

The background of the image is a wide-angle photograph of a rural landscape. It features rolling hills covered in green and brown agricultural fields. Several white wind turbines are scattered across the hills, particularly on the right side. The sky is a clear, vibrant blue with no clouds. In the bottom right corner, there's a small, light-colored building with a blue roof and some yellow barrels nearby.

JSP & Servlet
(ver3.4.0)

목차 (Table of Contents)

1. Web 기본 개념
2. Servlet 이해
3. JSP(Java Server Pages) 이해
4. Java 웹 프로그래밍 I
5. Java 웹 프로그래밍 II

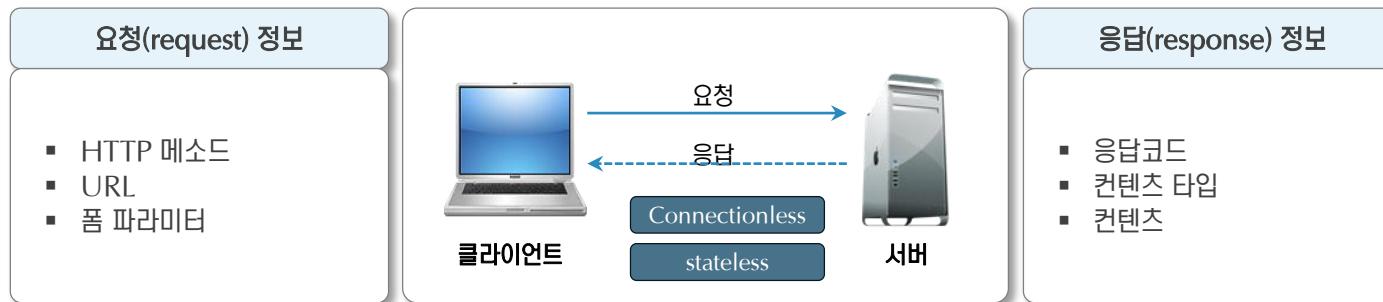


1. Web 기본 개념

-
- 1.1 HTTP 프로토콜
 - 1.2 TCP 포트와 서비스
 - 1.3 웹 서버
 - 1.4 CGI (Common Gateway Interfaces)
 - 1.5 요약

1.1 HTTP 프로토콜 (1/5) – 개요

- ✓ HTTP(HyperText Transfer Protocol)는 인터넷 상에서 데이터를 주고 받기 위한 규약입니다.
- ✓ 클라이언트와 서버는 반복적인 요청과 응답을 통해 서로 데이터를 주고 받습니다.
- ✓ HTTP 프로토콜은 비 연결지향으로 클라이언트가 응답을 받으면 서버와 연결이 끊어집니다. (Connectionless)
- ✓ 이러한 특징으로 인해 서버는 클라이언트의 이전 상태를 알 수 없습니다. (Stateless)



1.1 HTTP 프로토콜 (2/5) – HTTP 메소드

- ✓ HTTP 메소드는 요청의 종류를 서버에 알려주기 위해 사용하는 것으로 여러 방식이 존재합니다.
- ✓ 대부분의 웹 사이트에서, 서버에 정보를 요청할 때는 GET / 폼을 전송할 때는 POST 방식을 사용합니다.
- ✓ RESTful 웹 서비스에서는 HTTP 메소드의 목적에 맞게 사용할 것을 강조합니다.
- ✓ HEAD, TRACE, OPTIONS 등의 메소드는 서버의 상태를 확인하기 위해 사용합니다.

GET	URL을 이용하여 자원을 요청 (Retrieve)
POST	Request에 첨부한 Body 정보를 서버로 전송 (Create)
PUT	Body정보를 요청한 URL로 저장 (Update)
DELETE	URL에 해당하는 자원을 삭제 (Delete)
HEAD	헤더정보만 요청. 해당 자원이 존재하는지 또는 서버의 문제가 없는지를 확인하기 위해 사용
TRACE	요청을 그대로 반환. 테스트 목적으로 서버에서 무엇을 받았는지 알고 싶을 때 사용 (루프백 테스트)
OPTIONS	요청 URL에 응답할 수 있는 HTTP Method 종류들이 무엇인지 요청

1.1 HTTP 프로토콜 (3/5) – GET 메소드

- ✓ GET 메소드는 가장 단순한 HTTP 메소드로 서버에게 자원을 요청할 때 사용합니다.
- ✓ URL 뒤에 요청 파라미터를 붙이는 방식으로 서버에 데이터를 전달합니다.
- ✓ 파라미터를 포함한 URL은 최대길이가 제한되어 있으며 브라우저마다 차이가 있습니다.
- ✓ 예를 들어, IE는 URL 최대길이가 2048 글자를 넘을 수 없습니다.



출처 : Bryan Basham, Kathy Sierra, Bert Bates, 「Head First Servlets and JSP」 : O'Reilly Media, 2008, p.49

1.1 HTTP 프로토콜 (4/5) – POST 메소드

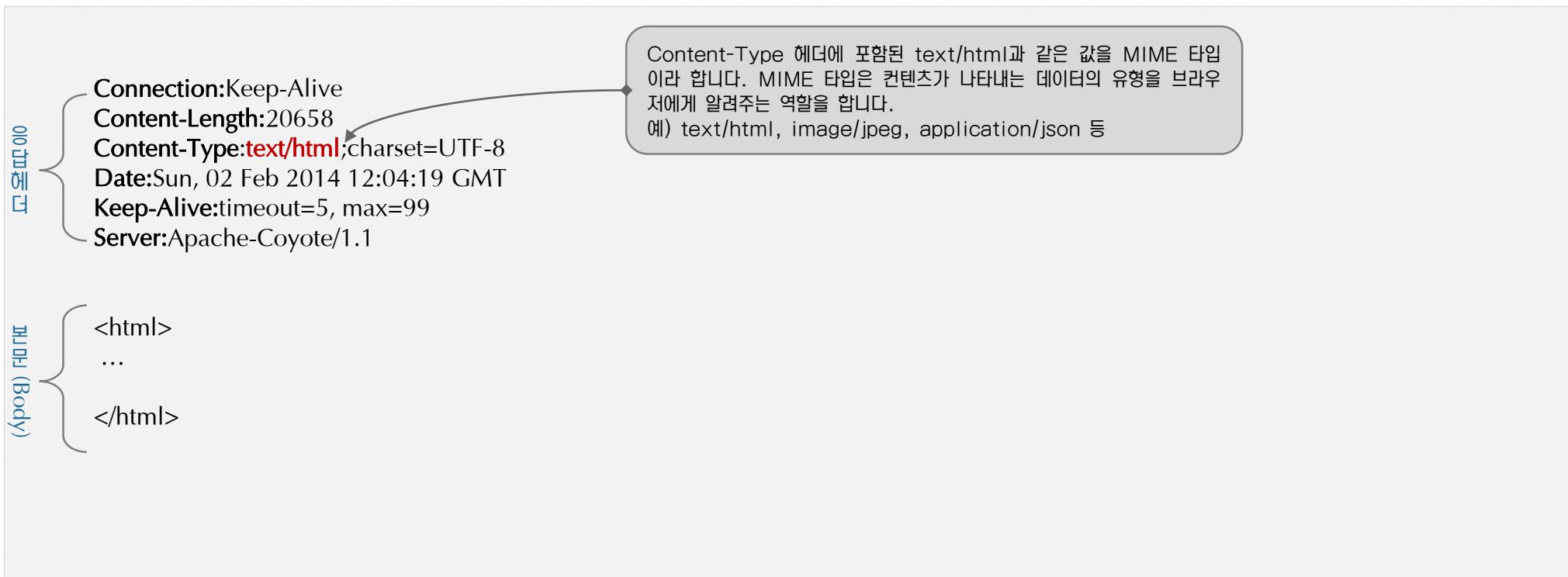
- ✓ POST 메소드는 서버에 저장할 데이터를 전달하기 위하여 사용합니다.
- ✓ 서버에 전달하는 데이터는 요청 본문에 포함되며 URL에 파라미터 값들이 노출되지 않습니다.
- ✓ GET과는 달리 전달하는 파라미터 길이의 제약이 없습니다.
- ✓ 웹 페이지에서 서버로 파일을 업로드하거나 폼을 전송할 때 주로 사용합니다.



