

제03장

# Spring MVC

Spring

# 학습목표

1. Java Web Development의 개념에 대해서 알 수 있다.
2. JSP 에 대해서 알 수 있다.
3. Spring MVC Project 를 구현하는 방법에 대해서 알 수 있다.

```
each: function(e, t, n) {  
    i = 0,  
    o = e.length,  
    a = M(e);  
    if (n) {  
        if (a) {  
            for (; o > i; i++)  
                if (r = t.apply(e[i], n), r ===  
        } else  
            for (i in e)  
                if (r = t.apply(e[i], n), r ===  
    } else if (a) {  
        for (; o > i; i++)  
            if (r = t.call(e[i], i, e[i])  
    } else  
        for (i in e)  
            if (r = t.call(e[i], i, e[i])  
    return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
} : function(e) {  
    return null == e ? "" : (e + "  
},  
makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
},  
isArray: function(e, t, n) {  
    var r;  
    if (t) {  
        if (n) return m.c  
        for (n = t.length  
            if (n in t  
    }  
}
```

# 목차

1. Java Web Development
2. JSP
3. Spring MVC Project

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; r--;)
            if (n in t && t[r] === e) return r;
    }
}

```

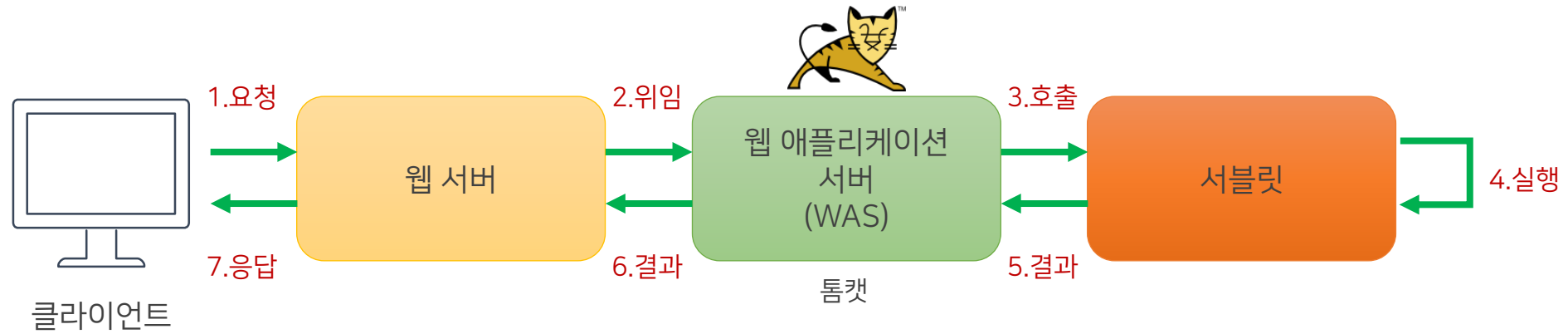
# 1. Java Web Development

# Servlet

## ■ Servlet

- 클라이언트의 요청에 따라 동적으로 서비스를 제공하는 자바 클래스
- 서버 쪽에서 실행되는 Server Side 자바 클래스
- 일반 자바 클래스와 달리 단독적으로 실행할 수 없고  
톰캣(Tomcat)과 같은 JSP/Servlet 컨테이너에 의해서 실행됨
- 서블릿 내부에서 HTML 문서를 만들어서 클라이언트에게 응답할 수 있음

## ■ 동작과정



# Context Path

## ■ Context Path

- Context Root 라고도 함
- WAS에서 웹 애플리케이션을 구분하기 위한 기본 경로를 의미함
- URL 작성 시 호스트 바로 뒤에 나오는 주소를 의미함
  - 프로토콜://호스트:포트/ContextPath/
- WAS에 웹 애플리케이션을 배포할 때 하나의 폴더에 모두 담아 배포하는데 이 때 배포하는 폴더 이름이 Context Path임
- Spring에서는 Context Path 경로를 "슬래시(/)"로 표기함

# 요청과 응답

## ■ 요청

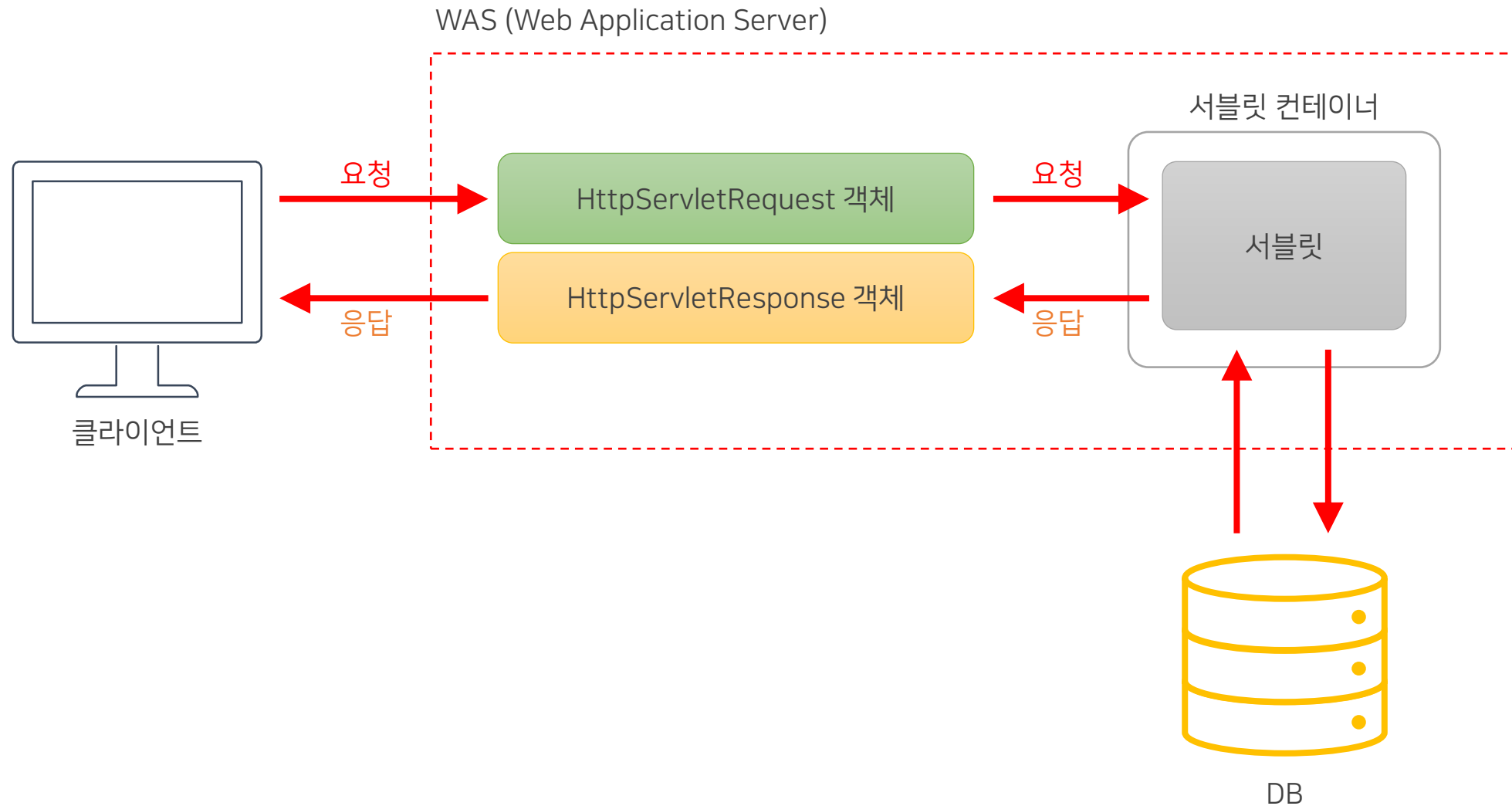
- 클라이언트가 서버 측으로 보내는 데이터 또는 그 흐름
- HttpServletRequest 인터페이스가 담당
- Header, Parameter, Cookie, URL 등의 정보를 처리할 수 있는 기능이 존재함

## ■ 응답

- 서버가 클라이언트 측으로 보내는 데이터 또는 그 흐름
- HttpServletResponse 인터페이스가 담당
- 출력 스트림 활용, 응답 주소 처리, 응답 데이터 타입 및 문자셋 설정 등의 기능이 존재함

# 요청과 응답

## ■ 요청과 응답의 기본 흐름





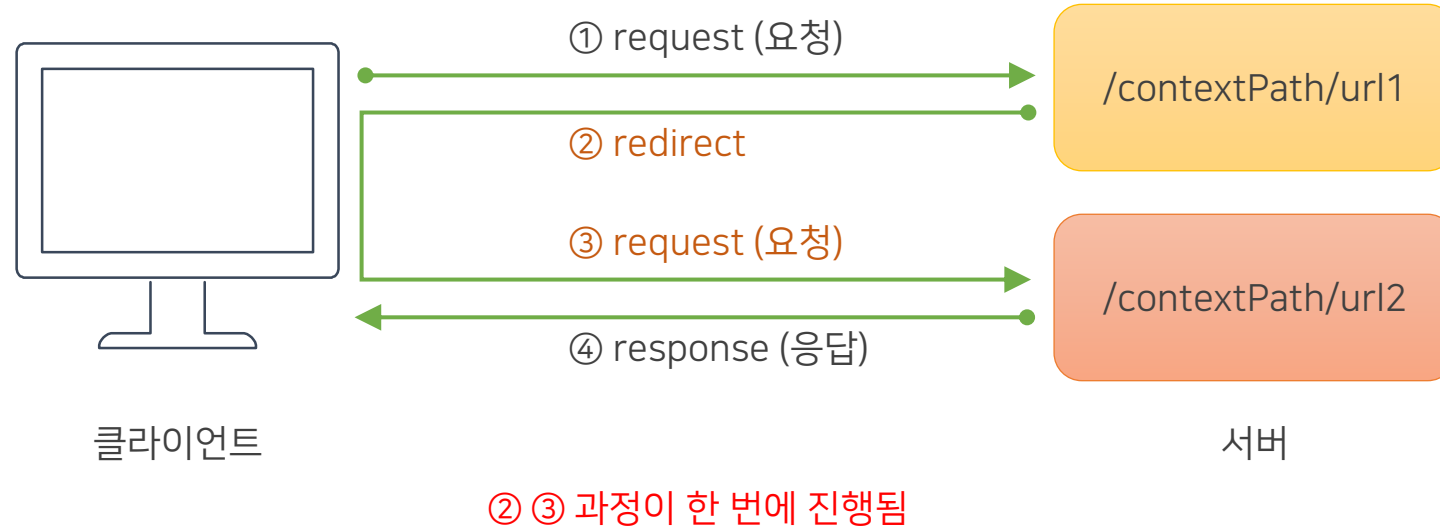
# Redirect

## ■ Redirect

- 기존 요청을 유지하지 않고 새로운 요청을 발생하는 이동 방식
- 서버가 클라이언트에게 이동할 장소를 알려주는 것으로 이동하는 방식
- 이동할 경로를 작성할 때는 ContextPath를 포함한 전체 경로가 필요함
- 클라이언트가 직접 이동하는 방식이기 때문에 URL을 통해서 redirect 경로를 확인할 수 있음
- DB가 변경되는 작업 이후에는 Redirect를 진행해야 함
  - INSERT, UPDATE, DELETE 이후에는 Redirect 방식으로 이동해야 함
- HttpServletResponse 인터페이스의 sendRedirect() 메소드를 이용해 redirect 할 수 있음
- Spring에서는 "redirect:URL" 방식의 값을 반환하는 것으로 redirect 할 수 있음

# Redirect

## ■ Redirect 흐름



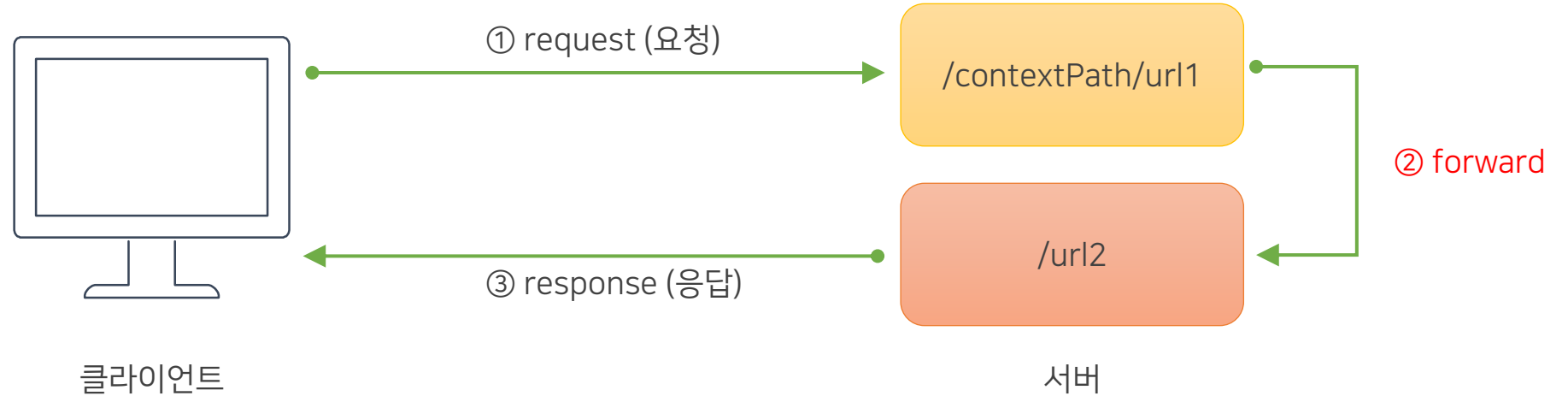
# Forward

## ■ Forward

- 기존 요청을 그대로 다시 사용하는 이동 방식
  - 이동할 때 데이터를 전달할 수 있음
- 클라이언트의 요청을 서버 내 다른 곳으로 이동하는 서버 내부의 이동 방식
- 이동할 경로는 ContextPath를 제외하고 서버 내부 경로만 작성해야 함
- 서버 내부에서 이동하는 방식이기 때문에 클라이언트는 URL을 통해서 forward한 경로를 확인할 수 없음
- DB 변경이 없는 작업이나 단순 이동의 경우 Forward를 진행
  - SELECT 이후에는 Forward 방식으로 이동해야 함
