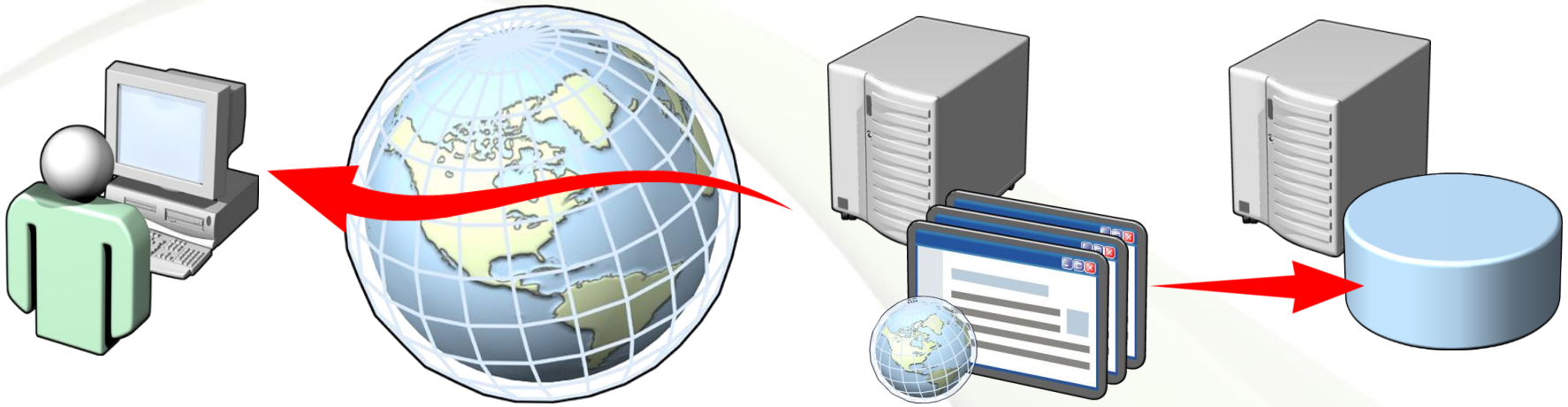
Introduction to Web Programming

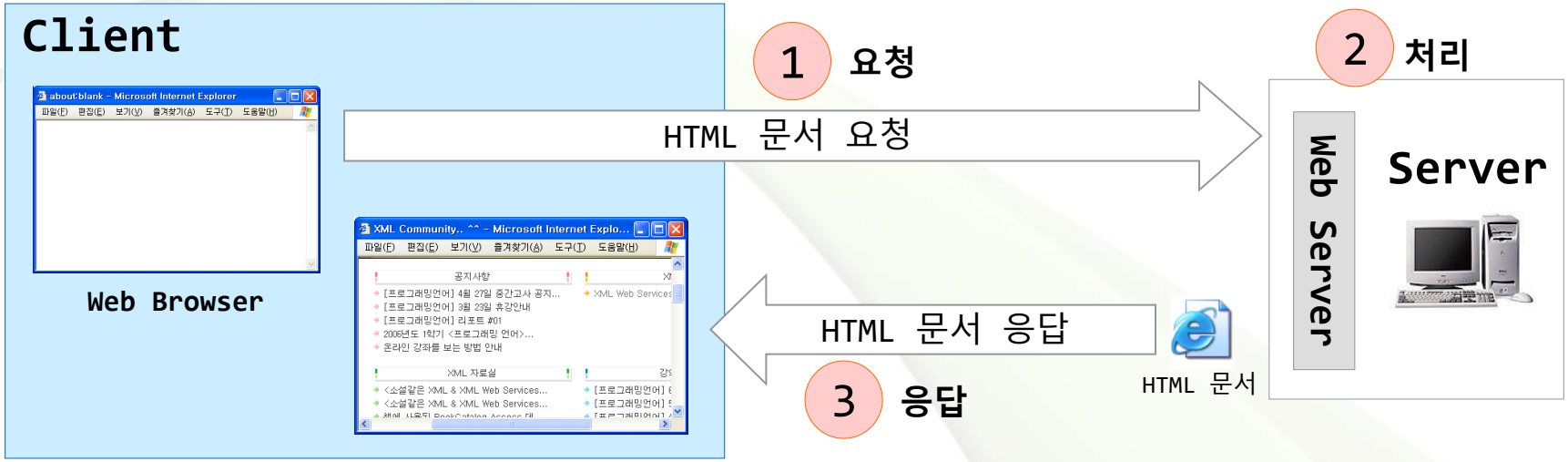
웹 애플리케이션 구조



Browser	Internet	Web Application	Data Server
HTML + CSS + Javascript	HTTP 기반 Request & Response	Static Resources (html, image…) Dynamic Resource (Servlet, JSP)	DBMS

웹 구성 요소

- 웹 애플리케이션 작동 구조



- 웹 서버(Web Server)
  - » 클라이언트의 요청을 받아서 처리한 후 결과를 클라이언트에 전송해 주는 주체

- 웹 클라이언트(Web Client)
  - » 필요한 데이터를 웹 서버에 요청하는 주체
  - » 일반적으로 웹 브라우저(Web Browser)

정적 웹응용프로그램 vs 동적 웹 응용프로그램

- 정적 웹 응용프로그램
  - » Html, image 등 미리 만들어진 자원으로 사용자의 요청에 대해 응답
  - » 미리 생성된 자원이 없으면 응답할 수 없으며
  - » 작은 차이라도 각각 다른 페이지로 구성해야 합니다.
  - » 웹 서버 수준에서 응답이 처리됩니다.

- 동적 웹 응용프로그램
  - » 요청이 들어오면 요청에 따라 동적으로 응답할 결과를 생성
  - » Web Server는 요청을 받고 적절한 웹 응용프로그램에 요청을 전달
  - » CGI 를 이용한 프로세스 기반 웹 응용프로그램 서비스 또는
  - » php, jsp, asp (→ asp.net) 등의 쓰레드 기반 동적 웹 응용프로그램 기술 사용

자바의 동적 웹 응용프로그램 기술

- 서블릿 (Servlet)
  - » 자바 기반 웹 개발 표준
  - » 서블릿 규약에 따라 만든 클래스
  - » 특정한 웹 요청을 처리하고 응답을 생성하기 위해 클래스를 만들고 등록
  - » 서블릿 컨테이너에 의해 관리
  - » 응답 결과물이 HTML인 경우 코드로 HTML을 제어하는 데 따른 비 효율성 노출

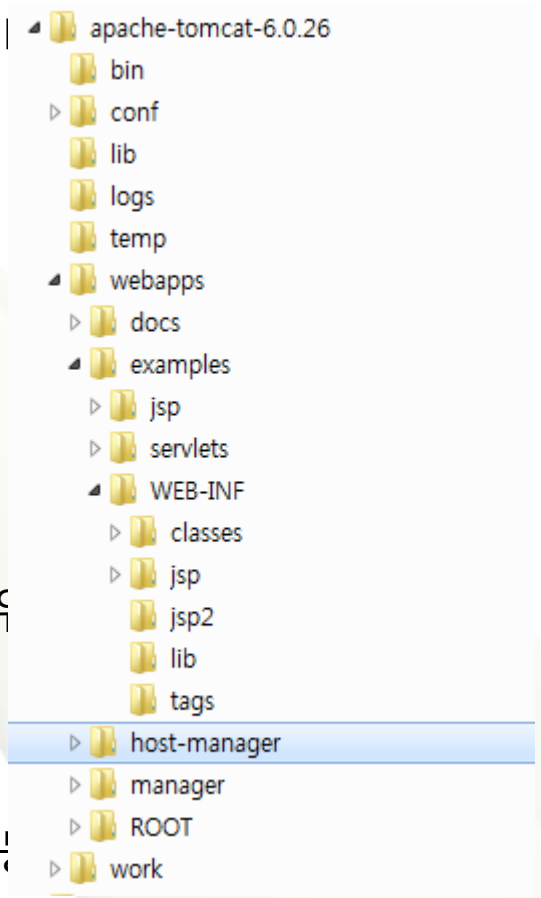
- JSP (Java Server Pages)
  - » 자바 언어를 기반으로 하는 스크립트 언어
  - » HTTP 프로토콜을 기반 웹 요청에 대해 HTML, XML 등의 결과를 응답하는데 최적화
  - » JSP 컨테이너에 의해 관리
  - » 페이지는 실행 시점에 서블릿 클래스로 변환 및 컴파일 되어 실행

컨테이너

- 서블릿 컨테이너
  - » 서블릿 클래스를 관리
  - » 요청이 발생하면 요청을 처리할 서블릿 객체를 생성하고 실행해서 결과 반환
- JSP 컨테이너
  - » JSP 페이지를 관리
  - » 요청이 발생하면 처리할 페이지를 파싱해서 클래스를 만들고 컴파일 및 실행
- 대부분의 컨테이너는 서블릿 컨테이너와 JSP 컨테이너의 기능을 모두 제공하기 때문에 일반적으로 웹 컨테이너로 통용

자바 웹 애플리케이션 구조

- 웹 애플리케이션
  - » 톰캣 설치 경로의 webapps 디렉터리 각 하위 디렉
- 웹 애플리케이션의 구성
  - » 웹 애플리케이션 디렉터리
    - › JSP(.jsp), HTML(.html) 등의 파일들이 위치
    - › 웹을 통해 공개되는 영역
  - » WEB-INF 디렉터리
    - › .class 파일, .jar 파일, web.xml 파일 등이 위
    - › .class 파일은 classes 디렉터리에 위치
    - › .jar 파일은 lib 디렉터리에 위치
    - › 웹을 통해 공개되지 않는 영역 (직접 접근 불가능
  - » web.xml
    - › 웹 애플리케이션의 환경설정 파일



ROOT 웹 애플리케이션과 기본 페이지

- ROOT 웹 애플리케이션
  - » 톰캣의 기본 애플리케이션으로 애플리케이션 이름을 지정하지 않고 접근
    - › http://localhost:8080/ 형식으로 접근
  - » 톰캣 설치 경로\webapps\ROOT 경로

- 커스텀 웹 애플리케이션
  - » 애플리케이션 이름을 지정해서 접근
    - › http://localhost:8080/appname/ 형식으로 접근
  - » 톰캣 설치 경로\webapps\appname 경로

- 기본 페이지
  - » 페이지 이름을 명시하지 않았을 경우 실행되는 페이지
  - » web.xml 파일에 등록

서블릿 작성

- HttpServlet 상속 클래스 정의
- doGet(HttpServletRequest, HttpServletResponse) 및 doPost(HttpServletRequest, HttpServletResponse) 재정의
- 컴파일 및 .class 파일을 WEB-INF\classes 디렉터리에 저장

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GreetingsServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        //Process Request
    }
    public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        //Process Request
    }
}
```

서블릿 등록

- web.xml 파일에 등록
- 특정 요청과 등록된 서블릿 클래스 바인딩

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>

    <servlet>
        <servlet-name>GreetingsServlet</servlet-name>
        <servlet-class>GreetingsServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>GreetingsServlet</servlet-name>
        <url-pattern>/GetGreetings</url-pattern>
    </servlet-mapping>

</web-app>
```

- 등록 후 http://localhost:8080/appname/GetGreetings 로 요청

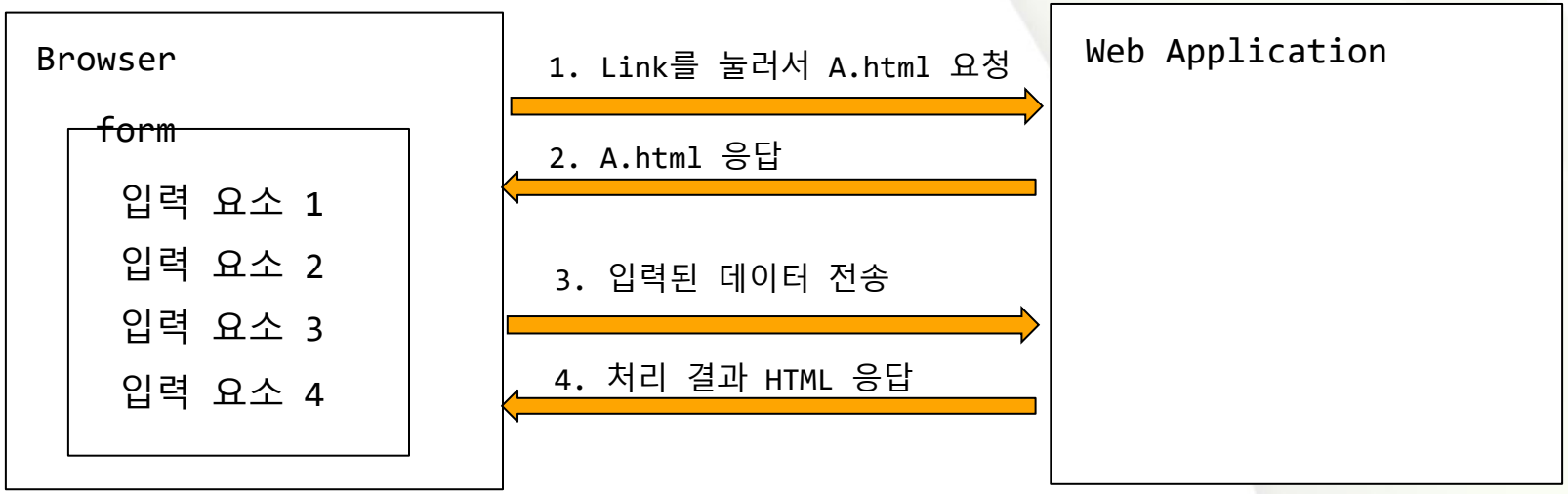
JSP 작성 및 실행

- 웹 애플리케이션 하위 경로에 .jsp 로 저장
- 요청시 파싱되어 클래스 파일로 변경된 후 컴파일되어 실행
- 컴파일 결과는 서블릿 클래스

```
<%@ page language="java"
         contentType="text/html; charset=utf-8"
         pageEncoding="EUC-KR"%>

<html>
<head>
         <meta http-equiv="Content-Type“  content="text/html; charset=utf-8">
         <title>Insert title here</title>
</head>
<body>
         <%= "Hello, JSP Programming World !!!!!" %>
</body>
</html>
```