출처 : Bryan Basham, Kathy Sierra, Bert Bates, 「Head First Servlets and JSP」 : O'Reilly Media, 2008, p.50

1.1 HTTP 프로토콜 (5/5) – Response

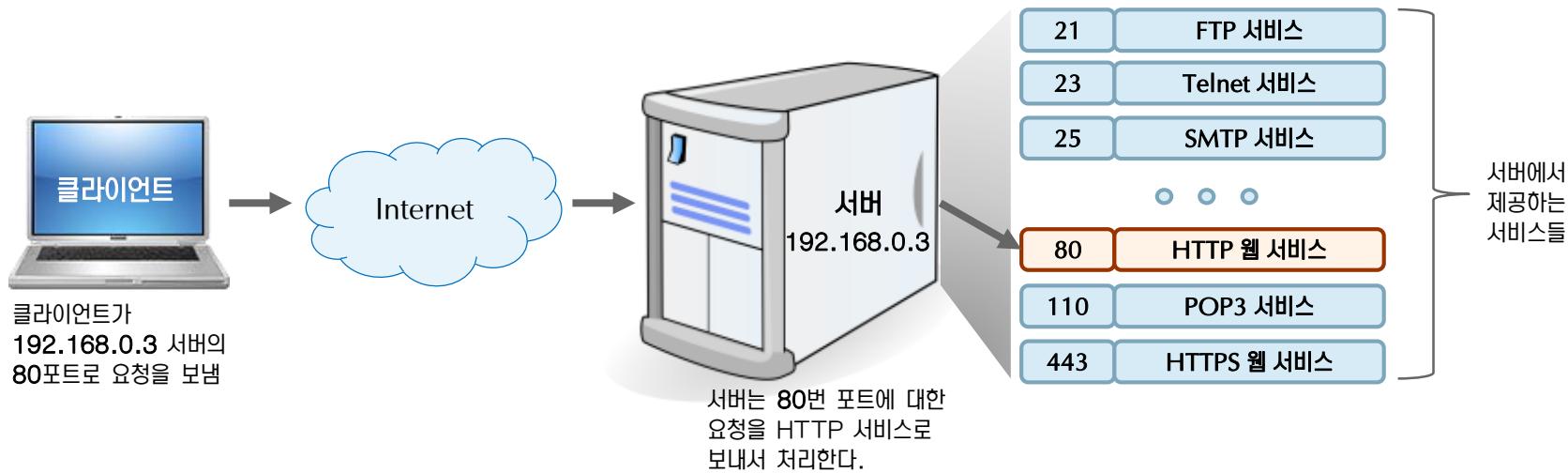
- ✓ 응답(Response)이란 서버가 클라이언트로 보내는 메시지를 말합니다.
- ✓ 응답 메시지는 헤더와 본문(Body)으로 구성됩니다.
- ✓ 헤더에는 사용 프로토콜, 요청 성공 여부, 본문에 포함된 컨텐츠의 종류 등이 있습니다.
- ✓ 본문은 HTML, 이미지와 같은 클라이언트에서 사용할 컨텐츠가 있습니다.



출처 : Bryan Basham, Kathy Sierra, Bert Bates, 「Head First Servlets and JSP」 : O'Reilly Media, 2008, p.51

1.2 TCP 포트와 서비스

- ✓ TCP 포트는 서버(하드웨어)에서 구동되는 소프트웨어(서비스)를 구별하기 위한 번호입니다. (0~65535)
- ✓ 포트번호는 서버(하드웨어)에서 구동되는 특정 서비스에 대한 논리적인 연결을 나타냅니다.
- ✓ TCP 포트번호 0 ~1023까지는 널리 알려진 서비스를 위하여 예약되어 있습니다.
- ✓ 서버는 포트를 통해 클라이언트가 어느 애플리케이션에 접속하기를 원하는지 알 수 있습니다.



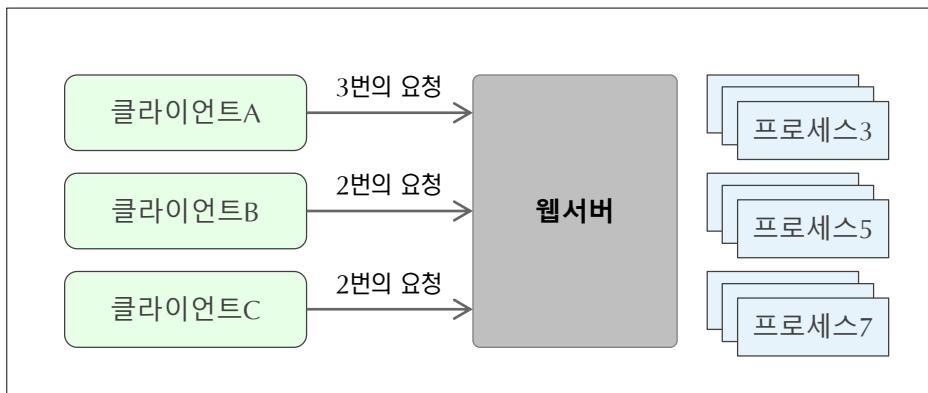
1.3 웹 서버

- ✓ 웹 서버는 HTTP 프로토콜을 통해 클라이언트가 요청한 자원을 찾아 제공하는 서버입니다.
- ✓ 웹 서버는 단지 정적 리소스(HTML 문서, 이미지 등)만 찾아서 그대로 클라이언트에게 넘겨줍니다.
- ✓ 클라이언트가 요청한 자료가 서버에 없는 경우, 서버는 404 Not Found 응답메시지를 보냅니다.
- ✓ 웹 서버는 동적으로 컨텐츠를 생성할 수 없기 때문에 대안으로 CGI와 같은 기술이 등장하였습니다.