- Spring의 기본 이동 방식임

# Forward

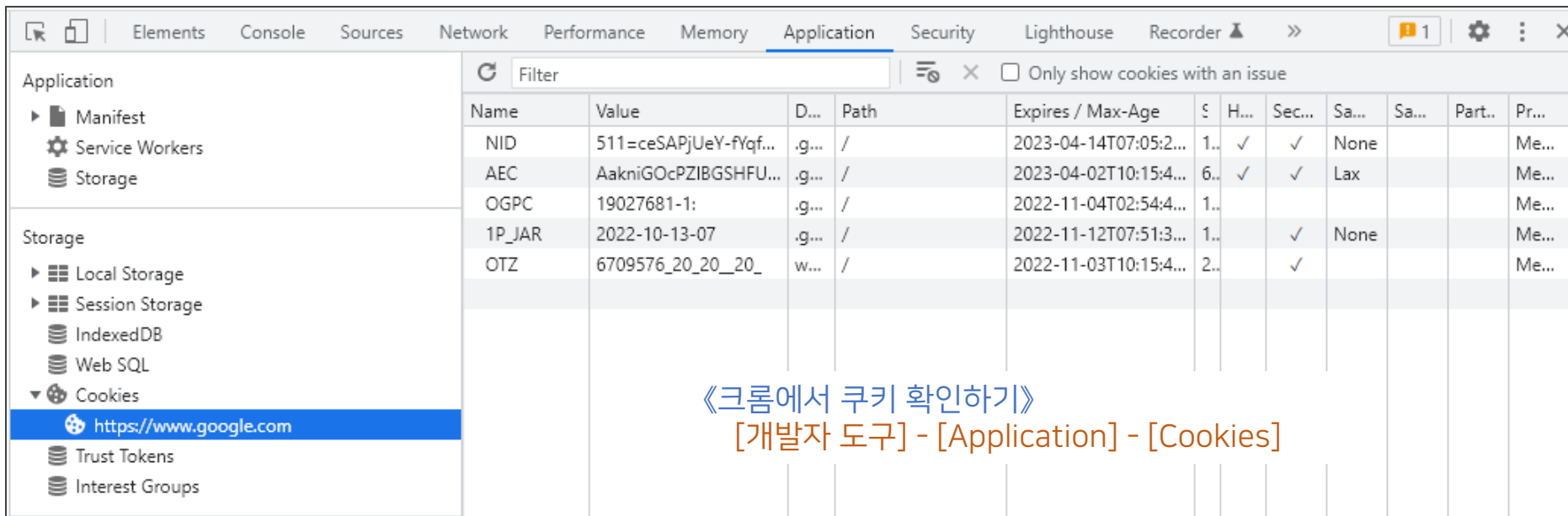
## ■ Forward 흐름



# Cookie

## ■ Cookie

- 클라이언트 측 브라우저에 저장되는 정보를 의미함
- 서버에서 관리하지 않기 때문에 보안이 취약함
- 웹 페이지들에서 참조해야 하는 공유 정보를 저장해 두고 사용하기 위해 쿠키를 활용할 수 있음
  - 오늘 더 이상 열지 않기, 아이디 저장 등에서 활용
- 4KB 용량 제한이 있음
- 웹 브라우저에서 사용 유무를 설정할 수 있음



The screenshot shows the Chrome DevTools Application tab with the 'Cookies' section expanded for the URL <https://www.google.com>. The table below lists the cookies found.

| Name   | Value                 | D...  | Path | Expires / Max-Age     | §   | H... | Sec... | Sa... | Sa... | Part.. | Pr... |
|--------|-----------------------|-------|------|-----------------------|-----|------|--------|-------|-------|--------|-------|
| NID    | 511=ceSAPjUeY-fYqf... | .g... | /    | 2023-04-14T07:05:2... | 1.. | ✓    | ✓      | None  |       |        | Me... |
| AEC    | AakniGOcPZIBGSHFU...  | .g... | /    | 2023-04-02T10:15:4... | 6.. | ✓    | ✓      | Lax   |       |        | Me... |
| OGPC   | 19027681-1:           | .g... | /    | 2022-11-04T02:54:4... | 1.. |      |        |       |       |        | Me... |
| 1P_JAR | 2022-10-13-07         | .g... | /    | 2022-11-12T07:51:3... | 1.. |      | ✓      | None  |       |        | Me... |
| OTZ    | 6709576_20_20_20_     | w...  | /    | 2022-11-03T10:15:4... | 2.. |      | ✓      |       |       |        | Me... |

《크롬에서 쿠키 확인하기》  
[개발자 도구] - [Application] - [Cookies]

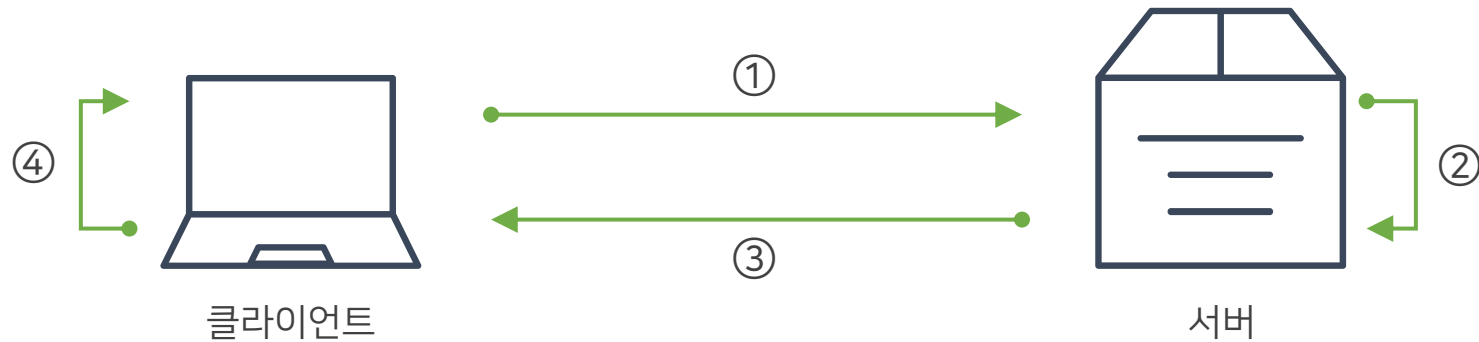
## ■ Session

- 서버 메모리에 저장되는 정보를 의미함
- 서버에서 관리하므로 보안이 우수함
- 웹 페이지들에서 참조해야 하는 공유 정보를 저장해 두고 사용하기 위해 세션을 활용할 수 있음
  - 로그인, 장바구니 등에서 활용
- 세션에 저장된 정보는 유효 시간을 가짐 (기본 30분)
- 브라우저 당 하나의 세션이 생성됨 (세션아이디를 통해서 각 세션을 구분함)
- HttpSession 인터페이스가 세션 처리를 담당함

# Session

## ■ Session 실행 과정

- ① 브라우저로 특정 사이트에 접속함
- ② 접속한 브라우저의 세션 객체가 생성됨
- ③ 생성된 세션 객체의 아이디(ID)를 접속한 브라우저로 응답해 줌
- ④ 브라우저는 서버로부터 받은 세션 아이디(Session ID)를 세션 쿠키 형태로 저장 (쿠키명 : JSESSIONID)



# Servlet Attribute

## ■ Servlet Attribute

- 서블릿 속성
- 웹 프로그램 실행 시 데이터를 서블릿 관련 객체에 저장하는 방법
- 속성으로 저장해 둔 데이터를 서블릿이나 JSP들이 공유해서 사용할 수 있음

## ■ Attribute 관련 메소드

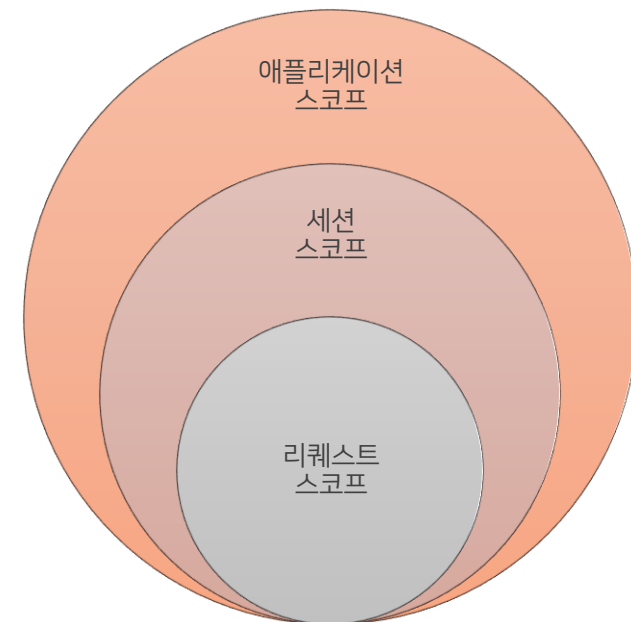
| 메소드   | 역할  |
|---|---|
| <code>void setAttribute(String name, Object obj)</code> | 지정한 name 을 가진 속성으로 obj 값을 저장함                                 |
| <code>Object getAttribute(String name)</code>           | 지정한 name 을 가진 속성 값을 반환함<br>Object 타입으로 반환하기 때문에 캐스팅이 필요할 수 있음 |
| <code>void removeAttribute(String name)</code>          | 지정한 name 을 가진 속성을 삭제함   |



# Scope

## ■ Scope

- 서블릿 속성(Attribute)에 접근할 수 있는 접근 범위를 의미함
- 스코프가 다른 경우에는 해당 속성에 접근할 수 없음
- 동일한 이름의 속성을 서로 다른 스코프에 저장할 수 있음
- **스코프간 우선 순위**
  - 리퀘스트 스코프 > 세션 스코프 > 애플리케이션 스코프
- **스코프별 접근 범위**
  - 리퀘스트 스코프 : 하나의 요청과 응답 사이클에서만 접근이 가능함
  - 세션 스코프 : 세션이 만료되기 전 브라우저를 닫기 전까지 접근이 가능함
  - 애플리케이션 스코프 : 웹 애플리케이션이 종료되기 전까지 접근이 가능함
- **스코프별 저장 정보**
  - 리퀘스트 스코프 : 단순 요청 정보
  - 세션 스코프 : 로그인 회원 정보, 장바구니 정보
  - 애플리케이션 스코프 : 총 방문자 수



