

Servlet & JSP 기초

- 강사 김현오



목차

1. Web 기본 개념
2. Servlet
3. 세션 관리
4. JSP
5. JSTL
6. Servlet 활용기술
7. 웹 프로그래밍 전략



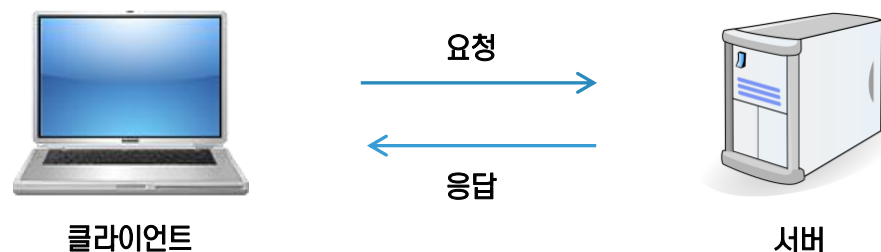
1. Web 기본 개념

1.1 Web 기본 개념

- HTTP 프로토콜
- HTTP Method
- URL
- TCP 포트
- 웹서버
- CGI

HTTP 프로토콜

- ✓ 웹에서만 사용하는 프로토콜
- ✓ TCP/IP를 기반으로 하여, TCP/IP를 이용해서 한 지점에서 다른 지점으로 요청과 응답을 전송
 - TCP : 한쪽 노드에서 다른 쪽 노드로 파일을 보내기 위한 프로토콜
 - IP : 한 호스트에서 목적지 호스트까지 패킷을 옮기고 이동하기 위한 기반 프로토콜
- ✓ HTTP의 구조는 “요청/응답의 끊임없는 주고 받음” 이라고 말할 수 있다.
- ✓ 요청(request)의 주요 구성 요소
 - HTTP 메소드
 - 접근하고자 하는 페이지(URL)
 - 폼 파라미터
- ✓ 응답(response)의 주요 구성 요소
 - 상태코드(요청이 성공했는지 아닌지 등등)
 - 콘텐츠 타입(텍스트, 그림, HTML 등)
 - 콘텐츠(HTML 코드, 이미지 등)



HTTP Method

✓ HTTP Method는 HTTP 프로토콜을 사용하여 서버로 요청을 보내는 방법이다.

- GET
 - URL을 이용하여 자원을 요청
- POST
 - Request에 첨부한 Body 정보를 서버로 전송
- HEAD
 - 헤더정보만 요청. 요청한 URL로부터 정보는 빼고 헤더 정보만 가져옴
- TRACE
 - 요청한 메시지의 루프백 테스트를 요청. 서버에서 무엇을 받았는지 알고 싶을 때 테스트 목적으로 사용
- PUT
 - Body정보를 요청한 URL로 저장
- DELETE
 - URL에 해당하는 자원을 삭제
- OPTION
 - 요청한 URL이 응답할 수 있는 HTTP 메소드가 무엇인지 요청

HTTP GET

- ✓ GET은 HTTP 메소드 중 가장 단순한 것, 단순히 서버에게 자원을 요청하는 것 뿐(HTML, 이미지, PDF 등등)
- ✓ GET의 데이터 전송방식은 브라우저 주소란에 기입하는 URL 뒤에 붙이는 방식
- ✓ GET으로 보낼 수 있는 글자 수는 제한이 있음

HTTP 메소드

```
GET /select/selectBeerTaste.jsp?color=dark&taste=malty
Accept:image/webp,*/*;q=0.8
Accept-Encoding:gzip,deflate,sdch
Accept-Language:ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
Cache-Control:max-age=0
Connection:keep-alive
Host:ssstatic.naver.net
If-Modified-Since:Tue, 12 Oct 2010 09:50:21 GMT
If-None-Match:"218-4926867b4d140"
Referer:http://search.naver.com/search.naver?sm=tab_hy_top&where=nexsearch&ie=utf8&query=HTTP+%EA%B8%B0%EB%B0%98&x=0&y=0
User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.102 Safari/537.36
```

HTTP POST

- ✓ 서버에 좀더 복잡한 요청을 하기 위한 메소드
- ✓ 요청 파라미터에 대한 길이 제한이 없음

HTTP 메소드

웹서버 상 자원에 대한 경로

POST /advisor/selectBeerTaste.do

Accept:image/webp,*/*;q=0.8

Accept-Encoding:gzip,deflate,sdch

Accept-Language:ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4

Cache-Control:max-age=0

Connection:keep-alive

Host:sstatic.naver.net

If-Modified-Since:Tue, 12 Oct 2010 09:50:21 GMT

If-None-Match:"218-4926867b4d140"

Referer:http://search.naver.com/search.naver?sm=tab_hyt.top&where=nexearch&ie=utf8&query=HTTP+%EA%B8%B0%EB%B0%98&x=0&y=0

User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.102 Safari/537.36

color=dark&taste=malty

HTTP Response

✓ 서버가 클라이언트로 보내는 응답

✓ 헤더와 몸체로 구성

- 헤더에는 사용된 프로토콜이 뭔지, 보내준 요청이 성공했는지, 몸체에 포함된 콘텐츠의 종류는 무엇인지 등이 들어 있음
- 몸체에는 HTML과 같은 콘텐츠가 들어 있음

응답 헤더
→ Connection:Keep-Alive
Content-Length:20658
Content-Type:text/html;charset=UTF-8
Date:Sun, 02 Feb 2014 12:04:19 GMT
Keep-Alive:timeout=5, max=99
Server:Apache-Coyote/1.1

Content-Type의 값을 보통 MIME 타입이라고 한다.
MIME 타입이란 브라우저에게 화면에 보여줄 데이터 형식을 알려 주는 역할을 함

콘텐츠
→ <html>
...
</html>

URL

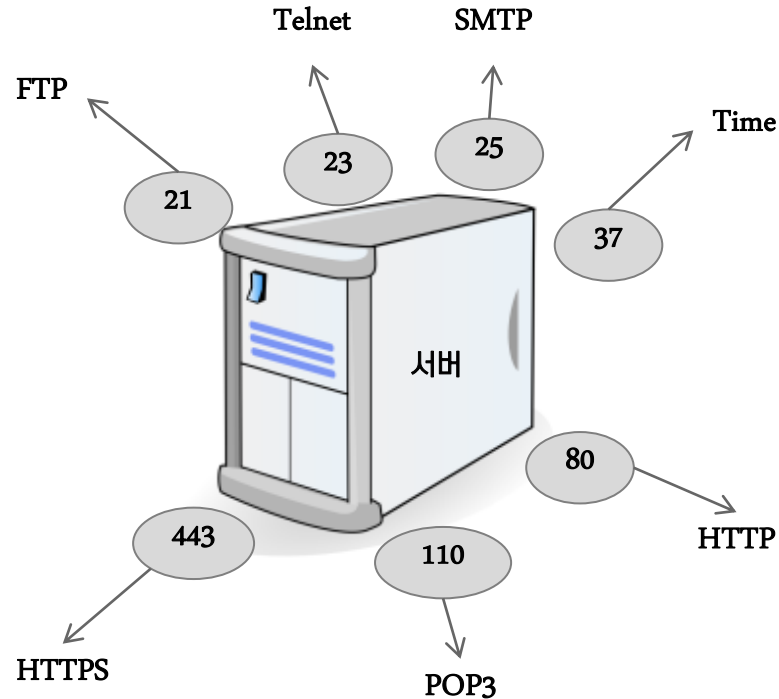
✓ 웹 상에 있는 모든 자원에 대한 주소

`http://www.namoosori.com:80/course/java/generic.html`

- `http://` : 서버와 대화하기 위하여 사용하는 커뮤니케이션 프로토콜
- `www.namoosori.com` : 서버 주소, 인터넷 상에 유일한 주소
- `80` : URL주소의 옵션, 보통 80포트는 생략함
- `/course/java/` : 서버에서 자원의 위치
- `generic.html` : 요청된 콘텐츠 이름

TCP 포트

- ✓ 서버 하드웨어 상에서 돌아가는 소프트웨어를 구별하기 위한 16비트 숫자 값
- ✓ 포트로 쓸 수 있는 값의 범위는 0~65535
- ✓ 포트는 서버 하드웨어 상에서 돌아가는 특정 소프트웨어에 대한 논리적인 연결을 나타냄
- ✓ 대개 0번에서 1023번까지 TCP 포트는 이미 널리 알려진 서비스를 위하여 예약되어 있음
- ✓ 포트란 것이 있기 때문에 서버는 클라이언트가 지금 어느 애플리케이션에 접속하기를 원하는지 알 수 있음



웹서버

- ✓ 클라이언트로 부터 요청을 받아 요청한 것을 넘겨주는 일을 하는 것
- ✓ 웹서버는 정적인 페이지만 서비스 함
- ✓ 웹서버는 단지 파일을 찾아서 있는 그대로 클라이언트에게 넘겨주는 일만 함
- ✓ 요청한 자료를 찾지 못 하면 404 Not Found 메시지를 보냄
- ✓ 동적인 콘텐츠 생성이 불가능 하다.
- ✓ Apache는 유명한 오픈 소스 웹서버

- ✓ CGI(Common Gateway Interface)는 초기 웹 프로그램 개발에 사용된 개발 방식이다.
- ✓ 동적인 콘텐츠 생성을 도와 주는 도우미 애플리케이션
- ✓ 대부분 CGI 프로그램은 펄(Perl) 스크립트로 작성한다. 그 외 C, 파이썬, PHP 등이 있다.

- ✓ CGI 방식의 문제점
 - 각각의 클라이언트 호출에 따른 개별 프로세스를 생성한다.
 - 접속자 만큼 프로세스가 생성되므로 시스템 부하가 많이 생긴다.
 - 윈도우에서 C언어 등으로 만들어진 CGI 애플리케이션은 리눅스에서 사용할 수 없다.



2. Servlet

2.1 Servlet 기본

2.2 Servlet 컨테이너

2.3 Request, Response

2.4 Redirect vs. Request Dispatch

- Servlet 개념
- Servlet 구현

Servlet 소개

✓ 정의

- 서블릿은 자바를 사용하여 웹 페이지를 동적으로 생성하는 서버 측 프로그램을 말하며 흔히 '서블릿'이라 한다.
- 자바를 이용하여 웹 애플리케이션을 만들기 위해 제공되는 기술이다.

✓ 장점

- 자바 언어를 이용해서 동적 웹 애플리케이션을 구현할 수 있다.
- 자바를 사용하기 때문에 플랫폼에 독립적이다.
- 프로세스 기반이 아닌 스레드 기반이기 때문에 좀 더 효과적으로 멀티 태스킹을 지원한다.

✓ 단점

- 화면에 표현될 HTML 코드를 프로그램적으로 작성해야 한다.
- 서비스하기 전에 반드시 컴파일을 해야 한다.
- 이러한 단점을 보완하기 위해서 JSP가 탄생했다.

✓ 웹 애플리케이션 개발시 서블릿을 사용하면 좋은점

- 콘텐츠와 비즈니스 로직을 분리할 수 있다.
- 컨트롤러와 뷰 역할 분담으로 인해 웹디자이너와 개발자간의 원활한 작업이 가능하다.
- 유지 보수가 용이하다.
- 기능 확장이 용이하다.
- JSP 페이지에 HTML, CSS, 자바스크립트와 자바 소스 코드가 뒤섞이는 스파게티 소스를 막을 수 있다.

Servlet 시작하기

✓ 서블릿을 개발하고 서비스하기 위해서는 JVM, 자바 서블릿 API, 서블릿 컨테이너, 웹 서버가 필요하다.

✓ 자바 서블릿 API

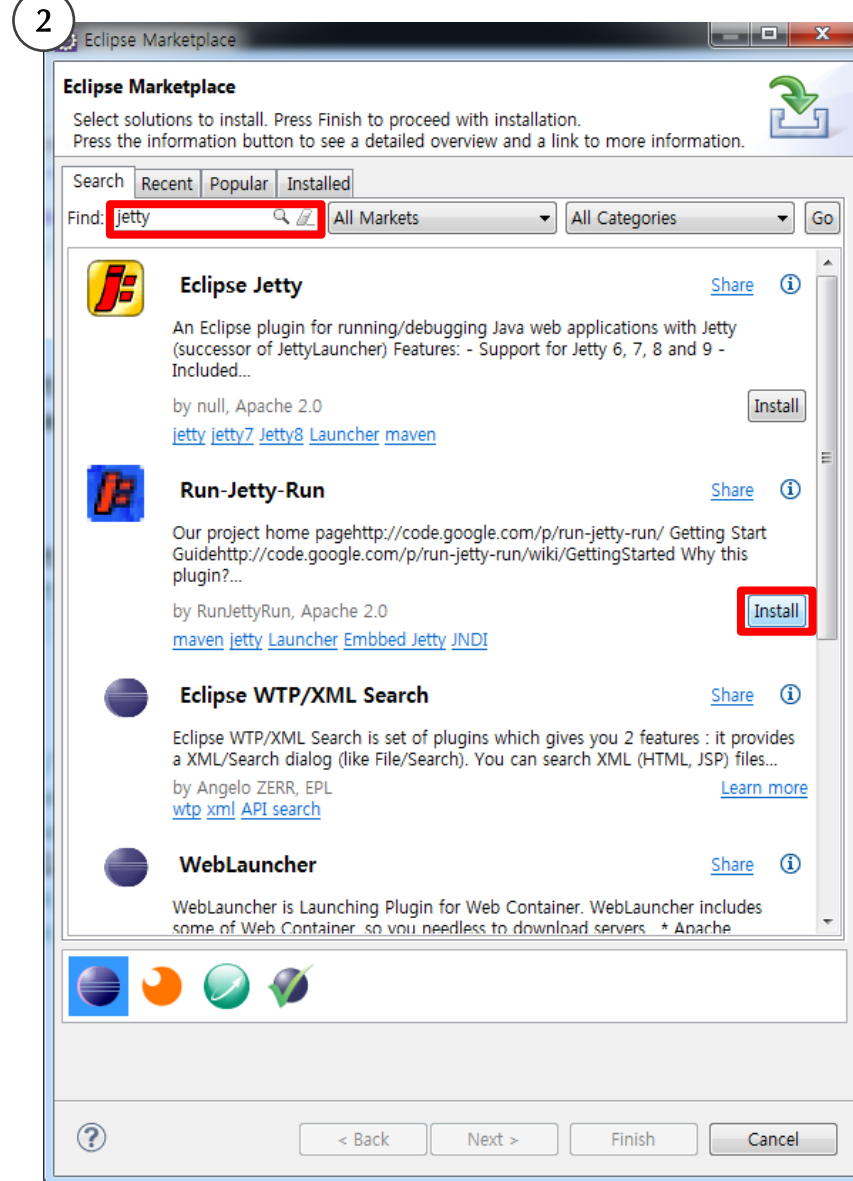
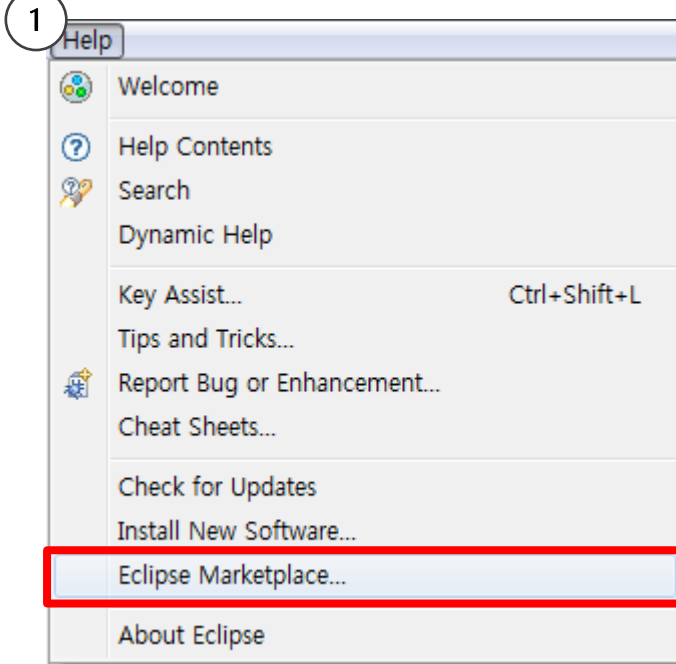
- 서블릿은 응답/요청 모델에서 작동하도록 디자인되었다.
- 요청/응답 모델에서 클라이언트는 서버에게 요청 메시지를 보내고 서버는 이에 대한 응답으로 메시지를 돌려보낸다.
 - 서버란, 웹 브라우저 클라이언트로부터 HTTP요청을 받아들이고, 그 HTTP요청에 응답하는 컴퓨터 프로그램을 말한다.
 - HTTP란, WWW 상에서 클라이언트와 서버 사이에서 정보를 주고 받을 수 있는 프로토콜이다. 주로 HTML문서를 주고 받을 때 사용된다.
 - WWW란, 인터넷에 연결된 컴퓨터들을 통해 사람들이 정보를 공유할 수 있는 전 세계적인 정보 공간을 말한다.
- 자바 서블릿 API는 순수 자바로 되어 있으면서 클라이언트로부터 HTTP요청을 받아들이고 그 요청에 응답하는 메커니즘을 가지고 있다.
- 그러므로 서블릿은 클라이언트의 요청을 받아들여 연속성을 가지는 데이터 저장소를 이어주는 중간 계층에 위치한다.
- javax.servlet 패키지, javax.servlet.http 패키지
- 서블릿을 생성할 경우 javax.servlet.http.HttpServlet 상속 받게 된다.

✓ 서블릿 컨테이너

- 서블릿을 동작시키기 위한 실행환경을 의미한다. 또한 서블릿 컨테이너는 서블릿을 생성, 호출, 소멸하는 등의 생명주기를 관리한다.
- 서블릿 컨테이너는 TCP/IP 연결을 생성하고 HTTP프로토콜을 해석하는 과정을 수행하고 있어 웹 프로그래머들이 웹을 쉽게 구현할 수 있게 지원해준다.
- 대표적인 서블릿 컨테이너(웹 서버에 포함되어 있음)
 - 아파치 톰캣(Apach Tomcat)
 - 제티(Jetty)
 - 레진

서블릿 컨테이너 설치(jetty)

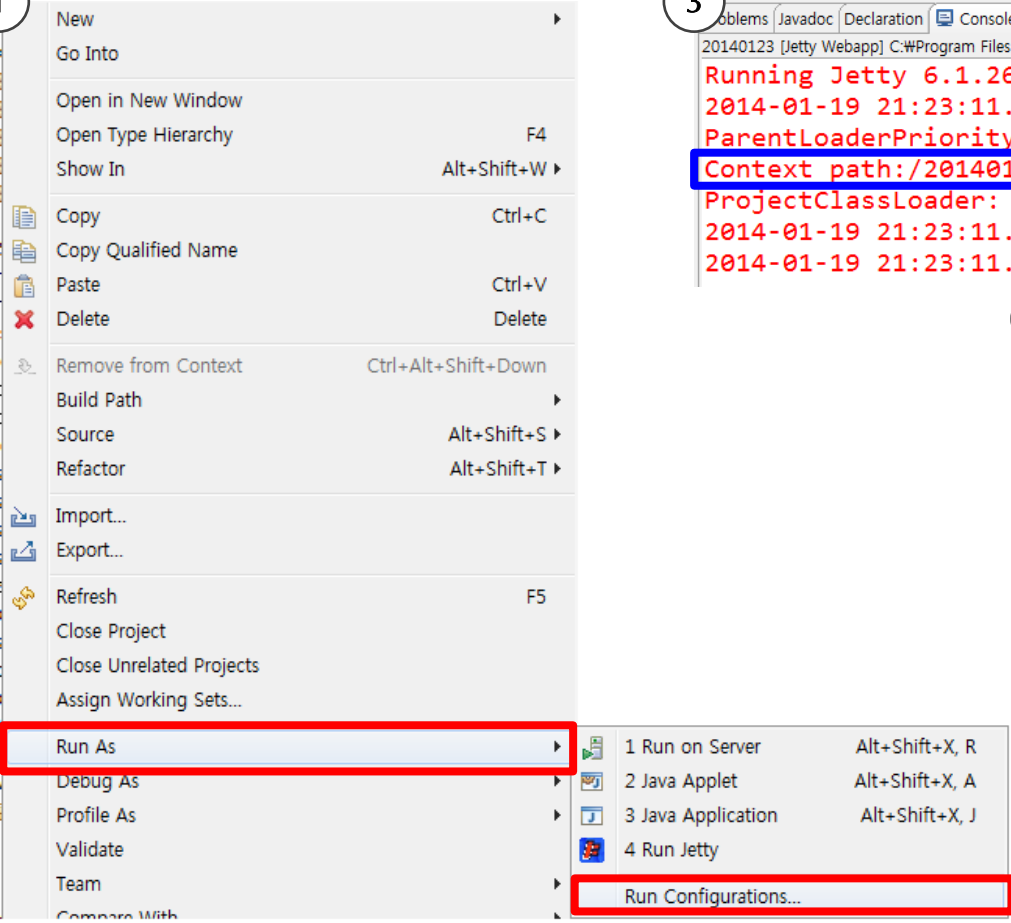
✓ 이클립스의 플러그인으로 jetty 설치



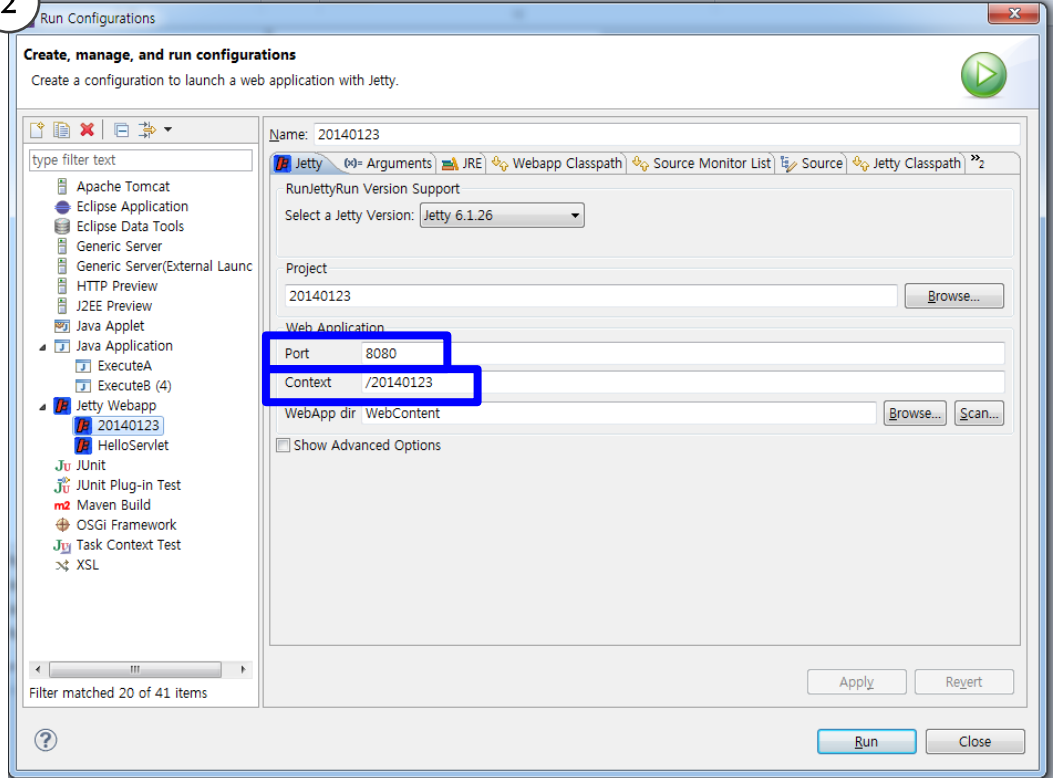
서블릿 컨테이너 설정(jetty)

✓ Jetty서버 설정 확인

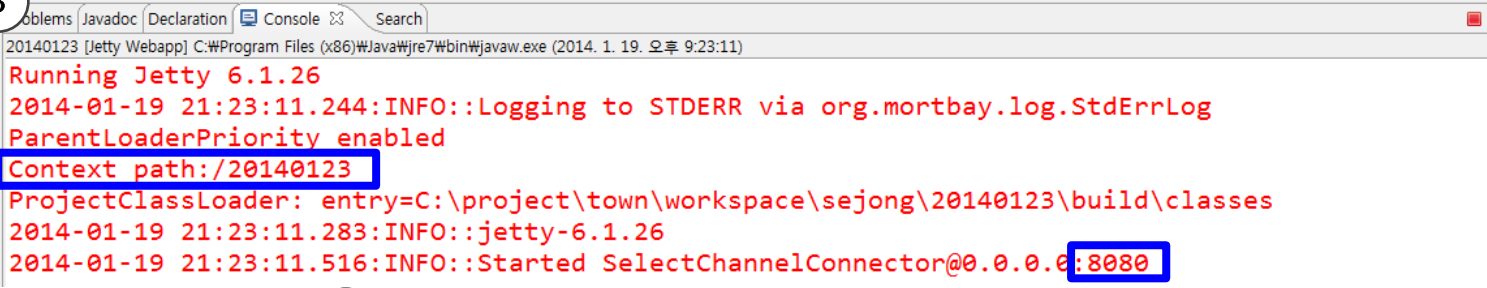
1



2



3



```
20140123 [Jetty Webapp] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (2014. 1. 19. 오후 9:23:11)
Running Jetty 6.1.26
2014-01-19 21:23:11.244:INFO::Logging to STDERR via org.mortbay.log.StdErrLog
ParentLoaderPriority enabled
Context path:/20140123
ProjectClassLoader: entry=C:\project\town\workspace\sejong\20140123\build\classes
2014-01-19 21:23:11.283:INFO::jetty-6.1.26
2014-01-19 21:23:11.516:INFO::Started SelectChannelConnector@0.0.0.0:8080
```

서블릿 컨테이너 실행(jetty)

✓ 웹 프로젝트를 jetty서버에 빌드 및 배포, 서버 구동

1

New

Go Into

Open in New Window

Open Type Hierarchy

Show In

Copy

Copy Qualified Name

Paste

Delete

Remove from Context

Build Path

Source

Refactor

Import...