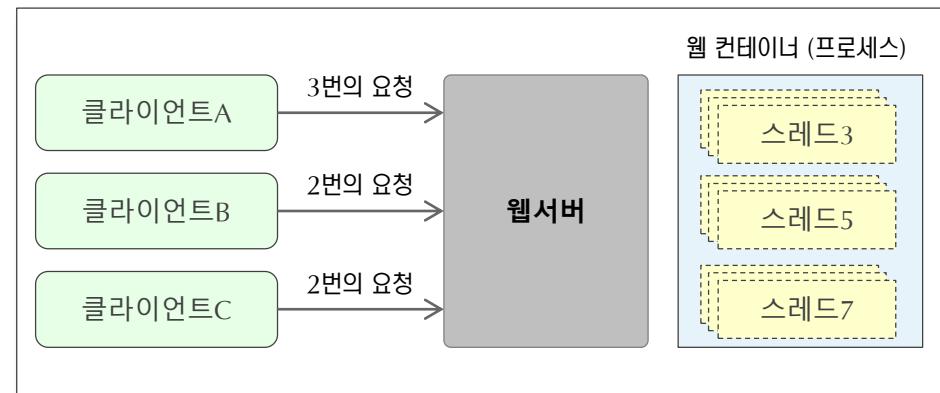


1.4 CGI (Common Gateway Interfaces)

- ✓ CGI는 웹 서버에서 동적인 컨텐츠를 생성하여 클라이언트에 제공할 수 있는 인터페이스입니다.
- ✓ 전통적인 CGI 방식은 클라이언트 요청마다 개별 프로세스가 생성되어 시스템 부하가 많이 생기는 단점이 있었습니다.
- ✓ 또한 플랫폼에 종속적이어서 윈도우에서 C언어 등으로 만들어진 CGI 애플리케이션은 리눅스에서 사용할 수 없었습니다.
- ✓ 이러한 단점을 해결하기 위하여 개별 스레드가 요청을 처리하는 방식으로 발전하였습니다. (ASP, PHP, Servlet 등)



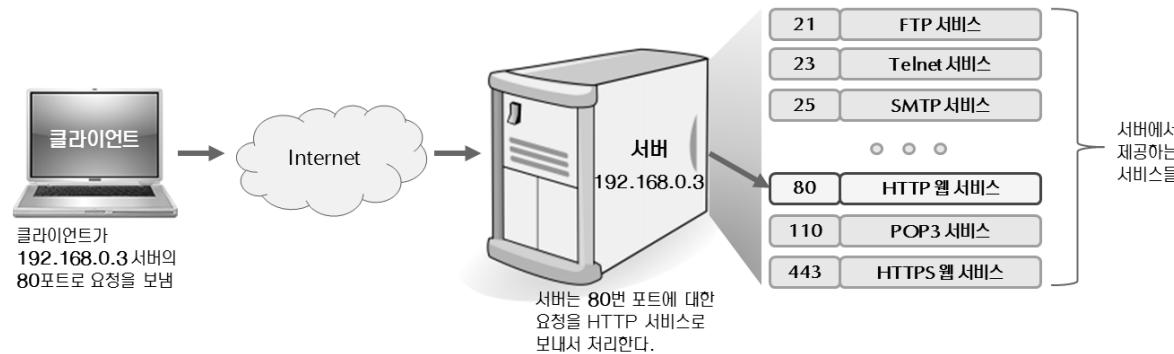
전통적인 CGI 방식



요청을 스레드로 처리하는 방식

1.5 요약

- ✓ HTTP는 인터넷 상에서 데이터를 주고 받기 위한 규약으로써 반복적인 요청과 응답을 통해 데이터를 주고 받습니다.
- ✓ 클라이언트가 서버에 데이터를 요청할 때 요청방식으로는 GET, POST, PUT, DELETE가 있습니다.
- ✓ 클라이언트가 파라미터를 전송할 때 GET은 URL뒤에 붙여서 전송하고, POST는 요청 본문에 포함하여 전송합니다.
- ✓ TCP포트는 서버에서 서비스 구별을 위한 번호입니다. 서버는TCP포트로 클라이언트가 어느 서비스를 요청한지 식별합니다.