```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r < n; r++)
      if (n in t && t[r] === e) return r;
  }
}

```

## 2. JSP

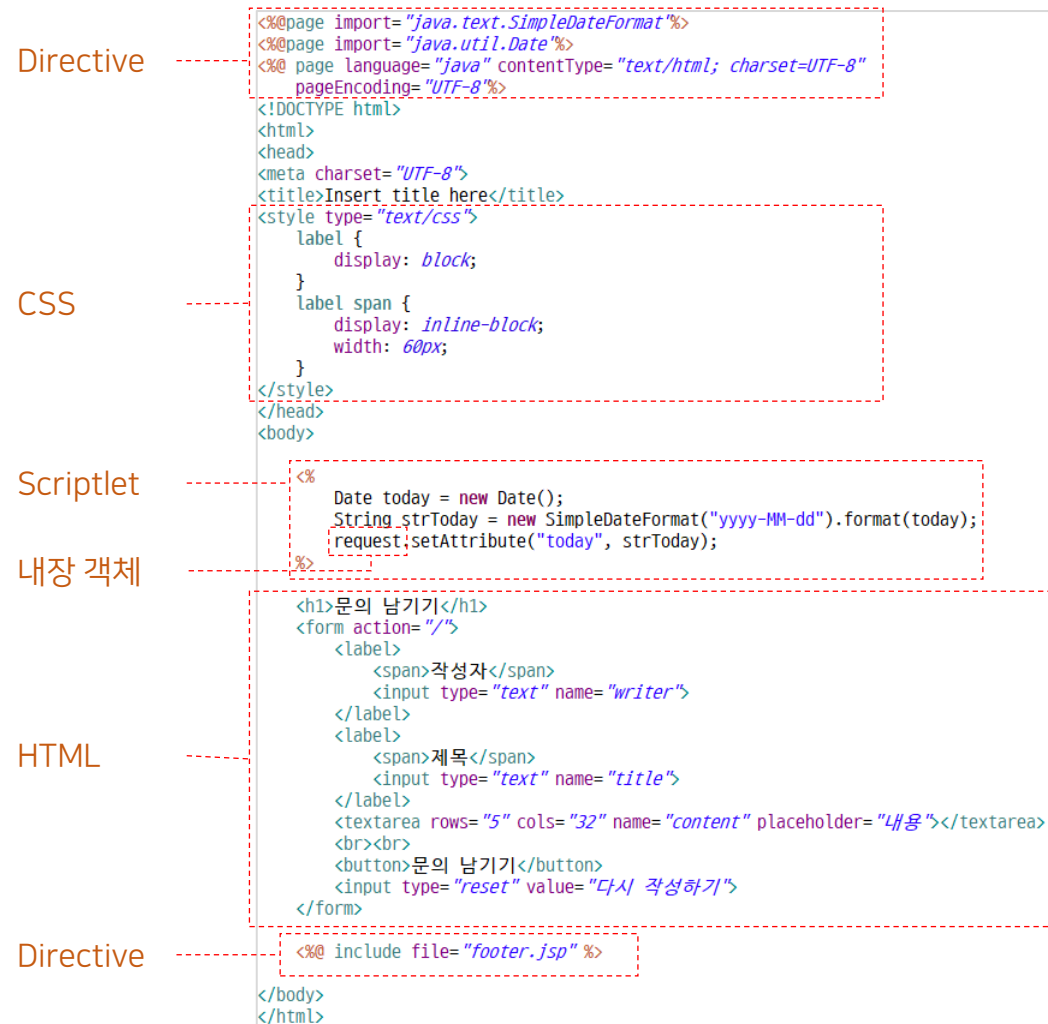
## ■ JSP

- Jakarta Server Page (구 Java Server Page)
- 사용자 화면(View)을 만드는 구성 요소
- HTML 템플릿 기반으로 구성되어 있으나 Java 를 함께 사용할 수 있음
- Servlet의 경우 Java 코드로 HTML 화면을 만들어 사용이 불편하였으나 JSP의 경우 HTML 내부에 Java 코드를 넣는 방식을 채택하여 보다 편리한 웹 개발이 가능해짐
- 구성 요소
  - HTML / CSS / Javascript
  - JSP 액션 태그
  - JSP 스크립트 요소

# JSP 구성 요소

## ■ JSP 페이지 구성 요소

- 디렉티브 태그(Directive Tag)
- 스크립트 요소(Script Element)
  - 주석문 (Comment)
  - 표현식 (Expression)
  - 선언식 (Declaration)
  - 스크립트릿 (Scriptlet)
- 내장 객체
- 액션 태그(Action Tag)
- 커스텀 태그(Custom Tag)



# page directive

## ■ page directive 속성

| 속성           | 기본값        | 작성방법                                | 의미  |
|--------------|------------|-------------------------------------|---|
| info         | 없음         | <%@ page info="메인페이지" %>            | 페이지를 설명하는 문자열                                   |
| language     | java       | <%@ page language="java" %>         | 사용할 언어  |
| contentType  | text/html  | <%@ page contentType="text/html" %> | 페이지 출력 형식                                       |
| pageEncoding | ISO-8859-1 | <%@ page pageEncoding="UTF-8" %>    | 페이지 문자열 인코딩                                     |
| import       | 없음         | <%@ page import="java.util.Date" %> | 다른 패키지의 클래스 import                              |
| session      | true       | <%@ page session="true" %>          | HttpSession 객체 사용 유무                            |
| errorPage    | 없음         | <%@ page errorPage="error.jsp" %>   | 예외 처리 담당 페이지                                    |
| isErrorPage  | false      | <%@ page isErrorPage="false" %>     | 현재 JSP 페이지의 예외 처리 담당 유무                         |
| isELIgnored  | true       | <%@ page isELIgnored="true" %>      | EL 사용 유무  |
| buffer       | 8KB        | <%@ page buffer="8KB" %>            | 사용할 버퍼 크기                                       |
| autoFlush    | true       | <%@ page autoFlush="true" %>        | 페이지 내용 출력 전에 버퍼가 다 채워질 경우<br>자동으로 버퍼를 비울 것인지 여부 |

# include directive

## ■ include directive

- 공통으로 사용하는 JSP 페이지를 다른 JSP 페이지에 추가할 때 사용함
- 페이지 상단(header 영역)이나 페이지 하단/footer 영역)의 공통 화면을 만들 때 주로 사용
- JSP 페이지의 재사용성이 높고 유지관리가 용이함
- 정적인 페이지를 포함하는 방식으로 동작함
  - 페이지마다 값이 바뀌는 변수를 포함할 수 없음
- 형식
  - `<%@ include file="포함할 JSP" %>`

# taglib directive

## ■ taglib directive

- JSTL을 사용할 때 추가하는 directive
- 사용하려는 태그 라이브러리에 따라서 형식이 다름
- 형식 (CORE LIBRARY 의 경우)
  - Tomcat 9.0 이하  
`<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
  - Tomcat 10.1 이상  
`<%@ taglib prefix="c" uri="jakarta.tags.core" %>`

# 스크립트 요소

## ■ 선언부

- 전역 변수 선언 및 메소드 정의
- `<%! void method() { } %>`

## ■ 표현식

- 변수 및 메소드 호출 결과
- `<%= age %>`
- `<%= getArea() %>`

## ■ 주석문

- 주석 처리
- 페이지 소스 보기를 통한 주석의 확인이 불가능함
- `<%-- 주석 --%>`

## ■ 스크립트릿

- 일반 자바 코드
- 지역 변수 선언, 제어문 등 모든 자바 코드
- `<% if(age >= 20) { %>`



# JSP 내장 객체

## ■ JSP 내장 객체

- JSP가 Servlet으로 변환될 때 컨테이너가 자동으로 생성시키는 서블릿 객체를 의미함
- Servlet을 사용할 때 자주 사용하던 객체 9개를 컨테이너가 자동으로 생성함
- 개발자는 JSP 내장 객체를 일일이 생성할 필요 없이 곧바로 사용할 수 있음

## ■ JSP 내장 객체 종류

| 내장 객체       | 서블릿 타입                                 | 의미                   | 저장소 역할 |
|-------------|--|----------------------|--------|
| request     | javax.servlet.http.HttpServletRequest  | 클라이언트 요청 정보 저장       | ○      |
| response    | javax.servlet.http.HttpServletResponse | 서버 응답 정보             |        |
| out         | javax.servlet.jsp.JspWriter            | JSP 페이지에 결과 출력       |        |
| session     | javax.servlet.http.HttpSession         | 세션 정보 저장             | ○      |
| application | javax.servlet.ServletContext           | 컨텍스트 정보 저장           | ○      |
| pageContext | javax.servlet.jsp.PageContext          | JSP 페이지에 대한 정보 저장    | ○      |
| page        | java.lang.Object                       | JSP 페이지의 서블릿 인스턴스 저장 |        |
| config      | javax.servlet.ServletConfig            | JSP 페이지의 설정 정보 저장    |        |
| exception   | java.lang.Exception                    | 예외 처리                |        |

# JSP 저장 영역

## ■ JSP 저장 영역

- JSP 내장 객체 중에서 데이터를 저장할 수 있는 4개 객체를 의미함
- 데이터를 속성(Attribute)으로 저장할 수 있음

## ■ JSP 저장 영역과 스코프

| 저장 영역       | 서블릿 타입             | 스코프              |
|-------------|--------------------|------------------|
| pageContext | this               | 동일한 JSP 내에서 공유   |
| request     | HttpServletRequest | 한 번의 요청에 대해서 공유  |
| session     | HttpSession        | 같은 브라우저 내에서 공유   |
| application | ServletContext     | 같은 애플리케이션 내에서 공유 |

# JSP 액션 태그

## ■ JSP 액션 태그

- 스크립트릿(Scriptlet)에 작성하는 자바 코드 대신 사용할 수 있는 JSP 태그
- 웹 디자이너/퍼블리셔들의 입장에서는 좀 더 쉬운 형태로 자바 기능을 사용할 수 있음
- 태그 이름의 prefix로 "jsp:"을 사용함

## ■ 주요 JSP 액션 태그

| 태그                | 의미  |
|-------------------|---|
| <jsp:include>     | 다른 JSP를 현재 JSP에 포함시키는 태그                        |
| <jsp:param>       | <jsp:include> 태그의 하위 태그로 포함할 JSP에 파라미터를 전달하는 태그 |
| <jsp:forward>     | RequestDispatcher 인터페이스의 forward 기능을 수행하는 태그    |
| <jsp:useBean>     | 객체 생성(new) 태그                                   |
| <jsp:setProperty> | Setter  |
| <jsp:getProperty> | Getter  |

# 표현 언어

## ■ 표현 언어

- EL, Expression Language
- 스크립트 요소 중 데이터 출력을 담당하던 표현식(<%=값%>)을 대체하기 위해 표현 언어가 등장함
- 형식
  - `${값}`

## ■ 표현 언어 주요 특징

- 기존의 표현식보다 쉬운 형태를 제공
- 자체 연산자를 사용할 수 있음
- 표현 언어 자체 내장 객체를 제공
- JSP 저장 영역에 저장된 속성(Attribute)을 사용할 수 있음
- 사용 가능 데이터
  - true/false, 정수, 실수, 문자열('hello', "hello"), 널(null)

# 표현 언어 연산자

## ■ 표현 언어 연산자

| 종류        | 연산자           | 의미                                  |
|-----------|---------------|-------------------------------------|
| 산술 연산자    | + - *         | 더하기 빼기 곱하기                          |
|           | / 또는 div      | 나누기                                 |