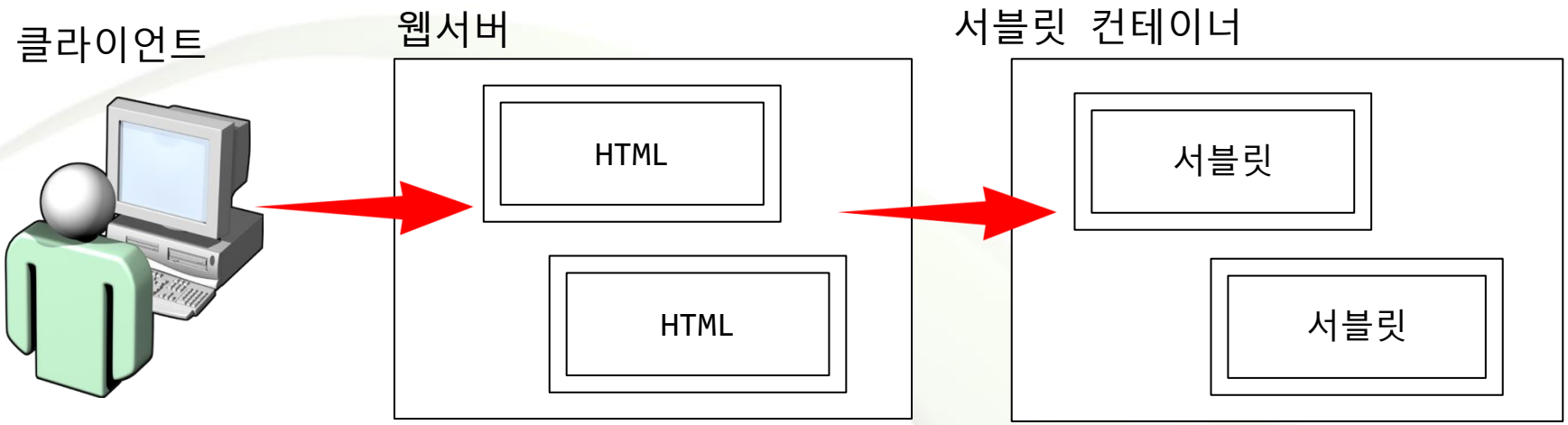
HTTP 요청 (Link와 Submit)

- Browser를 통해 두 가지 형태의 요청을 서버로 전달
  - » 주소 직접입력 또는 하이퍼링크를 클릭해서 특정 페이지 요청 (Link)
  - » 응답된 페이지에 사용자 데이터를 입력하고 서버로 전송 (Submit)
  - » Submit 할 데이터는 form 태그 단위로 입력되고 전송
  - » form 태그에 action 속성으로 전송 대상을 지정
  - » form 태그에 method 속성으로 데이터 전송방법을 지정
    - › post : 메시지 본문에 데이터를 기록하고 전송
    - › get : 주소 뒤에 ?이름=값&이름=값2의 형식으로 데이터 전송



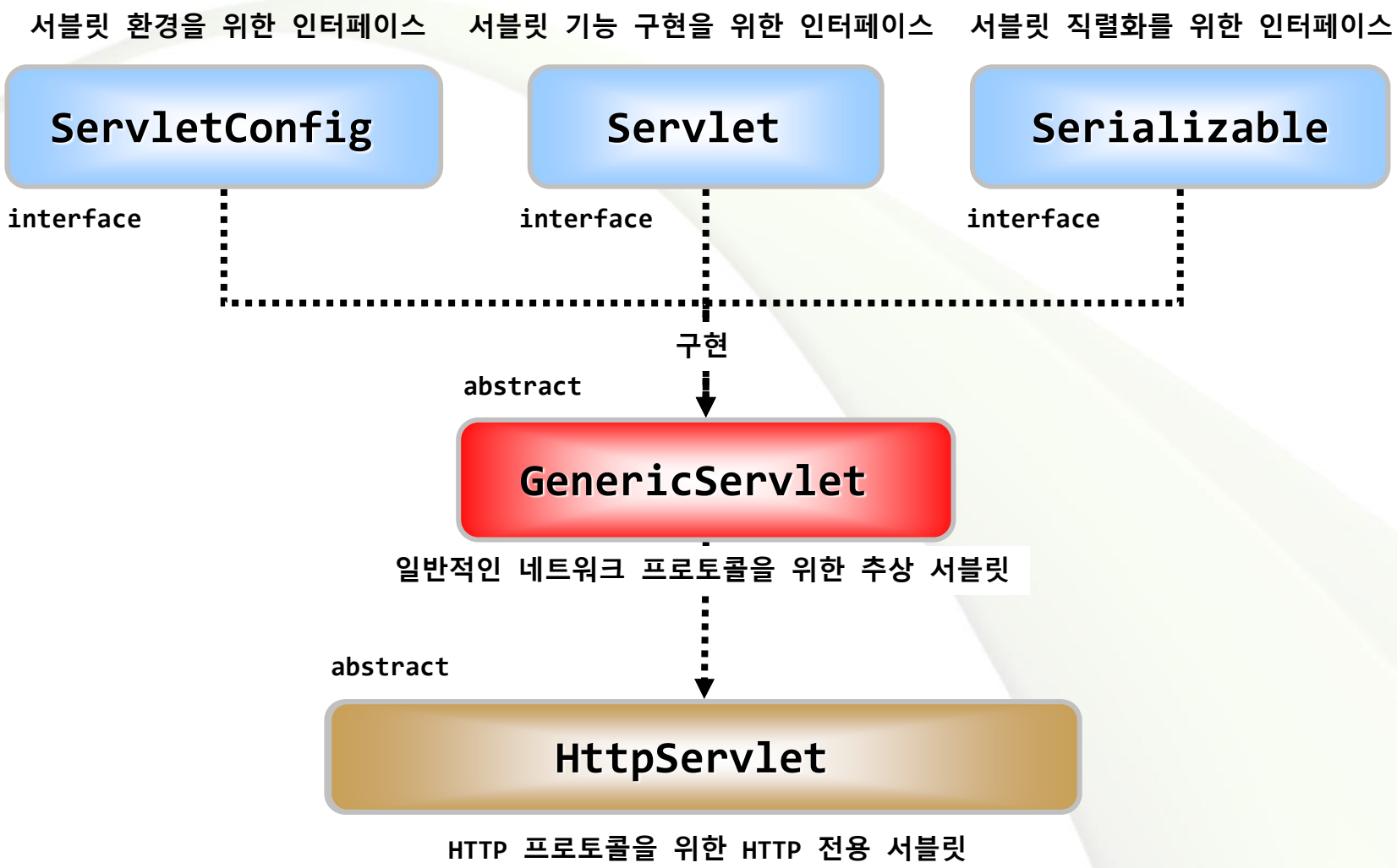
Servlet

서블릿 처리 구조 개요



- 일반적인 HTML 요청은 웹서버 수준에서 직접 HTML을 찾고 응답
- 서블릿 요청시 웹 서버는 요청 정보를 서블릿 컨테이너에게 전달
- 서블릿 컨테이너는 해당 서블릿을 찾아서 실행 및 처리 결과를 응답
  - » HttpServletRequest 및 HttpServletResponse 객체 생성
  - » 서블릿 클래스 로딩 및 객체 생성
  - » 요청에 따라 서블릿 클래스의 doGet 또는 doPost 메서드 호출

서블릿 클래스 상속 구조

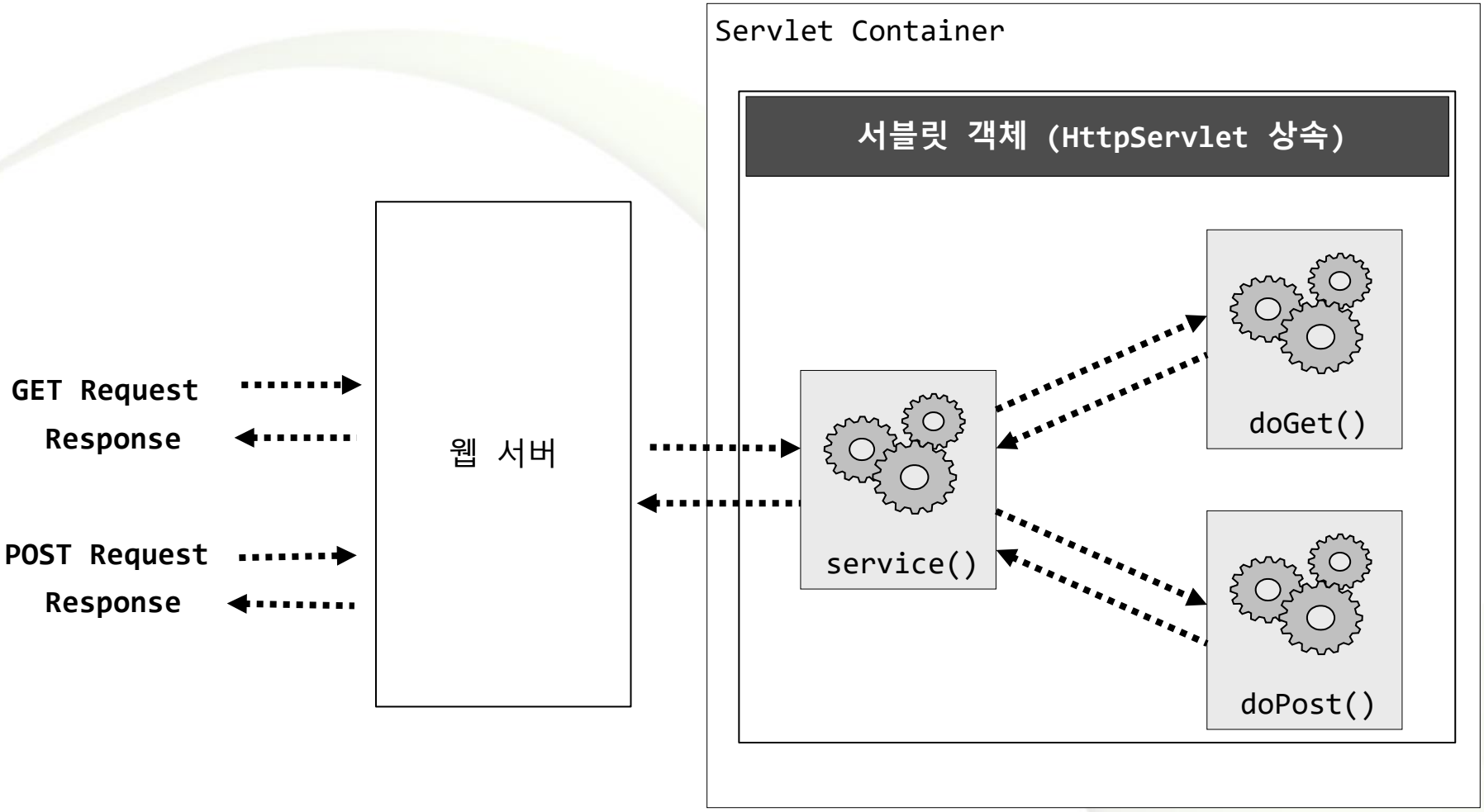

서블릿 환경을 위한 인터페이스
서블릿 기능 구현을 위한 인터페이스
서블릿 직렬화를 위한 인터페이스
ServletConfig
Servlet
Serializable
interface
interface
interface
구현
abstract
GenericServlet
일반적인 네트워크 프로토콜을 위한 추상 서블릿
abstract
HttpServlet
HTTP 프로토콜을 위한 HTTP 전용 서블릿

서블릿 클래스

- HttpServlet 클래스 상속
- doGet (GET 방식 요청시 호출) 및 doPost(POST 방식 요청시 호출) 메서드 재정의
- doGet / doPost 메서드 내부에 요청 처리 및 응답 코드 구현

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GreetingsServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        //Process Request
    }
    public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        //Process Request
    }
}
```

서블릿 요청 처리 메서드 호출

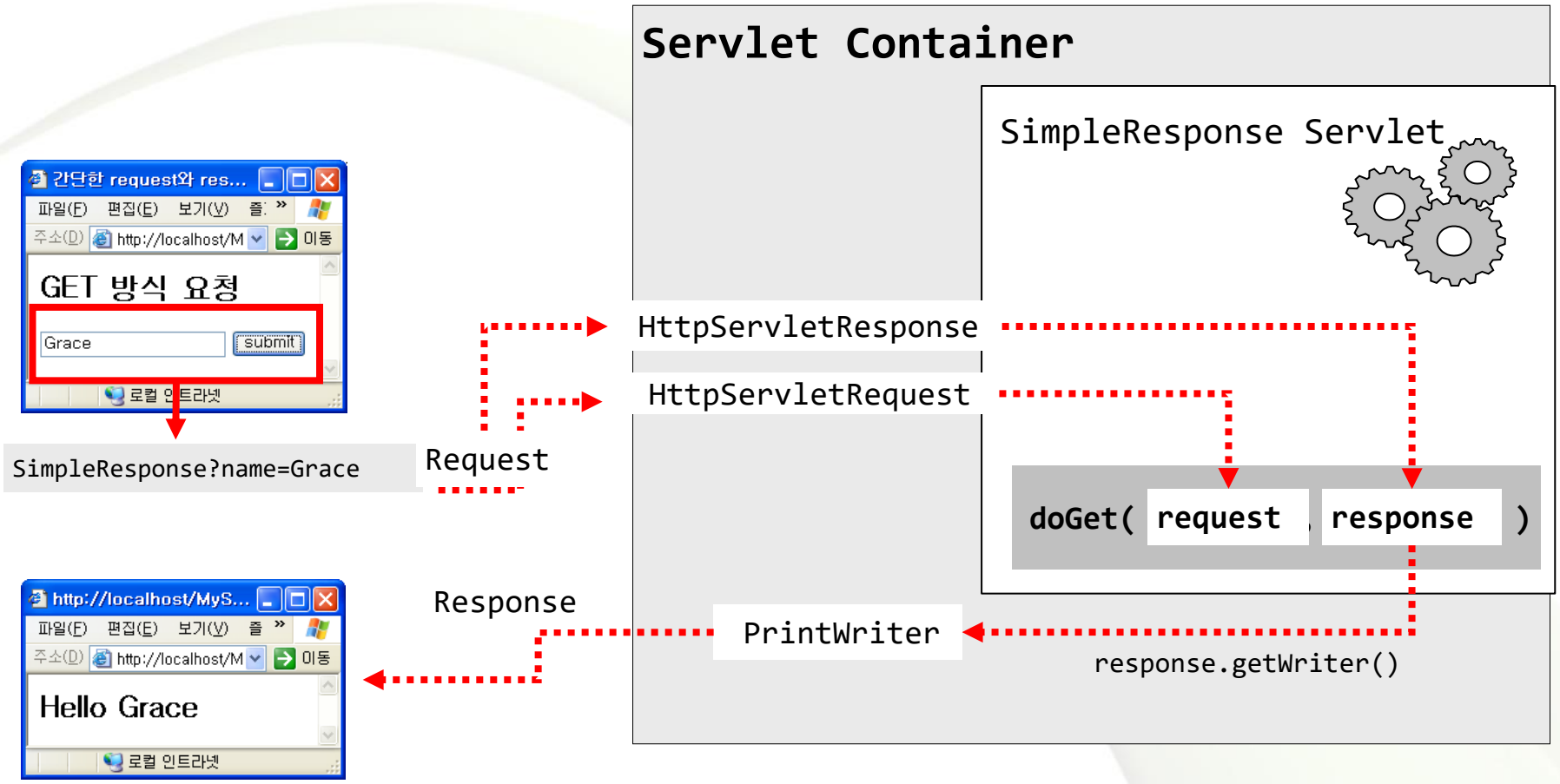


HttpServletRequest와 HttpServletResponse

- HttpServletRequest 객체는 클라이언트의 모든 요청 정보를 포함
  - » 요청 HTTP 헤더 정보
  - » 서블릿으로 전달된 파라미터 정보
  - » InputStream 형태의 클라이언트로부터 전송된 데이터
  - » 세션(Session)과 쿠키(Cookie)와 같은 기타 정보

- HttpServletResponse 객체는 클라이언트로 보내지는 응답 정보를 포함
  - » 응답 HTTP 헤더 정보
  - » OutputStream 형태의 클라이언트로 전송되는 데이터
  - » 세션과 쿠기 설정

HttpServletRequest와 HttpServletResponse 처리 과정



Servlet 한글 지원 문제

- response 객체의 한글 인코딩
  - » 응답으로 전송되는 response 객체의 한글 표현을 위해 한글 인코딩을 설정해 주어야 합니다.