Export...

Refresh

Close Project

Close Unrelated Projects

Assign Working Sets...

Run As

Debug As

Profile As

Validate

Team

2

Items Javadoc Declaration Console Search

20140123 [Jetty Webapp] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (2014. 1. 19. 오후 9:23:11)

Running Jetty 6.1.26

2014-01-19 21:23:11.244:INFO::Logging to STDERR via org.mortbay.log.StdErrLog

ParentLoaderPriority enabled

Context path:/20140123

ProjectClassLoader: entry=C:\project\town\workspace\sejong\20140123\build\classes

2014-01-19 21:23:11.283:INFO::jetty-6.1.26

2014-01-19 21:23:11.516:INFO::Started SelectChannelConnector@0.0.0.0:8080

3

Directory: /20140123/

localhost:8080/20140123/

1 Java Applet

2 Java Application

3 Run Jetty

Run Configurations...

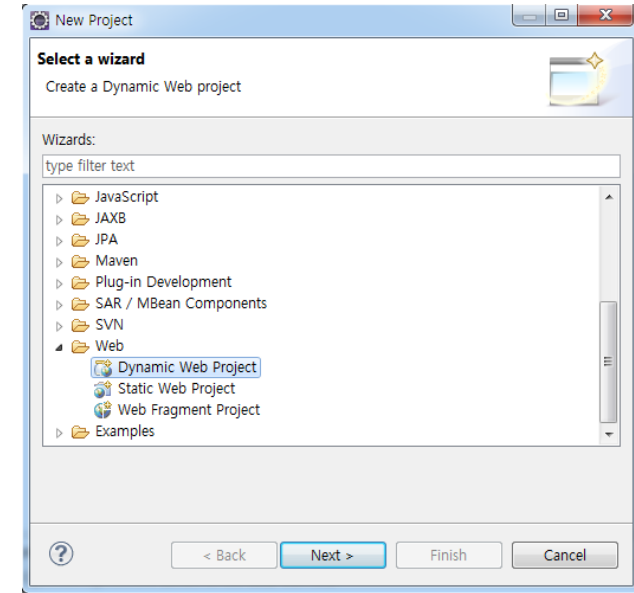
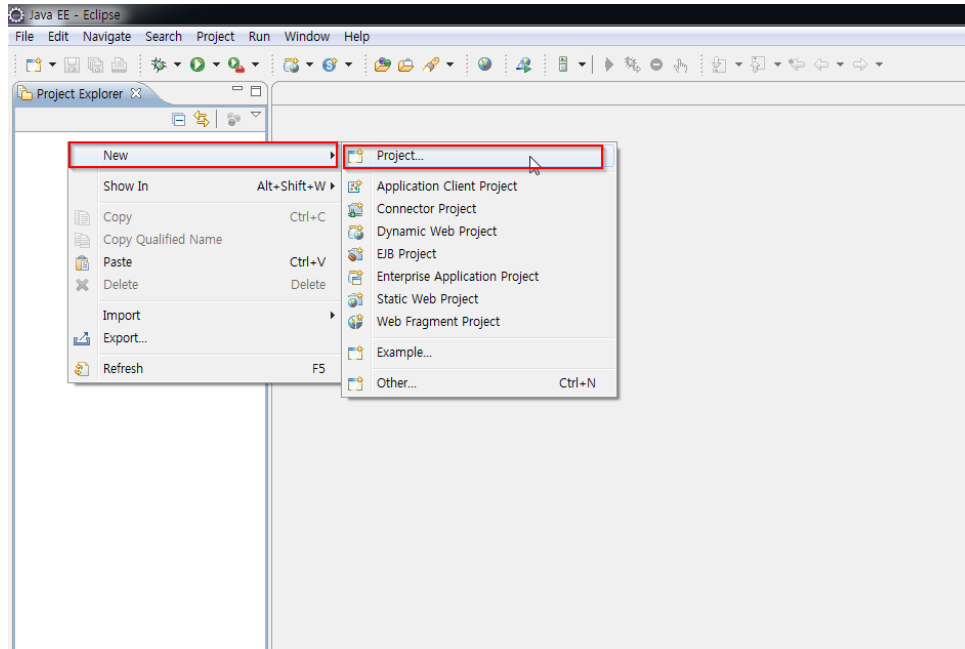
한국소프트웨어기술진흥협회
KOSTA Korea Software Technology Association

- 17 -

NEXTREE

웹 프로젝트 생성

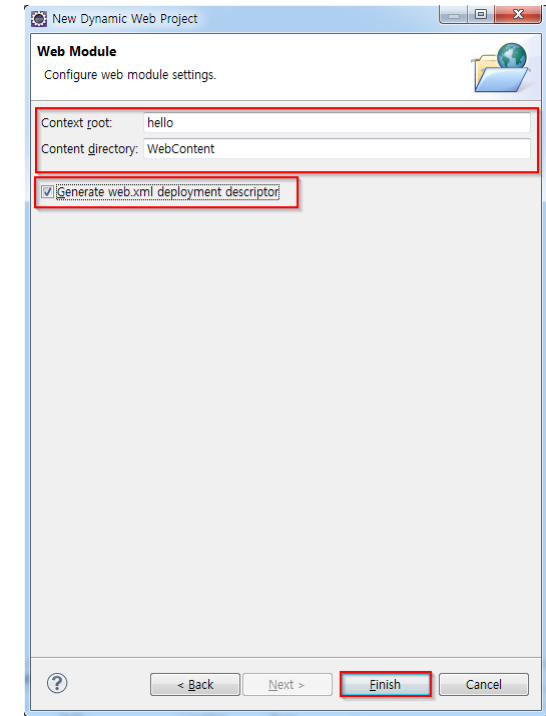
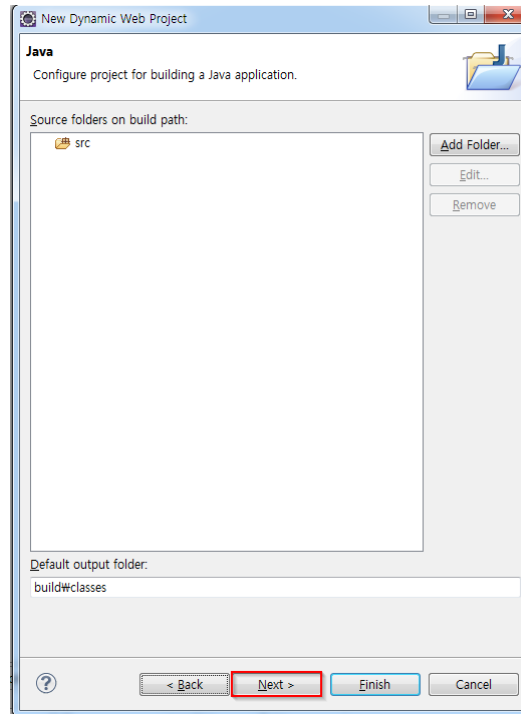
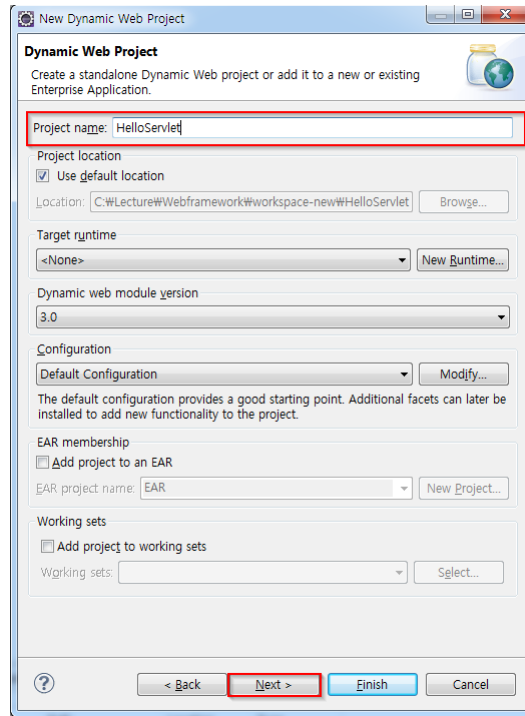
✓ Dynamic Web Project 생성



웹 프로젝트 설정

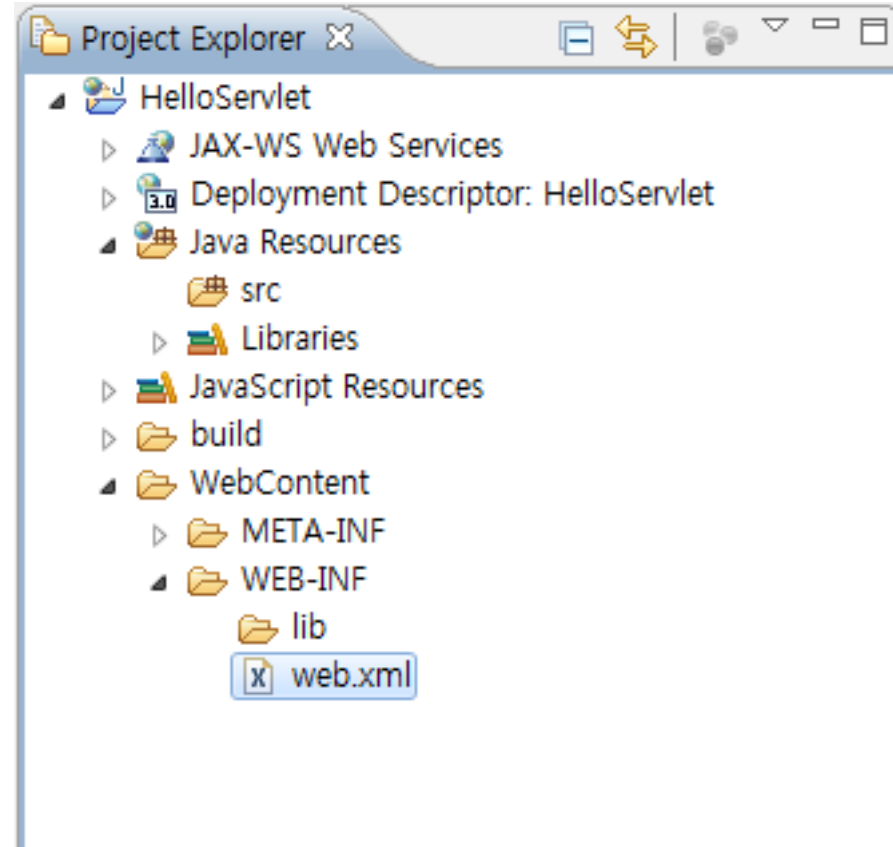
✓ Dynamic Web Project 생성

- Project Name : 20140123
- Context Root : 20140123
- Context Directory : WebContent



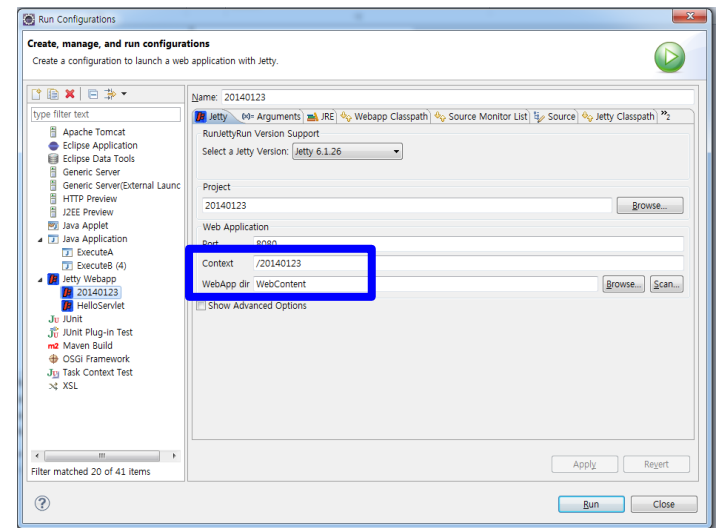
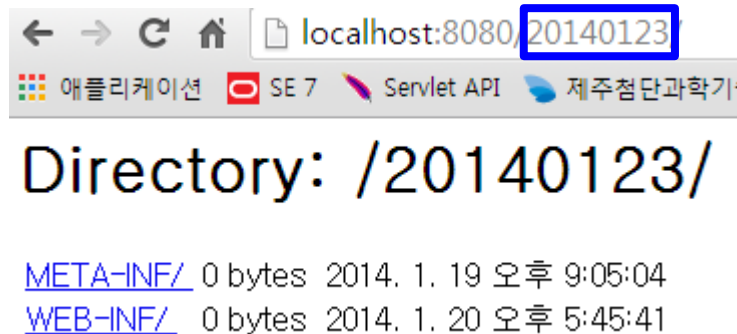
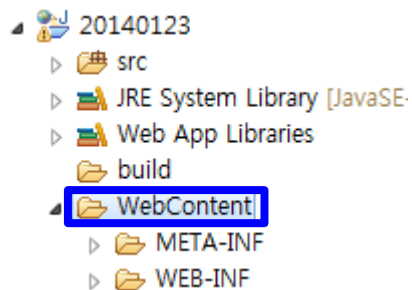
웹 프로젝트 구조

- ✓ Dynamic Web Project 생성완료
- ✓ WebContent 폴더
 - 웹 루트 폴더
 - JSP, HTML 등의 파일 위치
- ✓ WebContent/WEB-INF 폴더
 - xml, properties 등의 설정 파일 위치
- ✓ lib 폴더
 - 라이브러리 파일 위치
 - 서블릿 jar 추가
- ✓ src 폴더
 - JAVA 소스 파일 위치
- ✓ web.xml
 - 웹 프로젝트의 Deployment Descriptor(배포 설명자)
 - Servlet, JSP 매핑, 필터 등 정의



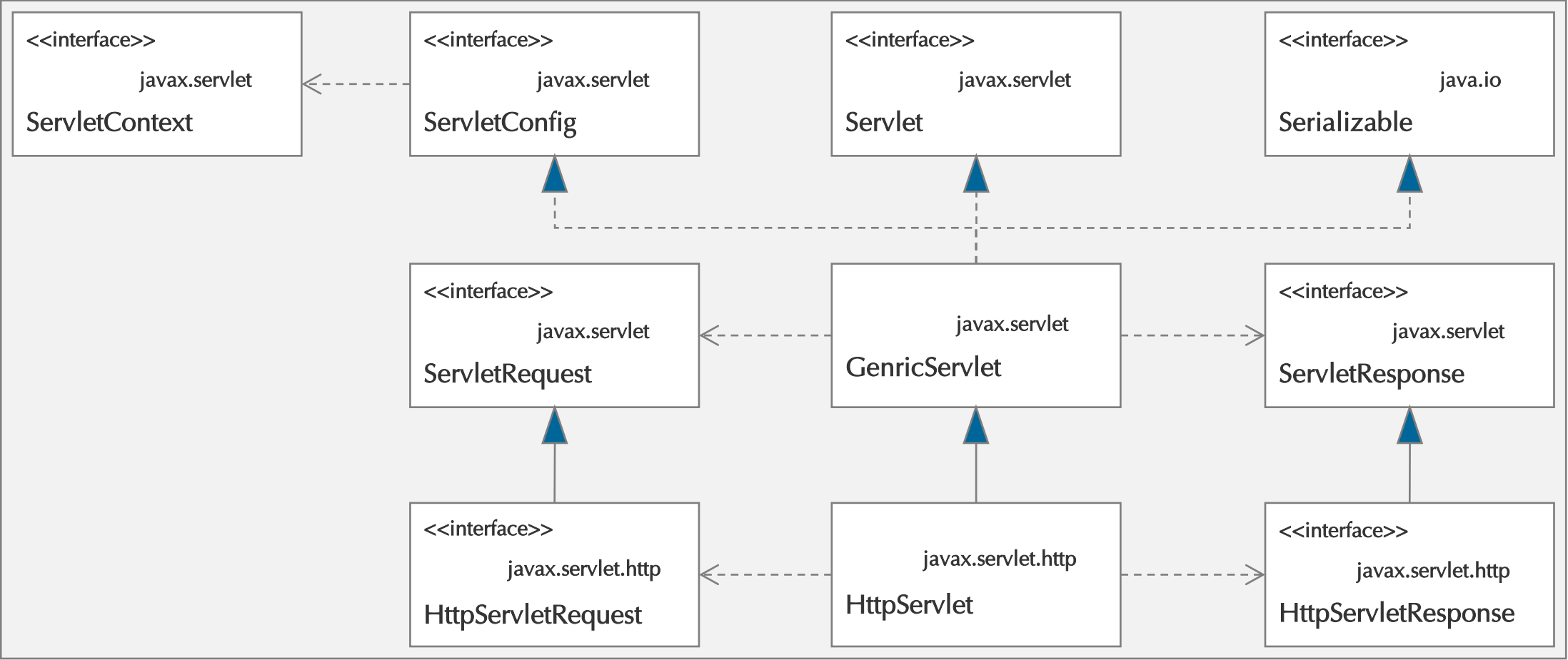
Context에 대한 이해

- ✓ 하나의 Context(물리적인 디렉토리와 개념적인 이름)는 하나의 웹 애플리케이션과 완벽하게 일치하는 개념이다.
 - Context : Context Directory 와 Context Path
- ✓ 웹 애플리케이션은 클라이언트가 접근할 수 있는 모든 자원(정적 또는 동적 콘텐츠)을 Context 디렉토리 아래 둔다.
 - Context 디렉토리 명은 설정이 가능하나 서버마다 달라질 수 있다.
 - 현재 우리는 WebContent라는 물리적인 디렉토리가 Context 디렉토리에 해당된다.
- ✓ Context Path는 클라이언트가 웹 애플리케이션으로 진입하는 시작점으로 볼 수 있다.
 - Context Path는 하나의 웹 애플리케이션의 논리적인 이름(경로이름)을 의미한다. 이름이 없어도 상관없다.
 - 클라이언트가 웹 애플리케이션에 접근 시, <http://localhost:8080/20140123> 으로 접근한다면 Context Path는 20140123 이다.
 - 기본적으로 서버에 접근하기 위해서는 서버 IP와 포트번호가 필요한데, 그 뒤에 Context Path가 붙는다.
 - Context Path는 위 Context Directory와 맞닿아 있다.



서블릿 클래스들 간의 관계

✓ 서블릿 관련 클래스 및 인터페이스 상속관계



Generic Servlet 추상클래스

✓ 서블릿 관리에 필요한 기능을 미리 구현하여 제공

- 서블릿을 위한 어댑터 역할

✓ javax.servlet.GenericServlet

- void destroy()
- java.lang.String getInitParameter(java.lang.String name)
- java.util.Enumeration<java.lang.String> getInitParameterNames()
- ServletConfig getServletConfig()
- ServletContext getServletContext()
- java.lang.String getServletInfo()
- java.lang.String getServletName()
- void init()
- void init(ServletConfig config)
- void log(java.lang.String msg)
- void log(java.lang.String message, java.lang.Throwable t)
- abstract void service(ServletRequest req, ServletResponse res)

HTTP Servlet 추상클래스

- ✓ 일반적인 서블릿을 의미함
- ✓ Generic Servlet 상속
- ✓ Service 메서드를 HTTP 프로토콜 요청에 적합하게 재 구현

- ✓ `javax.servlet.HttpServlet`
 - `void doDelete(HttpServletRequest req, HttpServletResponse resp)`
 - `void doGet(HttpServletRequest req, HttpServletResponse resp)`
 - `void doHead(HttpServletRequest req, HttpServletResponse resp)`
 - `void doOptions(HttpServletRequest req, HttpServletResponse resp)`
 - `void doPost(HttpServletRequest req, HttpServletResponse resp)`
 - `void doPut(HttpServletRequest req, HttpServletResponse resp)`
 - `void doTrace(HttpServletRequest req, HttpServletResponse resp)`
 - `long getLastModified(HttpServletRequest req)`
 - `void service(HttpServletRequest req, HttpServletResponse resp)`
 - `void service(ServletRequest req, ServletResponse res)`

Servlet 생성

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

//서블릿 중 99%는 HttpServlet를 상속한다.
public class ch2Servlet extends HttpServlet {

    //실제 프로젝트에서는 서블릿 중 99%는 doGet()과 doPost() 메소드만 재정의 한다.
    @Override //컨테이너가 생성한 Request와 Response 객체 참조를 넘겨받는 곳
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter writer = response.getWriter();
        writer.append("<HTML><BODY>");
        writer.append("<H2>ch2Servlet doGet</H2>");
        writer.append("</BODY></HTML>");
    }
}
```


배포 서술자(Deployment Descriptor)

- ✓ web.xml – 서블릿과 JSP를 어떻게 실행하느냐에 관한 많은 정보들이 들어감.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
<display-name>my-servlet</display-name>

<!-- <servlet> 항목들을 보면 이 컨테이너가 관리하는 웹 애플리케이션에는 어떤 것들이 있는지 알 수 있다. -->
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.kosta.web.HelloServlet</servlet-class>
</servlet>

<!-- 런타임 시 요청이 들어오면 컨테이너는 <servlet-mapping>항목을 검색한다. -->
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/HelloServlet</url-pattern>
</servlet-mapping>

</web-app>
```



2. Servlet

2.1 Servlet 기본

2.2 Servlet 컨테이너

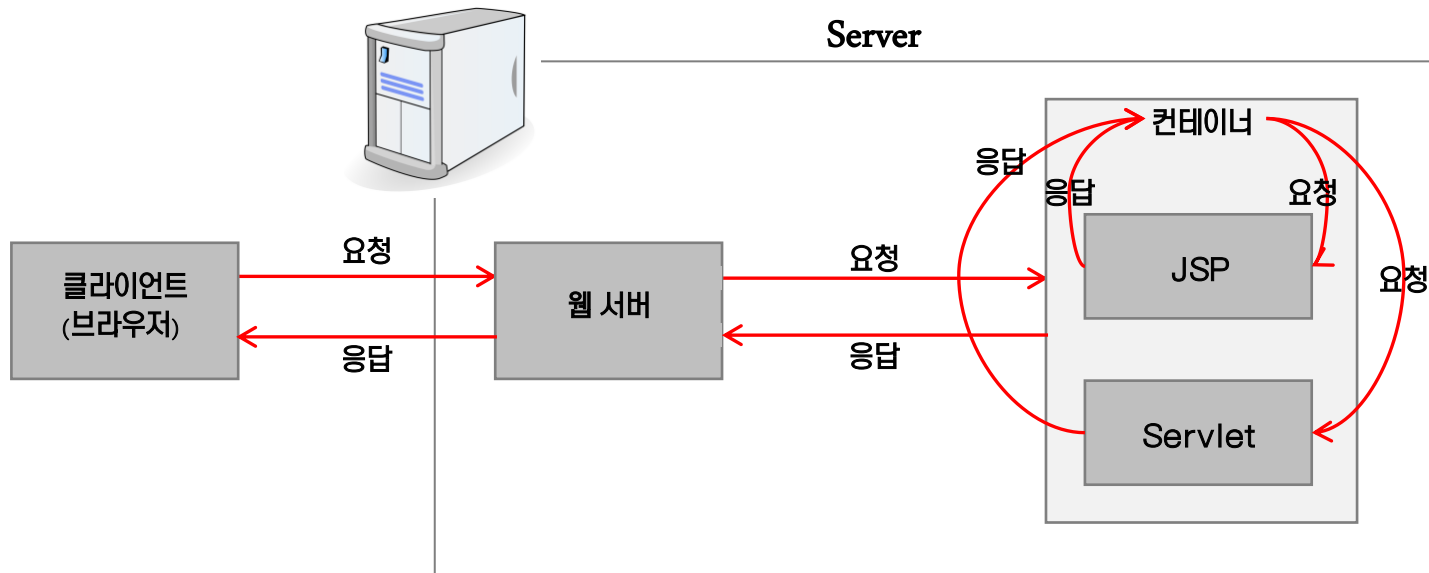
2.3 Request, Response

2.4 Redirect vs. Request Dispatch

- Servlet 요청처리 프로세스
- Servlet 컨테이너 역할
- Servlet 라이프사이클
- HTTP 메서드 종류

Servlet 요청 처리 프로세스

- ✓ 클라이언트(브라우저)가 HTTP 요청
- ✓ 웹 서버는 해당 요청을 받아들임 (GET / POST)
- ✓ 웹 서버는 요청을 컨테이너에게 전달
- ✓ 컨테이너는 HTTP Request, HTTP Response 객체를 만들어 서블릿 doPost()나 doGet() 메소드 중 하나를 호출
- ✓ 서블릿은 요청에 대한 처리
- ✓ 요청결과(응답)는 서블릿 -> 컨테이너 -> 웹 서버 -> 클라이언트(브라우저)에게 전달됨