|           | % 또는 mod      | 나머지                                 |
| 관계 연산자    | > 또는 gt       | 크다                                  |
|           | >= 또는 ge      | 크거나 같다                              |
|           | < 또는 lt       | 작다                                  |
|           | <= 또는 le      | 작거나 같다                              |
|           | == 또는 eq      | 같다                                  |
|           | != 또는 ne      | 같지 않다                               |
| 논리 연산자    | && 또는 and     | 논리 곱                                |
|           | 또는 or         | 논리 합                                |
|           | ! 또는 not      | 논리 부정                               |
| 조건 연산자    | 조건식 ? 값1 : 값2 | 조건식이 true이면 값1, 조건식이 false이면 값2를 반환 |
| empty 연산자 | empty 값       | 값이 null 또는 빈 문자열이면 true를 반환         |

# 표현 언어 내장 객체

## ■ 표현 언어 내장 객체

| 종류      | 내장 객체            | 의미  |
|---------|------------------|---|
| 스코프     | pageScope        | pageContext에 저장된 속성을 참조할 때 사용 (우선순위 1)                      |
|         | requestScope     | request에 저장된 속성을 참조할 때 사용 (우선순위 2)                          |
|         | sessionScope     | session에 저장된 속성을 참조할 때 사용 (우선순위 3)                          |
|         | applicationScope | application에 저장된 속성을 참조할 때 사용 (우선순위 4)                      |
| 요청 파라미터 | param            | 요청 파라미터를 참조할 때 사용(변수)<br>request.getParameter() 대신 사용       |
|         | paramValues      | 요청 파라미터를 참조할 때 사용(배열)<br>request.getParameterValues() 대신 사용 |
| 헤더      | header           | 요청 헤더 이름을 단일 값으로 반환   |
|         | headerValues     | 요청 헤더 이름을 여러 값(배열)으로 반환                                     |
| 쿠키      | Cookies          | 쿠키 값을 반환  |
| JSP 페이지 | pageContext      | pageContext 객체를 참조할 때 사용                                    |

## ■ JSTL

- Java Standard Tag Library
- 가장 많이 사용되는 커스텀 태그들을 표준화하여 라이브러리로 제공하는 태그 라이브러리
- Tomcat 홈페이지([tomcat.apache.org](https://tomcat.apache.org))에서 라이브러리를 제공함

JSTL 1.2.5 라이브러리 다운로드  
<https://tomcat.apache.org/download-taglibs.cgi>

**Standard-1.2.5**

**Source Code Distributions**

- [Source README](#)
- [zip](#) ([pgp](#), [sha512](#))

**Jar Files**

- [Binary README](#)
- Impl:
  - [taglibs-standard-impl-1.2.5.jar](#) ([pgp](#), [sha512](#))
- Spec:
  - [taglibs-standard-spec-1.2.5.jar](#) ([pgp](#), [sha512](#))
- EL:
  - [taglibs-standard-jstlel-1.2.5.jar](#) ([pgp](#), [sha512](#))
- Compat:
  - [taglibs-standard-compat-1.2.5.jar](#) ([pgp](#), [sha512](#))

# JSTL 태그 라이브러리

## ■ JSTL 태그 라이브러리 종류

| 라이브러리  | 주요 기능          | 접두어(prefix) | URI  |
|--------|----------------|-------------|--|
| 코어     | 변수 처리, 제어문 처리  | c           | <a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a> (Tomcat 9.0 이하)<br><a href="http://jakarta.tags.core">jakarta.tags.core</a> (Tomcat 10.1 이상) |
| 형식     | 숫자 및 날짜 형식, 지역 | fmt         | <a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a><br><a href="http://jakarta.tags.fmt">jakarta.tags.fmt</a>                                      |
| XML    | XML 코어, XML 반환 | x           | <a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a><br><a href="http://jakarta.tags.fmt">jakarta.tags.fmt</a>                                      |
| 데이터베이스 | SQL            | sql         | <a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a><br><a href="http://jakarta.tags.fmt">jakarta.tags.fmt</a>                                      |
| 함수     | 컬렉션 처리, 문자열 처리 | fn          | <a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a><br><a href="http://jakarta.tags.functions">jakarta.tags.functions</a>              |



# JSTL 태그 라이브러리

## ■ taglib directive

| 라이브러리  | Tomcat 9.0 이하 디렉티브 (Tag Library 1.2)                                   |
|--------|--|
| 코어     | <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>       |
| 형식     | <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>      |
| XML    | <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>        |
| 데이터베이스 | <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>      |
| 함수     | <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %> |

| 라이브러리  | Tomcat 10.1 이상 디렉티브 (Tag Library 3.0)                  |
|--------|--|
| 코어     | <%@ taglib prefix="c" uri="jakarta.tags.core" %>       |
| 형식     | <%@ taglib prefix="fmt" uri="jakarta.tags.fmt" %>      |
| XML    | <%@ taglib prefix="x" uri="jakarta.tags.xml" %>        |
| 데이터베이스 | <%@ taglib prefix="sql" uri="jakarta.tags.sql" %>      |
| 함수     | <%@ taglib prefix="fn" uri="jakarta.tags.functions" %> |

# JSTL CORE LIBRARY

## ■ JSTL CORE LIBRARY

- Java 의 변수 선언, 제어문(조건문, 반복문) 등의 기능을 지원하는 태그
- Java 의 주요 기능을 태그로 대신할 수 있기 때문에 매우 유용한 라이브러리임
- Core 라이브러리를 사용하면 JSP 페이지 내에 존재하는 대부분의 Java 코드를 없앨 수 있음
  - 가능하다면 JSP 에는 Java 코드가 전혀 없는 것이 좋은 구성임

# JSTL CORE LIBRARY

## ■ JSTL CORE LIBRARY 태그 종류

| 종류  | 태그            | 의미   |
|-----|---------------|--|
| 변수  | <c:set>       | 변수 생성 (정확히는 속성(Attribute) 생성)              |
|     | <c:remove>    | 변수 제거                                      |
| 제어문 | <c:if>        | 조건문  |
|     | <c:choose>    | switch문(하위 태그로 <c:when>, <c:otherwise> 있음) |
|     | <c:forEach>   | 반복문  |
|     | <c:forTokens> | 구분자를 이용해 토큰 분리                             |
| URL | <c:import>    | 다른 자원을 JSP 페이지로 가져옴                        |
|     | <c:redirect>  | 리다이렉트(하위 태그로 <c:param> 사용 가능)              |
|     | <c:url>       | 지정한 URL로 이동(하위 태그로 <c:param> 사용 가능)        |
| 기타  | <c:catch>     | 예외 처리                                      |
|     | <c:out>       | JspWriter에 내용 처리 후 출력                      |

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e)
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "")
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > r ? Math.max(0, r + n
      if (n in t && t[n] === e) return n
  }
}