```
response.setContentType("text/html;charset=utf-8");
```

- request 객체의 한글 인코딩
  - » 클라이언트로부터 POST 방식으로 전송된 요청 정보는 다음과 같이 한글 인코딩을 설정해 주어야 제대로 사용할 수 있습니다.

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

  - » 클라이언트로부터 GET 방식으로 전송된 요청 정보는 server.xml 파일에서 인코딩 설정

```
<Connector … URIEncoding=“euc-kr” … />
```

JSP (Java Server Pages) 개요

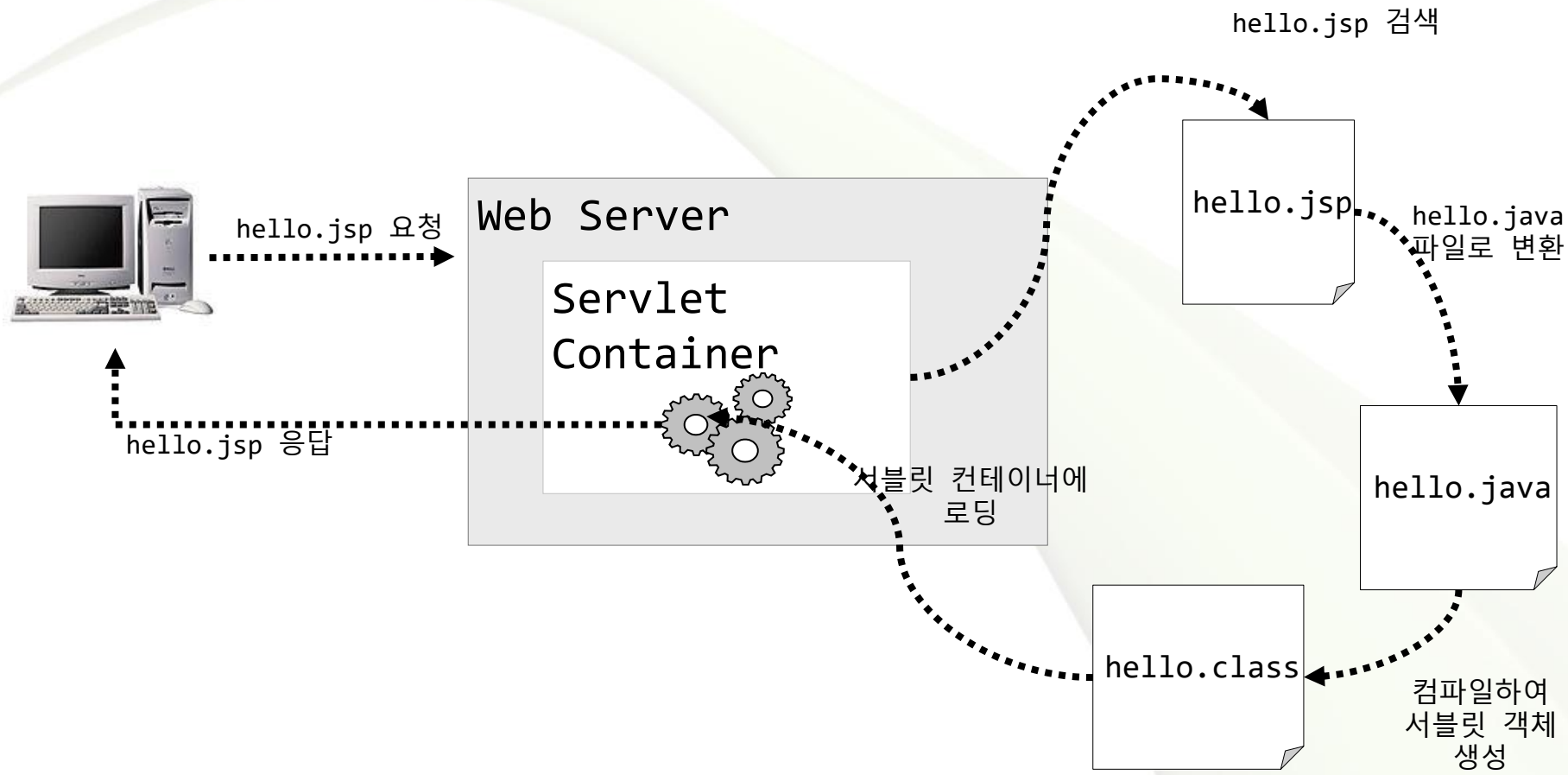
JSP?

- Servlet을 이용한 HTML 응답 처리는 코드로 HTML을 제어하는 구조로 코드와 디자인의 효율적인 협업이 어렵습니다.

- JSP는 HTML과 코드를 분리하고 HTML을 중심으로 코드를 구성해서 특히 HTML을 응답하는 데 효과적인 방법을 제공합니다.

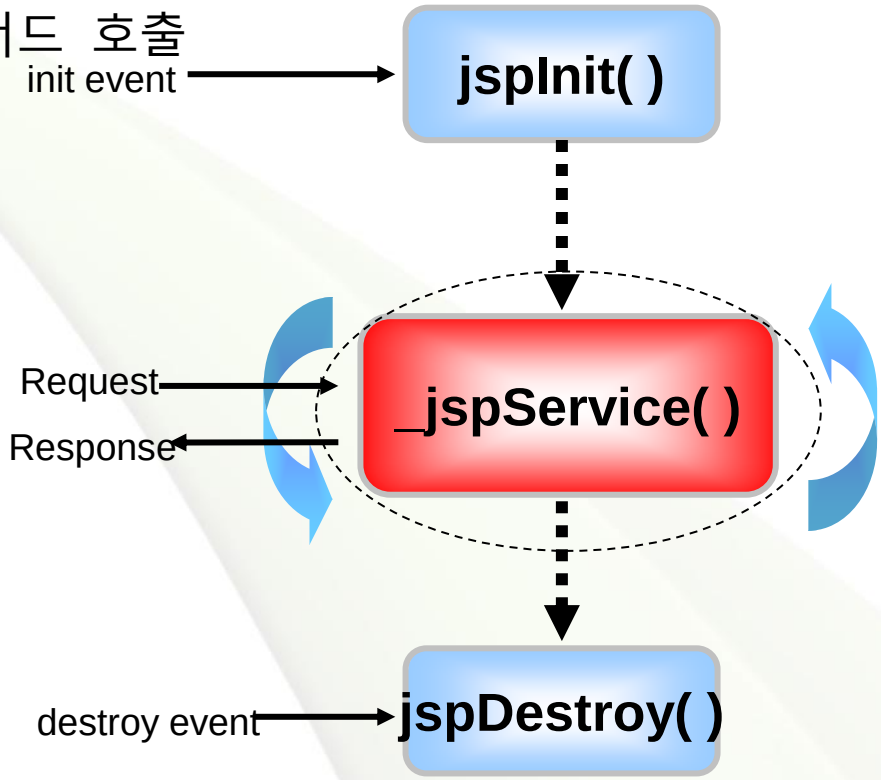
- 별도의 컴파일 과정을 직접 처리하지 않고 요청시에 Servlet 클래스를 동적으로 생성하고 컴파일 한 후 실행하는 과정을 수행합니다.

JSP 실행 구조


hello.jsp 검색
hello.jsp
hello.java
파일로 변환
hello.jsp 요청
Web Server
Servlet
Container
hello.jsp 응답
서블릿 컨테이너에
로딩
hello.java
hello.class
컴파일하여
서블릿 객체
생성

JSP 페이지 요청 라이프사이클

- jspInit()
  - » JSP에 대한 첫 요청이 올 경우 서블릿으로 변환된 후 jspInit() 메서드 호출
  - - 서블릿의 초기화 및 서비스를 시작하기 위한 준비
  - - _jspService() 메서드 호출
- _jspService()
  - » 클라이언트의 요청을 처리하는 메서드
- jspDestroy()
  - » 서블릿 객체가 메모리에서 제거될 때 호출되는 메서드



페이지 구성 요소

- 지시문 (Directive)
  - » JSP 페이지가 처리될 때 필요한 정보를 표시
  - » 형식 : <%@ 지시문 이름=“값” 이름2=“값2” … %>
  - » 종류 : page 지시문, include 지시문 등

```
<%@ page contentType="text/html; charset=utf-8" %>

<%@ page import="java.util.*" %>

<%@ include file="포함될 파일 이름" %>
```

페이지 구성 요소

- 주석
  - » HTML 주석
    - › 서버에서 처리되지만 브라우저에서 표시되지 않는 내용 포함
    - › 형식 : <!-- 주석 내용 -->
  - » 서버 주석
    - › 서버에서 처리되지 않는 내용으로 응답에 포함되지 않습니다.
    - › 형식 : <%-- 주석 내용 --%>

페이지 구성 요소

- HTML 및 정적 텍스트
  - » 별도의 처리 과정 없이 있는 그대로 HTML 응답에 포함되는 내용

- 실행문
  - » 스크립트릿
    - › 일반 실행문 포함
    - › 형식 : <% 실행문 %>
  - » 표현식
    - › 값으로 출력할 내용을 포함
    - › 형식 : <%= 값 또는 값을 반환하는 코드 %>
  - » 선언문
    - › 변수 또는 메서드 선언 영역
    - › 형식 : <%! 변수선언 또는 메서드 선언 %>

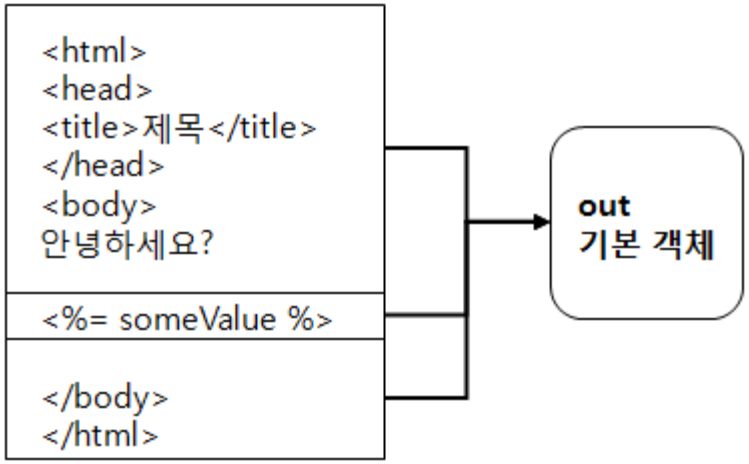
JSP 내장 객체

주요 기본 객체

기본 객체	실제 타입	설명
request	javax.servlet.http.HttpServletRequest 또는 javax.servlet.ServletRequest	클라이언트의 요청 정보를 저장한다.
response	javax.servlet.http.HttpServletResponse 또는 javax.servlet.ServletResponse	응답 정보를 저장한다.
pageContext	javax.servlet.jsp.PageContext	JSP 페이지에 대한 정보를 저장한다.
session	javax.servlet.http.HttpSession	HTTP 세션 정보를 저장한다.
application	javax.servlet.ServletContext	웹 어플리케이션에 대한 정보를 저장한다.
out	javax.servlet.jsp.JspWriter	JSP 페이지가 생성하는 결과를 출력할 때 사용되는 출력 스트림이다.
config	javax.servlet.ServletConfig	JSP 페이지에 대한 설정 정보를 저장한다.
page	java.lang.Object	JSP 페이지를 구현한 자바 클래스 인스턴스이다.
exception	java.lang.Throwable	예외 객체. 에러 페이지에서만 사용된다.

out 기본 객체

- JSP 페이지가 생성하는 모든 내용은 out 기본 객체를 통해서 전송



- 복잡한 if-else 사용시 out 기본 객체 사용하면 편리

```
<%
    if (grade > 10) {
        out.println(gradeStringA);
    } else if (grade > 5) {
        out.println(gradeStringB);
    }
%>
```

out 기본 객체 주요 메서드

- 출력 메서드
  - » print() - 데이터를 출력한다.
  - » println() - 데이터를 출력하고, \r\n(또는 \n)을 출력한다.
  - » newLine() - \r\n(또는 \n)을 출력한다.

- 버퍼 관련 메서드
  - » int getBufferSize() - 버퍼의 크기를 구한다.
  - » int getRemaining() - 현재 버퍼의 남은 크기를 구한다.
  - » clear() - 버퍼의 내용을 비운다. 만약 버퍼가 이미 플러시 되었다면 IOException을 발생시킨다.
  - » clearBuffer() - 버퍼의 내용을 비운다.
  - » flush() - 버퍼를 플러시 한다.
  - » boolean isAutoFlush() - 버퍼가 다 찼을 때 자동으로 플러시 할 경우 true를 리턴한다.

pageContext 기본 객체

- 다른 기본 객체에 대한 접근 메서드 제공

메서드	리턴타입	설명
getRequest()	ServletRequest	request 기본 객체를 구한다.
getResponse()	ServletResponse	response 기본 객체를 구한다.
getSession()	HttpSession	session 기본 객체를 구한다.
getServletContext()	ServletContext	application 기본 객체를 구한다.
getServletConfig()	ServletConfig	config 기본 객체를 구한다.
getOut()	JspWriter	out 기본 객체를 구한다.
getException()	Exception	exception 기본 객체를 구한다.
getPage()	Object	page 기본 객체를 구한다.

application 기본 객체: 초기화 파라미터

- 초기화 파라미터 설정: web.xml 파일 이용