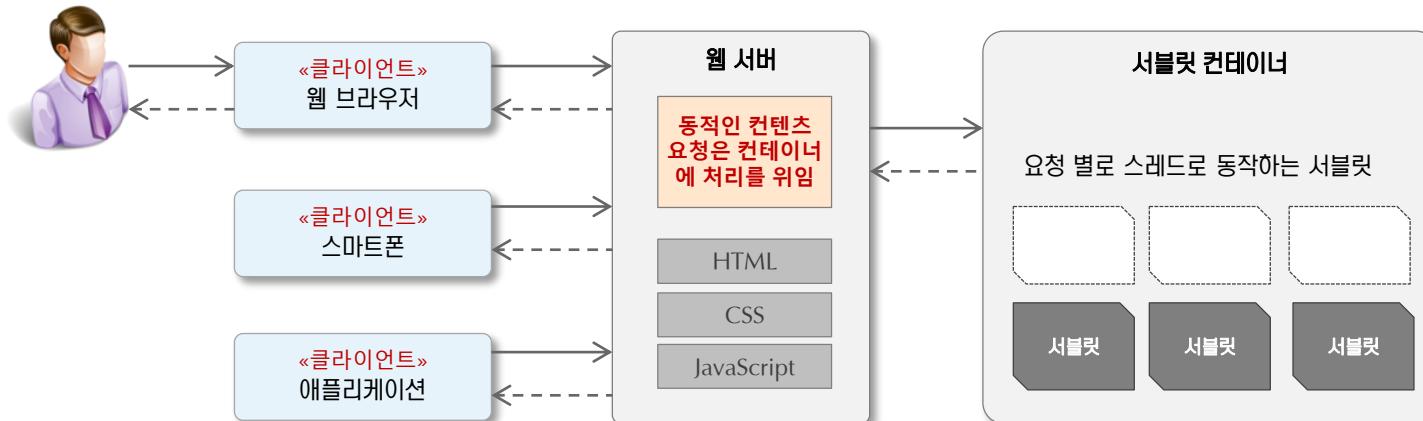


2. Servlet 이해

-
- 2.1 서블릿 기본
 - 2.2 서블릿 컨테이너
 - 2.3 요청과 응답
 - 2.4 요청 위임하기
 - 2.5 요약

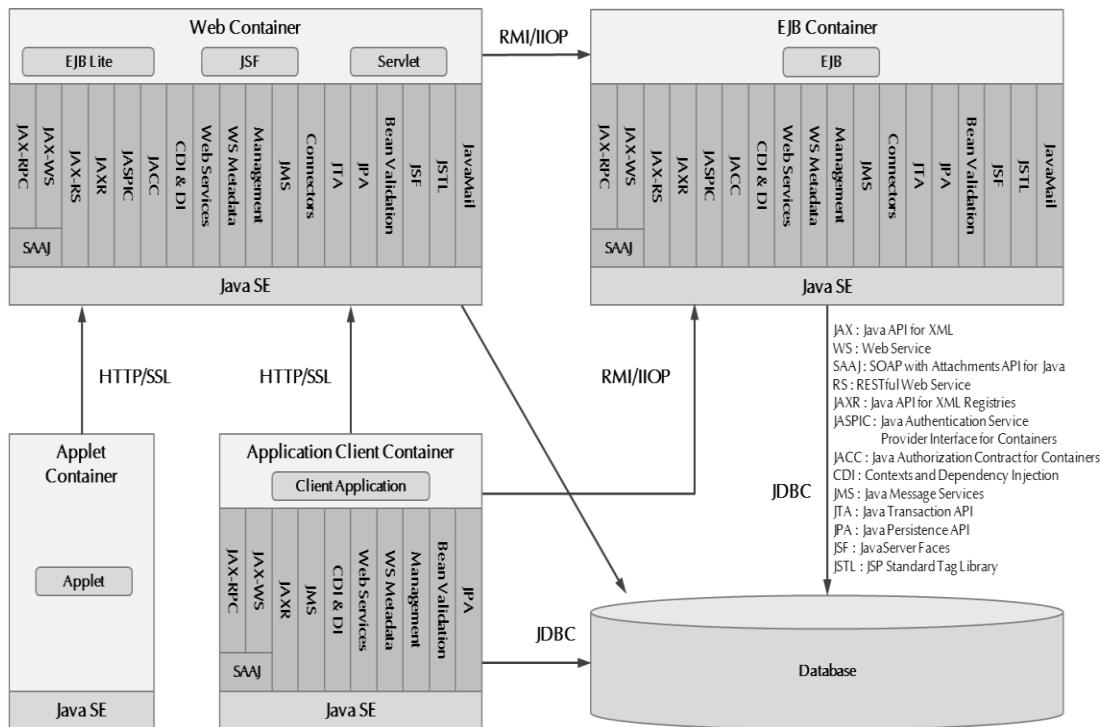
2.1 서블릿 기본 (1/6) – 개요

- ✓ 서블릿(Servlet)은 Java 언어를 사용하여 웹 페이지를 동적으로 생성할 수 있는 서버 측 프로그램입니다.
- ✓ Java를 사용하므로 플랫폼에 독립적이고, 스레드 기반으로 좀 더 효과적인 멀티 태스킹을 지원합니다.
- ✓ Servlet API는 서블릿 프로그램 개발에 필요한 다양한 클래스 및 인터페이스를 제공합니다.
- ✓ Servlet 컨테이너는 서블릿이 동작하는 실행환경으로 서블릿의 생명주기를 관리합니다.

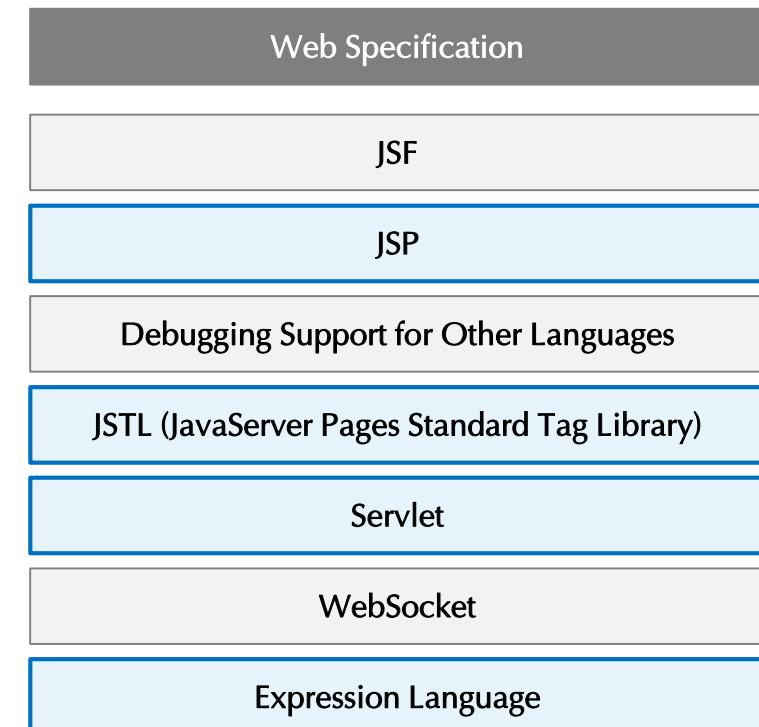


2.1 서블릿 기본 (2/6) – JEE 명세

- ✓ JEE 스펙은 엔터프라이즈 애플리케이션 개발에 필요한 기술들에 대한 표준으로 JCP에서 정의하였습니다.
- ✓ JEE 스펙 중 웹 애플리케이션 개발과 관련된 기술들을 모아둔 것을 Web Specification이라고 합니다.
- ✓ 모든 스펙을 구현한 서버를 WAS라고 하며, 이 중 웹과 관련된 기술만을 처리하는 서버를 웹 컨테이너라 합니다.
- ✓ 본 과정에서는 웹과 관련된 기술인 JSP, JSTL, Servlet, EL을 다루게 됩니다.