Servlet 컨테이너가 주는 이점

✓ 웹 서버와 서블릿의 커뮤니케이션 지원

- 개발자가 소켓 생성, 포트 리스닝 등 복잡한 부분을 직접 구현할 필요 없음

✓ 라이프사이클 관리

- 서블릿 클래스의 생성, 초기화, 호출, 소멸을 관리함
- 개발자가 직접 생성하고 초기화할 필요가 없음

✓ 멀티스레딩 지원

- 요청이 들어올 때마다 새로운 스레드를 생성하여 처리
- 다중요청에 대한 스레드 생성 및 운영에 대해서 관리

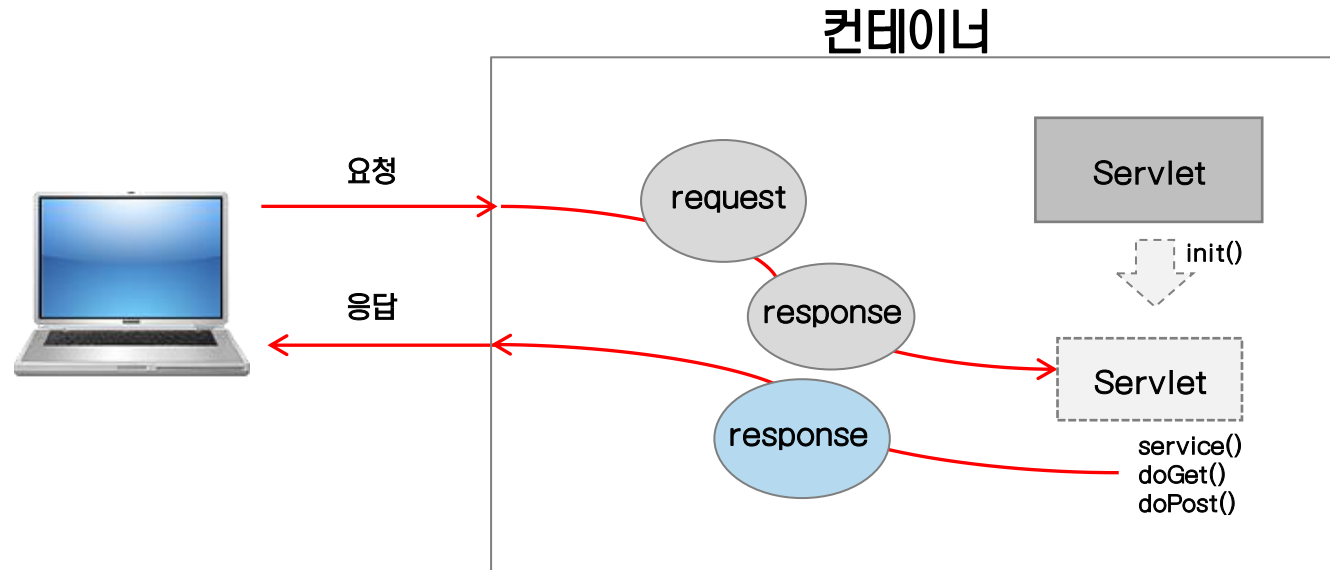
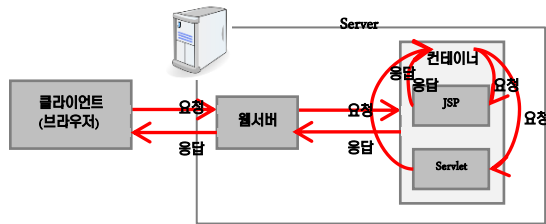
✓ 선언적인 보안관리

- 보안에 관련된 내용을 서블릿 내부에 하드코딩할 필요 없음
- XML 설정을 통해 컨테이너가 보안관리 하도록 함

✓ JSP 지원

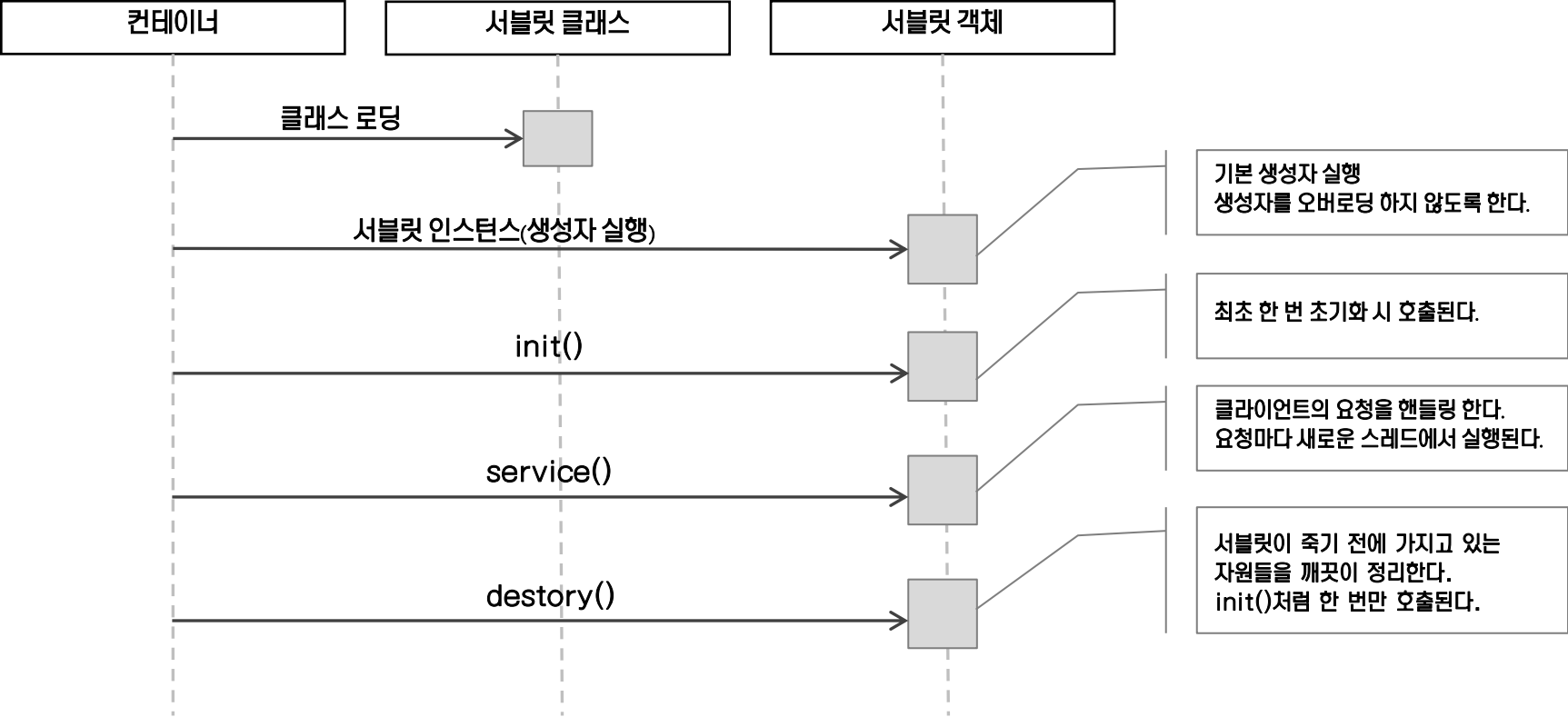
Servlet 컨테이너

- ✓ 서블릿이 요청되면 컨테이너는 HttpServletRequest, HttpServletResponse 객체 생성
- ✓ 요청을 처리할 해당 서블릿 스레드를 생성하여 Request, Response 객체를 전달
- ✓ 컨테이너는 서블릿의 service() 호출
- ✓ 요청방식에 따라 doGet() 또는 doPost () 호출
- ✓ 처리가 끝나면 컨테이너는 Response객체를 HttpServletResponse로 변환하여 클라이언트로 전송
- ✓ 마지막으로 서블릿 스레드와 Request, Response 객체는 소멸



Servlet 라이프사이클 (1/2)

✓ 컨테이너에 의해서 호출되고 소멸되는 서블릿의 라이프사이클



Servlet 라이프사이클 (2/2) – 주요 메서드

✓ init()

- 컨테이너가 서블릿 인스턴스를 생성한 후 호출
- 클라이언트의 요청을 처리하기 전 서블릿을 초기화하는 기능
- 필요한 경우 재 정의 가능 (예 : 데이터 베이스 접속)
- init() 메서드가 호출 되는 시점은 서버마다 다르게 동작할 수 있다.
- 서버가 구동하기 시점일 수도 있고 서블릿이 최초에 호출되는 시점일 수도 있다.

✓ service()

- 컨테이너가 클라이언트 요청에 대한 서블릿 인스턴스를 생성/초기화한 후 호출
- init() 메소드 이후에 호출됨
- 요청의 HTTP 메소드(GET/POST 등)를 참조하여 doGet() 또는 doPost()를 호출하는 기능
- 재정의하지 않음 (상속받은 상위 클래스의 service() 메소드를 그대로 사용)

✓ doGet() / doPost()

- service() 메소드에 의해서 호출됨
- 요청을 처리하기 위한 로직을 구현
- 반드시 재정의해서 사용

✓ destory()

- 요청처리가 끝나면 컨테이너가 호출
- 재정의하지 않음

HttpServlet의 메소드와 HTTP 메소드

HttpServlet Class Method	HTTP Method
void doGet(HttpServletRequest , HttpServletResponse)	GET
void doPost(HttpServletRequest , HttpServletResponse)	POST
void doPut(HttpServletRequest , HttpServletResponse)	PUT
void delete(HttpServletRequest , HttpServletResponse)	DELETE



2. Servlet

2.1 Servlet 기본

2.2 Servlet 컨테이너

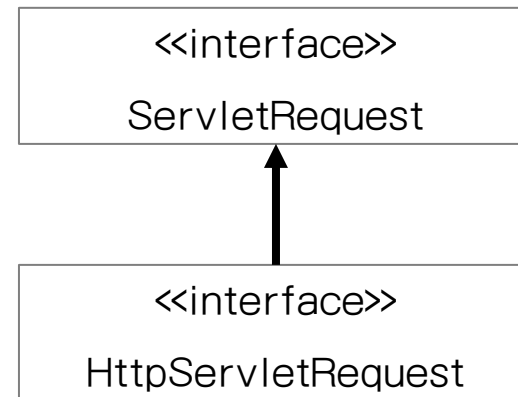
2.3 Request, Response

2.4 Redirect vs. Request Dispatch

- HttpServletRequest
- HttpServletResponse
- ContentType

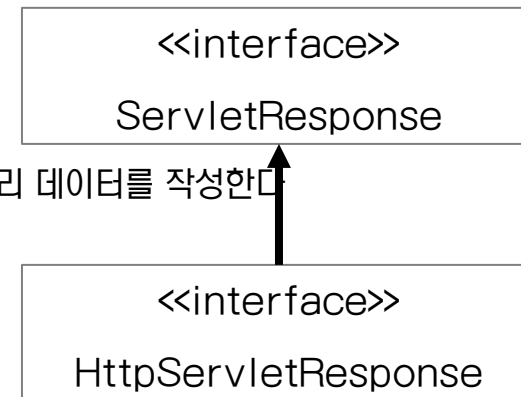
HttpServletRequest

- ✓ Servlet에서는 HttpServletRequest 객체로부터 다양한 요청정보를 처리한다.
- ✓ 다음은 HttpServletRequest 인터페이스의 메서드이다.
 - request.getHeader() : 요청 정보의 헤더정보를 반환한다.
 - request.getMethod() : 요청 정보의 HTTP Method 정보를 반환한다.
 - request.getParameter() : 요청 시 정보의 이름으로 요청 값을 반환한다.
 - request.getParameterValues() : 요청 시 같은 정보의 이름으로 요청 값들을 배열로 반환한다.
 - request.setCharacterEncoding() : 요청정보의 인코딩을 설정한다.
- ✓ 이 외에도 다양한 메서드가 있으므로 Servlet API를 살펴본다.
- ✓ <http://tomcat.apache.org/tomcat-5.5-doc/servletapi/>



HttpServletResponse

- ✓ Servlet에서는 HttpServletResponse 객체로부터 다양한 응답정보를 처리한다.
- ✓ 다음은 HttpServletResponse 인터페이스의 메서드이다.
 - response.setContentType() : 클라이언트로 보내는 컨텐츠 타입을 설정한다.
 - response.getWriter() : PrintWriter 객체를 반환하여 클라이언트에게 전달할 텍스트 데이터를 작성한다.
 - response.getOutputStream() : ServletOutputStream 객체를 반환하여 클라이언트에게 전달할 바이너리 데이터를 작성한다.
 - response.setStatus() : 응답에 대한 상태코드를 지정한다.
- ✓ 이 외에도 다양한 메서드가 있으므로 Servlet API를 살펴본다.
- ✓ <http://tomcat.apache.org/tomcat-5.5-doc/servletapi/>



ContentType

- ✓ ContentType은 HTTP 응답에 반드시 포함되어야 하는 헤더정보
- ✓ 브라우저에게 지금 반환하는 데이터가 무엇인지를 미리 알려줌
 - PDF : PDF보기 프로그램을 내부적으로 실행
 - VIDEO : 비디오 플레이어 실행
 - 파일다운로드를 위해 바이트 정보 저장
- ✓ setContentType() 메소드를 이용해 ContentType을 설정한 후 출력 스트림에 데이터 기록
- ✓ 일반적인 MIME TYPE
 - text/html
 - application/pdf
 - video/quicktime
 - application/java
 - image/jpeg
 - application/jar
 - application/octet-stream
 - application/x-zip

Servlet 응답 정보에 저장된 정보 조회하여 출력하기

✓ CourseServlet.java

```
public class CourseServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
        IOException {

        PrintWriter writer = resp.getWriter();
        writer.append("<HTML><BODY>");
        writer.append("<H2>CourseServlet doGet</H2>");

        EducationProcess process = new EducationProcessImpl();
        CourseDto courseDto = process.getCourseByName("청년취업과정");
        int courseId = courseDto.getCourseId();
        writer.append(String.format("<h3>%s, %s</h3>", courseId, courseDto.getName()));
        for (SubjectDto subjectDto : courseDto.getSubjects()) {
            writer.append(String.format("<p>%s, %s</p>", subjectDto.getSubjectId(), subjectDto.getName()));
        }

        writer.append("</BODY></HTML>");
        writer.close();
    }
}
```

Servlet 요청 정보로 저장된 정보 조회하여 출력하기

✓ CourseServlet.java

```
public class CourseServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
        IOException {

        PrintWriter writer = resp.getWriter();
        writer.append("<HTML><BODY>");
        writer.append("<H2>CourseServlet doGet</H2>");

        String name = req.getParameter("name");
        EducationProcess process = new EducationProcessImpl();
        CourseDto courseDto = process.getCourseByName(name);
        int courseId = courseDto.getCourseId();
        writer.append(String.format("<h3>%s, %s</h3>", courseId, courseDto.getName()));
        for (SubjectDto subjectDto : courseDto.getSubjects()) {
            writer.append(String.format("<p>%s, %s</p>", subjectDto.getSubjectId(), subjectDto.getName()));
        }
        writer.append("</BODY></HTML>");
        writer.close();
    }
}
```




2. Servlet

2.1 Servlet 기본

2.2 Servlet 컨테이너

2.3 Request, Response

2.4 Redirect vs. Request Dispatch

- Redirect와 Request Dispatch
- Redirect
- Request Dispatch

Redirect vs. RequestDispatch

- ✓ 사용자의 요청을 다른 URL에서 처리하도록 방향을 바꿀 수 있음
- ✓ 사용자의 요청을 다른 컴포넌트(대부분 JSP)에게 처리하도록 방향을 바꿀 수 있음
- ✓ Redirect
 - 요청을 받은 서블릿은 해당 요청이 다른 URL에서 처리해야 할 대상이라면 Response객체의 sendRedirect() 호출
 - HTTP Response에는 상태코드 302, Location 헤더에 새로운 URL 정보를 포함
 - 브라우저는 응답을 받은 후 상태코드(302)를 확인하고 Location 헤더에 설정된 URL로 새로운 요청
 - 사용자도 브라우저에 변경된 URL을 확인할 수 있음
- ✓ Request Dispatch
 - 요청을 다른 방향으로 위임하는 작업이 서버에서 일어남
 - 요청을 받은 서블릿은 해당 요청이 다른 컴포넌트(JSP)가 처리해야 할 대상이라면 dispatch forward 호출
 - 브라우저의 URL은 바뀌지 않기 때문에 사용자는 Dispatch 된 응답인지를 알 수 없음

Redirect

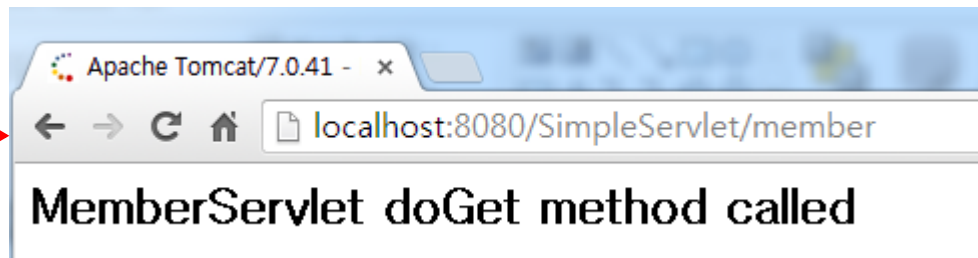
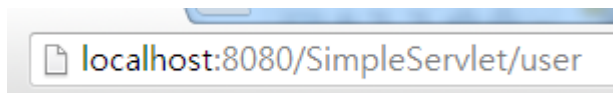
- ✓ 사용자는 /user 로 요청을 했지만 브라우저의 응답은 /member 로 변경됨
- ✓ 브라우저의 URL이 변경되어있음을 확인

UserServlet

```
@Override
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    resp.sendRedirect("member");
}
```

MemberServlet

```
@Override
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    PrintWriter writer = resp.getWriter();
    writer.append("<HTML><BODY>");
    writer.append("<H2>MemberServlet doGet method called</H2>");
    writer.append("</BODY></HTML>");
}
```



RequestDispatch

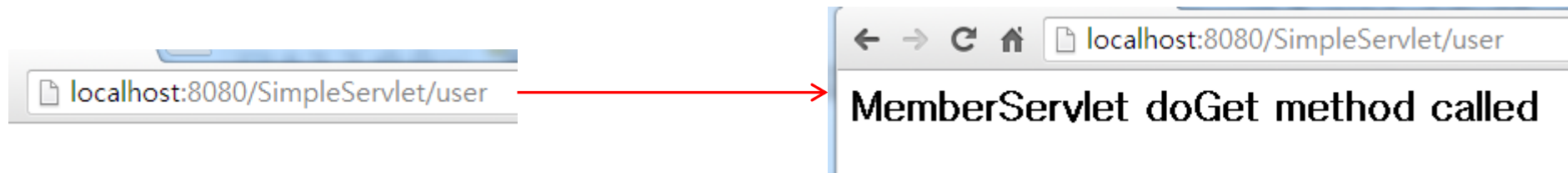
- ✓ 사용자는 /user 로 요청을 했지만 응답은 MemberServlet의 응답
- ✓ 브라우저의 URL은 변경되지 않음

UserServlet

```
@Override
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    //resp.sendRedirect("member");
    RequestDispatcher dispatcher = req.getRequestDispatcher("member");
    dispatcher.forward(req, resp);
}
```

MemberServlet

```
@Override
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    PrintWriter writer = resp.getWriter();
    writer.append("<HTML><BODY>");
    writer.append("<H2>MemberServlet doGet method called</H2>");
    writer.append("</BODY></HTML>");
}
```





3. 세션 관리

3.1 세션 개요

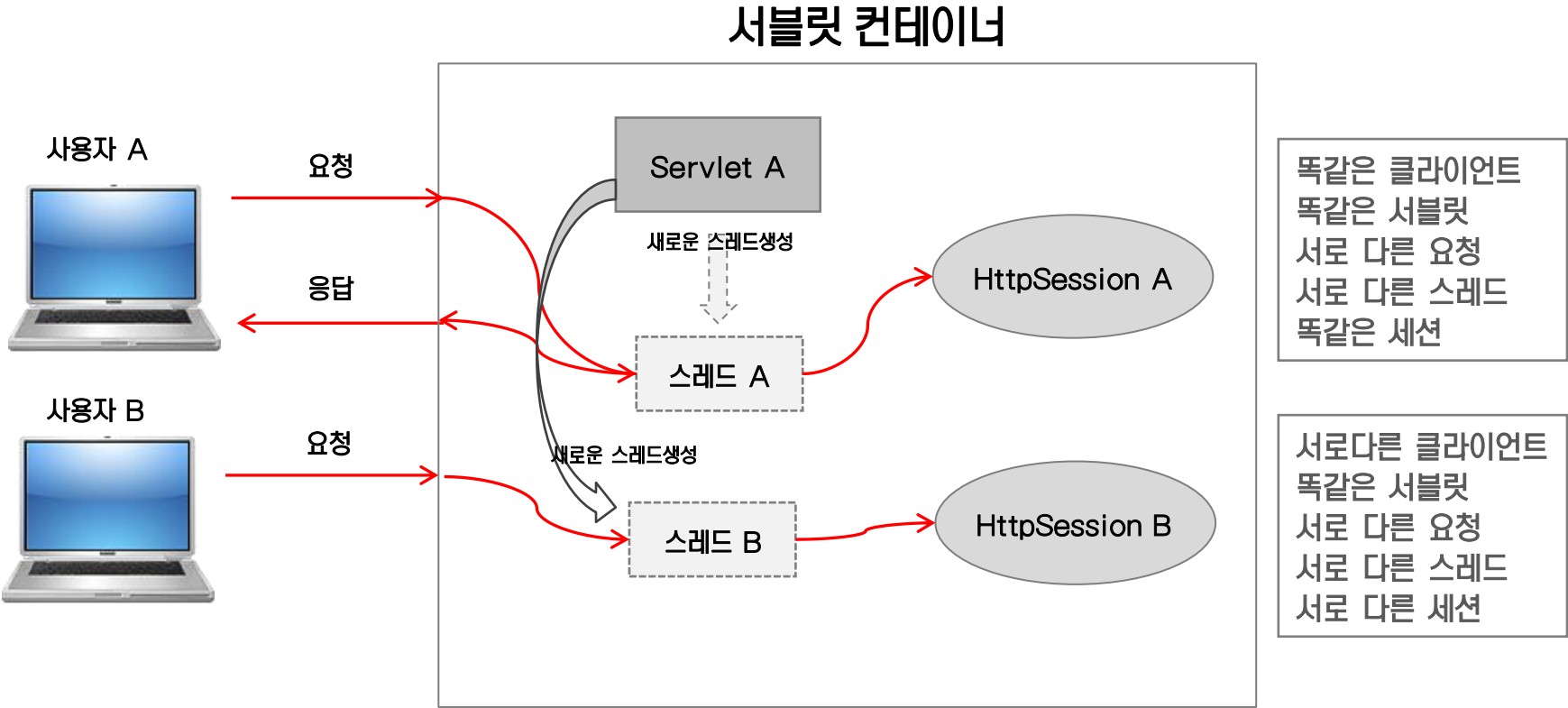
3.2 세션 관리

3.3 쿠키

- Session 개요

Session 개요

- ✓ 세션은 다중 요청간 대화 정보를 유지 하기 위한 방법이다.
- ✓ 클라이언트가 제일 처음 요청을 보낼 때, 컨테이너는 클라이언트의 유일한 세션 ID를 생성한다.
- ✓ 클라이언트 접속 시 세션ID로 서로 다른 클라이언트를 구분한다.