```
<context-param>
    <description>파라미터 설명(필수 아님)</description>
    <param-name>파라미터이름</param-name>
    <param-value>파라미터값</param-value>
</context-param>
```

- application 기본 객체의 초기화 파라미터 관련 기능

메서드	리턴타입	설명
getInitParameter(String name)	String	이름이 name인 웹 어플리케이션 초기화 파라미터의 값을 읽어온다. 존재하지 않을 경우 null을 리턴한다.
getInitParameterNames()	Enumeration	웹 어플리케이션 초기화 파라미터의 이름 목록을 리턴한다.

application 기본 객체: 자원 구하기

- 자원 접근 메서드

메서드	리턴타입	설명
getRealPath(String path)	String	웹 어플리케이션 내에서 지정한 경로에 해당하는 자원의 시스템상에서의 자원 경로를 리턴한다.
getResource(String path)	URL	웹 어플리케이션 내에서 지정한 경로에 해당하는 자원에 접근할 수 있는 URL 객체를 리턴한다.
getResourceAsStream(String path)	InputStream	웹 어플리케이션 내에서 지정한 경로에 해당하는 자원으로부터 데이터를 읽어올 수 있는 InputStream을 리턴한다.

속성(Attribute)

- 속성 기능 제공 기본 객체
  - » pageContext, request, session, application

- 속성 관련 메서드

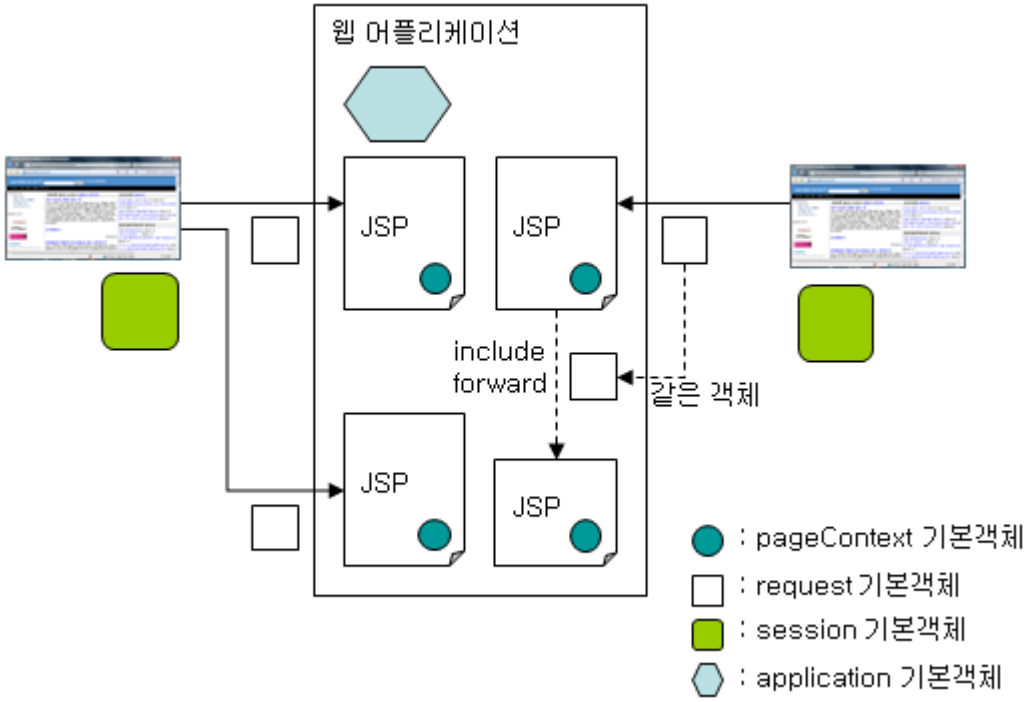
메서드	설명
void setAttribute(String name, Object value)	이름이 name인 속성의 값을 value로 지정한다.
Object getAttribute(String name)	이름이 name인 속성의 값을 구한다. 지정한 이름의 속성이 존재하지 않을 경우 null을 반환
void removeAttribute(String name)	이름이 name인 속성을 삭제한다.
Enumeration getAttributeNames()	속성의 이름 목록을 구한다. (pageContext 기본 객체는 지원하지 않음)

속성의 활용

기본 객체	영역	쓰임새
pageContext	PAGE	(한번의 요청을 처리하는) 하나의 JSP 페이지 내에서 공유될 값을 저장한다.
request	REQUEST	한번의 요청을 처리하는 데 사용되는 모든 JSP 페이지에서 공유될 값을 저장한다.
session	SESSION	한 사용자와 관련된 정보를 JSP 들이 공유하기 위해서 사용된다.
application	APPLICATION	모든 사용자와 관련해서 공유할 정보를 저장한다.

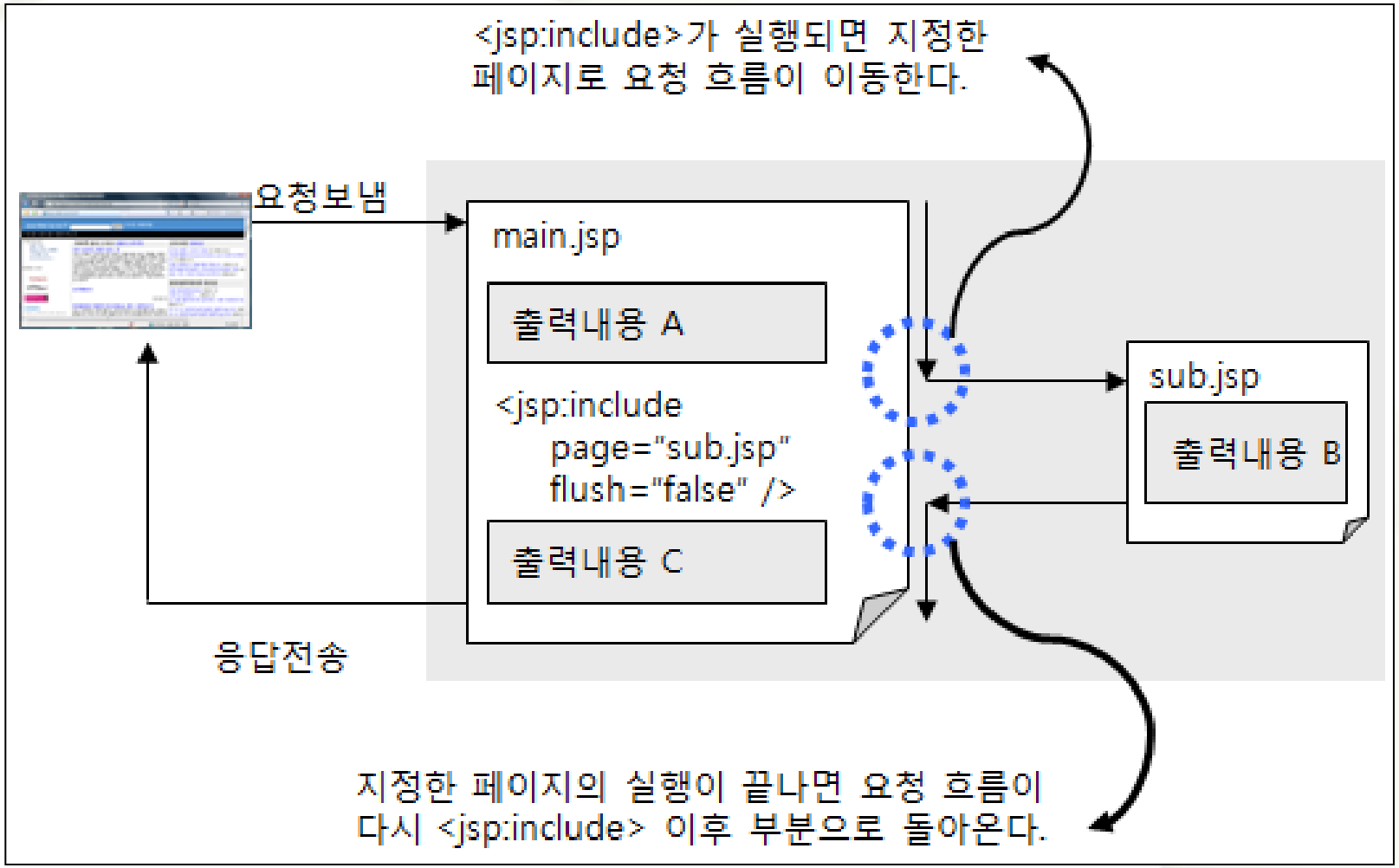
기본 객체와 영역

- 네 영역
  - » PAGE 영역 - 하나의 JSP 페이지를 처리할 때 사용되는 영역
  - » REQUEST 영역 - 하나의 HTTP 요청을 처리할 때 사용되는 영역
  - » SESSION 영역 - 하나의 웹 브라우저와 관련된 영역
  - » APPLICATION 영역 - 하나의 웹 어플리케이션과 관련된 영역



<jsp:include> 액션 태그

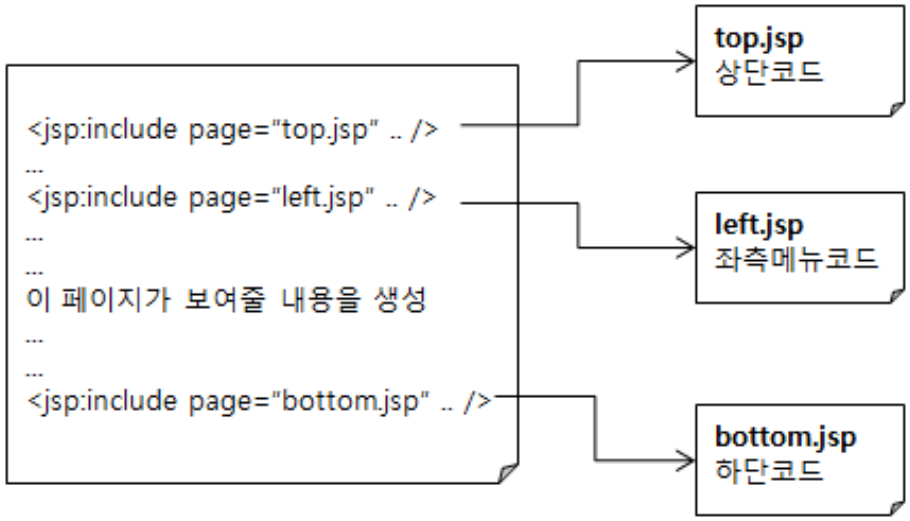
- 다른 JSP 페이지의 실행 결과를 현재 위치에 삽입
- 동작 방식



<jsp:include> 액션 태그

- 구문
  - » <jsp:include page="포함할페이지" flush="true" />
  - » page 속성 : 포함할 JSP 페이지
  - » flush 속성 : 지정한 JSP 페이지를 실행하기 전에 출력 버퍼를 플러시 할지의 여부를 지정한다. true이면 출력 버퍼를 플러시하고, false이면 하지 않는다.

- 중복 영역을 모듈화 하는 데 유용



<jsp:param> 액션 태그

- 신규 파라미터를 추가하는 데 사용
  - » <jsp:param name="파라미터이름" value="값" />

- 사용 사례

```
<jsp:include page="/module/top.jsp" flush="false">
    <jsp:param name="param1" value="value1" />
    <jsp:param name="param2" value="value2" />
</jsp:include>
```

<jsp:param> 액션 태그의 동작 방식

- 기존 파라미터는 유지하고 파라미터를 새로 추가
  - » <jsp:include>로 포함되는 페이지에서만 유효

```
[ request url ]
http://localhost:8080/examples-web/main.jsp?name=홍길동
```

```
main.jsp

<jsp:include page="sub.jsp">
  <jsp:param name="email"
             value="hkd@example.com" />
</jsp:include>
```

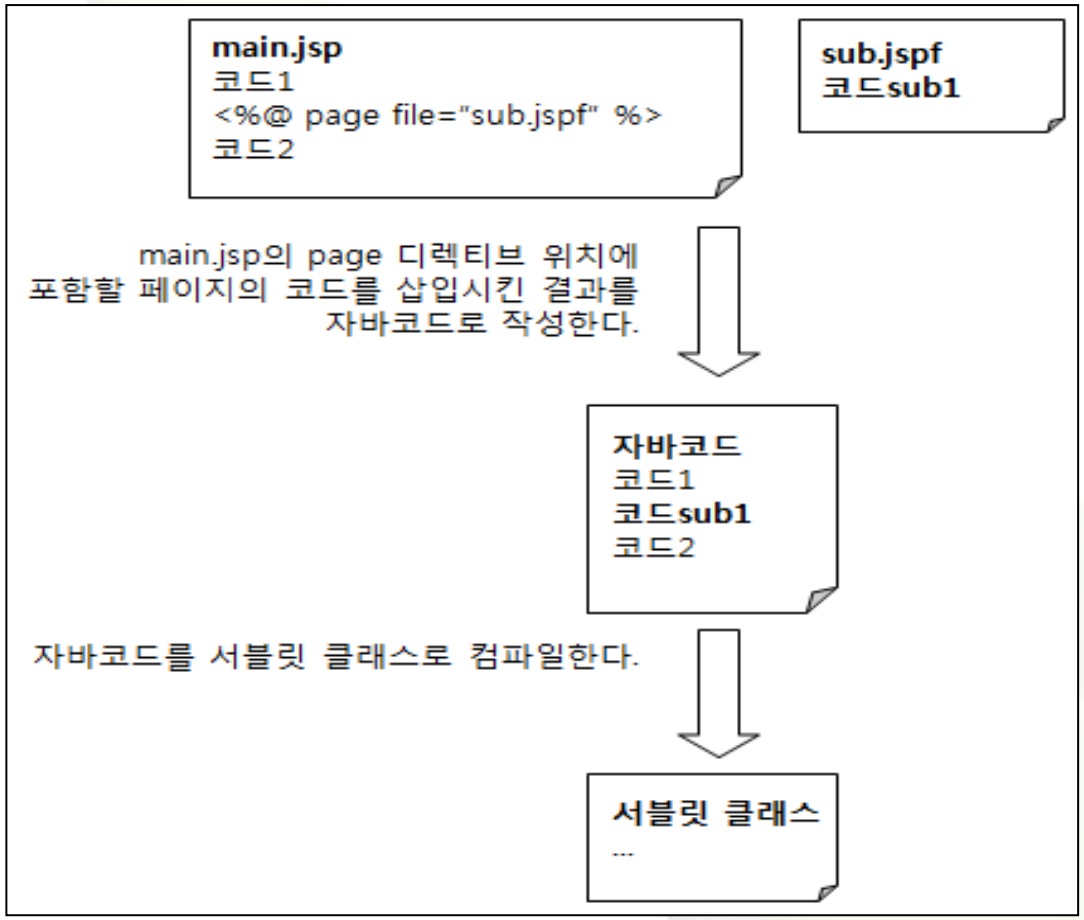
```
sub.jsp

String name =
  request.getParameter("name");
String email =
  request.getParameter("email");
```