```

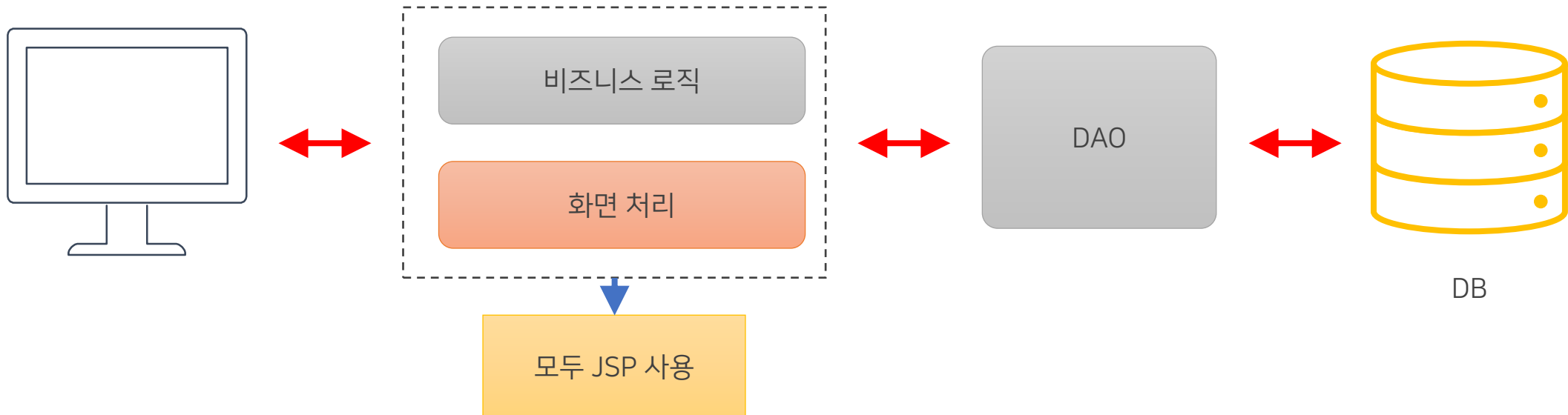
## 3. Spring MVC Project

# Model1

## ■ Model1

- 비즈니스 로직과 화면 처리를 모두 JSP로 처리하는 웹 애플리케이션 모델
- 기능 구현이 쉽고 편리함
- 디자이너와 개발자 모두 JSP를 사용하기 때문에 각각의 코드가 섞이므로 유지보수가 어렵고 코드 재사용이 어려움

## ■ Model1 동작 방식

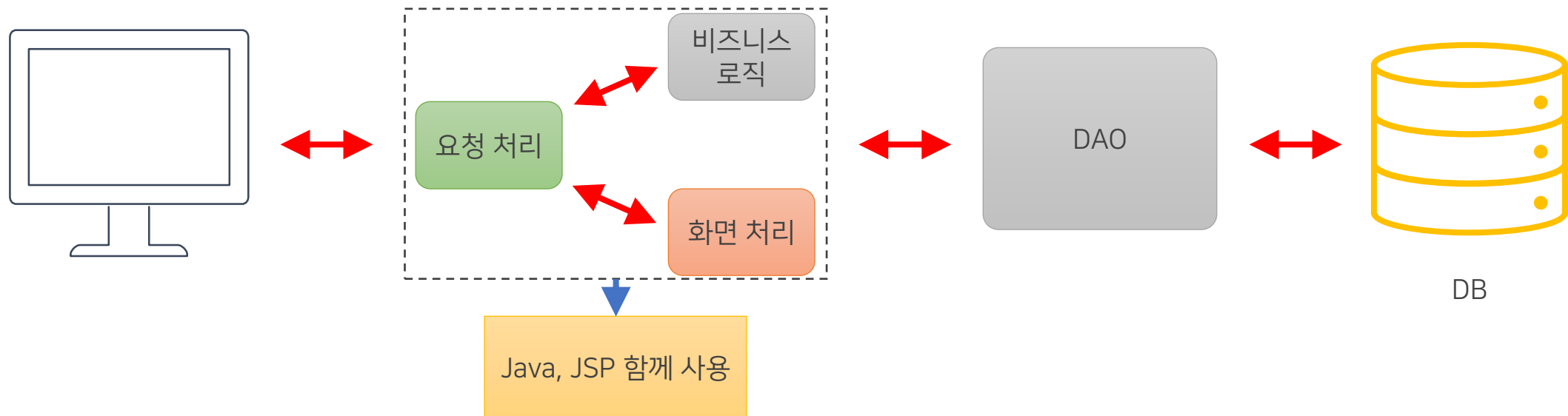


# Model2

## ■ Model2

- 비즈니스 로직과 화면 처리를 Java와 JSP로 분리하여 처리하는 웹 애플리케이션 모델
- 디자이너는 화면 기능을 구현하고, 개발자는 비즈니스 로직을 구현하기 때문에 업무 분할이 가능해짐
- 각 기능이 모듈화되어 처리되므로 개발 및 유지보수가 쉽고 코드 재사용이 가능함

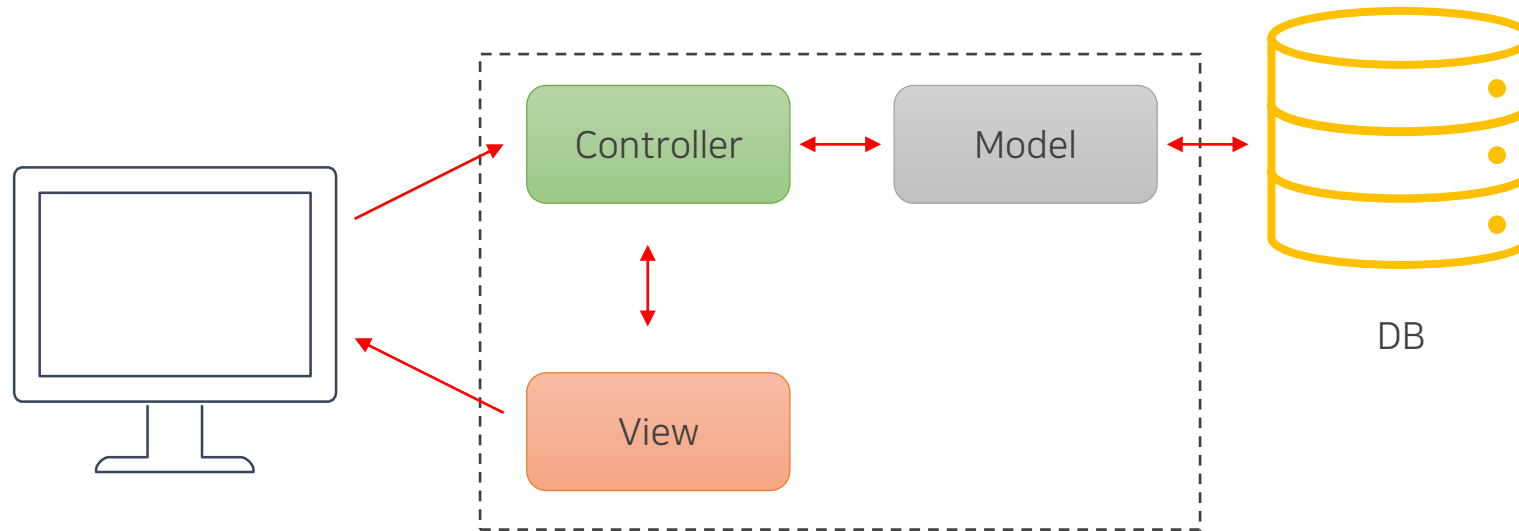
## ■ Model2 동작 방식



# MVC

## ■ MVC

- Model2 방식 중에서 가장 많이 사용하는 방식
- Model-View-Controller의 약자
- 화면 처리, 비즈니스 로직 처리, 요청 처리를 분리하여 프로그램을 개발하는 디자인 패턴