3. 세션 관리

3.1 세션 개요

3.2 세션 관리

3.3 쿠키

- HttpSession 사용방법
- Session 제거
- Session 타임아웃 설정

HttpSession 사용방법 (1/3)

- ✓ Response 객체에 세션 쿠키 보내기

```
HttpSession session = request.getSession(); //Request로 부터 세션을 리턴받는다.
```

- ✓ Request 객체로부터 세션 ID 가져오기

```
HttpSession session = request.getSession(); //보내메서드와 동일함
```

- ✓ isNew() : 세션이 새로 만들어졌는지 알 수 있는 메소드

```
HttpSession session = request.getSession(); //보내메서드와 동일함
```

```
if(session.isNew()) {  
    out.println("This is a new session.");  
} else {  
    out.println("Welcome back!");  
}
```

HttpSession 사용방법 (2/3)

✓ request.getSession(false);

- 이미 만들어져 있는 세션을 원할 때

```
HttpSession session = request.getSession(false); //false를 넘기는 이유는 존재하는 세션을 리턴하라는 의미
```

✓ response.encodeURL

- 클라이언트가 브라우저 설정을 통해 쿠키를 사용하지 않을 때 쿠키 사용방법
- URL 재작성 방법 : 세션 ID 정보를 모든 URL 뒤에 추가하는 방법이다.

```
response.setContentType("text/html");  
PrintWriter out = response.getWriter();  
HttpSession session = request.getWriter();
```

```
HttpSession session = request.getSession(); //세션을 리턴받음
```

```
out.println("<html><body>");  
out.println("<a href='\" + response.encodeURL(\"/BeerTest.do\") + '\">click me</a>"); //URL 뒤에 세션 ID를 추가함  
out.println("</body></html>");
```

//최초 컨테이너가 개발자가 코딩한 getSession()부분을 만나면, 클라이언트의 Request에서 세션 ID를 읽지 않는다.
//컨테이너는 클라이언트 쿠키가 이용 가능한지 아닌지 모르기때문에 첫 번째 응답은 쿠키와 URL 재작성을 동시에 사용한다.

HttpSession 사용방법 (3/3)

✓ response.encodeRedirectURL

- 들어온 요청을 다른 서블릿 이나 JSP로 세션을 유지하며 요청 하고 싶을 때 사용한다.

```
response.encodeRedirectURL("/BeerTest.do");
```

Session 제거

✓ HttpSession 주요 메서드

Method	설명	비고
getCreationTime()	세션이 생성된 시간을 리턴	세션이 얼마나 오래 되었는지 알고 싶을 때, 특정 시간만 세션을 사용하도록 제한하는 데 사용(예 로그인한후 10분 간만 사용하도록 하겠다.)
getLastAccessedTime()	이 세션으로 들어온 마지막 요청시간	클라이언트가 언제 마지막으로 세션에 접근했는지 알고 싶을 때.
setMaxInactiveInterval()	해당 세션에 대한 요청과 요청간의 최대 허용 시간(초 단위)을 지정	클라이언트의 요청이 정해진 시간이 지나도 들어 오지 않을 경우, 해당 세션을 제거하기 위하여 사용함.
getMaxInactiveInterval()	해당 세션에 대한 요청과 요청간의 최대 허용 시간(초 단위)을 리턴	세션이 얼마나 오랫동안 비활성화 상태였는지, 여전히 살아있기는 한지 알고 싶을 대. 세션이 invalidate() 되기까지 시간이 얼마나 남았는지 알기 위하여 사용
invalidate()	세션을 종료. 이 작업에는 현재 세션에 저장된 모든 세션 속성을 제거하는(unbind)작업이 포함	클라이언트가 비활성화이거나, 세션 작업이 완료되어 강제로 세션을 종료 할 . invalidate()는 세션 ID가 더 이상 존재하지 않으니, 관련 속성을 세션 객체에서 제거하라는 의미

✓ 세션이 종료되는 세 가지 이유

- 설정된 시간이 다 되어서(타임아웃)
- 개발자가 세션 객체에 invalidate() 메소드를 실행하는 경우.
- 애플리케이션이 다운되는 경우(문제가 생겨 다운되거나, 언디플로이 되는 경우)

Session 타임아웃 설정

- ✓ web.xml 에 설정 : 생성되는 모든 세션에 `setMaxInactiveInterval()` 메소드를 호출하는 것과도 같다.

```
<web-app ...>
  <servlet>
    ...
  </servlet>
  <session-config>
    <session-timeout>15</session-timeout> //15는 분을 의미한다.
  </session-config>
</web-app>
```

- ✓ 특정 세션만 타임아웃 설정 하기

- 특정 세션 인스턴스만 세션 타임아웃 값을 변경할 수 있다.(다른 세션의 타임아웃 값은 바뀌지 않는다.)

```
session.setMaxInactiveInterval(20*60); //메소드의 인자는 초 단위 시간 임.
```




3. 세션 관리

3.1 세션 개요

3.2 세션 관리

3.3 쿠키

- Cookie 란?
- 쿠키 관련 서블릿 API

Cookie 란?

- ✓ 원래 쿠키는 세션을 지원하기 위하여 고안된 것이지만 다른 목적으로 사용해도 무방함
- ✓ 서버와 클라이언트간에 교환하는 조그마한 데이터(이름/값의 String쌍)
- ✓ 서버는 클라이언트로 쿠키를 보내고, 이후 클라이언트는 매번 요청에 이 값을 전송함
- ✓ 쿠키는 기본적으로 세션과 생존 범위가 같다. 그러나 계속 유지할 수도 있다.

Cookie 관련 서블릿 API

✓ Cookie 객체 생성

```
Cookie cookie = new Cookie("username", name);
```

✓ 쿠키가 클라이언트에 얼마나 오랫동안 살아 있을지 설정

```
cookie.setMaxAge(30*60); //-1을 설정하면 쿠키는 브라우저가 빠져나가는 대로 쿠키를 지운다.
```

✓ 쿠키를 클라이언트로 보낼때

```
response.addCookie(cookie);
```

✓ 클라이언트 Request에서 쿠키를 읽어올때

```
Cookie[] cookies = request.getCookies();  
for(Cookie cookie : cookies) [  
    cookie.getName();  
    cookie.getValue();  
}
```



4. JSP

4.1 JSP 기본

4.2 JSP 태그

4.3 JSP 동작원리

- JSP 개요

- JSP 지시자

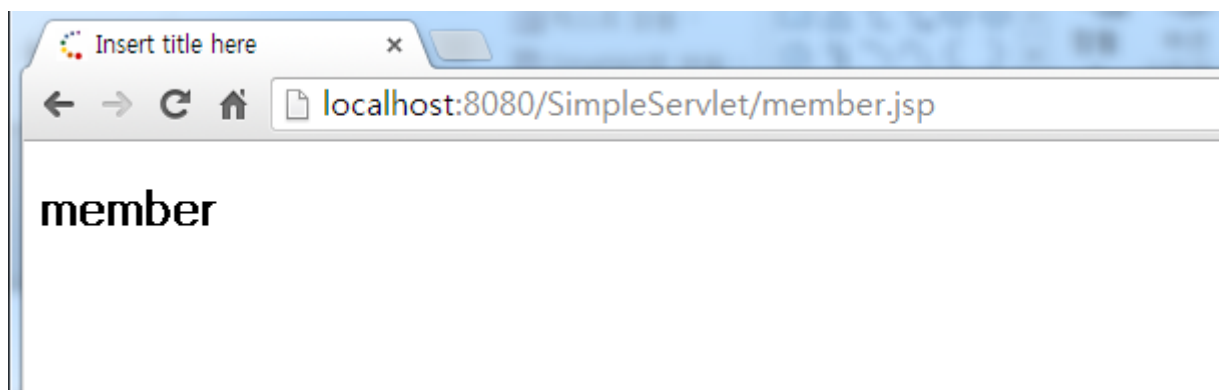
JSP(JavaServer Pages) 개요 (1/2)

- ✓ JSP는 HTML 내에 자바 코드를 삽입하여 웹 서버에서 동적으로 웹 페이지를 생성하여 클라이언트에 전달하는 언어이다.
- ✓ 확장자는 .jsp이며 HTML 코드와 자바 코드가 함께 존재한다.
- ✓ Servlet과 JSP 비교
 - Servlet은 Java 소스에 HTML 코드가 삽입된다.
 - JSP는 반대로 HTML코드에 Java코드가 삽입된다.
 - Servlet class는 컴파일과 web.xml에 서블릿을 등록하는 과정이 필요하지 하지만 JSP는 필요 없다.
 - 서블릿은 JSP로 대체될 수 있고, JSP는 서블릿으로 대체될 수 있다.
- ✓ JSP 장점
 - Java 코드에 HTML이 삽입되는 Servlet 단점을 해결한다.
 - HTML과 CSS 작업 시 웹 디자이너들이 작업하기가 수월하여 유지보수에 용이하다.
 - 간단한 로직을 구현할 때는 JSP가 더 간편하다.
- ✓ JSP 단점
 - 하지만 복잡한 로직을 구현할 때에 HTML 중심의 코드가 이해하기 어렵게 만들 수 있고, 프로그래밍 언어를 모르는 사람이 실수로 중요한 코드를 건들 우려도 있다. 그리고 힘들게 개발한 로직의 유출을 막기 위해서도 Servlet 기술이 필요하게 된다.
- ✓ 대안
 - 그래서 요즘은 JSP 기술과 Servlet 기술을 혼용한 새로운 프로그래밍 방법이 권장되고 있다. 프로그램의 기능을 구현하는 복잡한 로직은 서블릿 클래스 안에 기술하고, 그 결과를 가져다가 출력하는 일만 JSP 페이지가 담당하도록 만드는 방법이다. (모델 2방식)

JSP(JavaServer Pages) 개요 (2/2)

✓ 기본 JSP 소스코드와 브라우저 화면

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>member</h2>
</body>
</html>
```



JSP 지시자

✓ JSP가 서블릿이 되기 위해 필요한 중요정보를 지시자(<%@ %>)를 통해 제공

✓ page 지시자

- 페이지 관련 환경 정의
- character encoding, content-type 등의 속성을 정의

```
<%@ page import="kr.kosta.*" contentType="text/html" %>
```

✓ taglib 지시자

- <%@ taglib tagdir="/WEB-INF/tags/cool" prefix="cool" %>
- JSP 페이지에서 사용할 태그 라이브러리 정의

```
<%@ taglib tagdir="/WEB-INF/tags/cool" prefix="cool" %>
```

✓ include 지시자

- 현재 JSP 페이지에 포함될 코드나 문서를 정의
- 페이지의 Header, Tail, Navigation 등을 별도로 정의하여 include 할 수 있다.

```
<%@ include file="header.html" %>
```

page 지시자

✓ import

- 생성될 서블릿 클래스에 추가될 자바 import문을 정의
- java.lang, javax.servlet, javax.servlet.http는 기본적으로 import 됨

✓ contentType

- MIME 타입과 캐릭터셋 설정
- JSP파일 출력 시 문자코드 및 Content-Type 헤더에 출력할 문자코드명을 지정

✓ language

- 스크립트, 표현식, 선언문에 사용할 스크립팅 언어 정의

✓ pageEncoding

- JSP 문자 인코딩 정의
- JSP파일 작성시의 문자코드를 지정

✓ autoFlush

- 응답을 저장한 버퍼를 자동적으로 비울 것인지 여부를 설정
- true인 경우 버퍼가 다 채워졌거나 응답을 처리하는 서블릿 메소드가 종료되면 flush됨

include 지시자

- ✓ include 지시자는 지정한 파일을 JSP파일에 삽입한다.

```
<%@ include file="header.jsp" %>
```

- ✓ include 된 파일들은 내용이 그대로 복사되어 포함되고 이에 대한 서블릿 코드는 하나만 생성된다.
- ✓ 중첩하여 사용할 수 있다. (header.jsp 내부에서 또 다른 파일 include 가능)
- ✓ 주의사항 :
 - contentType 속성이 두 번 선언되지 않도록 주의한다.
예) 메인 페이지와 include된 페이지 모두 contentType 속성을 정의하면 오류발생

taglib 지시자

- ✓ 태그 라이브러리란 기본적인 JSP 내장 기능을 확장하기 위한 커스텀 태그들의 집합이다.
- ✓ taglib 지시자를 이용하여 core Tag를 추가한 예

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

- ✓ uri
 - 태그 라이브러리의 태그와 속성을 담고 있는 TLD(Tag Library Descriptor) 파일(.tld)이 존재하는 URI 지정
- ✓ prefix
 - 사용할 커스텀 태그들의 네임 스페이스 지정
- ✓ TLD 파일은 JSP페이지에서 사용 중인 커스텀 태그가 유효한 것인지를 검증하는데 사용하는 XML



4. JSP

4.1 JSP 기본

4.2 JSP 태그

4.3 JSP 동작원리

- 스크립틀릿 (scriptlets)
- 표현식 (expressions)
- 선언문 (declarations)
- 주석 (comment)

스크립틀릿 (scriptlets) (1/3)

✓ JSP 페이지에 자바 코드를 삽입하기 위해 사용한다.

```
package kr.kosta.util;

public class Counter {

    private static int count;

    public static synchronized int getCount() {
        count++;
        return count;
    }
}
```

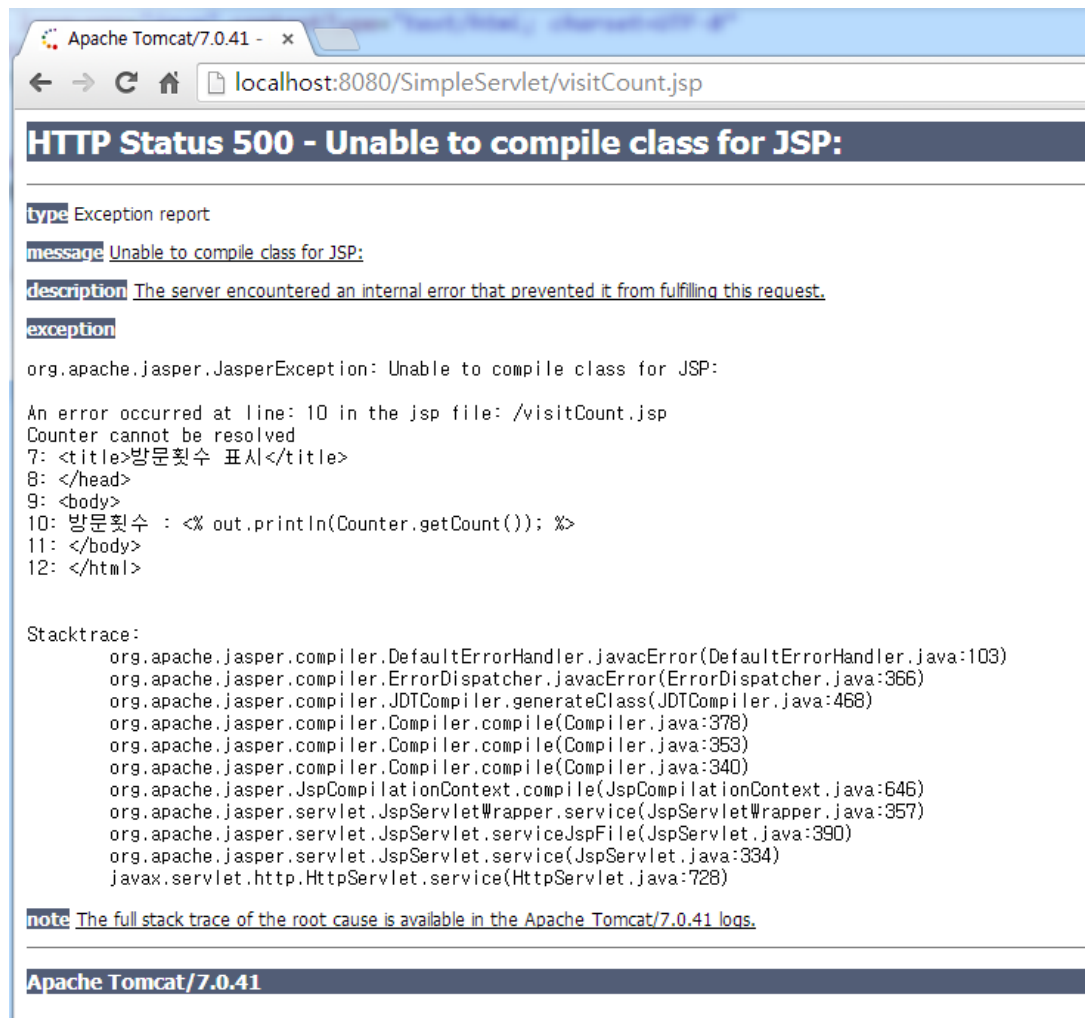
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "ht
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
<title>방문횟수 표시</title>
</head>
<body>
방문횟수 : <% out.println(Counter.getCount()); %>
</body>
</html>
```

스크립틀릿 (scriptlets) (2/3)

✓ Counter 클래스를 인식하지 못하는 경우 오류가 발생한다.

✓ 해결 방안

- 방법 #1. Counter 클래스의 패키지 명시
- 방법 #2. page 지시자 사용하여 import



Apache Tomcat/7.0.41 - x

localhost:8080/SimpleServlet/visitCount.jsp

HTTP Status 500 - Unable to compile class for JSP:

type Exception report

message Unable to compile class for JSP:

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: 10 in the jsp file: /visitCount.jsp
Counter cannot be resolved
7: <title>방문횟수 표시</title>
8: </head>
9: <body>
10: 방문횟수 : <% out.println(Counter.getCount()); %>
11: </body>
12: </html>
```

Stacktrace:

```
org.apache.jasper.compiler.DefaultErrorHandler.javacError(DefaultErrorHandler.java:103)
org.apache.jasper.compiler.ErrorDispatcher.javacError(ErrorDispatcher.java:366)
org.apache.jasper.compiler.JDTCompiler.generateClass(JDTCompiler.java:468)
org.apache.jasper.compiler.Compiler.compile(Compiler.java:378)
org.apache.jasper.compiler.Compiler.compile(Compiler.java:353)
org.apache.jasper.compiler.Compiler.compile(Compiler.java:340)
org.apache.jasper.JspCompilationContext.compile(JspCompilationContext.java:646)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:357)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:390)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:334)
javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
```

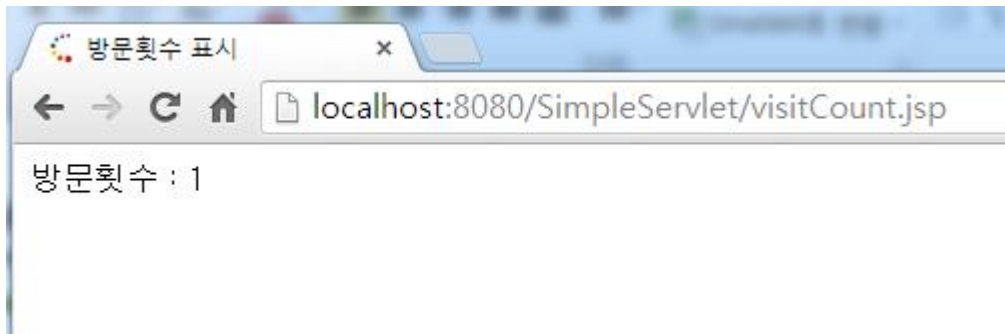
note The full stack trace of the root cause is available in the Apache Tomcat/7.0.41 logs.

Apache Tomcat/7.0.41

스크립틀릿 (scriptlets) (3/3)

- ✓ kr.kosta.util.Counter 와 같이 사용하는 클래스의 패키지명까지 명시
- ✓ JSP 내에서 사용하는 클래스가 많아질 경우 복잡해질 수 있음

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>방문횟수 표시</title>
</head>
<body>
방문횟수 : <% out.println(kr.kosta.util.Counter.getCount()); %>
</body>
</html>
```



지시자(1/2)

- ✓ page 지시자(<%@)의 import 속성을 사용하여 특정 패키지 또는 클래스를 import 할 수 있다.
- ✓ 컨테이너는 JSP 페이지를 자바 코드로 변환할 때 지시자를 참고한다.
- ✓ 패키지 하나만 import 할 경우

```
<%@ page import="kr.kosta.util.*" %>
```

- ✓ 여러 개의 패키지를 import 할 경우 (쉼표로 구분)

```
<%@ page import="kr.kosta.util.*, java.util.*" %>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="kr.kosta.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>방문횟수 표시</title>
</head>
<body>
방문횟수 : <% out.println(Counter.getCount()); %>
</body>
</html>
```

지시자[2/2]

✓ page 지시자

- 페이지 관련 환경을 정의함. 문자 인코딩, 응답 페이지 콘텐츠 타입, 페이지가 내장 세션 객체를 가질 것 인지 여부 등

```
<%@ page import="kr.kosta.util.*" session="false" %>
```

✓ taglib 지시자

- JSP에서 이용 가능한 태그 라이브러리를 정의한다.

```
<%@ taglib tagdir="/WEB-INF/tags/cool" prefix="cool" %>
```

✓ include 지시자

- 변환 시점에 현재 페이지에 포함할 코드나 문서를 정의함. 여러 JSP페이지에 공통으로 포함되어야 하는 코드가 있다면, 별도로 뽑아내어 재사용 가능한 단위로 만든 후 각각의 페이지에서 이를 포함한다.

```
<%@ include file="wickedHeader.html" %>
```


page 지시자 속성

✓ page지시자의 13가지 속성

주요속성	속성	설명
	import	생성될 서블릿 클래스에 추가될 자바 import문을 정의함.
	isThreadSafe	생성될 서블릿 클래스가 SingleThreadModel을 구현할 지 결정함. 디폴트 값은 true
	contentType	MIME 타입과 문자 인코딩을 설정.
	isELIgnored	페이지를 서블릿으로 전환할 때 EL 표현식을 무시할 것인지를 결정.
	isErrorPage	현재 페이지가 JSP 오류 처리용 페이지인지를 정의함. 디폴트 값은 false. true면 내장 예외사항 객체 를 사용할 수 있음
	errorPage	이 페이지에서 잡지 못한 예외사항을 보낼 오류 페이지 URL을 정의함. true라는 속성이 설정되어야 한다.
	language	스크립팅 언어를 정의. 현재는 java가 유일. 확장성을 위해 만들어 놓은것으로 추정
	extends	JSP가 상속할 부모 클래스를 정의함. 이값을 설정하면, 컨테이너가 제공하는 클래스 계층 구조를 재정의함
	session	내장 session 객체를 가질지 여부를 결정함. 디폴트 값은 true
	autoFlush	자동적으로 버퍼링된 출력을 비울 것인지를 결정함. 디폴트 값은 true
	info	변환될 페이지에 포함할 String 값을 기술함. 이 값은 나중에 getServletinfo() 메소드로 접근 가능
	buffer	내장 out 객체 버퍼링을 어떻게 관리할지 결정
	pageEncoding	JSP 문자 인코딩을 정의함.

표현식 (expressions)

- ✓ JSP 페이지 내에서 직접 클라이언트에 출력할 내용을 포함할 수 있다.
- ✓ `<%= Counter.getCount() %>`
- ✓ 표현식의 내용은 `out.print()`의 매개변수가 된다.
- ✓ 문장의 마지막에 세미콜론(;)은 붙이지 않음
- ✓ 리턴 타입이 `void`일 경우 표현식을 사용할 수 없음

표현식 코드

```
<%= Counter.getCount() %>
```

컨테이너에 의해
아래와 같이 변경된다.