```
[ main.jsp에서 사용되는 파라미터 ]

  name = "홍길동"
```

```
[ sub.jsp에서 사용되는 파라미터 ]

  name = "홍길동"
  email = "hkd@example.com"
```

include 디렉티브

- 코드 차원에서 포함
- 구문 : <%@ include file="포함할파일" %>
- 활용
  - » 모든 JSP 페이지에서 사용되는 변수 지정
  - » 저작권 표시와 같은 간단하면서도 모든 페이지에서 중복되는 문장

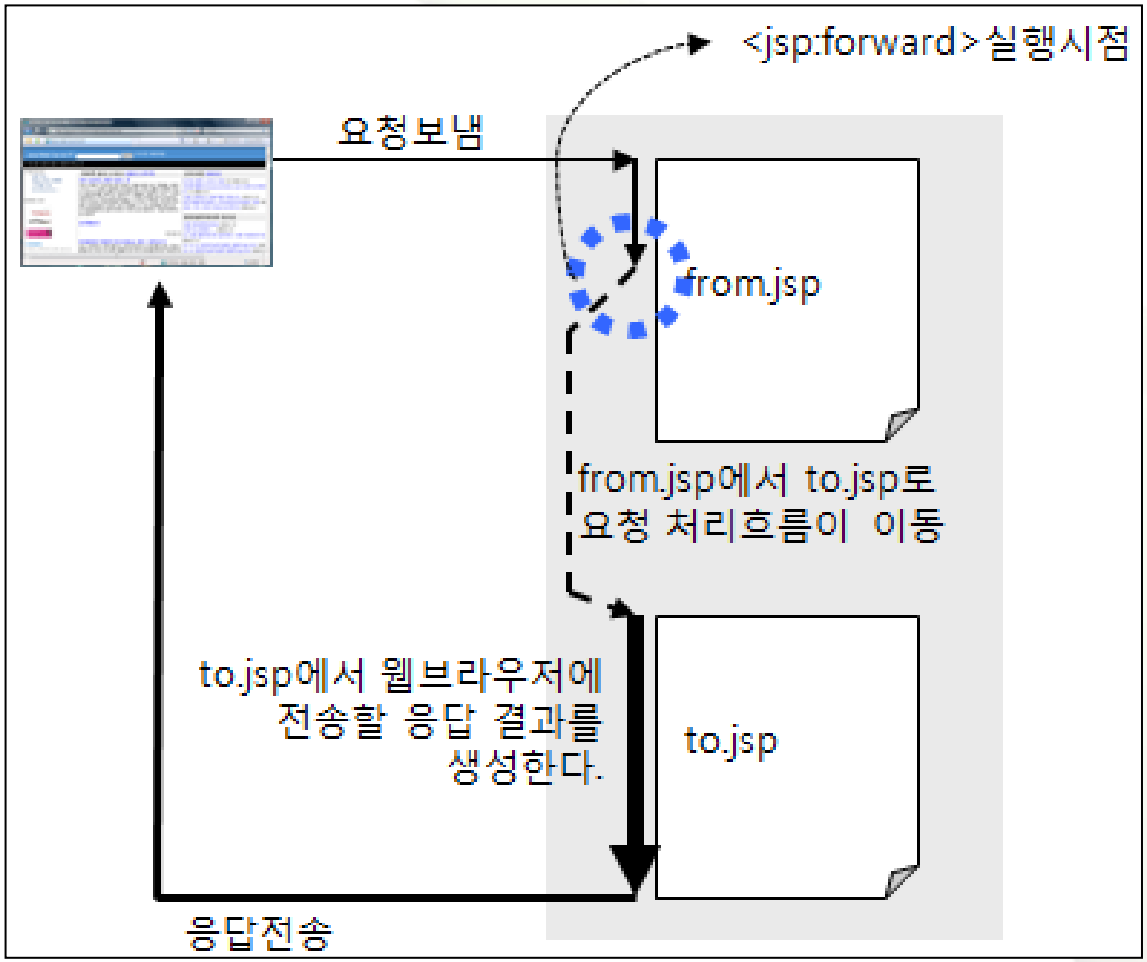


<jsp:include> 액션 태그와 include 디렉티브

비교항목	<jsp:include>	include 디렉티브
처리시간	요청 시간에 처리	JSP 파일을 자바 소스로 변환할 때 처리
기능	별도의 파일로 요청 처리 흐름 이동	현재 파일에 삽입
데이터 전달방법	request 기본 객체나 <jsp:param>을 이용한 파라미터 전달	페이지 내에 변수를 선언하고 변수 공유
용도	화면 레이아웃의 일부분을 모듈화할 때 주로 사용된다.	다수의 JSP 페이지에서 공통으로 사용되는 변수를 지정하는 코드나 저작권과 같은 문장을 포함한다.

<jsp:forward> 액션 태그

- 하나의 JSP 페이지에서 다른 JSP 페이지로 요청 처리를 전달할 때 사용
- 구문 : <jsp:forward page="이동할 페이지" />
- 동작 방식



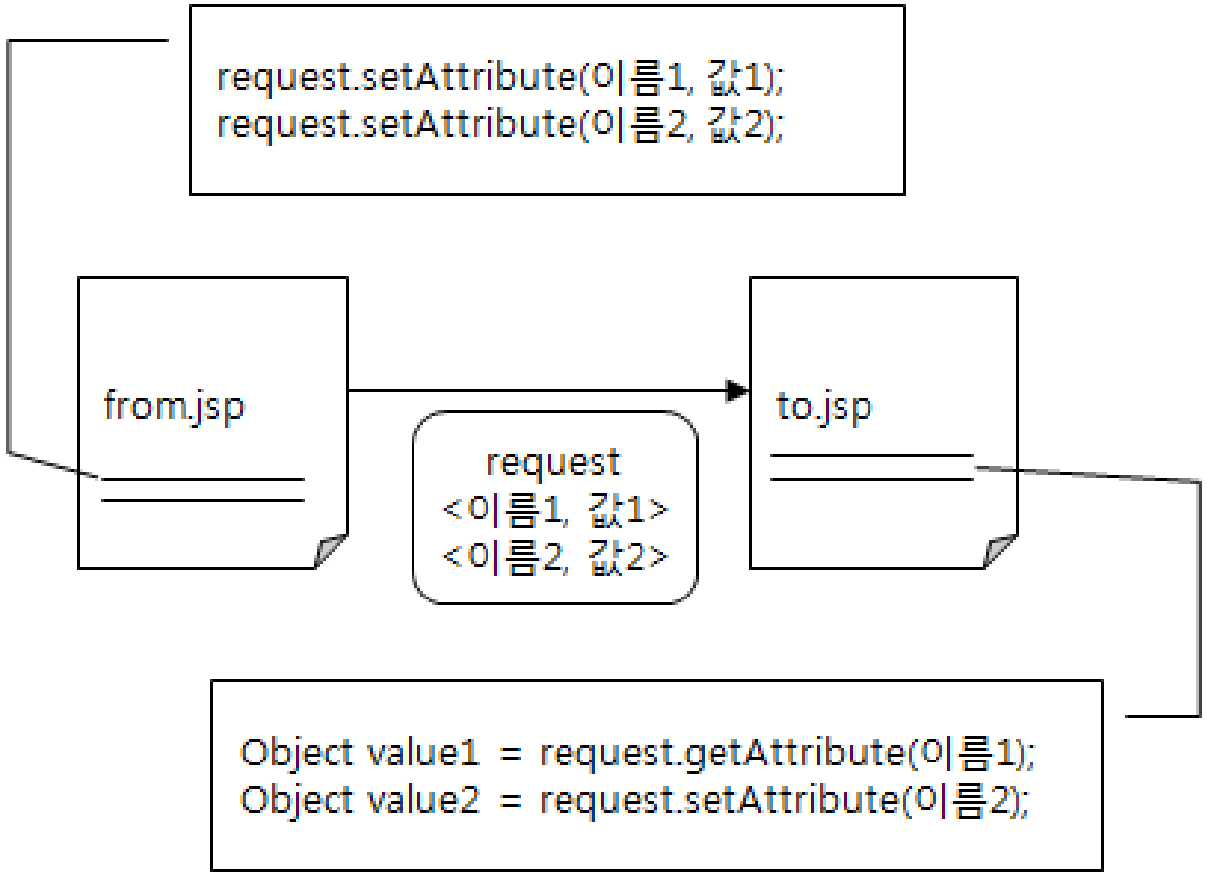
<jsp:forward> 액션 태그

- 사용 사례

```
<%@ page contentType = "text/html; charset=utf-8" %>

<%
    String forwardPage = null;

    // 조건에 따라 이동할 페이지 지정
    if (조건판단1) {
        forwardPage = "페이지URI1";
    } else if (조건판단2) {
        forwardPage = "페이지URI2";
    } else if (조건판단3) {
        forwardPage = "페이지URI3";
    }
%>

<jsp:forward page="<%= forwardPage %>" />
```

기본 객체의 속성을 이용한 값 전달

- 속성을 이용해서 JSP 페이지 간 값 전달
  - » <jsp:include>나 <jsp:forward>에서 사용



상태 유지 개요

- Http 프로토콜은 모든 요청을 독립적인 개별 요청으로 처리
- 이로 인해 이전 요청과 관련된 정보가 유지되지 않습니다.
- 다수의 상태유지 객체를 통해 상태 정보 유지를 지원

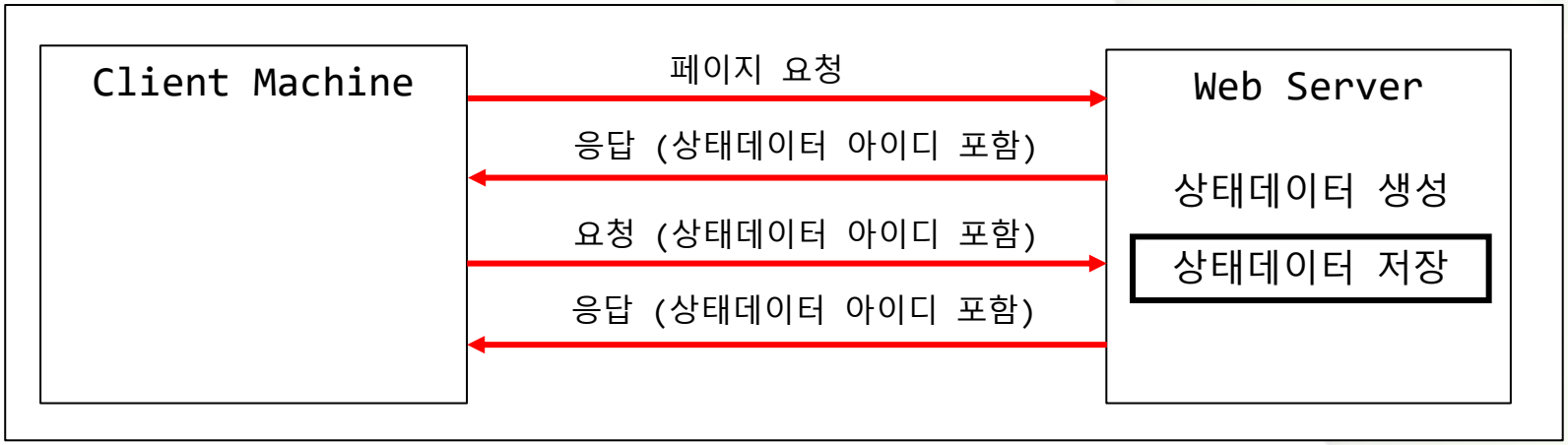
- 상태 유지 객체 종류

객체명	저장위치	유지범위
cookie	Browser / Client Machine	동일 브라우저/컴퓨터
session	Server	동일 브라우저
application	Server	동일 웹 애플리케이션

상태 유지 방법

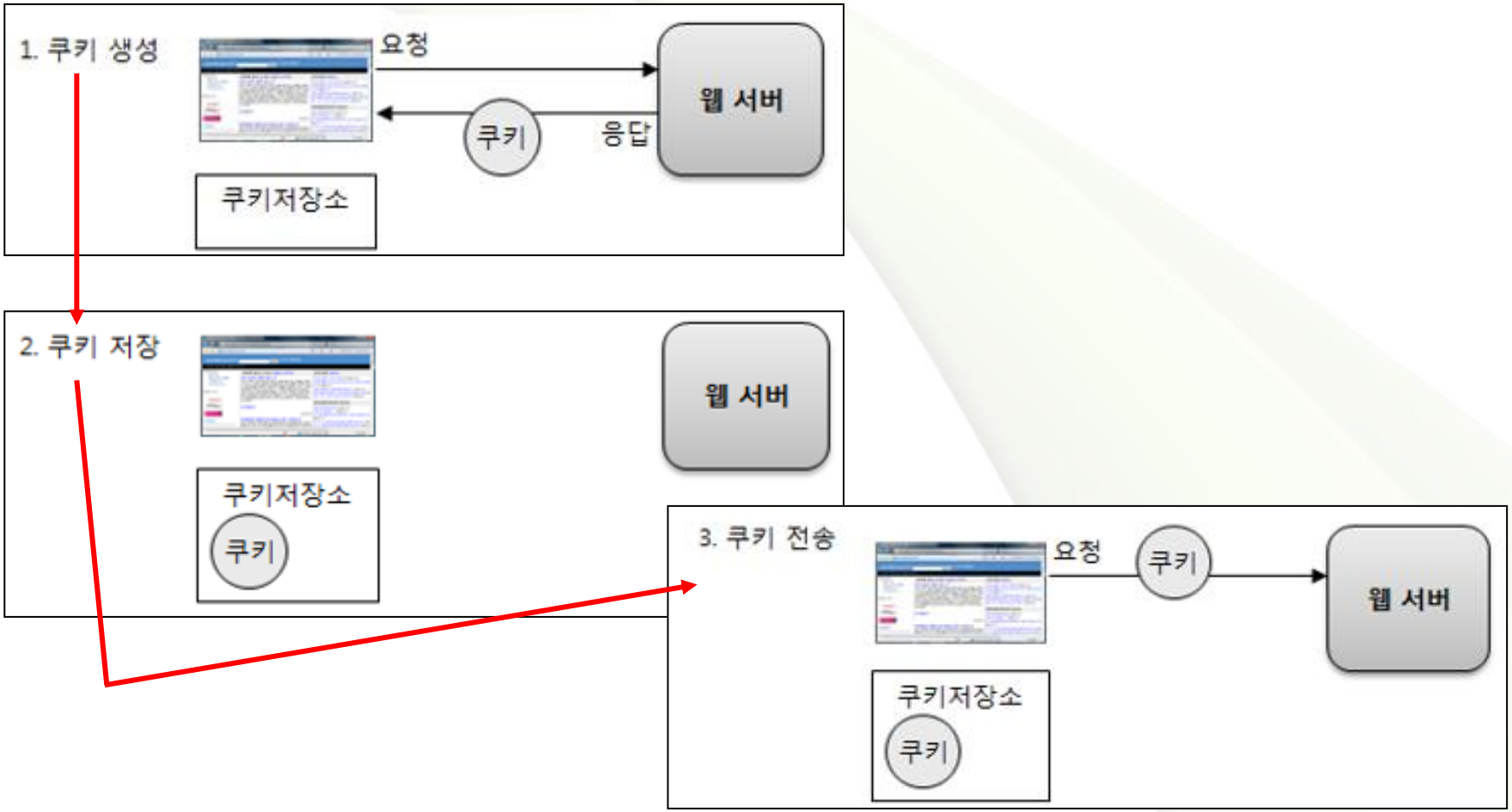
- ClientSide State Management

- ServerSide State Management



쿠키(Cookie)

- 쿠키(cookie)는 웹 브라우저가 보관하고 있는 데이터로서 웹 서버에 요청을 보낼 때 함께 전송
- 동작 방식



쿠키의 구성

- 구성 요소
  - » 이름 - 각각의 쿠키를 구별하는 데 사용되는 이름
  - » 값 - 쿠키의 이름과 관련된 값
  - » 유효시간 - 쿠키의 유지 시간
  - » 도메인 - 쿠키를 전송할 도메인
  - » 경로 - 쿠키를 전송할 요청 경로

- 쿠키 이름의 제약
  - » 쿠키의 이름은 아스키 코드의 알파벳과 숫자만을 포함할 수 있다.
  - » 콤마(,), 세미콜론(;), 공백(' ') 등의 문자는 포함할 수 없다.
  - » '$'로 시작할 수 없다.

JSP에서 쿠키 생성 / 읽기

- Cookie 클래스를 이용해서 쿠키 생성