# MVC 구성 요소

## ■ Controller

- 사용자 요청 및 응답 처리, 흐름 제어
- Servlet을 사용

## ■ Model

- 비즈니스 로직 처리
- Java Class를 사용

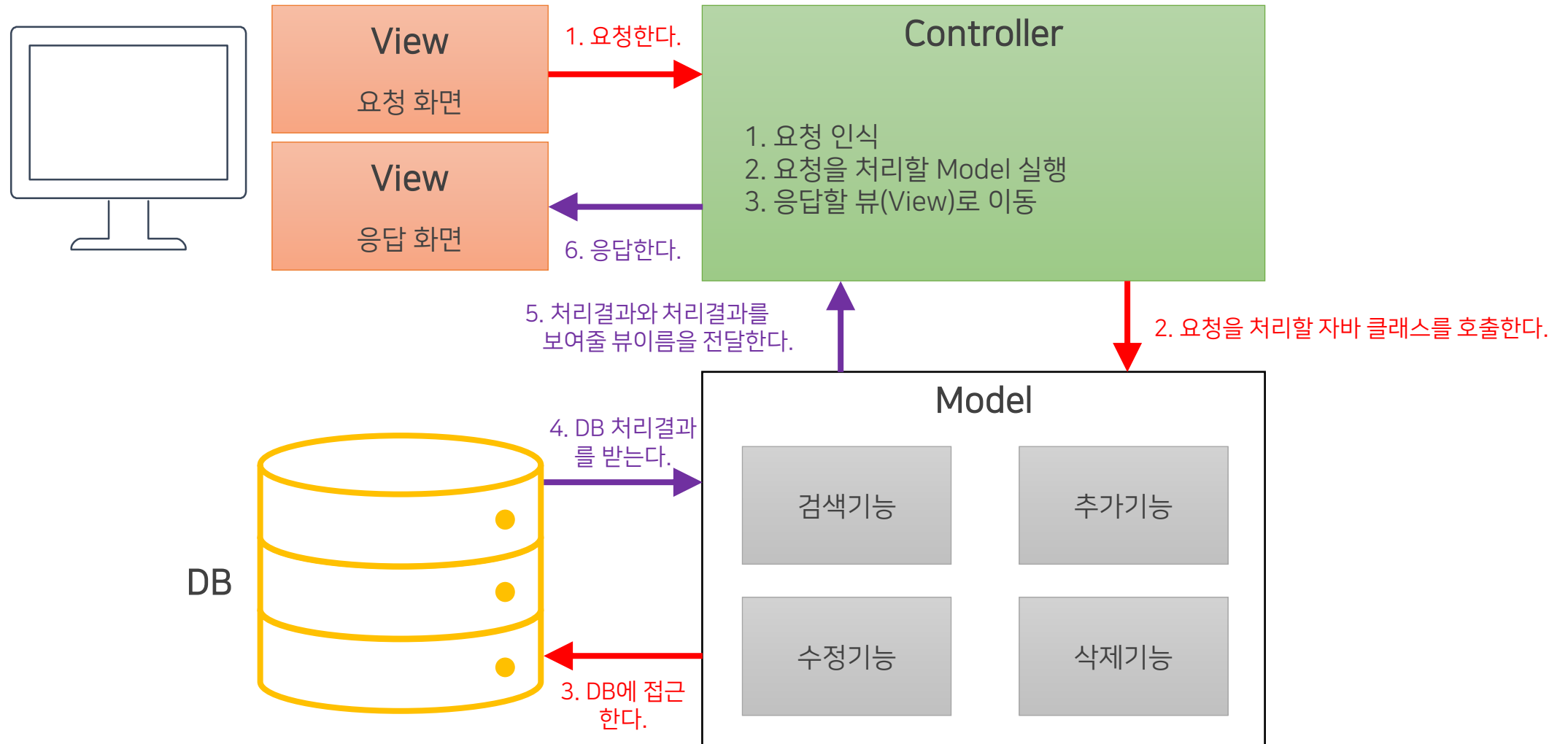
## ■ View

- 화면 처리
- JSP를 사용



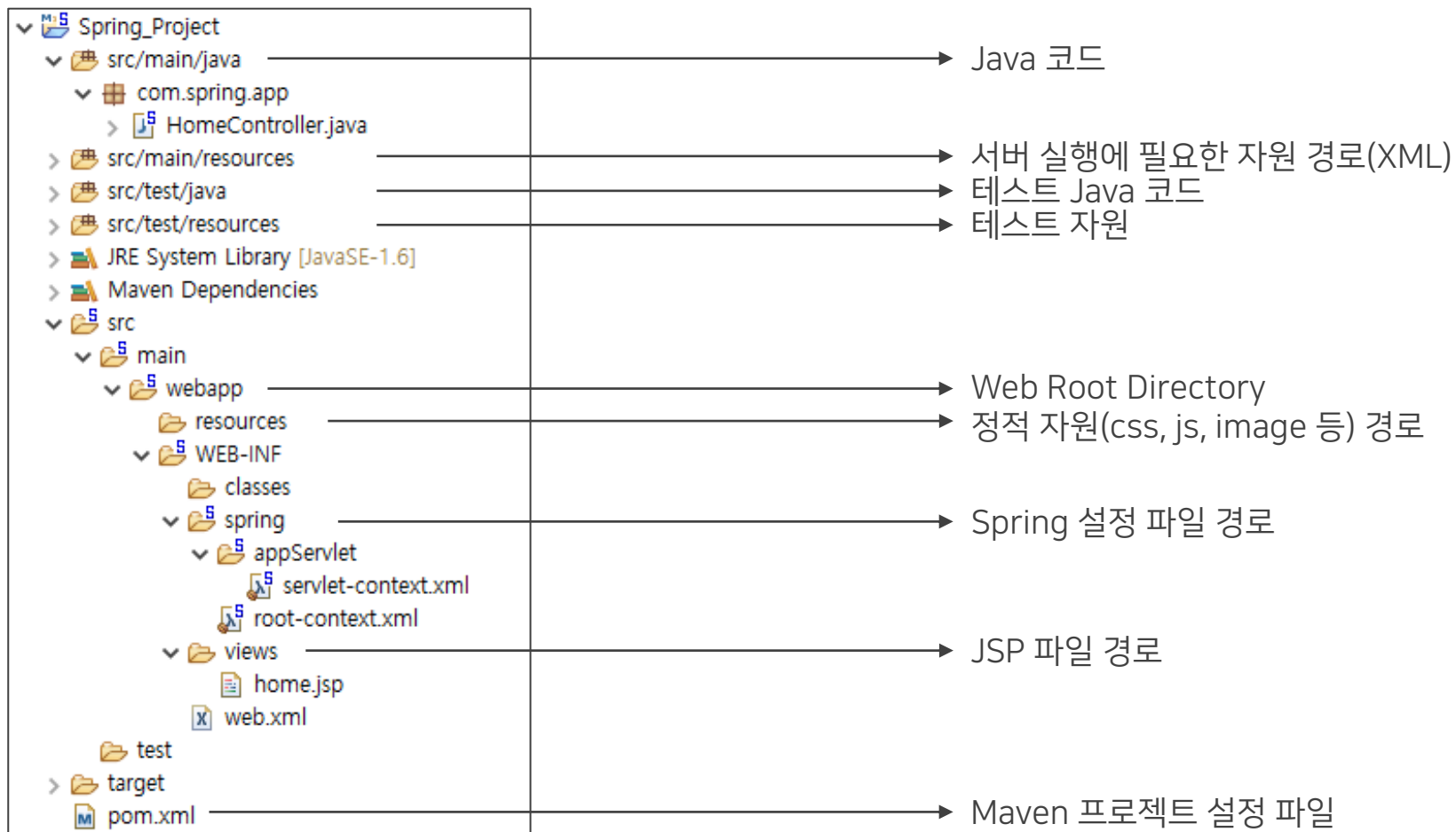
# MVC 동작

## ■ MVC 동작 순서

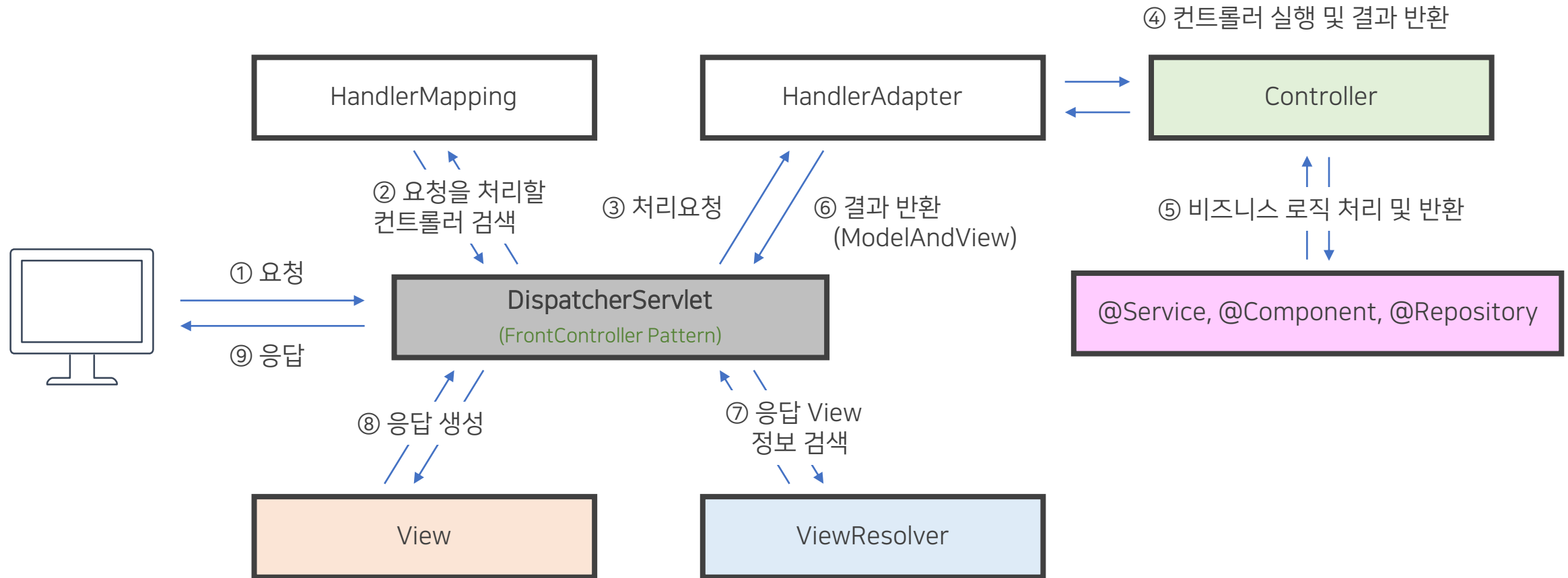


# Spring MVC

## ■ Spring MVC Project Template



# Spring MVC 동작 원리



# DispatcherServlet

## ■ DispatcherServlet

- 스프링 프레임워크의 핵심 서블릿
- /WEB-INF/web.xml 파일에 DispatcherServlet 관련 설정이 등록되어 있음
- 기본적으로 servlet-context.xml 설정 파일을 읽어서 스프링 프레임워크를 구동함

```
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

DispatcherServlet 을 의미함

DispatcherServlet은servlet-context.xml의  
내용을 이용해서 동작함

DispatcherServlet이 동작하는 경로는 "/"(ContextPath) 로 설정되어 있음  
ContextPath를 가진 모든 경로에서 DispatcherServlet이 동작함

# servlet-context.xml

## ■ <annotation-driven />

- @Controller Annotation을 활성화
- Spring MVC에서 Controller에게 요청하기 위해 필요한 HandlerMapping과 HandlerAdapter를 자동으로 bean으로 등록함

## ■ HandlerMapping

- @Controller가 적용된 클래스를 컨트롤러하고 함
- HandlerMapping은 요청을 처리할 컨트롤러를 @RequestMapping을 이용해서 검색함

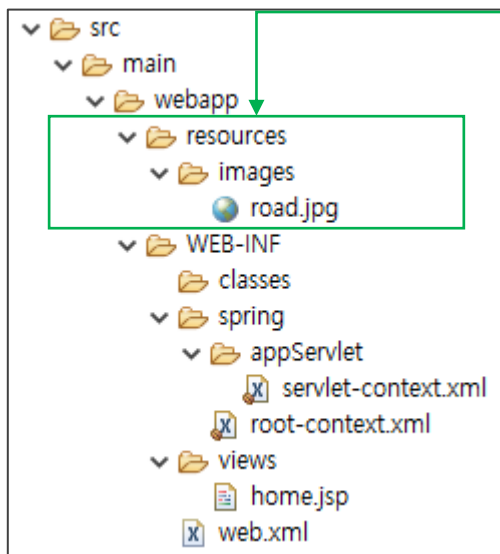
## ■ HandlerAdapter

- 요청을 처리할 컨트롤러의 메소드를 실행함
- 메소드 실행 결과를 ModelAndView 객체로 변환한 뒤 DispatcherServlet에게 반환함

# servlet-context.xml

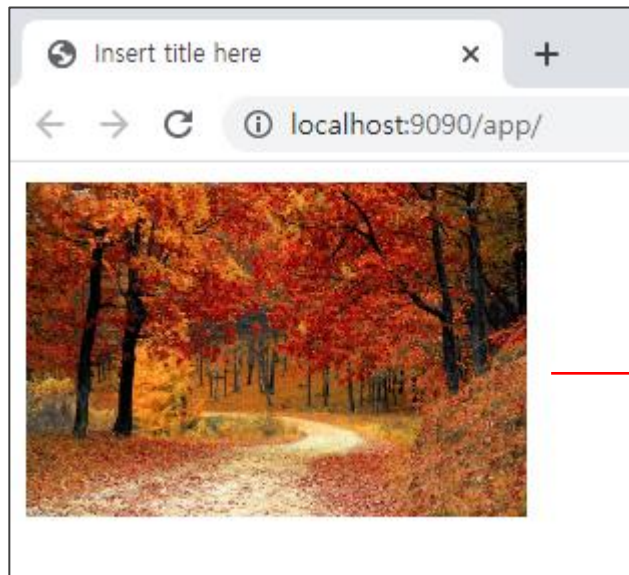
- `<resources mapping="/resources/**" location="/resources/">`
  - 웹 구성 요소 중에서 정적 자원들의 경로와 호출 방법을 기술
  - 정적 자원 : 멀티미디어 데이터(이미지, 오디오, 비디오 등), CSS, JS 등

```
<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources directory -->  
<resources mapping="/resources/**" location="/resources/" />
```