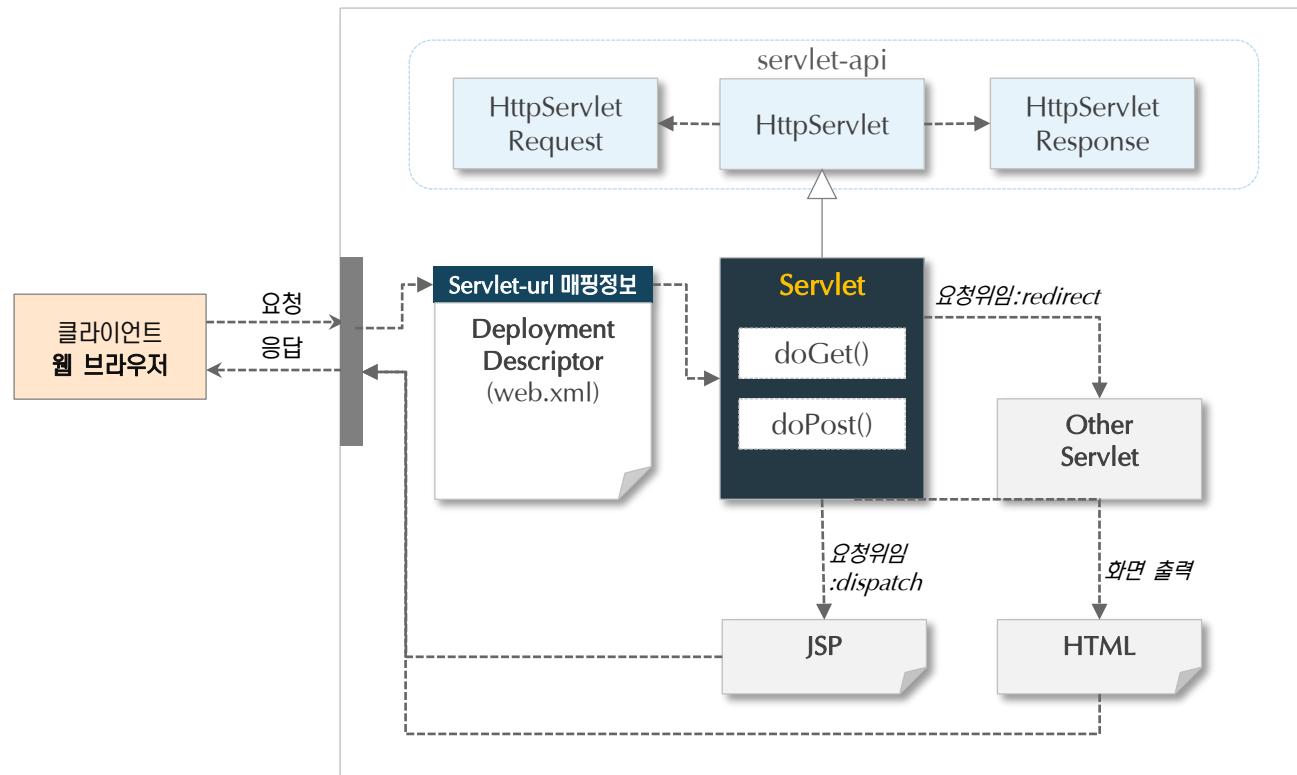


출처 : JavaEE Platform Spec EDR : www.oracle.com , p.8



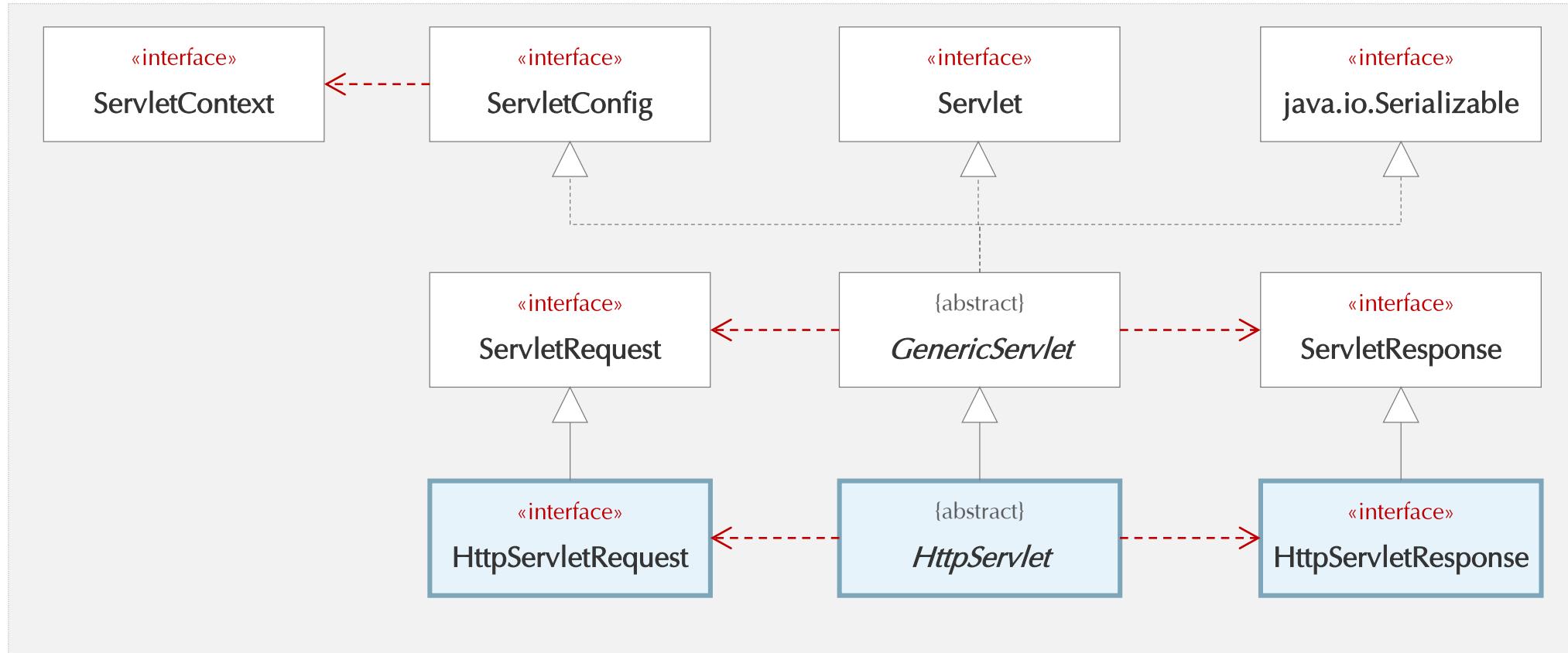
2.1 서블릿 기본 (3/6) – 서블릿 프로그래밍 핵심 구조

- ✓ 하나의 요청은 Deployment Descriptor(배포서술자, DD)의 URL에 매핑된 서블릿으로 연결됩니다.
- ✓ Servlet Spec 3.0 부터는 @WebServlet 어노테이션으로 매핑 가능합니다.
- ✓ 서블릿은 servlet-api의 HttpServlet을 상속받아 구현하고, HTTP 메소드에 따른 처리를 구현할 수 있습니다.
- ✓ 매핑된 서블릿이 요청을 직접 처리하여 출력하거나, 다른 서블릿이나 JSP로 요청을 위임할 수도 있습니다.