```
<%
    Cookie cookie = new Cookie("cookieName", "cookieValue");
    response.addCookie(cookie);
%>
```

- 클라이언트가 보낸 쿠키 읽기

```
<%
    Cookie[] cookies = request.getCookies();
%>
```

- 읽기 관련 주요 메서드

메서드	리턴타입	설명
getName()	String	쿠키의 이름을 구한다.
getValue()	String	쿠키의 값을 구한다.

쿠키 값의 인코딩/디코딩 처리

- 쿠키는 값으로 한글과 같은 문자를 가질 수 없음
  - » 쿠키의 값을 인코딩해서 지정할 필요 있음

- 쿠키 값 인코딩/디코딩 처리
  - » 값 설정 : URLEncoder.encode("값", "utf-8")

```
<%
    new Cookie("name", URLEncoder.encode("값", "utf-8"));
%>
```

  - » 값 조회 : URLDecoder.decode("값", "utf-8")

```
<%
    Cookie cookie = …;
    String value = URLDecoder.decode(cookie.getValue(), "utf-8");
%>
```

쿠키 값 변경

- 기존에 존재하는 지 확인 후, 쿠키 값 새로 설정

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

if (cookies != null && cookies.length > 0) {

    for (int i = 0 ; i < cookies.length ; i++) {

        if (cookies[i].getName().equals("name")) {
            Cookie cookie = new Cookie(name, value);
            response.addCookie(cookie);
        }

    }

}
```

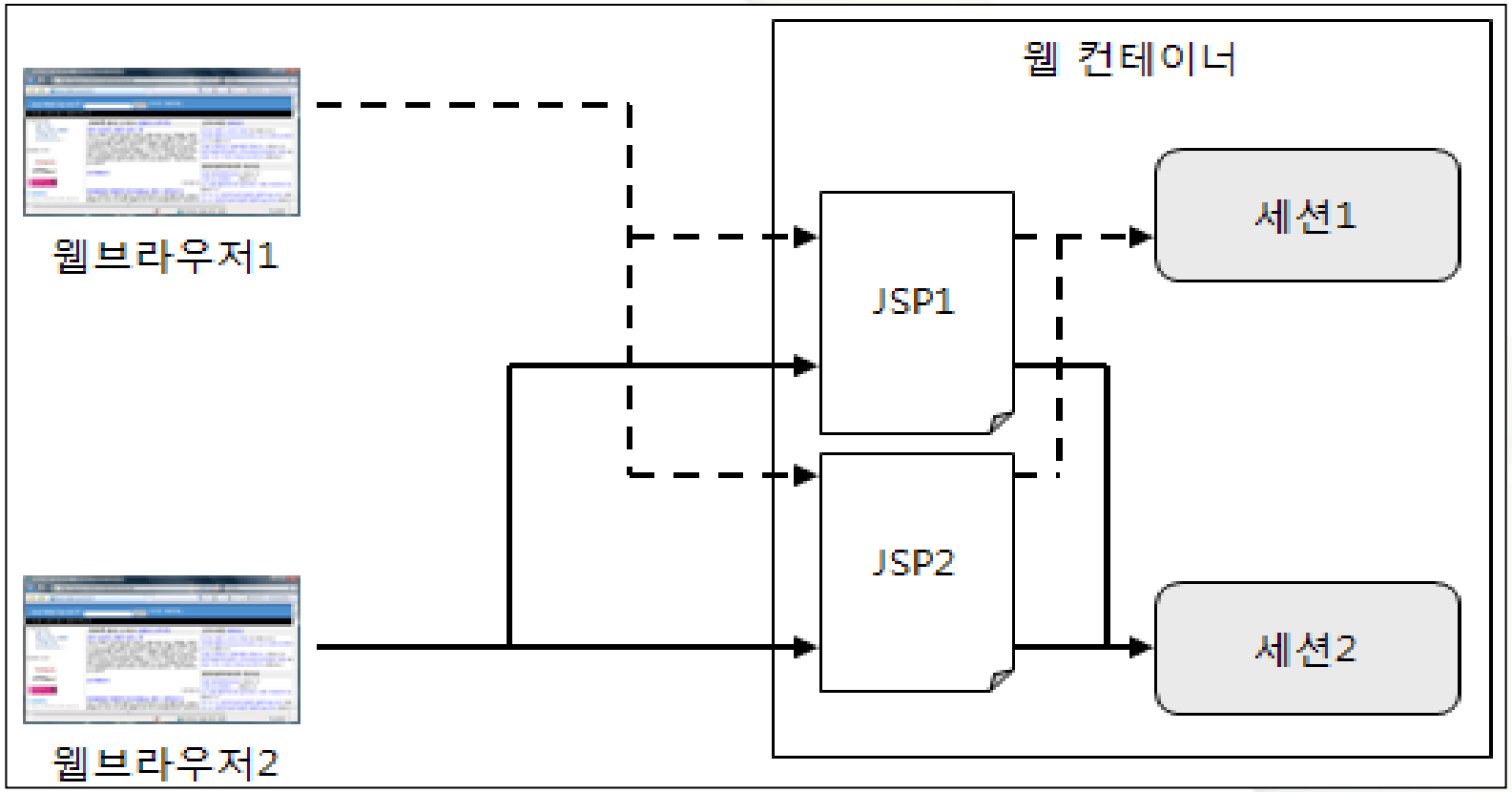
쿠키의 유효 시간

- 유효 시간을 지정하지 않으면 웹 브라우저를 닫을 때 쿠키도 함께 삭제

- Cookie.setMaxAge()로 쿠키 유효 시간 설정
  - » 유효 시간에 도달하지 않은 경우 웹 브라우저를 닫더라도 쿠키가 삭제되지 않고 이후 웹 브라우저를 열었을 때 해당 쿠키 전송
  - » 유효 시간은 초 단위로 설정

세션 (session)

- 웹 컨테이너에서 클라이언트의 정보를 보관할 때 사용
- 서버에서 생성하고 저장
- 클라이언트(브라우저)마다 각각의 세션 사용



세션과 session 기본 객체

- page 디렉티브의 session 속성 값을 true로 지정
  - » 세션이 존재하지 않을 경우 세션이 생성되고, 세션이 존재할 경우 이미 생성된 세션을 사용

- session 기본 객체를 이용해서 세션에 접근

```
<%@ page contentType="text/html;charset=utf-8" … %>

<%@ page session = "true" %>

<%
    ...
    session.setAttribute("userInfo", userInfo);
    ...
%>
```

- 속성 이용해서 클라이언트 관련 정보 저장

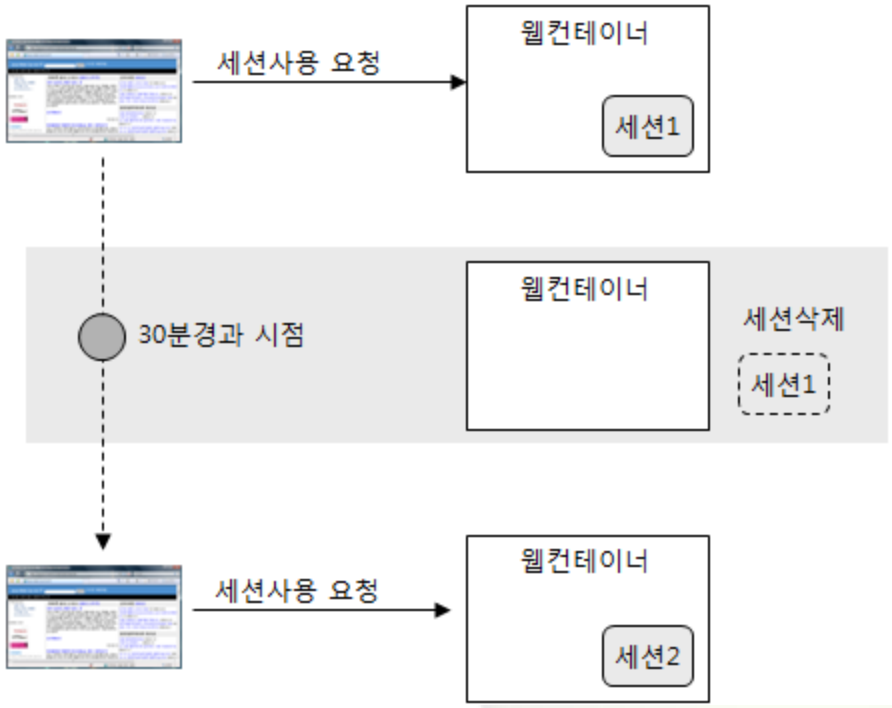
세션 종료 / 타임 아웃

- session.invalidate() 을 이용해서 세션 종료
  - » 세션이 종료되면 기존에 생성된 세션 삭제
  - » 이후 접근 시 새로운 세션 생성 됨

- 마지막 세션 사용 이후 유효 시간이 지나면 자동 종료
  - » web.xml 파일에서 지정

```
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
```

  - » 또는 session 기본 객체의 setMaxInactiveInterval() 메서드를 이용해서 초 단위 값 지정



session 기본 객체에 저장될 값 타입

- 모든 종류의 데이터 형식 저장 가능 (Object)

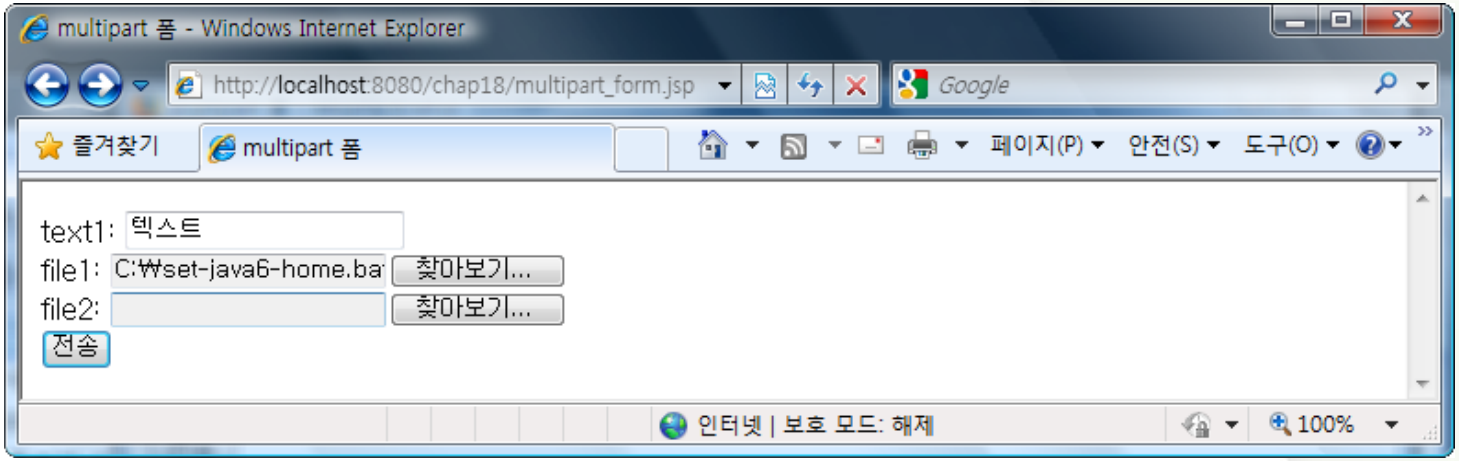
- 필요한 값을 개별 속성으로 저장하는 것 보다는
  - » 유지 보수 불편함 (신규 속성 추가시 변경할 코드 증가 등)

- 속성을 하나로 묶은 자바 클래스를 작성해서 저장하는 것이 좋음
  - » 유지 보수 편리함

파일 전송을 위한 브라우저 FORM 설정

- method는 POST 방식으로 지정
- enctype은 multipart/form-data로 설정
- type 속성이 file인 <input> 태그를 통해 파일 선택

```
<form action="..." method="post" enctype="multipart/form-data">
     ...
    <input type="file" name="file1" />
</form>
```



multipart/form-data로 전송된 데이터 형식

```
[multipart/form-data; boundary=---------------------------7d9202102082c]
-----------------------------7d9202102082c
Content-Disposition: form-data; name="text1"

텍스트
-----------------------------7d9202102082c
Content-Disposition: form-data; name="file1"; filename="set-java6-home.bat"
Content-Type: application/octet-stream

set JAVA_HOME=c:₩java₩jdk1.6.0_07
set PATH=%JAVA_HOME%₩bin;%PATH%
-----------------------------7d9202102082c
Content-Disposition: form-data; name="file2"; filename=""
Content-Type: application/octet-stream


-----------------------------7d9202102082c--
```

일반 파라미터

파일 데이터

FileUpload API

- multipart/form-data로 전송된 데이터 처리
  - » form-data와 파일 데이터를 분리하고 별도의 변수에 저장하는 등의 처리 과정 필요
  - » 규칙에 의한 전형적인 처리 과정이므로 이미 구현된 라이브러리 활용
- 파일 업로드 라이브러리
  - » commons-fileupload-1.x.x.jar
- 추가 의존 라이브러리
  - » commons-io-2.x.x.jar