스크립틀릿 코드

```
<% out.print(Counter.getCount()); %>
```

JSP 선언문 (declarations)

✓ 변수 선언문

```
<%! int count=0; %>
```

<Servlet code>

```
public class basicCounter_jsp extends SomeSpecialHttpServlet {
```

```
    int count=0;
```

```
    public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException {  
        PrintWriter out = response.getWriter();
```

```
        ...
```

```
    }
```

```
}
```

✓ 메소드 선언문

```
<%! int doubleCount() {  
    count = count*2;  
    return count;  
}  
%>
```

<Servlet code>

```
public class basicCounter_jsp extends SomeSpecialHttpServlet {
```

```
    int doubleCount() {  
        count = count*2;  
        return count;  
    }
```

```
    public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException {  
        PrintWriter out = response.getWriter();
```

```
        ...
```

```
    }
```

```
}
```

주석처리

✓ HTML 주석

- `<!-- Comment -->`
- 이 주석의 내용을 브라우저로 표시함
- 브라우저는 HTML 주석으로 인지
- 사용자가 [소스보기]를 통해 확인할 수 있음

✓ JSP 주석

- `<%-- Comment -->`
- 서블릿으로 변환될 때 자바 소스 코드 주석으로 변환
- 사용자가 [소스보기]를 통해 확인할 수 없음



4. JSP

4.1 JSP 기본

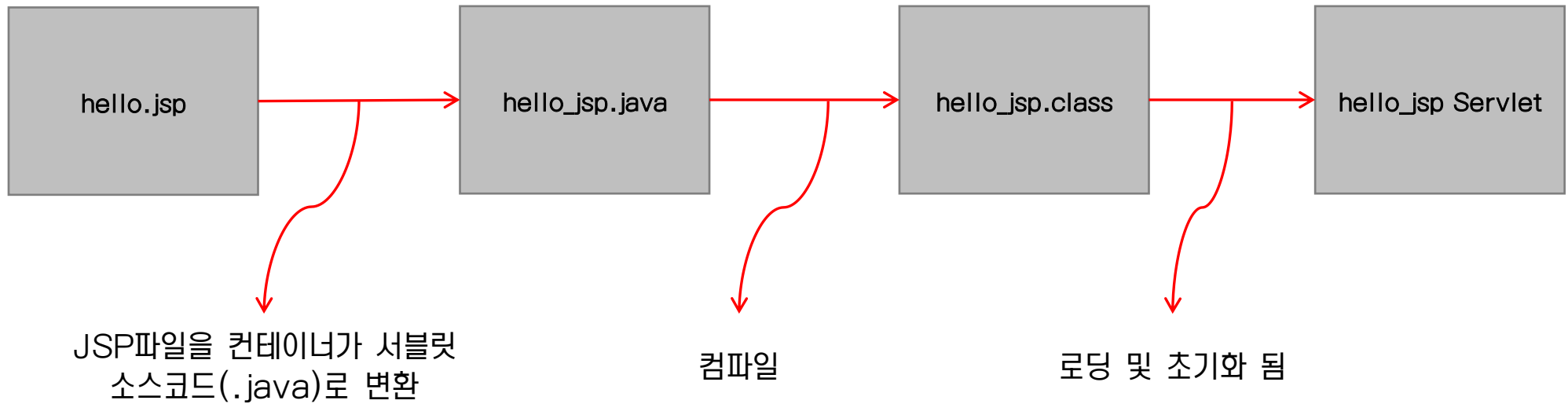
4.2 JSP 태그

4.3 JSP 동작원리

- Servlet으로 변환되는 JSP
- JSP 라이프사이클
- 영역객체와 속성

Servlet으로 변환되는 JSP (1/6)

- ✓ JSP는 컨테이너에 의하여 웹 애플리케이션에서 동작하는 서블릿으로 변환된다.
- ✓ 이러한 변환과 컴파일은 최초요청 시 딱 한번만 처리된다.
 - 대부분의 컨테이너는 컨테이너가 구동될 때 미리 처리할 수 있는 옵션을 제공한다.



서블릿으로 변환되는 JSP (2/6)

✓ JSP가 컨테이너에 의해서 서블릿으로 변환된 모습

- 모든 스크립틀릿 코드와 표현식 코드는 service() 메소드 안으로 들어감
- 스크립틀릿 안에 선언된 변수는 모두 지역변수

```
<HTML><BODY>
방문횟수 : <%= Counter.getCount() %>
</BODY></HTML>
```

```
public class visitCount_jsp extends HttpServlet {
    public void _jspService(HttpServletRequest request,
                            HttpServletResponse response)
                            throws java.io.IOException, ServletException{

        PrintWriter out = response.getWriter();
        response.setContentType( "text/html" );
        out.write( "<HTML><BODY>" );
        out.write( "방문횟수 : " );
        out.print(Counter.getCount());
        out.write( "</BODY></HTML>" );

    }
}
```

서블릿으로 변환되는 JSP (3/6)

✓ 선언문으로 정의된 변수와 메소드

```
<HTML><BODY>
<%!
    int doubleCount() {
        count = count * 2;
        return count;
    }
%>
<%! int count = 1; %>
방문횟수 : <%= Counter.getCount() %>
</BODY></HTML>
```

```
public class visitCount_jsp extends HttpServlet {
    int doubleCount() {
        count = count * 2;
        return count;
    }
    int count = 1;
    public void _jspService(HttpServletRequest request,
                               HttpServletResponse response)
                               throws java.io.IOException, ServletException{

        PrintWriter out = response.getWriter();
        response.setContentType( "text/html" );
        out.write( "<HTML><BODY>" );
        out.write( "방문횟수 : " );
        out.print(Counter.getCount());
        out.write( "</BODY></HTML>" );

    }
}
```

서블릿으로 변환되는 JSP (4/6)

✓ 컨테이너가 JSP를 Servlet으로 변환하는 과정

- 지시자가 있는지 확인하고 변환 작업시 필요한 정보를 로딩
- HttpServlet의 하위 클래스 생성 (예 : org.apache.jasper.runtime.HttpJspBase)
- import 속성을 가진 page 지시자가 있으면 클래스 파일에 import문 삽입
- 선언문(declarations)이 있으면 멤버변수 선언 위치에 삽입
- service() 메소드 생성 – 실제 이름은 _jspService() (서블릿의 부모 클래스에 정의된 service()메소드에서 _jspService()를 호출)
- 템플릿 텍스트 (HTML, 스크립틀릿, 표현식)를 Response 객체의 출력 스트림인 PrintWriter의 출력 형식에 맞게 삽입

서블릿으로 변환되는 JSP (5/6)

- ✓ `_jspService()`에 선언된 내장객체
 - 컨테이너가 `_jspService()`를 생성하면서 JSP에 작성된 코드 외에 내장객체를 자동으로 선언하고 초기화

내장객체 변수명	내장 객체의 타입	설 명
request	<code>javax.servlet.HttpServletRequest</code> (<code>javax.servlet.http.HttpServletRequest</code>)	클라이언트의 HTTP 요청을 담는 객체
response	<code>javax.servlet.HttpServletResponse</code> (<code>javax.servlet.http.HttpServletResponse</code>)	요청에 대한 응답 객체
pageContext	<code>javax.servlet.jsp.PageContext</code>	페이지 실행에 필요한 컨텍스트 정보를 담는 객체
session	<code>javax.servlet.http.HttpSession</code>	클라이언트 세션 정보를 담는 객체
application	<code>javax.servlet.ServletContext</code>	동일한 어플리케이션 컨텍스트의 모든 페이지가 공유할 데이터를 담는 객체
out	<code>javax.servlet.jsp.JspWriter</code>	응답 전송을 위한 출력 스트림
config	<code>javax.servlet.ServletConfig</code>	해당 페이지 서블릿 설정 데이터(초기화 정보)를 담는 객체
page	<code>javax.servlet.jsp.HttpJspPage</code>	해당 페이지 서블릿 인스턴스
exception	<code>java.lang.Throwable</code>	서블릿 실행 시 발생한 예외 중 처리하지 못한 객체

서블릿으로 변환되는 JSP [6/6]

- ✓ JSP가 변환된 서블릿에는 몇 가지 주요 API가 구현되어 있다.
- ✓ `jspInit()`
 - 서블릿의 `init()` 메소드에서 호출
 - JSP에서 선언문을 통해 재정의할 수 있다.
- ✓ `jspDestroy()`
 - 서블릿의 `destroy()` 메소드에서 호출
 - JSP에서 선언문을 통해 재정의할 수 있다.
- ✓ `_jspService()`
 - 서블릿의 `service()` 메소드에서 호출
 - 각 요청마다 새로운 스레드로 실행됨
 - 재정의할 수 없다.

JSP 라이프사이클 [1/2]

✓ 웹 애플리케이션 배포

- 웹 컨테이너는 배포된 애플리케이션의 DD(web.xml)파일을 로딩
- 이 단계에서 JSP는 아무 일도 하지 않고 서버에 그대로 존재함

✓ 사용자가 JSP 요청

- 컨테이너는 요청에 해당하는 JSP를 .java 파일로 변환
- JSP 문법에 오류가 있다면 이 시점에 오류 발생

✓ 변환된 Java 파일 컴파일

- 앞 단계에서 변환된 .java 파일을 .class로 컴파일
- Java 문법에 오류가 있다면 이 시점에 오류 발생

✓ 서블릿 로딩

- 컨테이너가 앞 단계에서 컴파일된 서블릿 클래스(.class)를 메모리로 로딩

✓ 서블릿 인스턴스화

- 컨테이너가 로딩한 서블릿을 인스턴스화
- jspInit() 메소드 호출
- 인스턴스화 된 객체는 클라이언트 요청을 처리할 수 있는 서블릿이 됨

JSP 라이프사이클 (2/2)

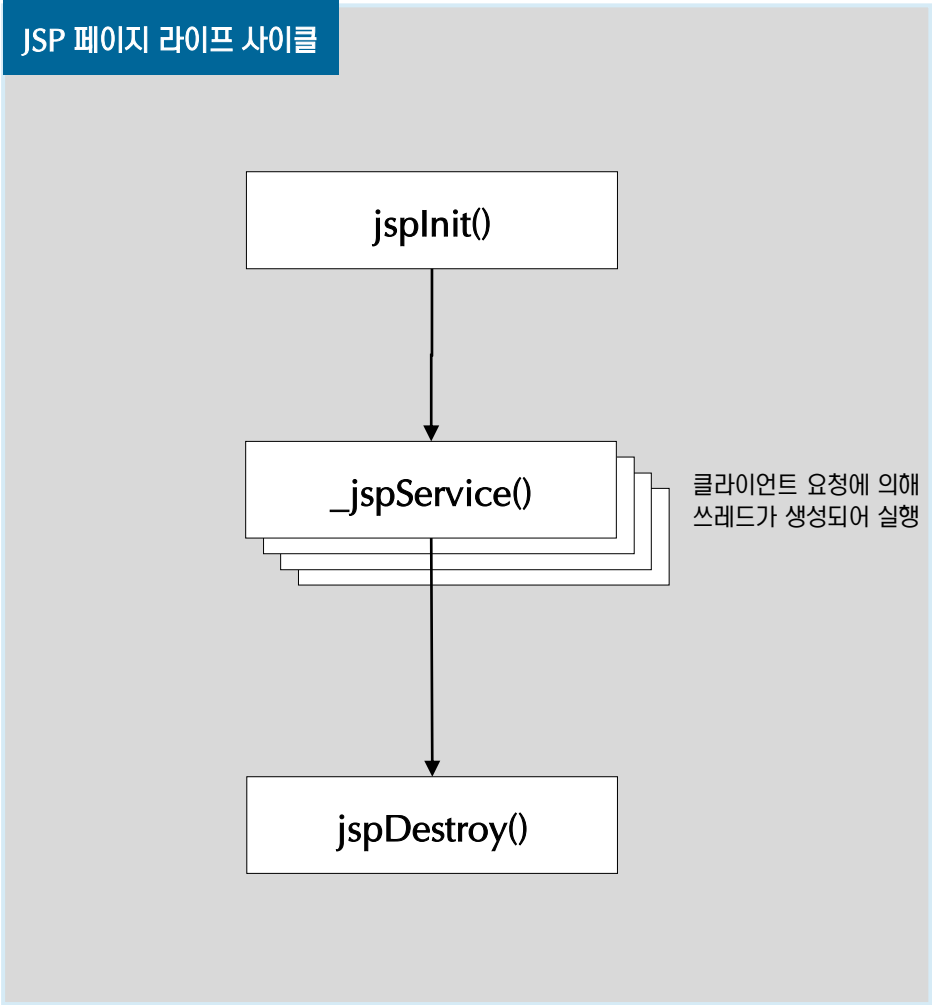
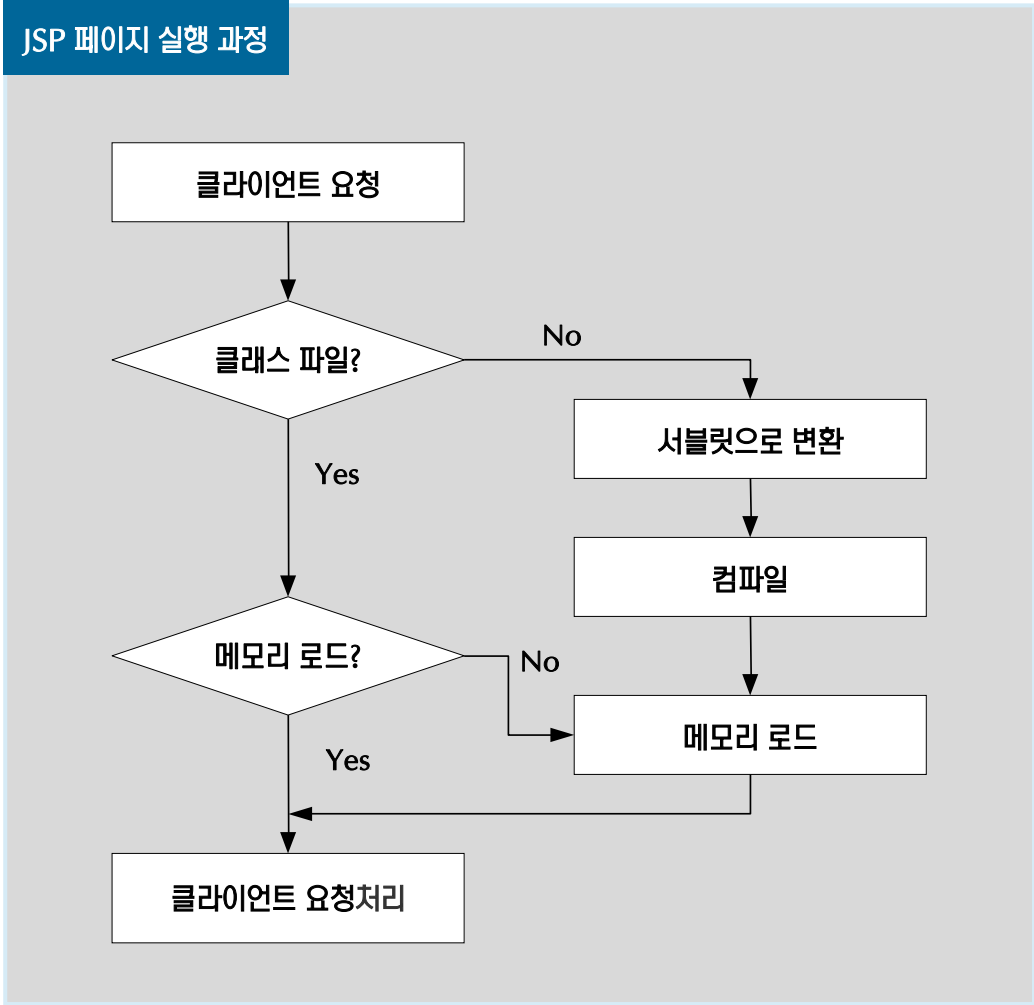
✓ 스레드 생성하여 요청 처리

- 요청이 들어올 때 마다 컨테이너는 새로운 스레드를 만들어 `_jspService()` 메소드 실행
- 다음 요청부터는 앞의 컴파일 과정은 생략하고 일반 서블릿과 같이 동작

✓ 클라이언트로 처리 결과 응답

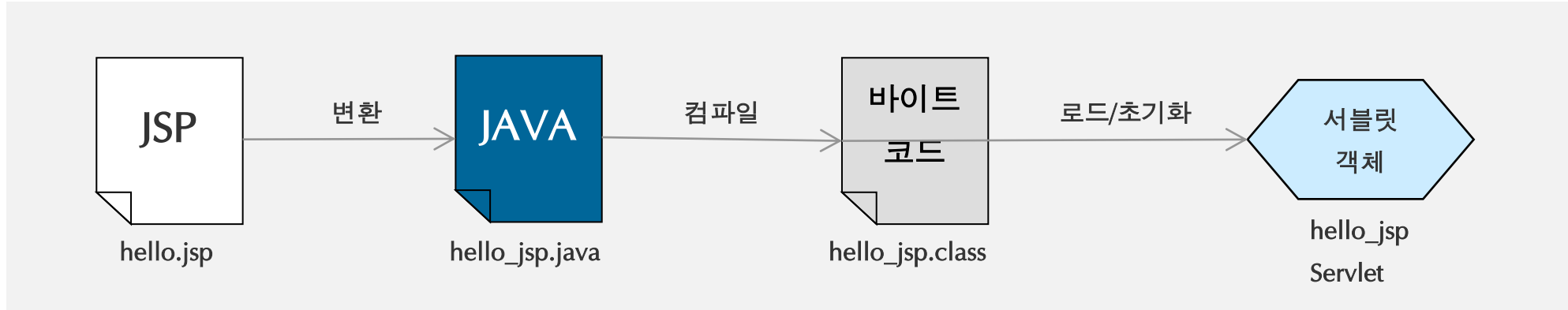
- 클라이언트 또는 다른 웹어플리케이션 컴포넌트로 요청을 넘길 수도 있음

JSP 실행 과정



JSP 실행 과정

- ✓ JSP 페이지는 서블릿으로 변환되어 실행된다.
- ✓ 서블릿으로 변환하는 것은 컨테이너의 역할이다.



```
public class hello_jsp extends HttpServlet {
    public void _jspService(HttpServletRequest request, HttpServletResponse response) throws
        java.io.IOException, ServletException {

        PrintWriter out = response.getWriter();
        response.setContentType("text/html; charset=KSC5601");
        out.write("<html><head><title>Hello JSP</title></head><body>");
        out.println("이 줄은 자바 프로그램에서");
        out.println("만들어내는 내용입니다.");
        out.write("<hr>");
        out.write("이곳은 HTML 부분입니다.");
        out.write("</body></html>");
    }
}
```

JSP 내장 속성 [1/2]

✓ 내장객체 session, request, application 객체는 각 속성들의 유효한 범위를 갖는다.

Scope	서블릿	JSP(내장 객체를 사용)
Application	<code>getServletContext().setAttribute("foo", barObj);</code>	<code>application.setAttribute("foo", barObj);</code>
Request	<code>request.setAttribute("foo", barObj);</code>	<code>request.setAttribute("foo", barObj);</code>
Session	<code>request.getSession().setAttribute("foo", barObj);</code>	<code>session.setAttribute("foo", barObj);</code>
Page	제공하지 않는다.	<code>pageContext.setAttribute("foo", barObj);</code>

JSP 내장 속성 (2/2)

- ✓ 내장 객체 중 영역 객체의 속성을 설정 및 조회하는 메소드
- ✓ `pageContext`, `request`, `session`, `application`들이 동일하게 정의하고 있음

메소드명	속성 유효 범위
<code>public Object getAttribute(String key)</code>	key값으로 등록된 속성을 Object로 리턴 해당 key값이 존재하지 않을 경우 null 리턴
<code>public void setAttribute(String key, Object value)</code>	속성값을 key값으로 등록
<code>public Enumeration getAttributeNames()</code>	해당 영역에 지정된 속성들의 이름을 Enumeration으로 리턴 해당 key값이 존재하지 않을 경우 null 리턴
<code>public void removeAttribute(String key)</code>	key 값에 해당하는 해당 영역의 속성을 삭제

PageContext로 속성 접근하기(1/2)

- ✓ pageContext는 자신에 설정되어 있는 page 범위 속성뿐만 아니라, 어떤 다른 생존 범위 속성도 접근 할 수 있을 정도로 막강하다.
- ✓ page 생존범위의 속성 세팅하기

```
<% Float one = new Float(42.5); %>  
<% pageContext.setAttribute("foo", one); %>
```

- ✓ page 생존범위의 속성 읽기

```
<%= pageContext.getAttribute("foo"); %>
```

- ✓ pageContext를 이용하여 session 생존범위 속성 세팅하기

```
<% Float one = new Float(42.5); %>  
<% pageContext.setAttribute("foo", two, pageContext.SESSION_SCOPE); %>
```

- ✓ pageContext를 이용하여 session 생존범위 속성 읽기

```
<%= pageContext.getAttribute("foo", pageContext.SESSION_SCOPE) %>  
(이것과 동일한 것 : <%= session.getAttribute("foo") %>
```

PageContext로 속성 접근하기(2/2)

- ✓ pageContext를 이용하여 application 생존범위 속성 읽기

```
<%= pageContext.getAttribute("mail", PageContext.APPLICATION_SCOPE) %>  
(이것과 동일한 것 : <%= application.getAttribute("foo") %>
```

- ✓ pageContext를 이용하여 어떤 생존 범위인지 모르는 속성 찾기

- “foo”라는 이름의 속성이 page 생존범위에 없다면, 가장 작은 범위에서부터 시작해서 가장 넓은 범위의 순으로 속성을 찾는다.

```
<%= pageContext.findAttribute("foo"); %>
```



4. JSP

4.1 JSP 기본

4.2 JSP 태그

4.3 JSP 동작원리

4.4 표준액션

4.5 EL(Expression Language)

4.6 레이아웃 템플릿

- 자바 빈 관련 표준 액션

표준 액션 – 자바빈 (1/4)

✓ JSP는 스크립팅을 사용하지 않고 자바빈의 프로퍼티를 출력할 수 있는 방법을 제공한다.

✓ Scriptlet을 이용하는 방법

```
<html><body>  
<% foo.Person p = (foo.Person) request.getAttribute("person"); %>  
Person is : <%= p.getName() %>  
</body></html>
```

✓ 표준 액션을 사용하는 방법(Scriptlet 사용 안함)

```
<html><body>  
<jsp:useBean id="person" class="foo.Person" scope="request" />  
Person created by servlet: <jsp:getProperty name="person" property="name" />  
</body></html>
```


표준 액션 – 자바빈 (2/4)

✓ <jsp:useBean>은 빈을 선언하고 초기화하는 태그

```
<jsp:useBean id="person" class="foo.Person" scope="request" />
```

- jsp:useBean : 표준 액션임을 나타냄
- id="person" : 빈 객체 식별자를 선언한다.
- class="foo.Person" : 빈 객체의 클래스 타입을 선언
- scope="request" : 빈 객체 속성의 생존범위 지정

✓ <jsp:getProperty>은 속성 빈 프로퍼티를 읽어오는 태그

```
<jsp:getProperty name="person" property="name" />
```

- jsp:getProperty : 표준 액션임을 나타냄
- name : 빈 객체 이름
- property="name" : 빈의 프로퍼티 명(getter/setter)

✓ <jsp:setProperty> 은 속성 빈 프로퍼티를 설정하는 태그

```
<jsp:setProperty name="person" property="name" value="Fred" />
```


표준 액션 – 자바빈 (3/4)

- ✓ <jsp:useBean>은 “person” 이라는 속성 객체를 찾다가 못 찾으면, 하나 만든다.

```
<jsp:useBean id="person" class="foo.Person" scope="request" />
```

<Servlet code>

// id에 있는 값을 가지고 변수를 선언한다.

```
foo.Person person = null;
```

```
synchronized (request) {
```

//태그에 정의된 생존범위(scope)에서 속성을 찾아본다.

```
person = (foo.Person)_jspx_page_context.getAttribute("person", PageContext.REQUEST_SCOPE);
```

//해당 생존범위에 이런 이름을 가진 속성이 없다면...

```
if(person == null) {
```

//객체를 생성해 person에 할당한다.

```
person = new foo.Person();
```

//마지막으로 방금 만든 객체를 태그에서 정의한 생존범위에 설정 한다.

```
_jspx_page_context.setAttribute("person", person, PageContext.REQUEST_SCOPE);
```

```
}
```

```
}
```

표준 액션 – 자바빈 (4/4)

✓ <jsp:useBean>은 자신의 몸체를 가질 수 있다.