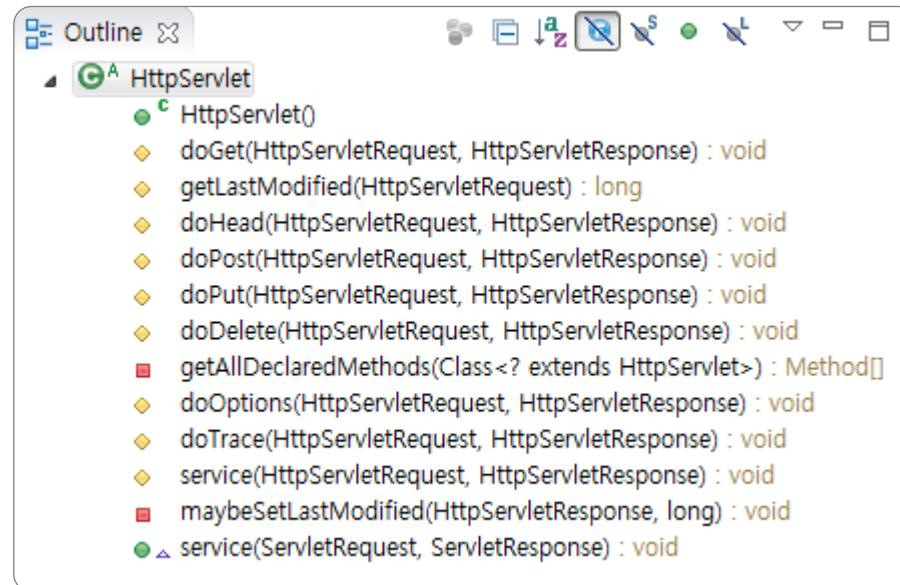
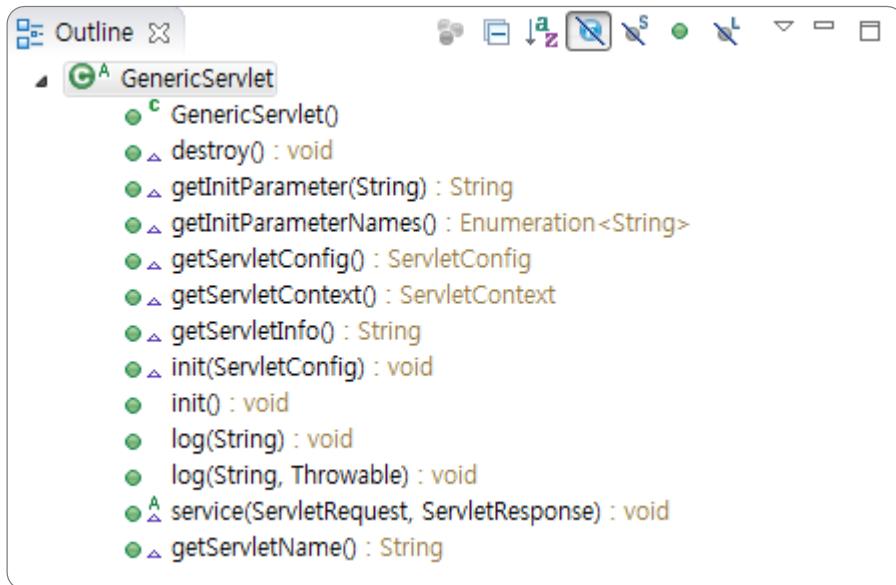
2.1 서블릿 기본 (4/6) – 서블릿 API

- ✓ 서블릿 API는 서블릿 프로그램을 작성을 위한 인터페이스 및 클래스로 javax.servlet 패키지 하위에 존재합니다.
- ✓ 서블릿을 사용한 Java 웹 프로그래밍을 할 때는 거의 대부분 HttpServlet 을 상속하여 개발합니다.



2.1 서블릿 기본 (5/6) – HttpServlet 추상클래스

- ✓ GenericServlet은 서블릿 관리에 필요한 기능들이 미리 구현되어 있는 추상클래스입니다.
- ✓ HttpServlet는 GenericServlet을 상속하며 HTTP 요청 메소드 유형별로 메소드가 정의되어 있습니다.
- ✓ 대부분의 웹 애플리케이션에서는 HttpServlet 추상클래스를 상속받아 요청방식에 해당하는 메소드를 재정의합니다.
- ✓ HttpServlet 의 service() 메소드에서는 요청방식에 따라 적합한 메소드를 호출합니다. (예, GET방식은 doGet() 호출)



2.1 서블릿 기본 (6/6) – 서블릿 등록

- ✓ 웹 애플리케이션에 서블릿을 등록하는 방법에는 두 가지가 있습니다.
- ✓ 첫 번째는 web.xml 에 <servlet> 과 <servlet-mapping> 요소로 등록하는 방법입니다.
- ✓ 서블릿 3.0 부터는 @WebServlet 어노테이션을 사용하여 서블릿을 등록할 수 있습니다.

```
<servlet>
    <description>사용자 등록처리</description>
    <servlet-name>UserRegisterController</servlet-name>
    <servlet-class>kr.namoo.yorizori.web.controller.UserRegisterController</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>UserRegisterController</servlet-name>
    <url-pattern>/user/register.do</url-pattern>
</servlet-mapping>
```