FileUpload API를 이용한 업로드 데이터 처리

```
// 1. multipart/form-data 여부 확인
boolean isMultipart = ServletFileUpload.isMultipartContent(request);
if (isMultipart) {
    // 2. 메모리나 파일로 업로드 파일 보관하는 FileItem의 Factory 설정
    DiskFileItemFactory factory = new DiskFileItemFactory();
    // 3. 업로드 요청을 처리하는 ServletFileUpload 생성
    ServletFileUpload upload = new ServletFileUpload(factory);
    // 4. 업로드 요청 파싱해서 FileItem 목록 구함
    List<FileItem> items = upload.parseRequest(request);
    Iterator<FileItem> iter = items.iterator();
    while (iter.hasNext()) {
        FileItem item = iter.next();
        // 5. FileItem이 폼 입력 항목인지 여부에 따라 알맞은 처리
        if (item.isFormField()) { // 텍스트 입력인 경우
            String name = item.getFieldName();
            String value = item.getString("utf-8");
        } else { // 파일 업로드인 경우
            String name = item.getFieldName();
            String fileName = item.getName();
            String contentType = item.getContentType();
            boolean isInMemory = item.isInMemory();
            long sizeInBytes = item.getSize();
        }
    }
}
```

업로드 한 파일 처리 방법

- FileItem.write(File file) 메서드를 사용하는 방법
- FileItem.getInputStream() 메서드로 구한 입력 스트림으로부터 바이트 데이터를 읽어와 FileOutputStream을 사용해서 파일에 출력하는 방법
- FileItem.get() 메서드로 구한 바이트 배열을 FileOutputStream을 사용해서 파일에 출력하는 방법

파일 다운로드

- 응답 컨텐츠 타입은 application/octet-stream

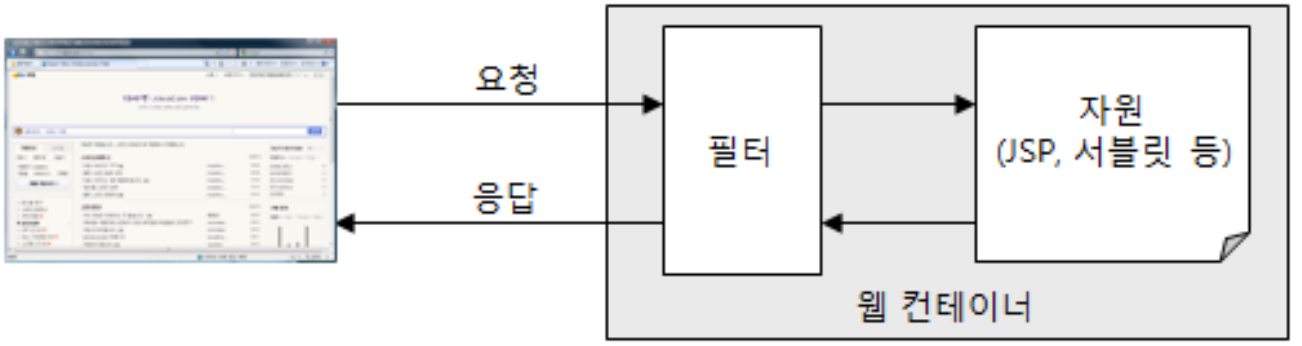
- Content-Disposition 헤더로 파일명 지정
  - » 파일명 설정시 ISO-8859-1로 인코딩 변환해서 설정

```
<%
  String fileName =
    new String(item.getFileName().getBytes("utf-8"), "iso-8859-1");
  response.setContentType("application/octet-stream");
  response.setHeader("Content-Disposition",
                     "attachment; filename=\"" + filename + "\"");
%>
```

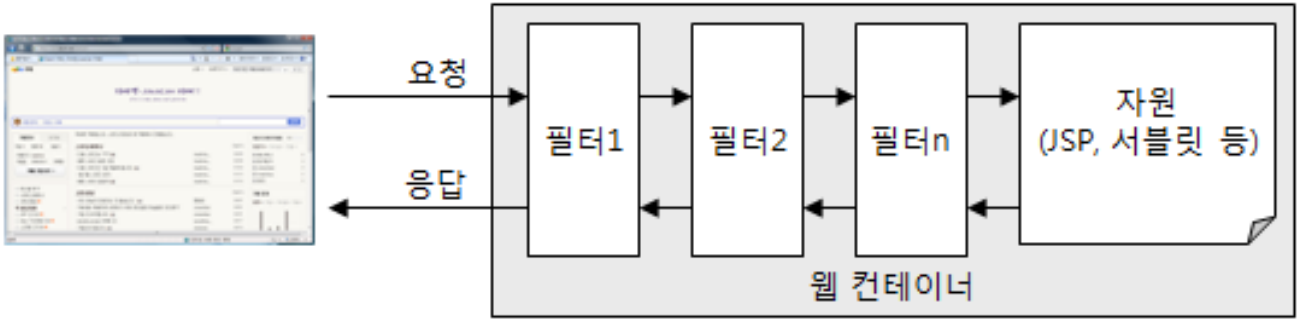
- 실제 파일 전송
  - » response.getOutputStream()으로 구한 OutputStream에 파일 데이터 출력

필터

- HTTP 요청과 응답을 변경할 수 있는 재사용 가능한 코드
- 필터의 기본 구조



- 요청의 내용을 변경하거나 응답의 내용을 변경 가능
- 1개 이상의 필터 연동 가능



필터 구현

- Filter 인터페이스 사용
- Filter 인터페이스의 메서드
  - » public void destroy()
    필터가 웹 컨테이너에서 삭제될 때 호출된다.

```
public void init(FilterConfig filterConfig) throws ServletException
```

- 필터를 초기화할 때 호출

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain c) throws IOException, ServletException
```

- 체인을 따라 다음에 존재하는 필터로 이동한다.
- 체인의 가장 마지막에는 클라이언트가 요청한 최종 자원이 위치한다.

```
public void destroy()
```

- 필터가 웹 컨테이너에서 삭제될 때 호출

필터 구현

- doFilter() 메서드에서 필터 기능 구현

```
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
        throws IOException, ServletException {

    public void doFilter(ServletRequest request,
                ServletResponse response
                FilterChain chain)
                throws IOException, ServletException {

        // 1. request 파리미터를 이용하여 요청의 필터 작업 수행
        ...

        // 2. 체인의 다음 필터 처리
        chain.doFilter(request, response);

        // 3. response를 이용하여 응답의 필터링 작업 수행
        ...
    }
}
```

필터 설정

- web.xml에 URL 별 매핑 설정 추가

```
<web-app ...>

    <filter>
        <filter-name>FilterName</filter-name>
        <filter-class>com.exampleweb.filter.AuthFilter</filter-class>
        <init-param>
           <param-name>paramName</param-name>
           <param-value>value</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>FilterName</filter-name>
        <url-pattern>*.action</url-pattern>
    </filter-mapping>

    ...
</web-app>
```

- annotation 사용

```
@WebFilter("*.action")
public class AuthFilter implements Filter { … }
```

필터 설정

- <dispatcher> 를 통한 필터 적용 대상 지정

```
<filter-mapping>
    <filter-name>AuthCheckFilter</filter-name>
    <servlet-name>FileDownload</servlet-name>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

  - » <dispatcher>의 값
    - › REQUEST - 클라이언트의 요청인 경우에 필터를 사용 (기본값)
    - › FORWARD - forward()를 통해서 제어를 이동하는 경우에 필터를 사용
    - › INCLUDE - include()를 통해서 포함하는 경우에 필터를 사용

필터의 응용

- 데이터 변환(다운로드 파일의 압축/데이터 암호화/이미지 변환 등)
- XSL/T를 이용한 XML 문서 변경
- 사용자 인증
- 캐싱 필터
- 자원 접근에 대한 로깅

ServletContextListener

- 웹 어플리케이션의 시작 이벤트나 종료 이벤트를 처리
- 웹 컨테이너는 ServletContextListener의 이벤트 메서드 호출
- ServletContextListener 인터페이스의 이벤트 처리 메서드
  - » public void contextInitialized(ServletContextEvent sce)
    - › 웹 어플리케이션이 초기화될 때 호출되는 메서드
  - » Public void contextDestroyed(ServletContextEvent sce)
    - › 웹 어플리케이션이 종료될 때 호출되는 메서드

HttpSessionListener

- 세션의 시작 이벤트나 종료 이벤트를 처리

- 웹 컨테이너는 HttpSessionListener의 이벤트 메서드 호출

- HttpSessionListenerListener 인터페이스의 이벤트 처리 메서드
  - » public void sessionCreated(HttpSessionEvent se)
    - › 세션이 처음 생성될 때 호출되는 메서드
  - » Public void sessionDestroyed(HttpSessionEvent se)
    - › 세션이 종료될 때 호출되는 메서드

web.xml 파일에 리스너 등록

- <listener> 태그 이용

```
<web-app ...>

    <listener>
        <listener-class>com.webexample.DemoWebListener</listener-class>
    </listener>
    ...

</web-app>
```

ServletContextEvent로부터 필요 정보 조회

- contextInitialized()와 contextDestroyed() 메서드에 전달되는 객체

- ServletContext를 구해주는 getServletContext() 메서드 제공
  - » ServletContext.getInitParameter() 메서드를 이용해서 web.xml 파일에 등록한 초기화 파라미터 값 조회

```
<web-app ...>
    <context-param>
        <param-name>jdbcdriver</param-name>
        <param-value>com.mysql.jdbc.Driver</param-value>
    </context-param>
</web-app>
```

```
public class DBCPInitListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        try {
            ServletContext context = sce.getServletContext();
            String drivers = context.getInitParameter("jdbcdriver");
        ...
    }
}
```

자바빈(JavaBeans)

- 자바빈 - 웹 프로그래밍에서 데이터의 표현을 목적으로 사용

- 일반적인 구성
  - » 값을 저장하기 위한 필드
  - » 값을 수정하기 위한 setter
  - » 값을 읽기 위한 getter

```
public class BeanClassName {
    /* 값을 저장하는 필드 */
    private String value;

    /* BeanClassName의 기본 생성자 */
    public BeanClassName() {
    }

    /* 필드의 값을 읽어오는 값 */
    public String getValue() {
        return value;
    }

    /* 필드의 값을 변경하는 값 */
    public void setValue(String value) {
        this.value = value;
    }
}
```

자바빈 프로퍼티

- 프로퍼티는 자바빈에 저장되어 있는 값을 표현

- 메서드 이름을 사용해서 프로퍼티의 이름을 결정

- 규칙 : 프로퍼티 이름이 propertyName일 경우
  - » setter: public void setPropertyName(Type value)
  - » getter: public Type getPropertyName()
  - » boolean 타입일 경우 getter에 get대신 is 사용 가능
  - » 배열 지정 가능: 예) public void setNames(String[])

- 읽기/쓰기
  - » 읽기 전용 : get 또는 is 메서드만 존재하는 프로퍼티
  - » 읽기/쓰기 : get/set 또는 is/set 메서드가 존재하는 프로퍼티

<jsp:useBean> 태그

- JSP에서 자바빈 객체를 생성할 때 사용

- 구문
  - <jsp:useBean id="[빈이름]" class="[자바빈클래스이름]" scope="[범위]" />
    - id - JSP 페이지에서 자바빈 객체에 접근할 때 사용할 이름
    - class - 패키지 이름을 포함한 자바빈 클래스의 완전한 이름
    - scope - 자바빈 객체가 저장될 영역을 지정한다. page, request, session, application 중 하나를 값으로 갖는다. 기본값은 page.
  - 사용 사례

```
<jsp:useBean id="info"
             class="com.demoweb.vor.MemberInfo"
             scope="request" />

<%= info.getName() %>
```

<jsp:useBean> 액션 태그의 동작 방식

- id 속성에 해당하는 객체가 지정한 영역에 존재할 경우
  - » 존재하는 객체를 사용

- 지정한 영역에 존재하지 않을 경우
  - » class 속성에 명시한 타입을 이용해서 객체를 생성 및
  - » 지정한 영역에 생성한 객체를 저장

- scope 속성 값에 따라 저장되는 기본 객체
  - » "page" - pageContext 기본 객체
  - » "request" - request 기본 객체
  - » "session" - session 기본 객체
  - » "application" - application 기본 객체

<jsp:setProperty> 액션 태그

- 자바빈 객체의 프로퍼티 값 설정

- 구문
  - <jsp:setProperty name="id" property="이름" value="값" />
    - name - 자바빈 객체의 이름
    - property - 값을 설정할 프로퍼티
    - value - 프로퍼티의 값
  - <jsp:setProperty name="id" property="이름" param="파라미터이름" />
    - param - 프로퍼티의 값으로 사용할 파라미터 이름.
  - <jsp:setProperty name="id" property="*" />
    - 프로퍼티와 동일한 이름의 파라미터를 이용해서 값을 설정
    - 폼에 입력한 값을 자바 객체에 저장할 때 유용하게 사용

<jsp:getPropety> 액션 태그

- 프로퍼티의 값을 출력하기 위해 사용

- 구문
  - » <jsp:getProperty name="자바빈" property="프로퍼티" />

프로퍼티 타입에 따른 값 매핑

프로퍼티의 타입	변환 방법	기본 값
boolean 또는 Boolean	Boolean.valueOf(String)을 값으로 갖는다	false
byte 또는 Byte	Byte.valueOf(String)을 값으로 갖는다	(byte) 0
short 또는 Short	Short.valueOf(String)을 값으로 갖는다	(short) 0
char 또는 Character	입력한 값의 첫 번째 글자를 값으로 갖는다	(char) 0
int 또는 Integer	Integer.valueOf(String)을 값으로 갖는다	0
long 또는 Long	Long.valueOf(String)을 값으로 갖는다	0L
double 또는 Double	Double.valueOf(String)을 값으로 갖는다	0.0
float 또는 Float	Float.valueOf(String)을 값으로 갖는다	0.0f

표현 언어

- Expression Language

- JSP에서 사용가능한 새로운 스크립트 언어

- EL의 주요 기능
  - » JSP의 네 가지 기본 객체가 제공하는 영역의 속성 사용
  - » 집합 객체에 대한 접근 방법 제공
  - » 수치 연산, 관계 연산, 논리 연산자 제공
  - » 자바 클래스 메서드 호출 기능 제공
  - » 표현언어만의 기본 객체 제공

- 간단한 구문 때문에 표현식 대신 사용

구문

- 기본 문법
  - » ${expr}, #{expr}
  - » 사용예
    - › <jsp:include page="/module/${skin.id}/header.jsp" />
    - › ${sessionScope.member.id}님 환영합니다.
  - » ${expr}은 표현식이 실행되는 시점에 바로 값 계산
  - » #{expr}은 값이 실제로 필요한 시점에 값 계산
    - › JSP 템플릿 텍스트에서는 사용 불가