```

```



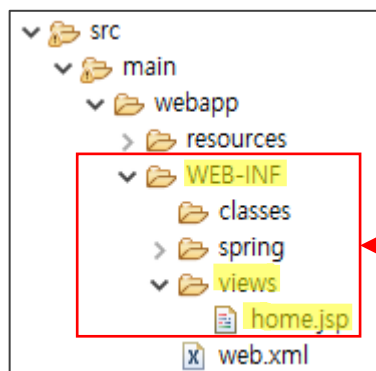
`<%=request.getContextPath()%>`는 ContextPath를 의미함

이미지 경로  
`http://localhost:9090/app/resources/images/road.jpg`

# servlet-context.xml

- <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  - 뷰 리졸버
  - HandlerAdapter에 의해서 반환된 ModelAndView 객체에 저장된 뷰 정보를 처리함

```
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->  
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
    <beans:property name="prefix" value="/WEB-INF/views/" />  
    <beans:property name="suffix" value=".jsp" />  
</beans:bean>
```



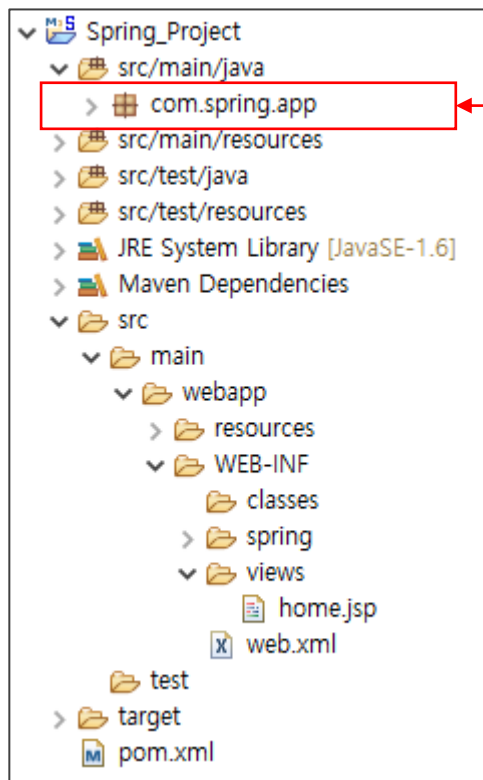
```
@RequestMapping(value="/", method=RequestMethod.GET)  
public String home() {  
    return "home";  
}
```

실제 응답할 뷰는 뷰 리졸버에 의해서 다음과 같이 처리됨  
"home" 앞에 "/WEB-INF/views/" 추가  
"home" 뒤에 ".jsp" 추가

```
return "/WEB-INF/views/home.jsp"
```

# servlet-context.xml

- `<context:component-scan base-package="com.spring.app" />`
  - base-package에 지정된 패키지에 저장된 클래스들을 스캔하고 자동으로 bean을 생성함
  - @Component, @Controller, @Service, @Repository 등의 Annotation이 추가된 클래스를 bean으로 등록함
  - Spring MVC Project의 top-level package와 동일해야 함



```
<context:component-scan base-package="com.spring.app" />
```

동일해야 함



# Spring MVC Annotation

## ■ 주요 Annotation

| Annotation        | 의미                                      | 사용        |
|-------------------|---|-----------|
| @Controller       | 스프링 MVC의 컨트롤러 객체임을 명시                   | 클래스       |
| @RequestMapping   | 특정 RequestURI에 매핑되는 컨트롤러(클래스)나 메소드임을 명시 | 클래스, 메소드  |
| @RequestParam     | 요청에서 특정 파라미터의 값을 가져올 때 사용               | 파라미터      |
| @RequestHeader    | 요청에서 특정 HTTP헤더 정보를 가져올 때 사용             | 파라미터      |
| @ModelAttribute   | 파라미터를 처리한 객체를 뷰까지 전달                    | 메소드, 파라미터 |
| @Component        | Bean으로 만들어 줘야 할 객체임을 명시                 | 클래스       |
| @Service          | 서비스 객체에 추가하는 @Component                 | 클래스       |
| @Repository       | DAO 객체에 추가하는 @Component                 | 클래스       |
| @PathVariable     | RequestURI에 포함된 값을 가져올 때 사용             | 파라미터      |
| @RequestBody      | 요청 본문에 포함된 데이터가 파라미터로 전달                | 파라미터      |
| @ResponseBody     | 반환 값이 HTTP 응답 메시지로 전송                   | 메소드, 리턴타입 |
| @CookieValue      | 쿠키가 존재하는 경우 쿠키 이름을 이용해서 쿠키 값을 가져올 때 사용  | 파라미터      |
| @SessionAttribute | Model의 정보를 세션에서 유지할 때 사용                | 클래스       |

# Controller

## ■ Controller

- 요청과 응답을 처리하는 클래스
- JSP/Servlet에서는 Servlet을 이용하여 구현함
- **@Controller**가 적용된 클래스
- 메소드 단위로 요청과 응답을 처리
- **@RequestMapping**을 이용해 요청 URL 및 요청 메소드 파악
  - @GetMapping, @PostMapping 등 요청 메소드에 따른 전용 Annotation이 존재함
- 반환 값을 이용해 응답을 처리

# Controller

## ■ Controller 예시

@Controller : 클래스를 컨트롤러로 인식시키는 Annotation

```
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );

        return "home";
    }
}
```

요청 주소가 Context Path 인 GET 방식의 요청을 처리함

model을 이용해 뷰로 전달(forward)할 속성(Attribute)을 저장

응답 결과가 나타날 뷰의 실제 경로와 이름은 "/WEB-INF/views/home.jsp"

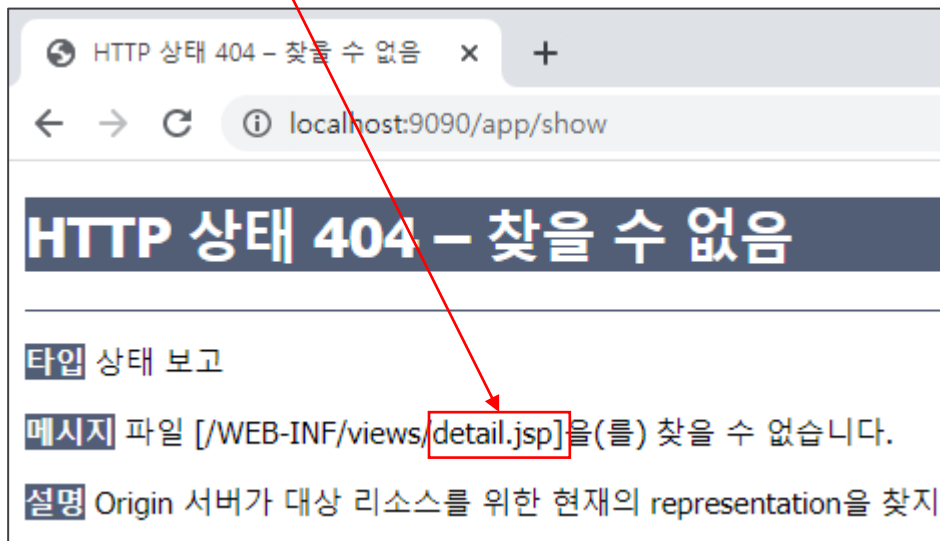
# Controller Method 반환 타입

## ■ Controller Method

- 컨트롤러는 하나의 요청을 하나의 메소드로 처리함
- Spring MVC Pattern에서는 메소드의 반환 타입을 String 또는 void로 설정할 수 있음

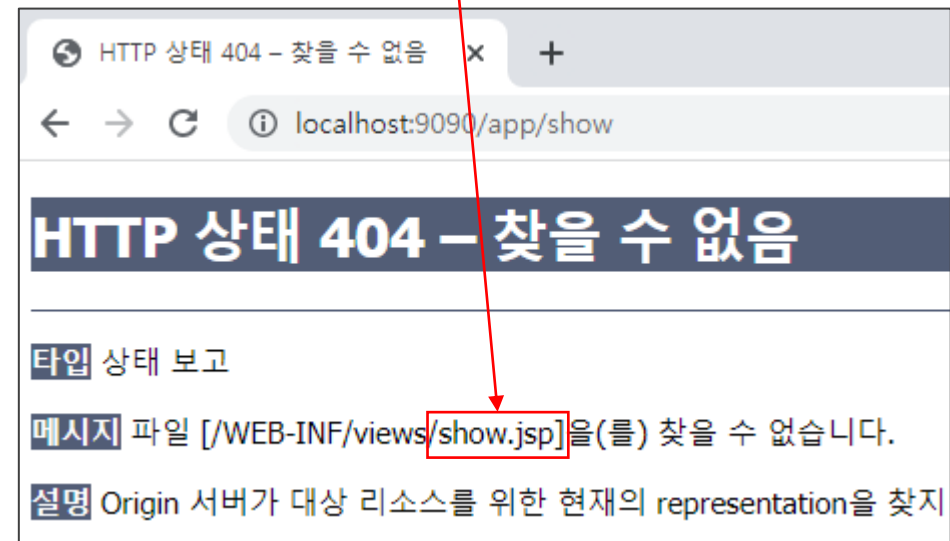
### ■ 반환 타입이 String인 경우 반환값을 뷰(JSP)의 이름으로 인식

```
@RequestMapping(value="show")  
public String method1() {  
    return "detail";  
}
```



### ■ 반환 타입이 void인 경우 매핑값을 뷰(JSP)의 이름으로 인식

```
@RequestMapping(value="show")  
public void method2() {  
}
```



# @RequestMapping

## ■ RequestMapping

- 요청 URL과 요청 메소드 등을 처리하는 Annotation
- @RequestMapping을 추가한 컨트롤러의 메소드는 요청을 처리하는 단위로 사용
- 요청 메소드에 따라서 @GetMapping, @PostMapping 등으로 바뀌어서 사용 가능함

| 구분           | 예시   | 의미                             |
|--------------|--|--------------------------------|
| value        | @RequestMapping(value="/")                   | "/" 요청                         |
|              | @RequestMapping(value={"/", "index"})        | "/"와 "index" 요청                |
|              | @RequestMapping(value="/member/*.do")        | "/member" 로 시작하고 ".do"로 끝나는 요청 |
| method       | @RequestMapping(method=RequestMethod.GET)    | GET 방식(조회)                     |
|              | @RequestMapping(method=RequestMethod.POST)   | POST 방식(삽입)                    |
|              | @RequestMapping(method=RequestMethod.PUT)    | PUT 방식(수정)                     |
|              | @RequestMapping(method=RequestMethod.DELETE) | DELETE 방식(삭제)                  |
| content type | @RequestMapping(consumes="application/json") | 요청 콘텐츠가 JSON임                  |
|              | @RequestMapping(produces="application/json") | 응답 콘텐츠가 JSON임                  |

# 기본 URL Pattern

## ■ web.xml의 기본 URL Pattern

- 기본 Pattern은 컨텍스트 패스(Context Path)로 설정된 상태
- 다른 Pattern으로 수정하면 전체 URL이 다르게 설정됨