- 새로운 빈이 만들어지는 경우에만 프로퍼티를 설정하고, 이미 빈이 존재한다면, 태그의 몸체 부분은 건너 뛴다.

```
<jsp:useBean id="person" class="foo.Person" scope="page" >  
  <jsp:setProperty name="person" property="name" value="Fred" />  
</jsp:useBean>
```

✓ 빈 표준 액션의 단점

- 표현식을 사용하면 중첩된 객체의 프로퍼티를 출력할 수 있다.

```
<html><body>  
<%= ((foo.Person) request.getAttribute("person")).getDog().getName() %>  
</body></html>
```

- 그러나, 빈 표준 태그는 중첩된 프로퍼티를 출력할 수 없다.
- Dog 객체의 name 속성 값을 출력할 방법이 없다.

```
<html><body>  
<jsp:useBean id="person" class="foo.Person" scope="request" />  
Dog's name is : <jsp:getProperty name="person" property="dog" />  
</body></html>
```



4. JSP

4.1 JSP 기본

4.2 JSP 태그

4.3 JSP 동작원리

4.4 표준액션

4.5 EL(Expression Language)

4.6 레이아웃 템플릿

- EL (Expression Language)
- EL의 [] 연산자
- EL 내장객체
- EL 함수
- 연산자

EL (Expression Language)

- ✓ EL은 표현식 언어(Expression Language)의 줄임말이다.
- ✓ JSP 2.0 스펙부터 공식적으로 포함
- ✓ 스크립트릿이나 표현식으로 지금까지 했던 작업을 EL로 간단하게 할 수 있음
- ✓ EL 표현식

```
<syntax>  
${person.dog.name}
```

```
<스크립팅>  
<%= ((foo.Person) request.getAttribute("person")).getDog().getName() %>
```

- ✓ 표현식에서 도트 연사자 왼쪽은 반드시 맵 또는 빈이어야 한다.

```
${person.name} // java.util.Map 또는 java bean
```

- ✓ 표현식에서 도트 연산자 오른쪽은 반드시 맵의 키이거나 빈 프로퍼티여야 한다.

```
${person.name} // 빈프로퍼티(getName(), setName())
```

- ✓ 오른쪽 오는 값은 식별자로서 일반적인 자바 명명 규칙을 따라야 한다.

```
${person.name} // 문자, _, $로 시작해야 함, 두 번째 글자부터 숫자를 써도 무방, 자바 예약어 사용금지
```

EL의 [] 연산자 (1/3)

✓ EL의 [] 연산자

```
${person["name"]} = ${person.name}
```

✓ [] 연산자의 왼편에는 맵, 빈, 배열, 리스트 변수가 올 수 있다.

```
${musicList["something"]}
```

✓ [] 연산자 안의 값이 문자열(따옴표로 묶여 있다면) 이라면, 이것은 맵 키가 될 수 있고, 빈 프로퍼티 또는 리스트나 배열 인덱스가 될 수 있다.

```
${musicList["something"]}
```

✓ 배열과 리스트인 경우 문자로 된 인덱스 값은 숫자로 바뀐다.

```
<servlet code>
String[] favoriteMusic = {"Zero 7", "Tahiti 80", "BT", "Frou Frou"};
request.setAttribute("musicList", favoriteMusic);
```

```
${musicList[0]} // Zero 7 출력
```

```
${musicList["1"]} //Tahiti 80
```


EL의 [] 연산자 (2/3)

- ✓ EL의 [] 연산자 안에 따옴표가 없다면 컨테이너는 안에 들어 있는 것이 뭔지 찾기 시작한다.
이런 이름으로 바인딩된 속성이 있으면 속성값을 [] 안에 넣는다. (똑같은 이름의 내장 객체가 있다면 , 내장 객체가 항상 우선한다.)

```
<servlet code>
java.util.Map musicMap = new java.util.HashMap();
musicMap.put("Ambient", "Zero 7");
musicMap.put("Surf", "Tahiti 80");
musicMap.put("DJ", "BT");
musicMap.put("Indie", "Frou Frou");

request.setAttribute("musicMap", musicMap);

request.setAttribute("Genre", "Ambient");
```

```
${musicMap[Genre]} =====> ${musicMap["Ambient"]} //return Zero 7
```

```
${musicMap["Genre"]} =====> ${musicMap["Genre"]} //return null
```


EL의 [] 연산자 (3/3)

✓ EL의 [] 연산자 안에 내장 표현식을 쓸 수 있다.

- 표현식을 실행할 때 안쪽에서 바깥쪽의 순서로 진행한다.

<servlet code>

```
java.util.Map musicMap = new java.util.HashMap();  
musicMap.put("Ambient", "Zero 7");  
musicMap.put("Surf", "Tahiti 80");  
musicMap.put("DJ", "BT");  
musicMap.put("Indie", "Frou Frou");
```

```
String[] musicTypes = {"Ambient", "surf", "DJ", "Indie"};  
request.setAttribute("MusicType", musicTypes);
```

Music id \${musicMap[MusicType[0]]}



Music is \${musicMap["Ambient"]}



Music is **Zero 7**

EL 내장 객체

✓ pageContext만 빼고 모두 맵이다.

내장 객체명	설명
pageScope requestScope sessionScope applicationScope	생존 범위 속성 맵
param paramValues	요청 파라미터 맵
header headerValues	요청 헤더 맵
cookie	쿠키 맵
initParam	컨텍스트 초기화 파라미터 맵
pageContext	pageContext 객체에 대한 참조. 맵이 아닌 유일한 빈

EL 내장 객체

✓ EL에서 요청 파라미터 사용

- 해당 파라미터 이름으로 값이 하나 밖에 없을 때 EL 내장 객체인 param을 사용하면 쉽게 그 값을 읽을 수 있다.
- 파라미터 이름으로 그 값이 하나 이상일 때는 param으로는 안되고 paramValues를 사용해야 한다.

```
<html code>
<form action="TestBean.jsp">
  Name : <input type="text" name="name">
  ID# : <input type="text" name="empID">

  First food : <input type="text" name="food">
  Second food : <input type="text" name="food">

  <input type="submit">
</form>
```

```
<jsp code>
Request param name is : ${param.name} <br>
Request param empID is : ${param.empID} <br>
Request param food is : ${param.food} <br>

First food request param : ${paramValues.food[0]} <br>
Second food request param : ${paramValues.food[1]} <br>

Request param name : ${paramValues.name[0]}
```

EL 내장 객체

✓ “host” 헤더 정보 읽기

- 스크립팅 코드

```
Host is : <%= request.getHeader("host") %>
```

- EL 내장 객체 header를 사용

```
Host is : ${header["host"]}, Host is : ${header.host}
```

✓ HTTP 요청 메소드 확인

- 스크립팅 코드

```
Method is : <%= request.getMethod() %>
```

- EL 사용

```
Method is : ${request.method} //오류 , Method is : ${requestScope.method} //오류
```

EL 내장 객체

✓ requestScope 와 request 객체

- requestScope : request 생존범위에 묶여 있는 속성에 대한 단순한 맵
- request 정보는 pageContext를 통해서 접근 할 수 있다.

```
Method is : ${pageContext.request.method} //pageContext 안에 request가 있고 request 안에 method가 있다.
```

✓ naming conflict(이름 충돌) 가 생존범위 안에서 의심 되면 명시적으로 표현한다.

```
${requestScope.person.name}
```

✓ 속성 이름이 문자열이라면

```
request.setAttribute("foo.person", p); //속성을 문자열로 입력한경우
```

```
${foo.person.name} //foo라는 속성은 존재 하지 않는다.
```

```
${requestScope["foo.person"].name} //[] 이용해서 접근 할 수 있다.
```

EL 내장 객체

✓ 쿠키 값 출력

- 스크립팅

```
<% Cookie[] cookies = request.getCookies();  
for(Cookie cookie : cookies) {  
    if(cookie.getName().equals("userName")) {  
        out.println(cookie.getValue());  
    }  
} %>
```

- EL Cookie 내장 객체 사용

```
${cookie.userName.value}
```

✓ 컨텍스트 초기화 파라미터 값 출력

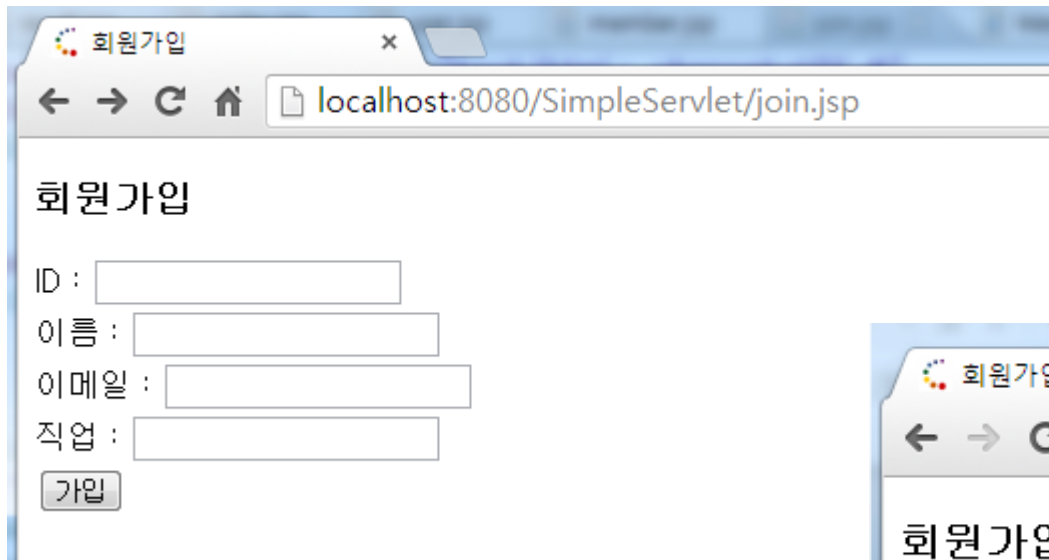
```
<web.xml>  
<context-param>  
    <param-name>mainEmail</param-name>  
    <param-value>likewecare@wickedlysmart.com</param-value>  
</context-param>
```

email is : <%= application.getInitParameter("mainEmail") %> // 스크립팅

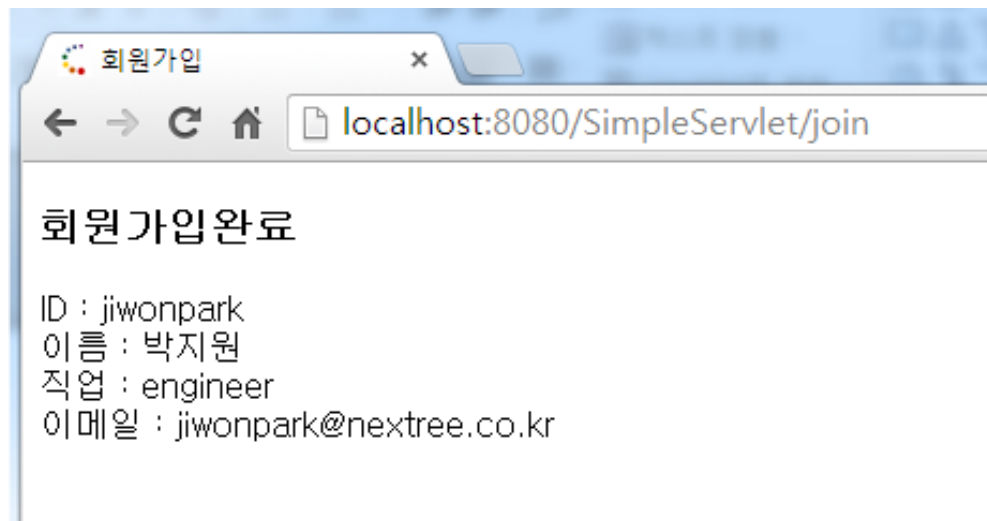
email is : \${initParam.mainEmail} //EL표현식

ET 실습(1/5)

- ✓ 회원가입 화면 구현
- ✓ 화면에 입력한 회원정보를 dispatch 된 화면(가입완료 화면)에서 다시 보여줌



A screenshot of a web browser window showing a form titled '회원가입' (Member Registration). The browser's address bar displays 'localhost:8080/SimpleServlet/join.jsp'. The form contains four input fields: 'ID :', '이름 :', '이메일 :', and '직업 :'. Below these fields is a button labeled '가입' (Join).



A screenshot of a web browser window showing a confirmation screen titled '회원가입완료' (Member Registration Complete). The browser's address bar displays 'localhost:8080/SimpleServlet/join'. The screen lists the registered information: 'ID : jiwonpark', '이름 : 박지원', '직업 : engineer', and '이메일 : jiwonpark@nextree.co.kr'.

EL 실습(2/5)

✓ join.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>회원가입</title>
</head>
<body>

<h3>회원가입</h3>
<form action="/SimpleServlet/join" method="POST">
    ID : <input type="text" name="id" /> <br>
    이름 : <input type="text" name="name" /> <br>
    이메일 : <input type="text" name="email" /> <br>
    직업 : <input type="text" name="job" /> <br>
    <input type="submit" value="가입" />
</form>

</body>
</html>
```

ET 실습(3/5)

✓ Member.java

```
public class Member {  
  
    private String id;  
    private String name;  
    private String job;  
    private String email;  
  
    public void setId(String id) {  
        this.id = id;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public void setJob(String job) {  
        this.job = job;  
    }  
    public String getId() {  
        return id;  
    }  
    public String getName() {  
        return name;  
    }  
    public String getJob() {  
        return job;  
    }  
    public String getEmail() {  
        return email;  
    }  
    public void setEmail(String email) {  
        this.email = email;  
    }  
}
```

EL 실습(4/5)

✓ JoinServlet.java

```
public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
    req.setCharacterEncoding("UTF-8");

    String id = req.getParameter("id");
    String name = req.getParameter("name");
    String job = req.getParameter("job");
    String email = req.getParameter("email");

    Member member = new Member();
    member.setId(id);
    member.setName(name);
    member.setJob(job);
    member.setEmail(email);

    req.setAttribute("member", member);
    RequestDispatcher view = req.getRequestDispatcher("/joinResult.jsp");
    view.forward(req, resp);
}
```

EL 실습(5/5)

✓ joinResult.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isELIgnored="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>회원가입</title>
</head>
<body>

<h3>회원가입완료</h3>
ID : ${member.id}<br>
이름 : ${member.name}<br>
직업 : ${member.job}<br>
이메일 : ${member.email}<br>
</body>
</html>
```

EL 함수(1/2)

✓ 태그 라이브러리를 구현하여 EL표현식으로 자바 메소드를 사용할 수 있다.

✓ 사용방법

- 정적인 공용 메소드를 제공하는 클래스 작성. 반드시 public static 메소드여야 한다. void메소드는 가능하나 리턴이 없으므로 무의미함
- 태그 라이브러리 서술자 파일을 만든다.
- JSP에 taglib 지시자를 코딩한다.
- 함수를 호출하는 EL을 작성한다.

✓ 메소드를 정의한 클래스 작성

```
<java>
package foo;

public class DiceRoller {
    public static int rollDice() {
        return (int) ((Math.random() * 6) + 1);
    }
}
```


EL 함수(2/2)

✓ 태그 라이브러리 서술자(TLD) 파일 작성

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

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd" version="2.0">

<tlib-version>1.2</tlib-version>
<uri>DiceFunctions</uri>
  <function>
    <name>rollIt</name>
    <function-class>foo.DiceRoller</function-class>
    <function-signature>int rollDice()</function-signature>
  </function>
</taglib>
```

✓ JSP에 taglib 지시자를 코딩 및 함수 호출 EL 작성

```
<%@ taglib prefix="mine" uri="DiceFunctions"%>
<html><body>
  ${mine:rollIt()}
</body></html>
```

연산자

✓ 산술연산자

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

✓ 논리연산자

- &&, and, ||, or, !, not

✓ 관계연산자

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

```
String num="2"; ${num > 3} = false, ${integer le 12} = true, ${42 div 0} = infinity
```

✓ 그 밖에 연산자들

- empty
- true
- false
- null



4. JSP

4.1 JSP 기본

4.2 JSP 태그

4.3 JSP 동작원리

4.4 표준액션

4.5 EL(Expression Language)

4.6 레이아웃 템플릿

- include 지시자
- <jsp:include> 표준 액션

include 지시자

- ✓ include 지시자는 포함할 파일 내용을 모두 복사하여 선언한 지점에 삽입한다. (포함 후 실행)

```
<header.jsp>  
 <br>  
<em><strong>We know how to make SOAP suck less.</strong></em> <br>
```

```
<Contact.jsp>  
<html><body>  
<%@ include file="Header.jsp"%>  
<br>  
<em>We can help.</em><br><br>  
Contact us at: ${iParam.mainEmail}  
</body></html>
```

<jsp:include> 표준 액션 (1/2)

- ✓ <jsp:include> 표준 액션도 include 지시자와 똑같은 곳에 둔다. (실행 후 포함)

```
<header.jsp>  
 <br>  
<em><strong>We know how to make SOAP suck less.</strong></em> <br>
```

```
<Contact.jsp>  
<html><body>  
<jsp:include page="Header.jsp" />  
<br>  
<em>We can help.</em><br><br>  
Contact us at: ${iParam.mainEmail}  
</body></html>
```

〈jsp:include〉 표준 액션 (2/2)

✓ <jsp:param/> 태그로 콘텐츠 추가하기

```
<main.jsp>
<html><body>
<jsp:include page="Header.jsp" >
    <jsp:param name="subTitle" value="We take the sting out of SOAP."/>
</jsp:include>
<br>
...
```

```
<Header.jsp>
 <br>
<em><strong>${param.subTitle}</strong></em> <br>
```




5. JSTL

5.1 JSTL 개요

5.2 JSTL 주요 태그

- JSTL 개요

JSTL 개요

✓ JSTL 개요

- JSP 표준 태그 라이브러리
- 스크립트 대신에 반복문, 조건문, 포맷, xml 조작 등을 HTML 태그처럼 사용할 수 있도록 지원한다.
- EL을 이용해서 효과적으로 프리젠테이션 역할을 수행

✓ JSTL 사용법

- JSTL을 사용하기 위해서는 WEB-INF/lib 에 jstl.jar, standard.jar 설치한다.
 - 톰캣에서 제공하는 샘플 프로젝트에 있는 jstl.jar, standard.jar을 찾아서 WEB-INF/lib에서 복사하면 된다.
- jsp 문서 상단에 jsp 태그라이브러리 지시자를 이용하여 jstl을 선언한다.

<JSTL 선언>

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

- JSTL 사용

<JSTL 사용>

```
<c:if test="${empty param.userName}">  
  <jsp:forward page="HandleIt.jsp" />  
</c:if>
```




5. JSTL

5.1 JSTL 개요

5.2 JSTL 주요 태그

- JSTL 코어 라이브러리
- 그 밖의 라이브러리

JSTL – <c:out> 태그

✓ 값을 출력 하기 위한 태그

- 기본 문법

<syntax>

```
<c:out value='${pageContent.rawHTML}' />
```

- escapeXml 속성 : 출력할 값에 브라우저가 해석하는 html등과 같은 값이 포함되어 있을 경우 해석되지 않고 그대로 표현해야 할 경우 사용한다.

<syntax>

```
<c:out value='${pageContent.rawHTML}' escapeXml='false'/> //true: html코드를 변환한다.
```

- default 속성 : 값이 없을 경우 기본 출력 값을 설정 할 수 있다.

<syntax 1.>

```
<c:out value='${user}' default='guest'/> //user가 null이거나 빈값일 경우 guest를 출력한다.
```

<syntax 2.>

```
<c:out value='${user}' > guest</c:out>
```

JSTL – <c:forEach> 태그(1/3)

- ✓ <c:forEach>태그는 컬렉션에 있는 각각의 항목에 대하여 몸체(바디) 부분을 반복한다.
- ✓ 여기서 말하는 컬렉션 : 배열(Array) , 컬렉션(Collection), 맵(Map), 콤마 분리 문자열(Comma-delimited String)
 - 기본 문법

<syntax>

```
<c:forEach var="movie" items="${movieList}" /> // var : 컬렉션에 있는 각각 항목을 나타내는 변수
    ${movie} //items : 루핑을 돌 실제 데이터(배열, 컬렉션, 맵, 콤마 분리 문자열)
</c:forEach>
```

- varStatus 속성 : 루핑 횟수를 알기 위한 속성

<syntax>

```
<c:forEach var="movie" items="${movieList}" varStatus="movieLoopCount" /> //movieLoopCount라는 인스턴스 변수선언
    ${movieLoopCount.count} //루핑이 돌때마다 값이 1씩 증가한다.
</c:forEach>
```

JSTL – <c:forEach> 태그(2/3)

✓ <c:forEach> 안에 <c:forEach>

- servlet code

<servlet code>

```
String[] movies1 = {"Matrix", "Kill Bill", "Boondock Saints"};
String[] movies2 = {"Amelie", "Return of the King", "Mean Girls"};
List movieList = new ArrayList();
movieList.add(movies1);
movieList.add(movies2);
request.setAttribute("movies", movieList);
```

- Jsp 코드

<syntax>

```
<c:forEach var="listElement" items="${movies}" />
    <c:forEach var="movie" items="$listElement" >
        ${movie}
    </c:forEach>
</c:forEach>
```


JSTL – <c:forEach> 태그(3/3)

✓ begin, end, step 속성

- begin, end : 컬렉션의 부분 집합만 루프를 돌릴때 사용한다.
- step : 값을 얼마씩 증가할 것인지를 설정

<syntax>

```
<c:forEach var="comma" items="${commaStr}" varStatus="commaCount" begin="1" end="5" step="2">
```

```
    ${commaCount.count}. ${comma}<br>
```

```
</c:forEach>
```

JSTL – 조건문 태그(1/2)

✓ <c:if>태그는 조건문을 만들때 사용한다.

- 기본 문법

<syntax>

```
<c:if test="${userType eq 'member'}">  
    <jsp:include page="inputComments.jsp"/>  
</c:if>
```

JSTL – 조건문 태그(2/2)

✓ <c:choose>태그는 <c:if> 태그가 표현 못하는 else를 등을 표현 할 수 있다.

- 기본 문법

<syntax>

<c:choose>

<c:when test="*\${userPref == 'performance'}*">

실행 할 내용....

</c:when>

<c:when test="*\${userPref == 'safety'}*">

실행 할 내용...

</c:when>

<c:otherwise>

해당 조건이 맞지 않을 때의 디폴트 실행문

</c:otherwise>

</c:choose>

- <c:otherwise> 태그가 반드시 있어야 하는 것은 아니다.

JSTL – <c:set> 태그(1/3)

✓ <c:set>태그는 속성, 빈, 맵의 값을 설정 할 수 있다.

- 속성 var 설정 하기

<syntax>

//session 생존범위에 “userLevel” 이란 이름의 속성이 없으면 하나를 새로 만든다. (value 속성값이 null이 아니라는 가정하에)

```
<c:set var="userLevel" scope="session" value="Cowboy"/>
```

//\$(person,dog} 의 Dog 객체를 Fido 변수에 설정 한다. Fido는 Dog 타입 변수가 된다.

```
<c:set var="Fido" value="${persion.dog}"/>
```

- 몸체가 있는 경우

<syntax>

```
<c:set var="userLevels" scope="sesssion">
```

Sheriff, Bartender, Cowgirl

```
</c:set>
```

- value가 null인 경우, 변수는 제거 된다.

JSTL – <c:set> 태그(2/3)

- 빈과 맵에 <c:set> 태그 사용하기

<syntax>

```
<c:set target="${Petmap}" property="dogName" value="Clover"/>
```

1. *target*은 null 이어서는 안된다. *target* 속성에는 객체가 들어가야 한다.
2. *target*이 빈일 경우 *property*는 프로퍼티명 이 되고, 맵일 경우 키 값이 된다.

- 몸체가 있는 경우

<syntax>

```
<c:set target="${person}" property="name">  
    ${foo.name}  
</c:set>
```

JSTL – <c:set> 태그(3/3)

✓ <c:set> 핵심 정리

- <c:set>태그에 var와 target을 동시에 사용할 수 없다.
- scope는 옵션이다. 없을 경우 디폴트는 page 생존범위이다.
- value가 null이면, var에 있는 속성은 제거된다.
- var 이름으로 속성이 없으면, 자동적으로 만든다. (value가 null이 아닐 경우)
- target 표현식이 null이면, 컨테이너는 예외사항을 던진다.
- target에는 실제 객체를 표현하는 표현식이 들어가야 한다. 빈이나 맵의 id 이름을 나타내는 문자열을 기입하면, 작동하지 않는다.
- target 표현식이 빈이나 맵이 아니면 컨테이너는 예외사항을 던진다.
- target 표현식이 빈이고, 해당 빈에 property에 명기된 이름으로 된 프로퍼티가 없다면 컨테이너는 예외사항을 던진다.
EL표현식\${bean.notAProperty} 또한 예외 사항을 던진다.

JSTL – <c:remove> 태그

✓ 속성을 제거 하기 위해 사용한다.

<syntax>

```
<c:set var="userStatus" scope="request" value="Brilliant" />
```

```
userStatus: ${userStatus} <br>
```

```
<c:remove var="userStatus" scope="request"/>
```

1. var 속성에는 문자열이 들어간다. (표현식을 넣으면 안된다.)

JSTL – <c:import> 태그

- ✓ 요청이 들어오는 시점에, url 속성에 명기한 파일을 현재 콘텐츠에 포함한다.
- ✓ <jsp:include> 표준 액션과 include 지시자로는 현재 웹 애플리케이션에 있는 페이지만 포함할 수 있지만, <c:import> 태그를 사용하면 컨테이너 외부에 있는 자원도 포함시킬 수 있다.