- 스크립트 요소(스크립트릿, 표현식, 선언부)를 제외한 나머지 부분에서 사용

EL에서 기본 객체

기본 객체	설명
pageContext	JSP의 page 기본 객체와 동일하다
pageScope	pageContext 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
requestScope	request 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
sessionScope	session 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
applicationScope	application 기본 객체에 저장된 속성의 <속성, 값> 매핑을 저장한 Map 객체
param	요청 파라미터의 <파라미터이름, 값> 매핑을 저장한 Map 객체
paramValues	요청 파라미터의 <파라미터이름, 값배열> 매핑을 저장한 Map 객체
header	요청 정보의 <헤더이름, 값> 매핑을 저장한 Map 객체
headerValues	요청 정보의 <헤더이름, 값 배열> 매핑을 저장한 Map 객체
cookie	<쿠키 이름, Cookie> 매핑을 저장한 Map 객체
initParam	초기화 파라미터의 <이름, 값> 매핑을 저장한 Map 객체

EL 데이터 타입

- 불리언(Boolean) 타입 - true 와 false
- 정수타입 - 0~9로 이루어진 정수 값.
- 실수타입 - 0~9로 이루어져 있으며, 소수점('.')을 사용할 수 있고, 3.24e3과 같이 지수형으로 표현 가능
- 문자열 타입 - 따옴표(' 또는 ")로 둘러싼 문자열.
  - » 작은 따옴표 사용시, 값에 포함된 작은 따옴표는 \'로 입력
  - » \ 기호 자체는 \\ 로 표시한다.
- 널 타입 - null

EL에서 객체에 접근

- ${<표현1>.<표현2>} 형식 사용
- 처리 과정
    1. <표현1>을 <값1>로 변환한다.
    2. <값1>이 null이면 null을 리턴한다.
    3. <값1>이 null이 아닐 경우 <표현2>를 <값2>로 변환한다.
        1. <값2>가 null이면 null을 리턴한다.
    4. <값1>이 Map, List, 배열인 경우
        1. <값1>이 Map이면
            1. <값1>.containsKey(<값2>)가 false이면 null을 리턴한다.
            2. 그렇지 않으면 <값1>.get(<값2>)를 리턴한다.
        2. <값1>이 List나 배열이면
            1. <값2>가 정수 값인지 검사한다. (정수 값이 아닐 경우 에러 발생)
            2. <값1>.get(<값2>) 또는 Array.get(<값1>, <값2>)를 리턴한다.
            3. 위 코드가 예외를 발생하면 에러를 발생한다.
    5. <값1>이 다른 객체이면
        1. <값2>를 문자열로 변환한다.
        2. <값1>이 이름이 <값2>이고 읽기 가능한 프로퍼티를 포함하고 있다면 프로퍼티의 값을 리턴한다.
        3. 그렇지 않을 경우 에러를 발생한다.

연산자

- 수치 연산자
  - » +, -, *, / 또는 div, % 또는 mod
- 비교 연산자
  - » == 또는 eq, != 또는 ne
  - » < 또는 lt, <= 또는 le, > 또는 gt, >= 또는 ge
- 논리 연산자
  - » && 또는 and
  - » || 또는 or
  - » ! 또는 not
- empty 연산자
  - » empty <값>
    - › 값이 null이면, true
    - › 값이 빈 문자열("")이면, true
    - › 값의 길이가 0인 배열이나 콜렉션이면 true
    - › 이 외의 경우에는 false
- 비교 선택 연산자
  - » <수식> ? <값1> : <값2>

EL의 용법

- request나 session 속성으로 전달한 값을 출력

- 액션 태그나 커스텀 태그의 속성 값
  - » <jsp:include page="/lo/${layout.module}.jsp" flush="true" />

- 함수 호출
  - » 코드의 간결함 및 가독성 향상

JSTL

- JSP Standard Tag Library - 널리 사용되는 커스텀 태그를 표준으로 만든 태그 라이브러리

- JSTL 태그 종류

라이브러리	하위 기능	접두어	관련URI
코어	변수지원 흐름 제어 URL 처리	c	http://java.sun.com/jsp/jstl/core
XML	XML 코어 흐름 제어 XML 변환	x	http://java.sun.com/jsp/jstl/xml
국제화	지역 메시지 형식 숫자 및 날짜 형식	fmt	http://java.sun.com/jsp/jstl/fmt
데이터베이스	SQL	sql	http://java.sun.com/jsp/jstl/sql
함수	콜렉션 처리 String 처리	fn	http://java.sun.com/jsp/jstl/functions

JSTL 1.2 관련 jar 파일 필요

- jstl-1.2.jar 파일 필요

코어 태그 라이브러리 종류

기능분류	태그	설명
변수 지원	set	JSP에서 사용될 변수를 설정한다.
	remove	설정한 변수를 제거한다.
흐름 제어	if	조건에 따라 내부 코드를 수행한다.
	choose	다중 조건을 처리할 때 사용된다.
	forEach	콜렉션이나 Map의 각 항목을 처리할 때 사용된다.
	forTokens	구분자로 분리된 각각의 토큰을 처리할 때 사용된다.
URL 처리	import	URL을 사용하여 다른 자원의 결과를 삽입한다.
	redirect	지정한 경로로 리다이렉트 한다.
	url	URL을 재작성 한다.
기타 태그	catch	예외 처리에 사용된다.
	out	JspWriter에 내용을 알맞게 처리한 후 출력한다.

변수 지원 태그

- 변수 설정
  - » EL 변수 값 설정 (생성 또는 변경)
    - › <c:set var="변수명" value="값" [scope="영역"] />
    - › <c:set var="변수명" value="값" [scope="영역"]>값</c:set>
  - » 특정 EL 변수의 프로퍼티 값 설정
    - › <c:set target="대상" property="프로퍼티이름" value="값" />
    - › <c:set target="대상" property="프로퍼티이름">값</c:set>

- 변수 삭제
  - » <c:remove var="varName" [scope="영역"] />
    - › scope 미지정시 모든 영역의 변수 삭제

흐름 제어

- if - 조건이 true일 경우 몸체 내용 실행

```
<c:if test="조건">
...
</c:if>
```

- choose - when - otherwise
  - » swich - case - default와 동일

```
<c:choose>
  <c:when test="${member.level == 'trial'}" >
    ...
  </c:when>
  <c:when test="${member.level == 'regular'}" >
    ...
  </c:when>
  <c:otherwise>
    ...
  </c:otherwise>
</c:choose>
```

반복 처리 (forEach)

- 집합이나 콜렉션 데이터 사용

```
<c:forEach var="변수" items="아이템">
  … ${변수사용} ...
</c:forEach>
```

- 특정 회수 반복

```
<c:forEach var="i" begin="1" end="10" [step="값"]>
  ${i} 사용
</c:forEach>
```

- varStatus 속성

```
<c:forEach var="item" items="<%= someItemList %>" varStatus="status">
${status.index + 1} 번째 항목 : ${item.name}
</c:forEach>
```

```
index - 루프 실행에서 현재 인덱스, count - 루프 실행 회수
begin - begin 속성 값, end - end 속성 값, step - step 속성 값
first - 현재 실행이 첫 번째 실행인 경우 true
last - 현재 실행이 루프의 마지막 실행인 경우 true
current - 콜렉션 중 현재 루프에서 사용할 객체
```

URL 관련 태그

- import - 외부/내부 페이지를 현재 위치에 삽입

```
<c:import url="URL" [var="변수명"] [scope="영역"] [charEncoding="캐릭터셋"]>
    <c:param name="파라미터이름" value="값" />
    ...
</c:import>
```

» 상대 URL import 시 <jsp:include>와 동일하게 동작

- url - 절대 URL과 상대 URL을 알맞게 생성

```
<c:url value="URL" [var="varName"] [scope="영역"]>
    <c:param name="이름" value="값" />
</c:url>
```

» 웹 컨텍스트 내에서 절대 경로 사용시 컨텍스트 경로 자동 추가

- redirect - 지정한 페이지로 리다이렉트

```
<c:redirect url="URL" [context="콘텍스트경로"]>
    <c:param name="이름" value="값" />
</c:redirect>
```

기타 코어 태그

- out - 데이터를 출력

```
<c:out value="value" [escapeXml="(true|false)"] [default="defaultValue"] />

<c:out value="value" [escapeXml="(true|false)"]>
    default value
</c:out>
```

- escapeXml 속성이 true일 경우 다음과 같이 특수 문자 처리

  » < → < , > → >

  » & → & , ' → ' , " → "

- catch - 몸체에서 발생한 예외를 변수에 저장

```
<c:catch var="exName">
...
예외가 발생할 수 있는 코드
...
</c:catch>
${exName} 사용
```

국제화 태그

기능분류	태그	설명
로케일 지정	setLocale	Locale을 지정한다.
	requestEncoding	요청 파라미터의 캐릭터 인코딩을 지정
메시지 처리	bundle	사용할 번들을 지정
	message	지역에 알맞은 메시지를 출력
	setBundle	리소스 번들을 읽어와 특정 변수에 저장
숫자 및 날짜 포맷팅	formatNumber	숫자를 포맷팅
	formatDate	Date 객체를 포맷팅
	parseDate	문자열로 표시된 날짜를 분석해서 Date 객체로 변환
	parseNumber	문자열로 표시된 날짜를 분석해서 숫자로 변환
	setTimeZone	시간대 정보를 특정 변수에 저장
	timeZone	시간대를 지정

로케일 지정 및 요청 파라미터 인코딩 지정

- <fmt:setLocale value="언어코드" scope="범위" />
  - » 국제화 태그가 Accept-Language 헤더에서 지정한 언어가 아닌 다른 언어를 사용하도록 지정하는 기능

- <fmt:requestEncoding value="캐릭터셋" />
  - » 요청 파라미터의 캐릭터 인코딩을 지정
  - » request.setCharacterEncoding("캐릭터셋")과 동일

<fmt:message> 태그

- 리소스 번들 범위에서 메시지 읽기

```
<fmt:bundle basename="resource.message" [prefix="접두어"]>
    <fmt:message key="GREETING" />
</fmt:bundle>
```

- 지정한 번들에서 메시지 읽기

```
<fmt:setBundle var="message" basename="resource.message" />
...
<fmt:message bundle="${message}" key="GREETING" />
```

- <fmt:message> 태그의 메시지 읽는 순서
  - » bundle 속성에 지정한 리소스 번들을 사용
  - » <fmt:bundle> 태그에 중첩된 경우 <fmt:bundle> 태그에서 설정한 리소스 번들 사용
  - » 1과 2가 아닐 경우 기본 리소스 번들 사용. 기본 리소스 번들은 web.xml 파일에서 javax.servlet.jsp.jstl.fmt.localizationContext 콘텍스트 속성을 통해서 설정 가능

formatNumber 태그

- 숫자를 포맷팅

```
<fmt:formatNumber value="숫자값" [type="값타입"] [pattern="패턴"]
    [currencyCode="통화코드"] [currencySymbol="통화심볼"]
    [groupingUsed="(true|false)"] [var="변수명"] [scope="영역"] />
```

- 주요 속성

속성	표현식/EL	타입	설명
value	사용 가능	String 또는 Number	양식에 맞춰 출력할 숫자
type	사용 가능	String	어떤 양식으로 출력할지를 정한다. number는 숫자 형식, percent는 % 형식, currency는 통화형식으로 출력. 기본 값은 number.
pattern	사용 가능	String	직접 숫자가 출력되는 양식을 지정한다. DecimalFormat 클래스에서 정의되어 있는 패턴 사용
var	사용 불가	String	포맷팅 한 결과를 저장할 변수 명. var 속성을 사용하지 않으면 결과가 곧바로 출력.
scope	사용 불가	String	변수를 저장할 영역. 기본 값은 page 이다.

parseNumber 태그

- 문자열을 숫자 데이터 타입으로 변환

```
<fmt:parseNumber value="값" [type="값타입"] [pattern="패턴"]
    [parseLocale="통화코드"] [integerOnly="true|false"]
    [var="변수명"] [scope="영역"] />
```

- 주요 속성

속성	표현식/EL	타입	설명
value	사용 가능	String	파싱할 문자열
type	사용 가능	String	value 속성의 문자열 타입을 지정. number, currency, percentage 가 올 수 있다. 기본 값은 number
pattern	사용 가능	String	직접 파싱할 때 사용할 양식을 지정
var	사용 불가	String	파싱한 결과를 저장할 변수 명을 지정
scope	사용 불가	String	변수를 저장할 영역을 지정한다. 기본 값은 page.

formatDate 태그

- 날짜 정보를 담은 객체(Date)를 포맷팅

```
<fmt:formatDate value="날짜값"
    [type="타입"] [dateStyle="날짜스타일"] [timeStyle="시간스타일"]
    [pattern="패턴"] [timeZone="타임존"]
    [var="변수명"] [scope="영역"] />
```

- 주요 속성

속성	표현식	타입	설명
value	사용 가능	java.util.Date	포맷팅할 날짜 및 시간 값
type	사용 가능	String	날짜, 시간 또는 둘 다 포맷팅 할 지의 여부를 지정
dateStyle	사용 가능	String	날짜에 대한 포맷팅 스타일을 지정
timeStyle	사용 가능	String	시간에 대한 포맷팅 스타일을 지정
pattern	사용 가능	String	직접 파싱할 때 사용할 양식을 지정
var	사용 불가	String	파싱한 결과를 저장할 변수 명을 지정
scope	사용 불가	String	변수를 저장할 영역을 지정

timeZone과 setTimeZone

- 국제화 태그가 사용할 시간대 설정

```
<fmt:timeZone value="Hongkong">
  <!-- 사용하는 시간을 Hongkong 시간대에 맞춘다. -->
  <fmt:formatDate ... />
</fmt:timeZone>
```

web.xml, 국제화 태그 콘텍스트 속성

속성 이름	설명
localizationContext	기본으로 사용할 리소드 번들을 지정한다. 리소스 번들의 basename을 입력한다.
locale	기본으로 사용할 로케일을 지정한다.
timeZone	기본으로 사용할 시간대를 지정한다.

JSTL이 제공하는 주요 EL 함수

함수	설명
length(obj)	obj가 List와 같은 Collection인 경우 저장된 항목의 개수를 리턴하고, obj가 문자열일 경우 문자열의 길이를 리턴한다.
toUpperCase(str)	str을 대문자로 변환한다.
toLowerCase(str)	str을 소문자로 변환한다.
substring(str, idx1, idx2)	str.substring(idx1, idx2)의 결과를 리턴한다. idx2가 -1일 경우 str.substring(idx1)과 동일하다.
trim(str)	str 좌우의 공백문자를 제거한다.
replace(str, src, dest)	str에 있는 src를 dest로 변환한다.
indexOf(str1, str2)	str1에서 str2가 위치한 인덱스를 구한다.
startsWith(str1, str2)	str1이 str2로 시작할 경우 true를, 그렇지 않을 경우 false를 리턴한다.
endsWith(str1, str2)	str1이 str2로 끝나는 경우 true를, 그렇지 않을 경우 false를 리턴한다.
contains(str1, str2)	str1이 str2를 포함하고 있을 경우 true를 리턴한다.
escapeXml(str)	XML의 객체 참조에 해당하는 특수 문자를 처리한다. 예를 들어, '&'는 '&'로 변환한다.