```
<servlet-mapping>  
  <servlet-name>appServlet</servlet-name>  
  <url-pattern>/</url-pattern>  
</servlet-mapping>
```

http://localhost:8080/app

```
<servlet-mapping>  
  <servlet-name>appServlet</servlet-name>  
  <url-pattern>/home/*</url-pattern>  
</servlet-mapping>
```

http://localhost:8080/app/home



Context Path 는 /app 로 가정



# Encoding Filter

## ■ Encoding Filter

- Filter : 컨트롤러가 동작하기 이전에 먼저 동작하는 구성 요소
- web.xml에 CharacterEncodingFilter를 추가
- request.setCharacterEncoding("UTF-8")와 동일한 역할을 수행함

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

문자셋 인코딩

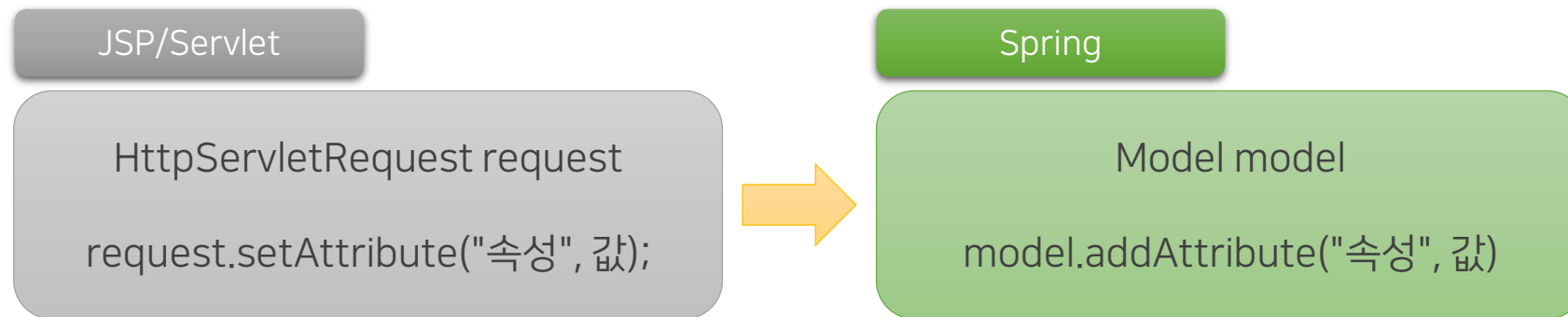
UTF-8

동일한 ContextPath를 가지는 모든 경로에 적용

# Model

## ■ Model

- 뷰가 응답 화면을 구성할 때 필요로하는 데이터를 전달하는 인터페이스
- forward 할 데이터를 저장할 때 사용함
- JSP/Servlet에서는 HttpServletRequest를 이용해서 데이터를 전달하였으나, 스프링에서는 Model을 이용하여 데이터를 전달함





# Model

## ■ Model 선언 및 활용

컨트롤러의 메소드 매개변수로 Model model 선언

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
    logger.info("Welcome home! The client locale is {}.", locale);

    Date date = new Date();
    DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);

    String formattedDate = dateFormat.format(date);

    model.addAttribute("serverTime", formattedDate );

    return "home";
}
```

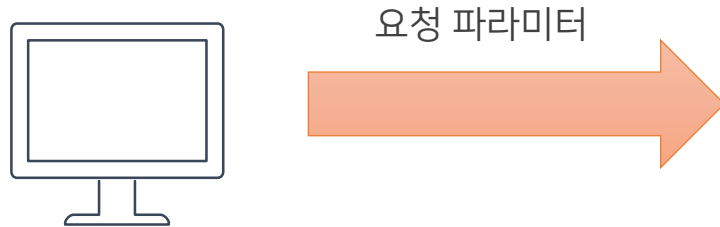
model을 이용해 뷰로 전달(forward)할 속성(Attribute)을 저장

home.jsp에서 \${serverTime}으로 전달된 값을 확인

# 요청 파라미터

## ■ 스프링의 요청 파라미터 처리 방식

- ① HttpServletRequest 인터페이스
- ② @RequestParam Annotation
- ③ 커맨드 객체



### Controller

#### JSP/Servlet

한 가지 방법만 지원함

HttpServletRequest

#### Spring

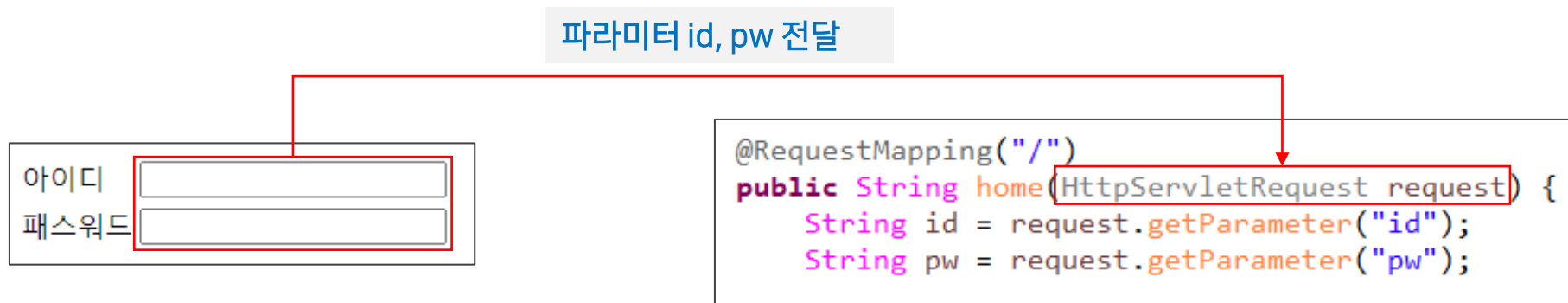
다양한 방법을 지원함

- HttpServletRequest
- @RequestParam
- MemberVo member

# HttpServletRequest

## ■ HttpServletRequest

- 요청을 처리하는 HttpServletRequest 인터페이스
- 단일 파라미터는 `String getParameter(String parameter)` 메소드를 이용해서 처리함
- 여러 파라미터는 `String[] getParameterValues(String parameter)` 메소드를 이용하여 처리함



# @RequestParam

## ■ @RequestParam

- 요청 파라미터를 인식한 뒤 변수에 저장하는 Annotation
- @RequestParam Annotation을 생략하면 스프링이 요청 파라미터 이름을 추론하여 변수에 저장함

## ■ @RequestParam 선택 요소 (Optional Element)

- name : 파라미터 이름 작성
- value : 파라미터 이름 name 의 별명 (Alias)
- required : 필수 여부 지정 (디폴트 true)
- defaultValue : 파라미터가 없는 경우 사용할 문자열 기본값

# @RequestParam

## ■ @RequestParam 활용

```
<a href="${contextPath}/read?pid=admin&week=sat">
```

파라미터 pid, week 전달

```
@RequestMapping(value="read", method=RequestMethod.GET)
public String read(
    @RequestParam(value="pid") String pid,
    @RequestParam(value="week", required=false, defaultValue="mon") String week) {
```

파라미터 pid는 필수이므로 전달되지 않는다면 Exception 발생  
파라미터 week는 필수가 아니며(required=false) 만약 전달되지 않는다면 week=mon으로 처리(defaultValue)

# 커맨드 객체

## ■ 커맨드 객체

- 요청 파라미터를 커맨드 객체의 필드로 전달 받는 방식
- 여러 개의 요청 파라미터를 전송하는 경우 일일이 처리하는 것보다 편리하게 개선된 방식임
- 내부적으로 커맨드 객체의 Setter를 이용해서 파라미터를 저장하므로 반드시 Setter를 작성해 놓아야 함

|     |                      |
|-----|----------------------|
| 아이디 | <input type="text"/> |
| 이름  | <input type="text"/> |
| 연락처 | <input type="text"/> |

```
@RequestMapping(value="register", method=RequestMethod.POST)
public String register(
    @RequestParam("id") String id,
    @RequestParam("name") String name,
    @RequestParam("tel") String tel) {
```

커맨드 객체 방식으로 개선

```
@RequestMapping(value="register", method=RequestMethod.POST)
public String register(MemberVO vo) {
```

# 커맨드 객체

## ■ 커맨드 객체의 특징

- 스프링은 커맨드 객체를 자동으로 뷰까지 전달함 (forward)
- 커맨드 객체의 클래스명을 CamelCase로 수정한 이름으로 전달함

|     |  |
|-----|--|
| 아이디 | <input type="text" value="admin"/>       |
| 이름  | <input type="text" value="관리자"/>         |
| 연락처 | <input type="text" value="02-500-1000"/> |
|     | <input type="button" value="전송"/>        |



```
@RequestMapping(value="register", method=RequestMethod.POST)
public String register(MemberVO vo) {
    return "home";
}
```

전달될 때 클래스명(MemberVO)을 CamelCase로 변환한 이름을 사용함  
(객체명 vo를 사용하는 것이 아님을 주의할 것!)

```
<div>아이디 ${memberVO.id}</div>
<div>이름 ${memberVO.name}</div>
<div>연락처 ${memberVO.tel}</div>
```

home.jsp



아이디 admin  
이름 관리자  
연락처 02-500-1000

# @ModelAttribute

## ■ @ModelAttribute

- 스프링에 의해서 자동으로 뷰까지 전달되는 커맨드 객체의 이름을 변경하고자 하는 경우에 사용

|                                   |  |
|-----------------------------------|--|
| 아이디                               | <input type="text" value="admin"/>       |
| 이름                                | <input type="text" value="관리자"/>         |
| 연락처                               | <input type="text" value="02-500-1000"/> |
| <input type="button" value="전송"/> |  |



```
@RequestMapping(value="register", method=RequestMethod.POST)
public String register(@ModelAttribute("member") MemberVO vo) {
    return "home";
}
```

클래스명 MemberVO를 CamelCase로 변경한 memberVO 대신 member를 사용!

```
<div>아이디 ${member.id}</div>
<div>이름 ${member.name}</div>
<div>연락처 ${member.tel}</div>
```

home.jsp



아이디 admin  
이름 관리자  
연락처 02-500-1000



# Redirect

## ■ Redirect

- 컨트롤러의 반환값이 "redirect:"으로 시작하면 리다이렉트로 이동함
- "redirect:" 뒤에는 새로운 요청 URL이 오기 때문에 특정 URLMapping값을 작성해야 함 (뷰 경로와 이름을 작성하는 것이 아님!)

```
@RequestMapping(value="register", method=RequestMethod.POST)
public String register(@ModelAttribute("member") MemberVO vo) {
    return "redirect:/view";
}

@RequestMapping(value="view", method=RequestMethod.GET)
public String view() {
    return "detail";
}
```

URLMapping "view"로 리다이렉트

리다이렉트는 기존 요청을 유지하지 않음

detail.jsp

```
<div>아이디 ${member.id}</div>
<div>이름 ${member.name}</div>
<div>연락처 ${member.tel}</div>
```



아이디  
이름  
연락처