```
<syntax>  
<c:import url="http://www.nextree.co.kr/sample/sample.html"/>
```

- <c:param> 를 활용한 내용 삽입

```
<syntax>  
<c:import url="Header.jsp">  
    <c:param name="subTitle" value="We take the sting out of SOAP." />  
</c:import>
```

- 포함될 파일(header.jsp)

```
<em> <strong>${param.subTitle}</strong></em>
```

JSTL – <c:url> 태그

✓ 하이퍼링크와 관련된 작업을 할 수 있다.

- URL 재작성

```
<syntax>
<a href="<c:url value='/inputComments.jsp'/>">click here</a>
```

- URL 인코딩

- <c:url> 태그는 URL 재작성은 하지만, URL 인코딩을 하지 않는다.
- <c:url> 몸체에 <c:param>을 사용하면 URI재작성도 하고 URL인코딩도 한다.

```
<syntax>
<c:url value="/inputComments.jsp" var="inputURL">
  <c:param name="firstName" value="${first}" />
  <c:param name="lastName" value="${last}" />
</c:url>
```

JSTL – <c:catch> 태그

- ✓ 페이지에 오류를 낼 만한 태그를 오류페이지로 넘기기 않고 직접 처리 하고 싶을 때 사용한다.

```
<syntax>
<%@ page errorPage="errorPage.jsp" %>
<c:catch>
    <% int x = 10/0; %> // 이 스크립트는 100% 예외사항을 던진다. 하지만 오류 페이지가 아닌 이 페이지에서 처리한다.
</c:catch>
```

- <c:catch> 태그에서 var 속성 사용하기

```
<syntax>
<%@ page errorPage="errorPage.jsp" %>
<c:catch var="myException">
    <% int x = 10/0; %> // 이 스크립트는 100% 예외사항을 던진다. 하지만 오류 페이지가 아닌 이 페이지에서 처리한다.
</c:catch>

<c:if test="{myException != null}">
    There was an exception: ${myException.message} <br>
</c:if>
```

그 밖의 라이브러리

✓ 포매팅 라이브러리

- 국제화 : <fmt:message> <fmt:setLocale> <fmt:bundle> <fmt:setBundle> <fmt:param> <fmt:requestEncoding>
- 포매팅 : <fmt:timeZone> <fmt:setTimeZone> <fmt:formatNumber> <fmt:parseNumber> <fmt:parseDate>

✓ SQL 라이브러리

- 데이터베이스 접근 : <sql:query> <sql:update> <sql:setDataSource> <sql:param> <sql:dateParam>

✓ XML 라이브러리

- 쿼어 XML 액션 : <x:parse> <x:out> <x:set>
- XML 흐름 제어 : <x:if> <x:choose> <x:when> <x:otherwise> <x:forEach>
- 변환 액션 : <x:transform> <x:param>



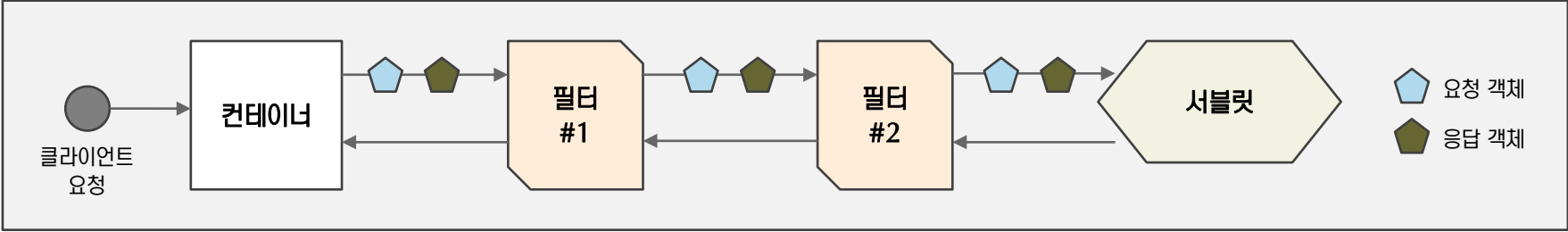
6. Servlet 활용기술

6.1 Servlet Filter

- Servlet Filter 개요
- Filter를 이용한 한글처리

Servlet Filter 개요

- ✓ 서블릿 필터는 서블릿으로 요청이 넘어가기 전에 요청을 가로채어 어떤 처리를 할 수 있다.
 - 필터의 동작 여부를 서블릿은 알 수 없다.



- ✓ 필터가 여러 개인 경우 실행 순서
 - 컨테이너는 DD(web.xml)에 선언한 정보를 바탕으로 필터를 언제 실행할지 결정한다.
- ✓ 필터의 구현
 - javax.servlet.Filter 인터페이스를 구현한다.
- ✓ Request 필터 활용
 - 보안 관련 내용 체크
 - 요청 헤더와 바디 포매팅 수정
 - 요청을 감시하거나 기록을 남김
- ✓ Response 필터 활용
 - 응답 스트림 압축
 - 응답 스트림에 내용을 추가하거나 수정

Filter를 이용한 한글처리 (1/2)

✓ 한글관련 처리에 Filter를 적용한 예

- web.xml 설정
- 모든 URL 요청(/*)에 대해서 필터 적용
- Filter Class의 초기 파라미터 설정

```
web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
  <display-name>HelloServlet</display-name>

  <servlet>
    <servlet-name>helloservlet</servlet-name>
    <servlet-class>kr.kosta.servlet.HelloServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>helloservlet</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>

  <filter>
    <filter-name>EncodingFilter</filter-name>
    <filter-class>kr.kosta.filter.CharacterEncoding</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>EncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Filter를 이용한 한글처리 (2/2)

✓ Filter Class 구현

✓ init()

- Filter가 등록될 때 호출 (Server Start)
- web.xml에서 정의한 파라미터 사용

✓ destory()

- Filter가 해제될 때 호출 (Server Stop)

✓ doFilter()

- Filter 동작 구현
- Filter Chain에 등록된 다른 Filter 호출

```
CharacterEncoding.java X
1 package kr.kosta.filter;
2
3 import java.io.IOException;
11
12 public class CharacterEncoding implements Filter {
13
14     private String encoding;
15
16     @Override
17     public void destroy() {
18         System.out.println("[filter destroy]");
19     }
20
21     @Override
22     public void doFilter(ServletRequest req, ServletResponse resp,
23                         FilterChain filter) throws IOException, ServletException {
24         System.out.println("[doFilter]");
25         req.setCharacterEncoding(encoding);
26         resp.setContentType("text/html; charset="+encoding);
27         filter.doFilter(req, resp);
28     }
29
30     @Override
31     public void init(FilterConfig config) throws ServletException {
32         encoding = config.getInitParameter("encoding");
33         System.out.println("[filter init] encoding : " + encoding);
34     }
35
36 }
37
38
39
40
```



7. 웹 프로그래밍 전략

7.1 Servlet & JSP

7.2 MVC 패턴

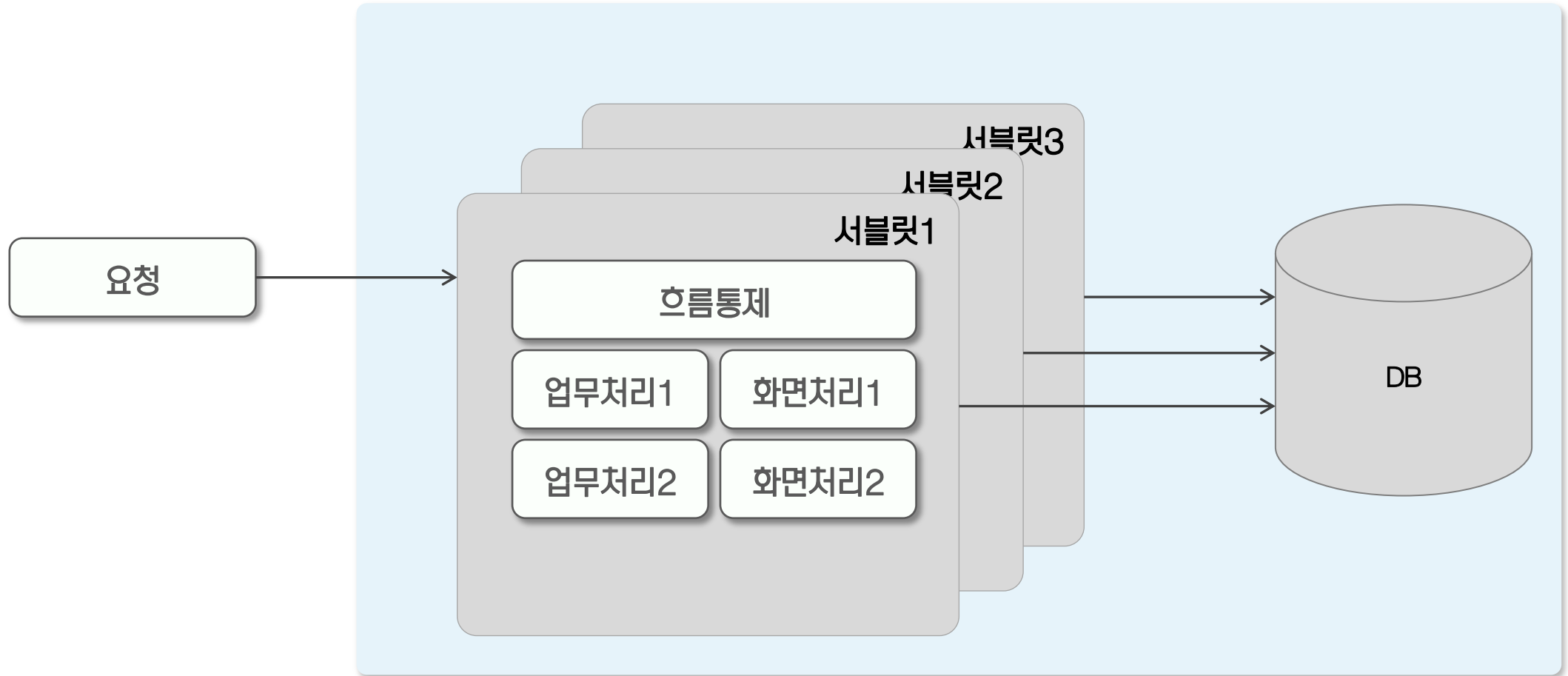
7.3 웹 프레임워크 구조

7.4 프레임워크 소개

- Servlet & JSP

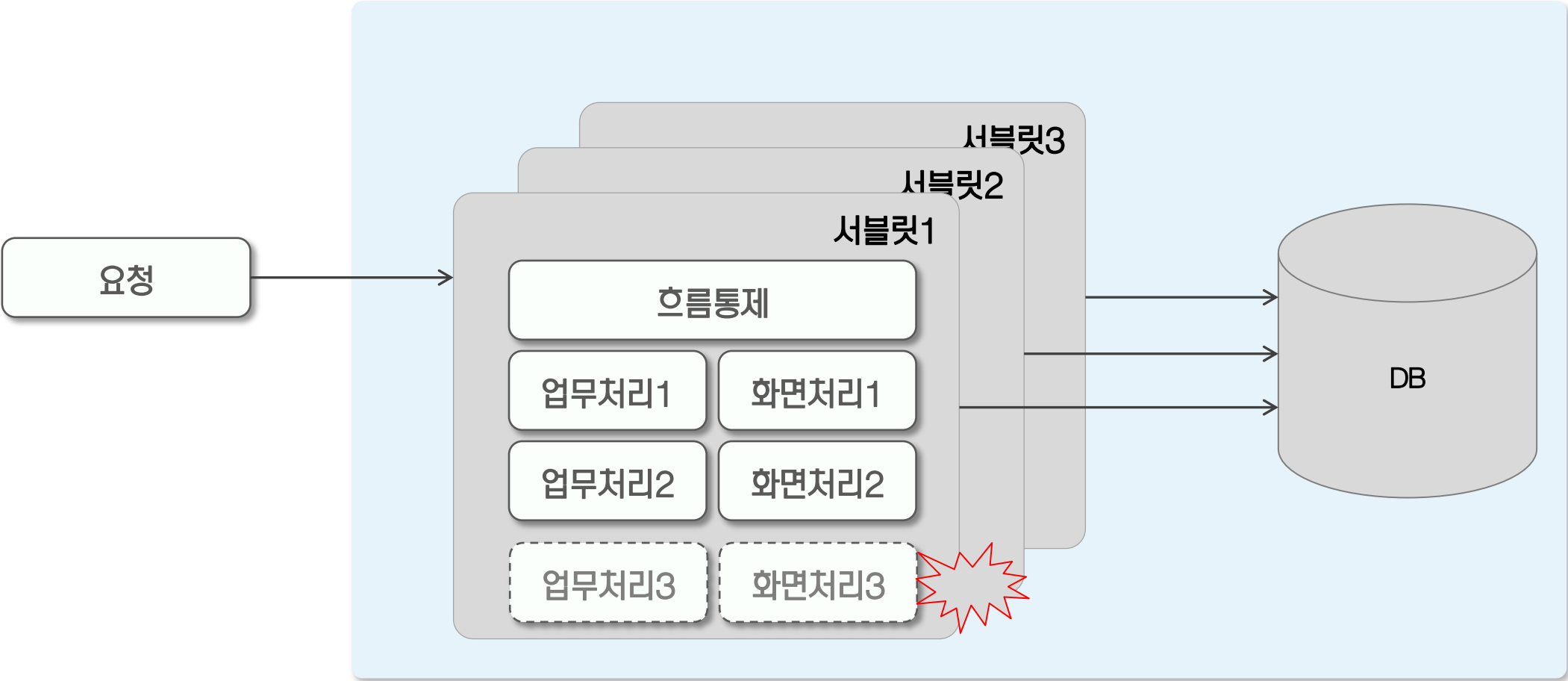
Servlet & JSP (1/4)

- ✓ 하나의 Servlet 또는 하나의 JSP에서 복합기능을 수행
- ✓ 업무복잡도가 증가할수록 유지보수가 어려워짐



Servlet & JSP (2/4)

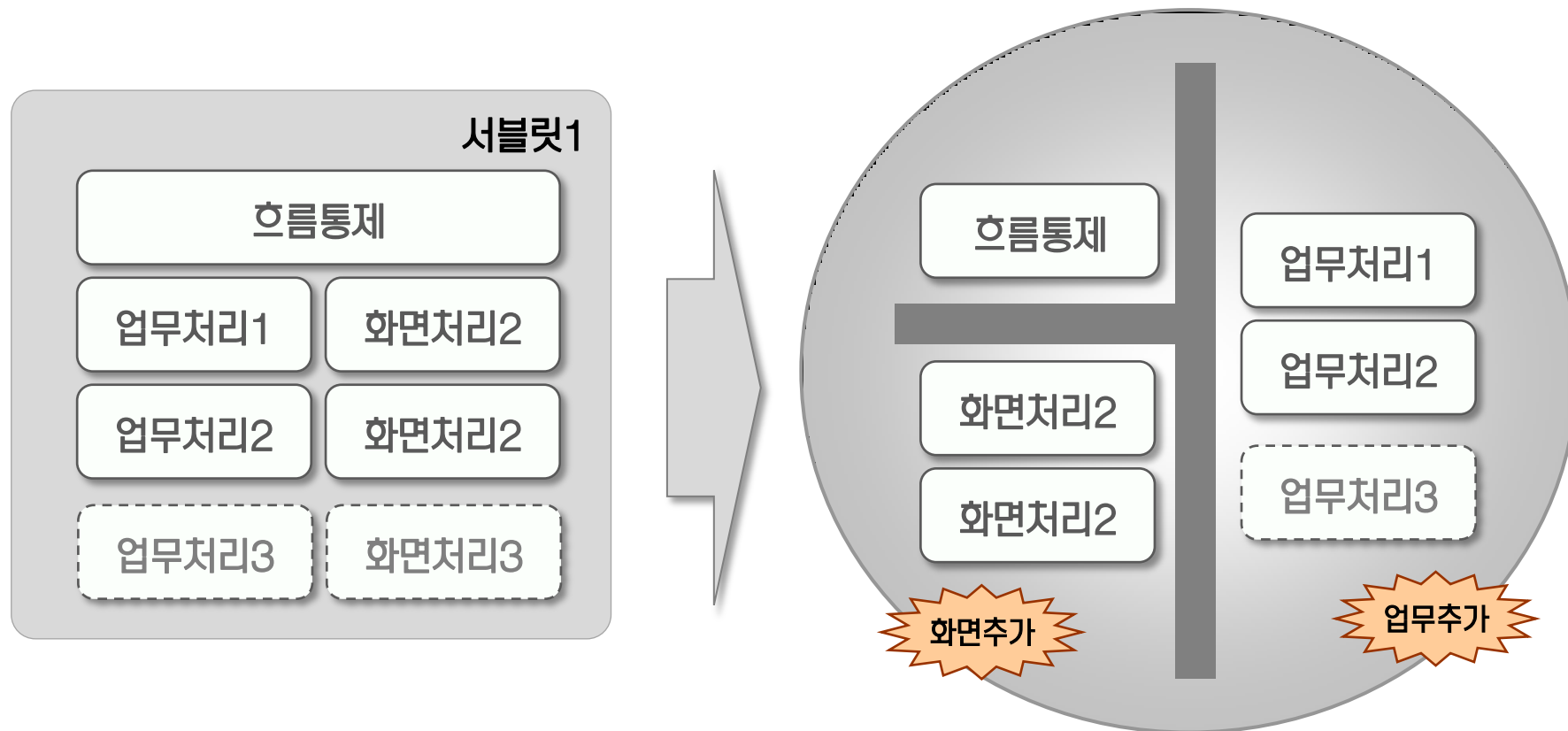
✓ 업무변화에 따라 기능 및 화면이 추가 또는 변경됨



Servlet & JSP (3/4)

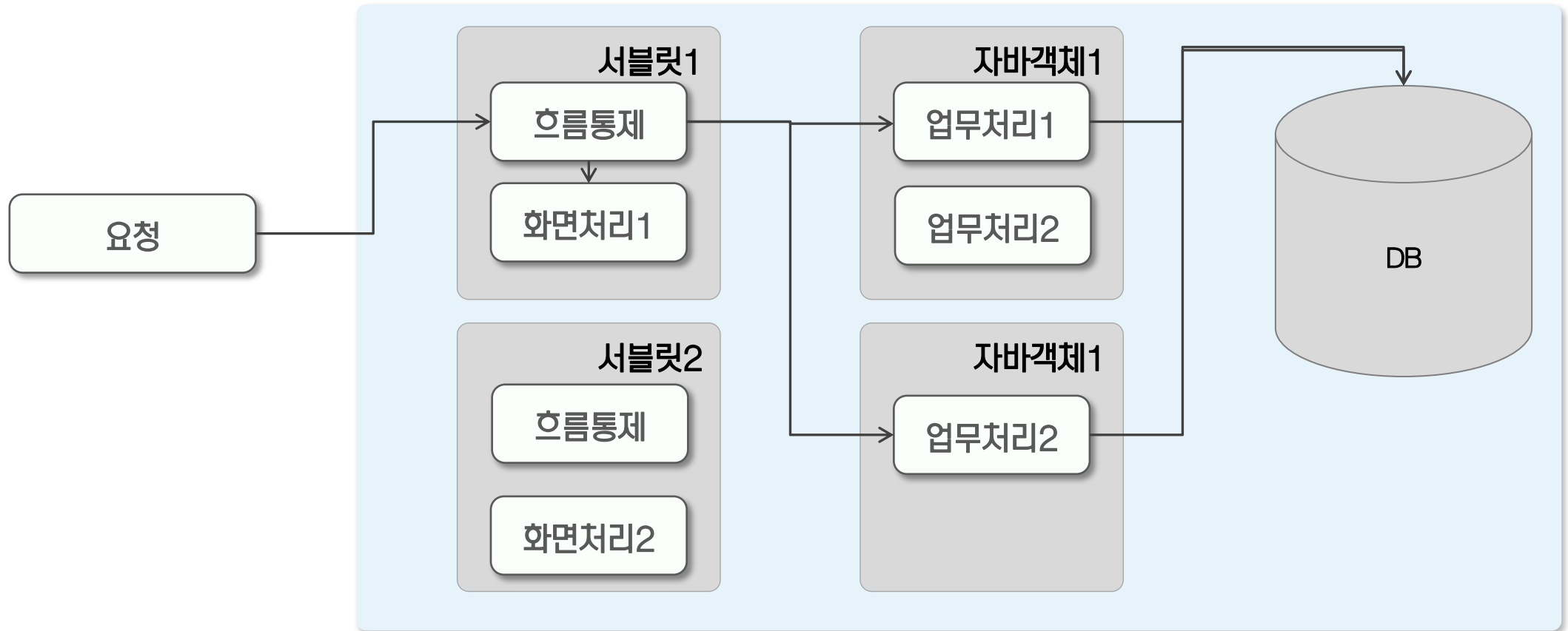
✓ 변경부와 고정부를 분리하여 확장에 열려있고 변경에 닫힌 구조를 고려

- 변화에 따른 변경을 최소화 하기
- 묶을 것 (비슷한 일을 하는 모듈)을 하나로 묶기
- 나눌 것 (다른 일을 하는 모듈)은 나누기



Servlet & JSP (4/4)

- ✓ 흐름통제, 화면처리, 업무처리를 분리
- ✓ 변화요구에 신속하게 대응
- ✓ 특정 모듈의 변경이 다른 모듈에 영향을 주지 않음





7. 웹 프로그래밍 전략

7.1 Servlet & JSP

7.2 MVC 패턴

7.3 웹 프레임워크 구조

7.4 프레임워크 소개

- 설계 방식 - 모델1
- 설계 방식 - 모델2
- MVC 패턴

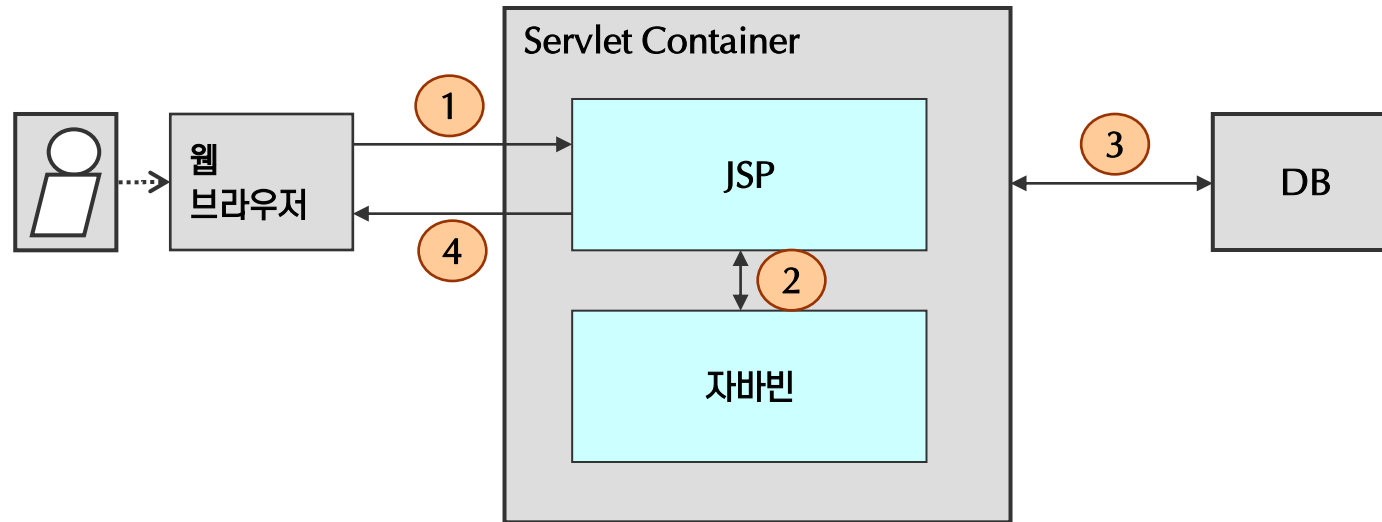
설계 방식 - 모델1

✓ 장점

- 규모가 작은 애플리케이션 개발에 유리
- 개발 시간이 단축 (?)
- 모델 2 개발 방식에 대한 추가적인 교육이 필요 없음

✓ 단점

- 프리젠테이션 로직과 비즈니스 로직이 JSP안에 혼재되어 있어 애플리케이션이 복잡해질수록 유지보수가 어려움
- 디자이너와 개발자간의 원활한 의사소통이 어려움
- 사용자의 증가되는 요구사항 대응이 어려움