방법 #1. web.xml 에 선언하는 방법

```
@WebServlet("/user/register.do")
public class UserRegisterController extends HttpServlet {

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

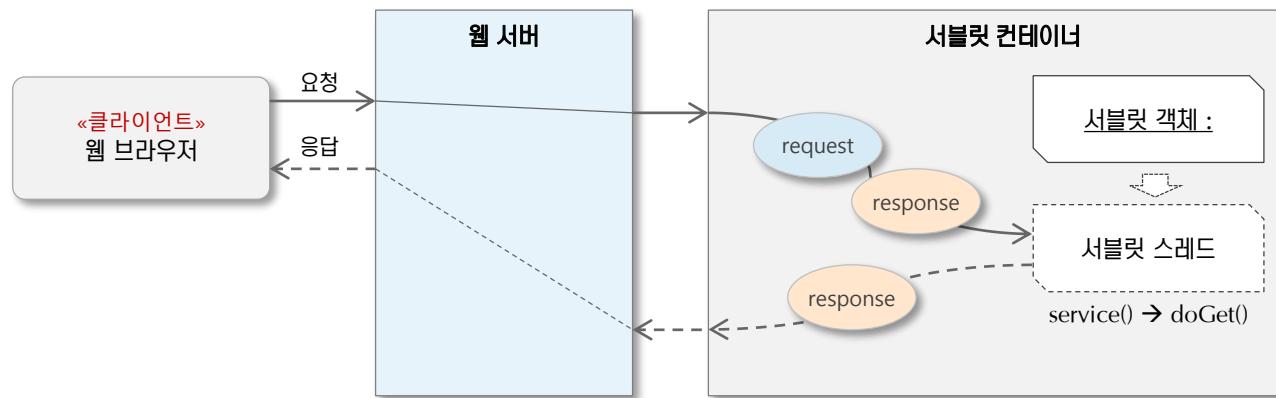
방법 #2. @WebServlet 어노테이션 선언 (Servlet 3.0+)

2.2 서블릿 컨테이너 (1/4) – 컨테이너는...

- ✓ 서블릿 컨테이너는 소켓 생성, 포트 리스닝 등과 같은 웹 서버와 서블릿의 커뮤니케이션을 지원합니다.
- ✓ 서블릿 컨테이너는 서블릿 객체의 생성, 초기화, 호출, 소멸과 같은 서블릿의 라이프사이클을 관리합니다.
- ✓ 서블릿 컨테이너는 다중 요청에 대한 스레드 생성 및 운영에 대해 관리합니다.
- ✓ 서블릿 컨테이너는 XML 설정을 통한 선언적인 보안관리를 제공합니다.
- ✓ 서블릿 컨테이너는 JSP를 지원합니다.

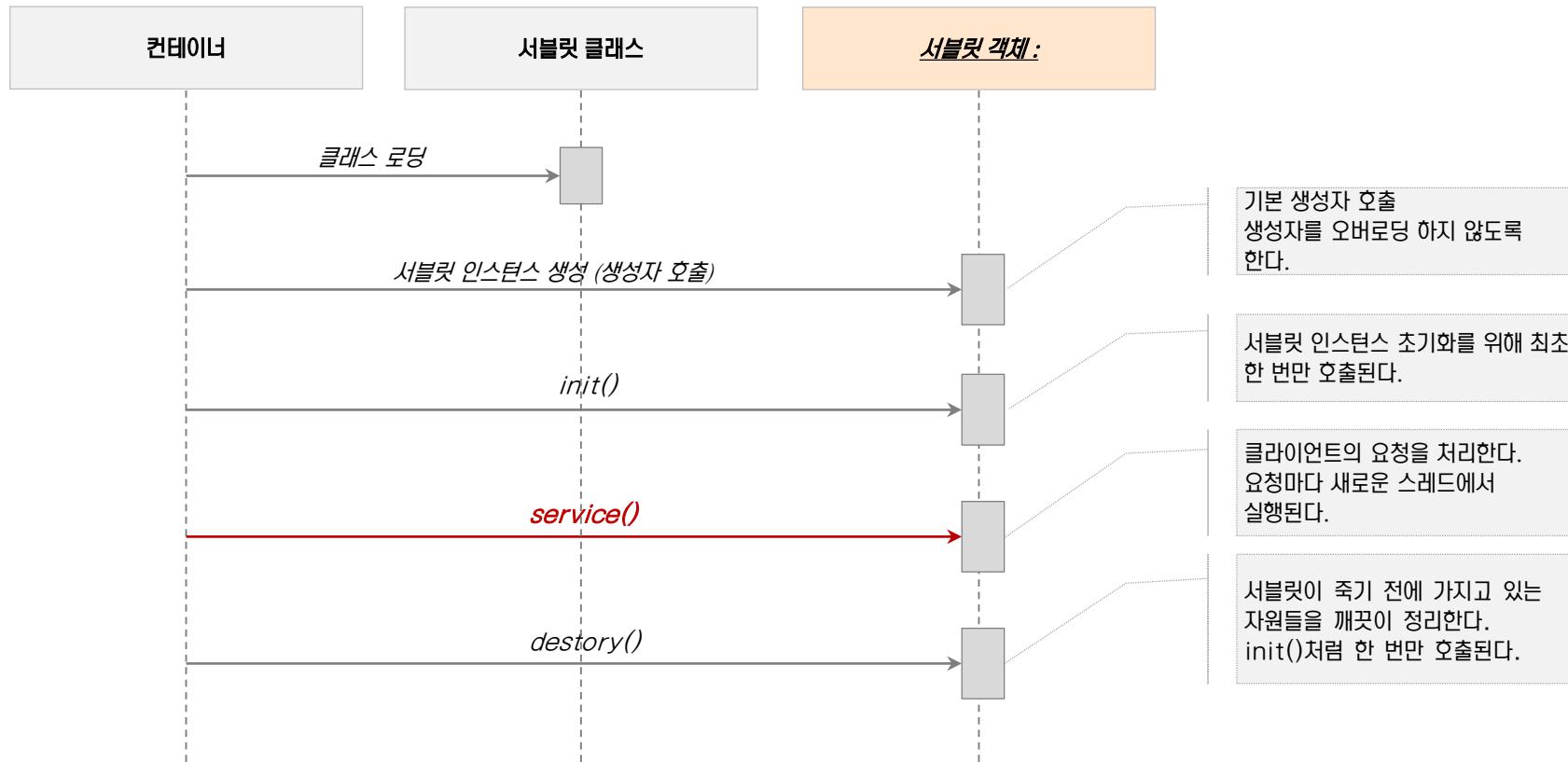
2.2 서블릿 컨테이너 [2/4] – Servlet 요청 처리 과정

- ✓ 클라이언트(웹 브라우저)가 웹 서버로 HTTP 요청을 보냅니다.
- ✓ 웹 서버는 클라이언트의 요청을 받아들여 컨테이너에게 전달합니다.
- ✓ 컨테이너는 HTTP Request, HTTP Response 객체를 만들어 서블릿을 호출합니다.
- ✓ 서블릿은 요청을 처리하고 결과인 응답객체를 요청 역순으로 클라이언트(브라우저)에게 전송합니다.



2.2 서블릿 컨테이너 (3/4) – Servlet 라이프사이클 (1/2)

- ✓ 서블릿 컨테이너는 서블릿의 라이프사이클을 관리합니다.



출처 : Bryan Basham, Kathy Sierra, Bert Bates, 「Head First Servlets and JSP」 : O'Reilly Media, 2008, p.131

2.2 서블릿 컨테이너 [4/4] – Servlet 라이프사이클 [2/2]

✓ init()

- 클라이언트의 요청을 처리하기 전에 서블릿을 초기화하는 메소드
- 컨테이너가 서블릿 인스턴스를 생성한 후 호출함
- 필요한 경우 재정의 가능 (예 : 데이터 베이스 접속)
- init() 메소드 호출 시점은 서버마다 그리고 설정에 따라 다를 수 있음

✓ service()

- 클라이언트 요청을 받아 처리하는 메소드
- 요청의 HTTP 메소드(GET/POST 등)를 참조하여 doXXX() 메소드를 호출함
- 재정의하지 않음 (상속받은 상위 클래스의 service() 메소드를 그대로 사용)

✓ doGet() / doPost()

- 개발이 필요한 부분
- 요청 메소드 유형에 따라 메소드를 재정의하여 구현함

✓ destory()

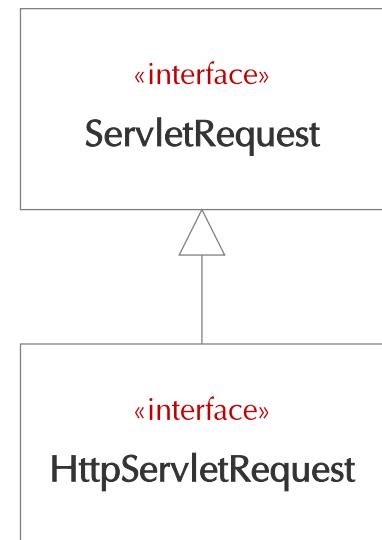
- 요청처리가 끝나면 컨테이너가 호출하는 메소드
- 재정의하지 않음

2.3 요청과 응답 (1/4) – HttpServletRequest

✓ HttpServletRequest 객체는 클라이언트의 요청정보를 관리합니다.

✓ HttpServletRequest 인터페이스의 메소드

- getHeader() : 요청 정보의 헤더정보를 반환
- getMethod() : 요청 정보의 HTTP Method 정보 반환
- getParameter() : 파라미터명으로 요청 값 반환
- getParameterValues() : 파라미터명으로 요청 값 배열 반환
- setCharacterEncoding() : 요청정보의 인코딩 설정

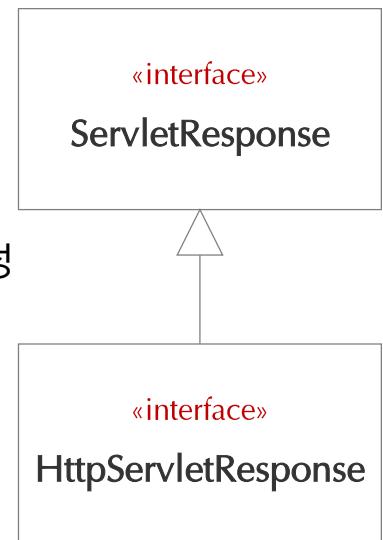


2.3 요청과 응답 (2/4) – HttpServletResponse

✓ **HttpServletResponse** 객체는 클라이언트에게 제공하는 응답정보를 관리합니다.

✓ **HttpServletResponse** 인터페이스의 메소드

- `setContentType()` : 클라이언트로 보내는 컨텐츠 타입 설정
- `getWriter()` : PrintWriter 객체반환, 클라이언트에게 전달할 텍스트 데이터 작성
- `getOutputStream()` : ServletOutputStream 객체반환, 클라이언트에게 전달할 바이너리 데이터 작성
- `setStatus()` : 응답에 대한 상태코드 지정



2.3 요청과 응답 (3/4) – ContentType

- ✓ ContentType은 브라우저에게 지금 반환하는 데이터가 무엇인지를 미리 알려주는 응답헤더입니다.
- ✓ 브라우저는 응답헤더의 ContentType을 보고 반환된 컨텐츠를 적절한 형태로 보여줍니다.
- ✓ 출력 스트림에 데이터를 기록하기 전에 setContentType() 메소드를 호출하여 ContentType을 설정해야 합니다.
- ✓ MIME TYPE : <http://www.iana.org/assignments/media-types/media-types.xhtml>
 - text/html
 - application/pdf
 - video/quicktime
 - application/java
 - image/jpeg
 - application/jar
 - application/octet-stream
 - application/x-zip

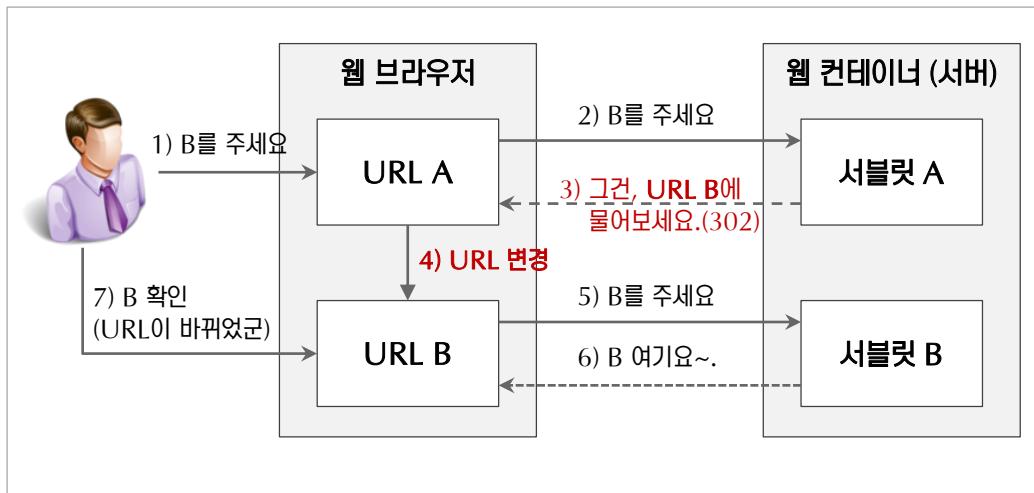
2.3 요청과 응답 (4/4) – 요청/응답처리 예

✓ LoginServlet.java

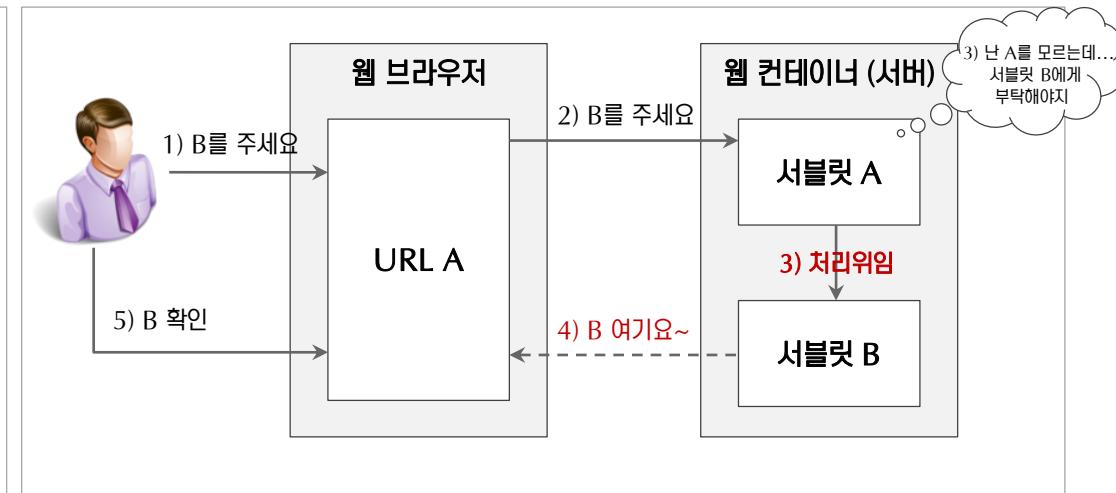
```
@Override  
protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
    throws ServletException, IOException {  
    req.setCharacterEncoding("UTF-8");  
    resp.setCharacterEncoding("UTF-8");  
    resp.setContentType("text/html");  
  