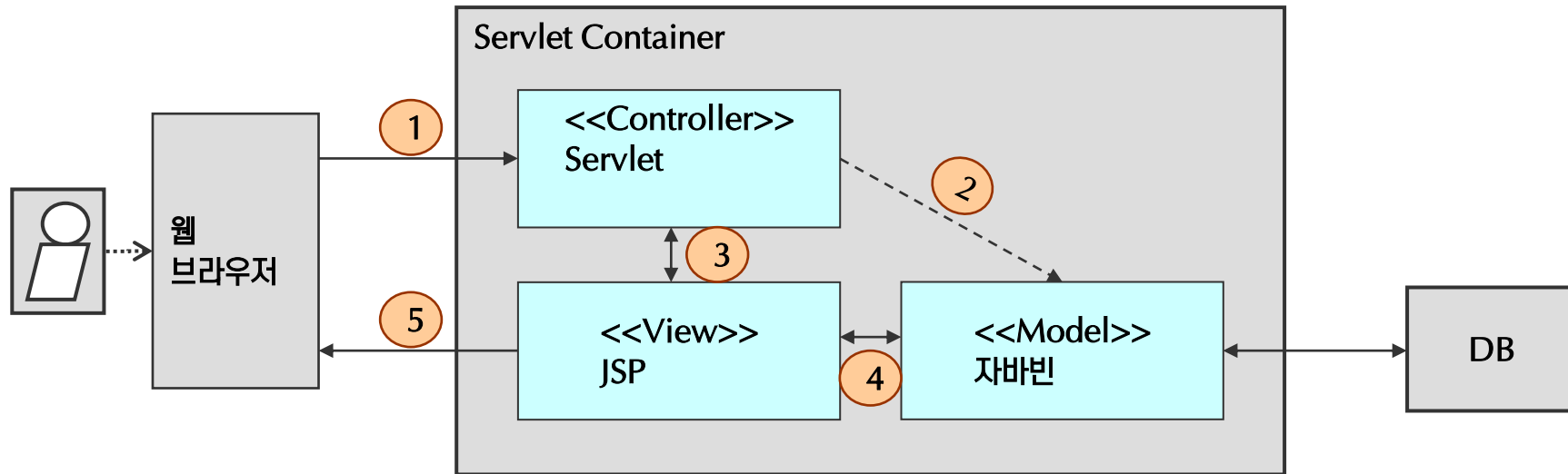
설계 방식 - 모델2

✓ 장점

- 로직과 프리젠테이션의 분리로 인해 애플리케이션이 명료해지며 유지보수와 확장이 용이함
- 디자이너와 개발자의 작업을 분리해 줌

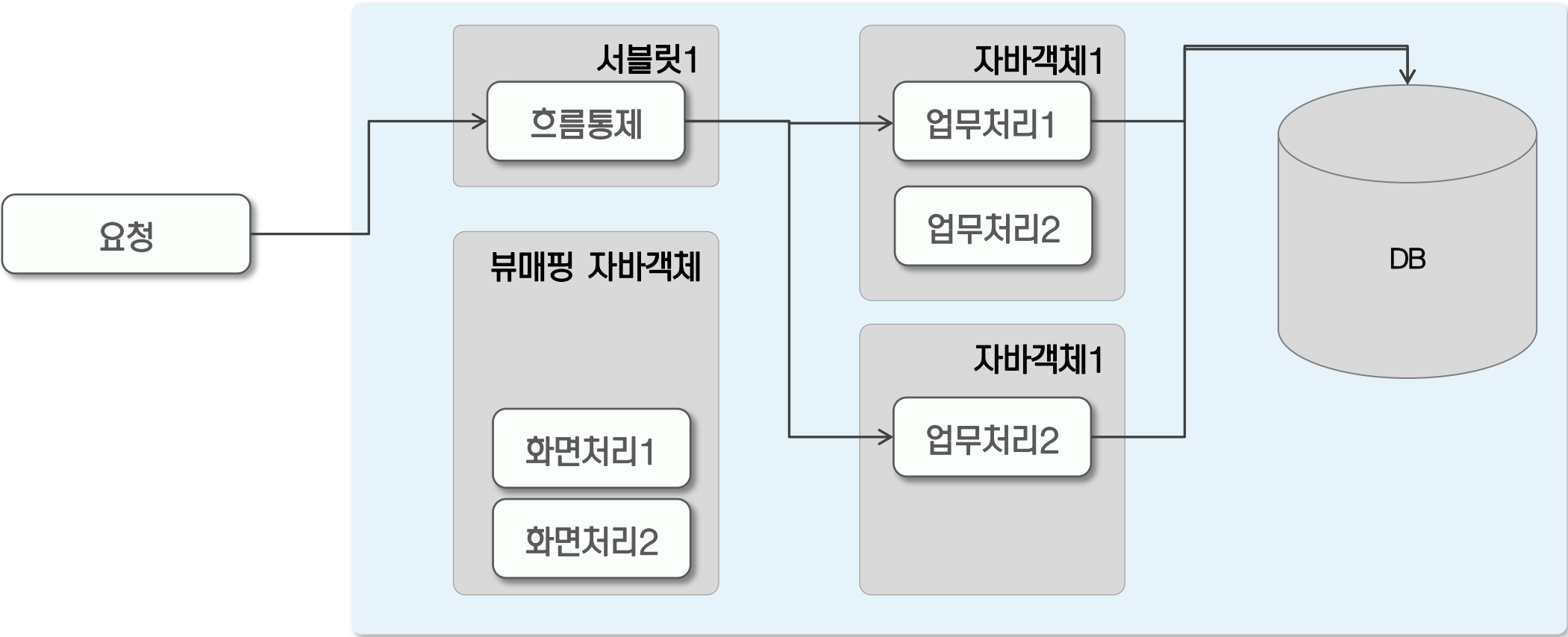
✓ 단점

- 개발 초기에 아키텍처 디자인을 위한 시간의 소요로 개발 기간이 늘어남
- MVC 구조에 대한 개발자들의 이해가 필요함



MVC 패턴

✓ Servlet & JSP 구조에서 흐름통제, 화면처리, 업무처리를 체계적으로 분리





7. 웹 프로그래밍 전략

7.1 Servlet & JSP

7.2 MVC 패턴

7.3 웹 프레임워크 구조

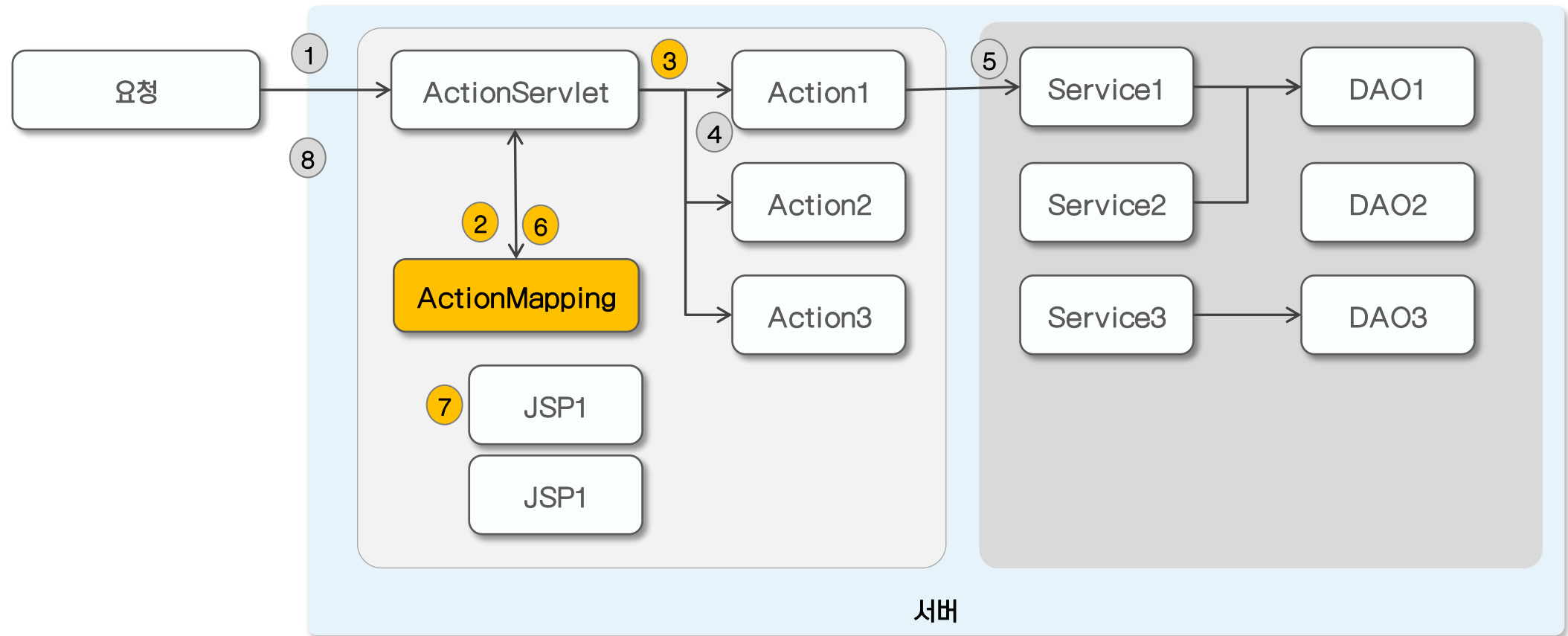
7.4 프레임워크 소개

- 웹 프레임워크 구조
- ActionServlet

웹 프레임워크 구조

✓ 웹프레임워크의 핵심업무

- URL 매핑 : URL분석하여 적절한 Action 클래스를 실행
- 뷰 매핑 : Action 클래스의 결과를 출력할 JSP로 Forward



※. 스트럿츠(Struts) 아키텍처 참조

ActionServlet

- ✓ 모든 요청을 받아 적절한 Action 클래스를 실행하는 역할을 담당한다.
- ✓ Action 클래스의 반환 값을 기반으로 적절한 JSP로 forward
- ✓ 매핑설정은 properties 또는 xml 로 관리



7. 웹 프로그래밍 전략

7.1 Servlet & JSP

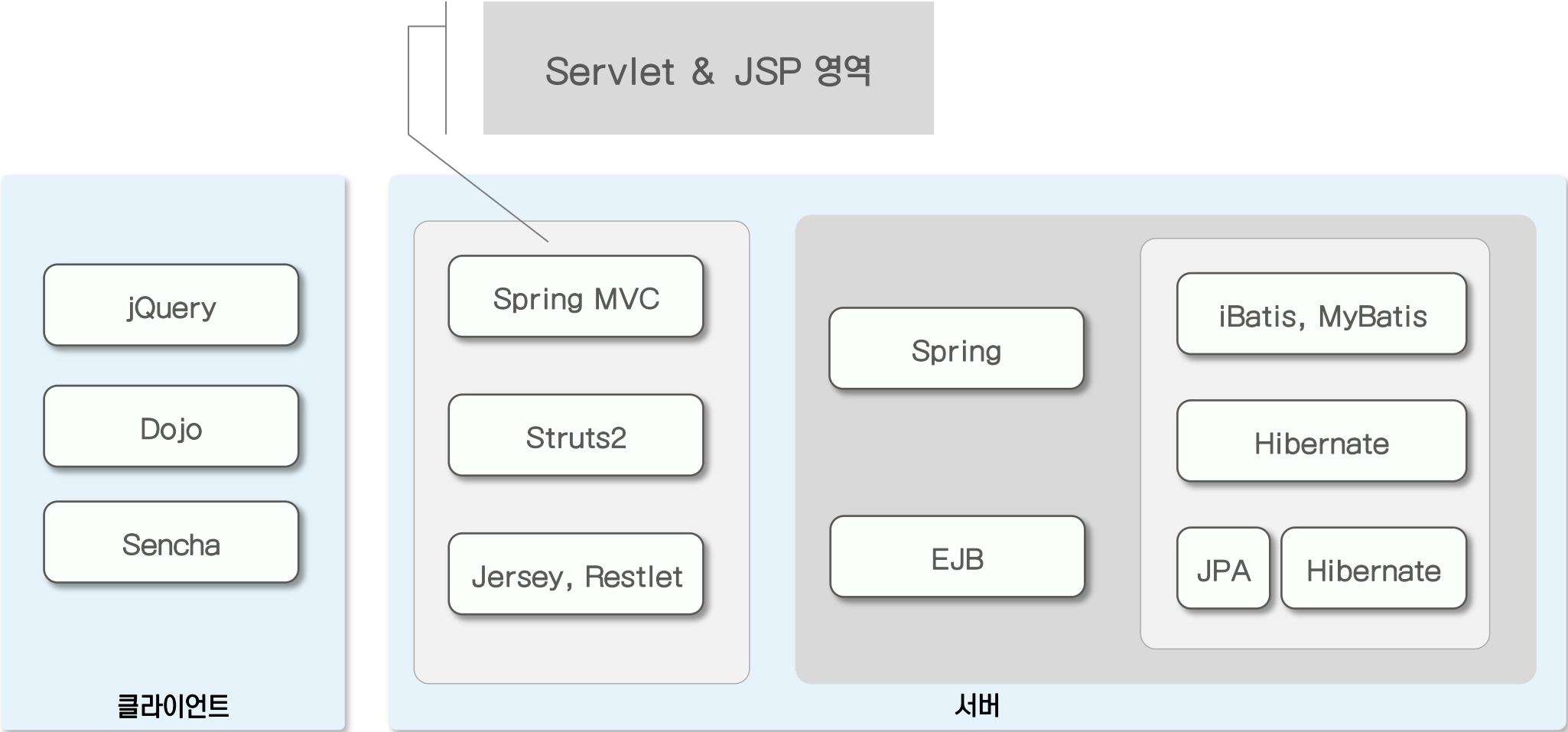
7.2 MVC 패턴

7.3 웹 프레임워크 구조

7.4 프레임워크 소개

- 웹 개발 자바 프레임워크
- 웹 프레임워크
- DB접근 프레임워크
- 컨테이너
- 프레임워크 구성도

웹 개발 자바 프레임워크



참고 : <http://nosql.findthebest.com/>

웹 프레임워크

✓ 스트럿츠(Struts)

- 2000년대 중반부터 사용되고 있는 기본에 충실한 웹프레임워크로 URL 매핑과 뷰매핑을 xml로 처리
- 최신버전 1.3.10 (2008.12 지원 중단)

✓ 스트럿츠2(Struts2)

- 스트럿츠(struts)의 후속으로 시작된 프레임워크이나 스트럿츠보다 웹워크(WebWork)의 영향을 받음
- 요청인자를 객체로 자동 매핑가능한 것이 특징
- 다양한 플러그인 사용 가능(RESTFul 서비스의 일부 지원 가능)
- 최신버전 2.3.x <http://struts.apache.org/>

✓ 스프링MVC(SpringMVC)

- 스프링 프레임워크의 일부로 기타 스프링 계열의 프레임워크와 호환성이 좋음
- 2.5.x 하위 버전에서는 모든 bean 생성을 xml로 관리하여 설정관리가 어려웠으나 2.5.x 상위 버전에서는 애노테이션을 도입하여 xml 설정을 대폭 단순화 함
- 일부 RESTFul 서비스 구현을 지원함
- 최신버전 4.0 <http://projects.spring.io/spring-framework/>

✓ Jersey, Restlet

- RESTFul 서비스 구현을 지원
- Jersey 최신 버전 2.5.1 <https://jersey.java.net/>
- Restlet 최신버전 2.1.6 <http://restlet.org/>

DB 접근 프레임워크

✓ 아이바티스(iBatis)

- JDBC의 불편한 점을 개선하기 위한 용도로 사용됨, SQL 재사용 및 동적 SQL 작성이 가능하여 JDBC 방식 대비 SQL문을 대폭 줄이는 효과를 보임
- 2010년 6월 16일로 아파치 재단에서 구글코드로 이전됨
- 최신버전 2.3.5.x <http://ibatis.apache.org/>

✓ 마이바티스(MyBatis)

- iBatis의 후속 프로젝트
- SQL 결과와 객체 매핑부분에 대폭 개선이 있었으며, 동적SQL 작성문법을 jstl과 유사하게 개선하여 친숙도를 높임

✓ 하이버네이트(Hibernate)

- ORM (Object Relation Mapper) 프레임워크로 객체간 관계설정이 되어 있으면 별도의 SQL문 없이 객체 조회를 통해 자동으로 SQL구문을 생성해주는 점이 강점
- 단 관계가 없는 객체조회는 불가능하며 이를 지원하기 위해 별도의 JPQL이라는 쿼리문법을 제공

컨테이너

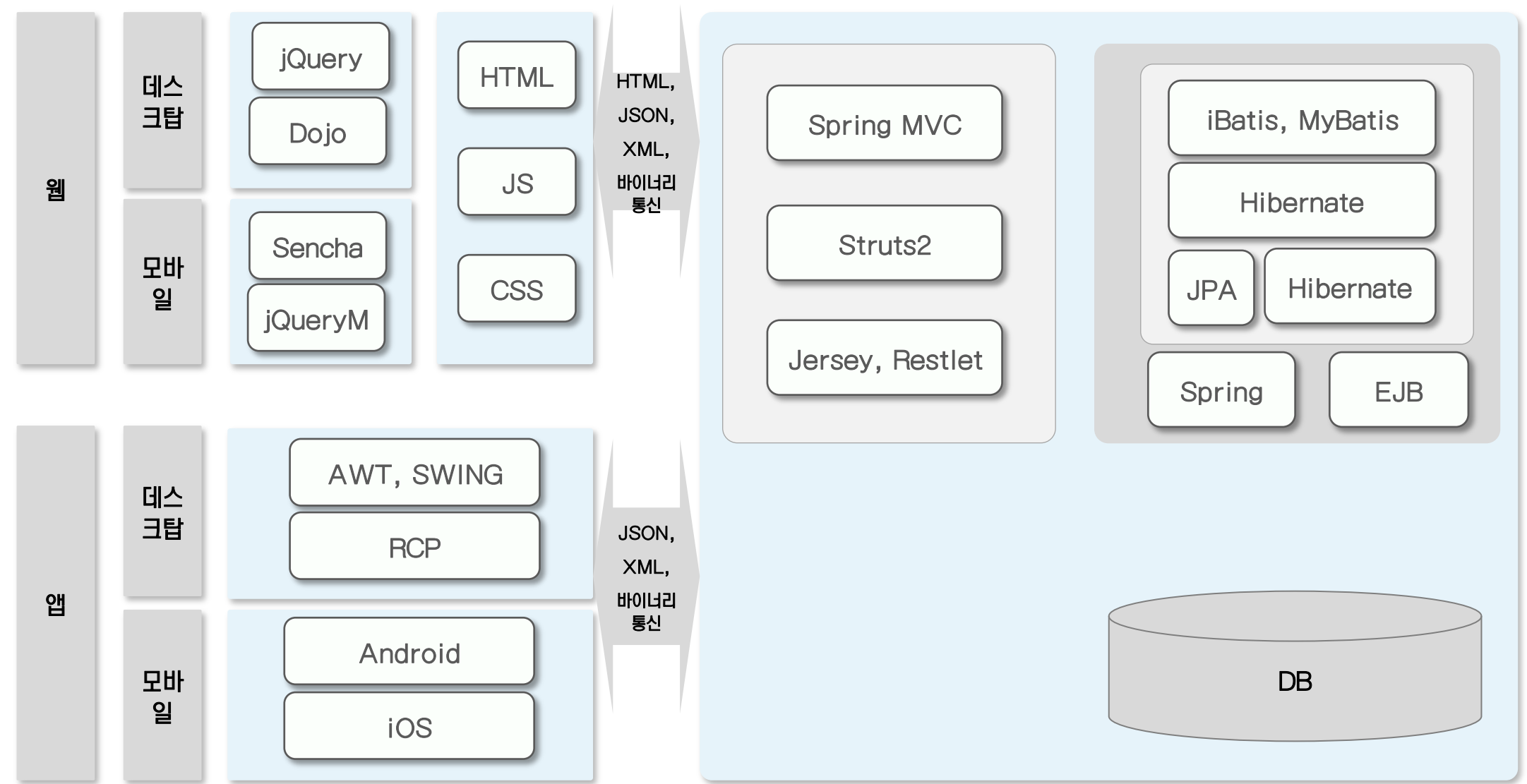
✓ EJB

- J2EE 표준을 준수하는 프레임워크로 웹애플리케이션서버(WAS)에서 지원, 기업형 웹애플리케이션 개발지원을 위해 분산처리, 트랜잭션등 기능을 제공하고 있지만 EJB를 지원하는 WAS를 사용해야만 하고 복잡한 EJB 스펙을 준수해야하는 등 어려움이 있음
- EJB2에서 문제로 제시된 복잡한 스펙 및 XML 설정을 벗어나기 위해 EJB3가 제시됨

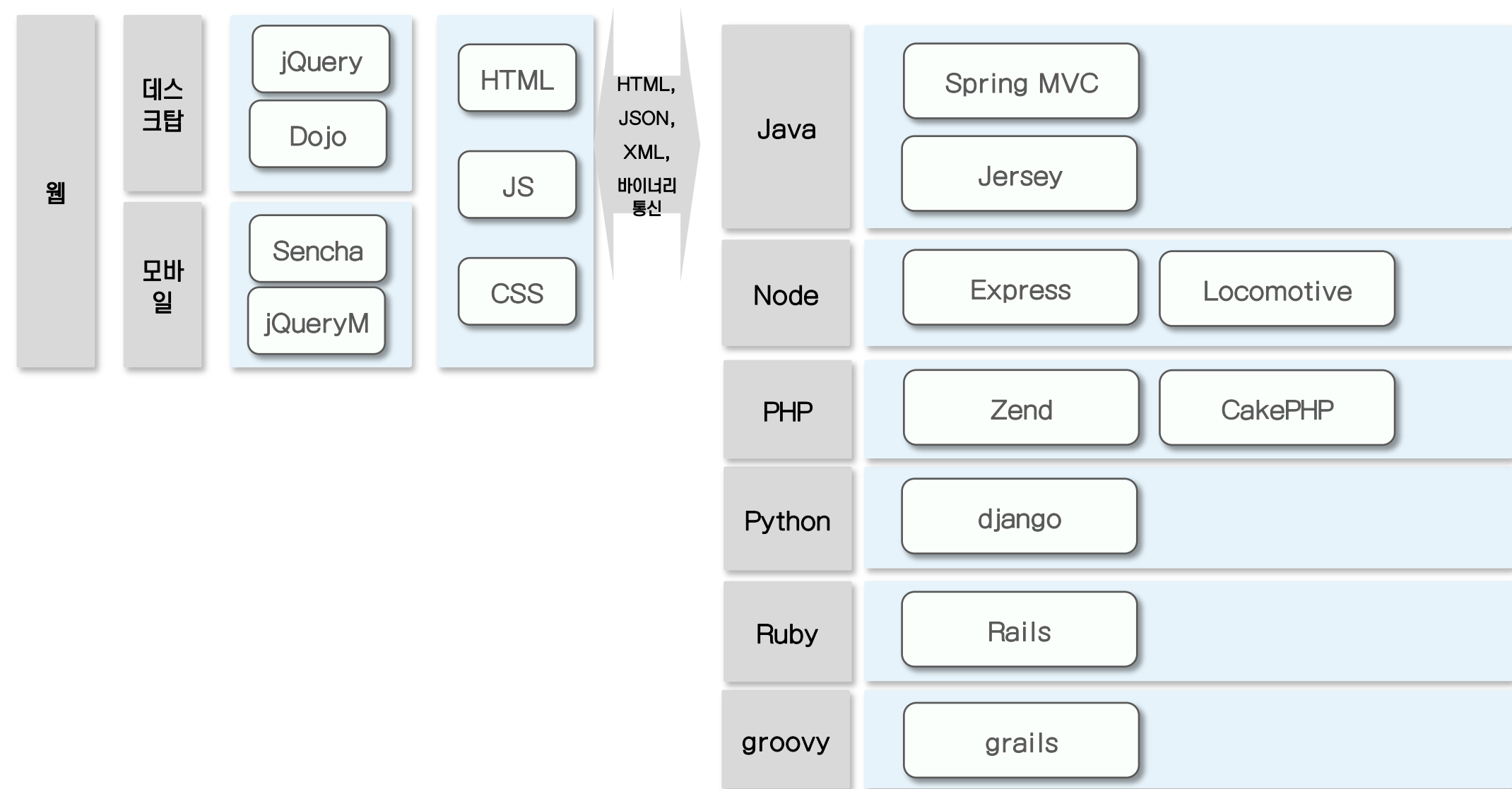
✓ 스프링

- EJB의 복잡함에서 벗어나 자바 본연의 모습으로 돌아가기 위한 의견이 제시될 시기에 시작됨
- EJB에서 제공하는 기능을 WAS에서 분리하여 경량 컨테이너 기능을 제공함
- 최근 수행되는 대다수의 프로젝트가 스프링을 이용하는 등 인기를 누리고 있으며, 수많은 하위 프로젝트로 인해 다양한 서비스 구축이 가능함
- (스프링MVC도 스프링의 하위 프로젝트)

프레임워크 구성도



기타 웹 애플리케이션 개발언어 및 프레임워크



토의

- ✓ 질의 응답
- ✓ 토론

감사합니다...

- ❖ 넥스트리소프트(주)
- ❖ 김현오 선임 (hyunohkim@nextree.co.kr)