    PrintWriter out = resp.getWriter();  
    String id = req.getParameter("id");  
    String pw = req.getParameter("pw");  
    if (id == null || id.trim().length() <= 0) {  
        out.println("<!DOCTYPE html><html lang=\"ko\">"  
            + "<head><meta charset=\"UTF-8\" />"  
            + "<title>로그인 예제</title></head>"  
            + "<body><h1>ID를 입력하세요.</h1>"  
            + "<a href='javascript:history.back()'>뒤로</a>"  
            + "</body></html>");  
        out.flush();  
        return;  
    }  
  
    ...중략...  
  
    req.getRequestDispatcher("welcome.jsp").forward(req, resp);  
}
```

2.4 요청 위임하기 (1/3) – 요청을 위임하는 두 가지 방식

- ✓ 클라이언트 요청 위임이란 요청을 받은 서블릿이 다른 URL에 요청을 위임하는 것을 말합니다.
- ✓ 요청의 위임처리가 서버에서 일어나는지, 클라이언트(브라우저)에서 일어나는지에 따라 두 가지 방식이 있습니다.
- ✓ Redirect 방식은 클라이언트 즉, 브라우저에서 위임할 URL로 다시 요청하는 방식입니다.
- ✓ Request Dispatch 방식은 서버 내부에서 다른 URL로 요청처리가 일어납니다.



Redirect 방식



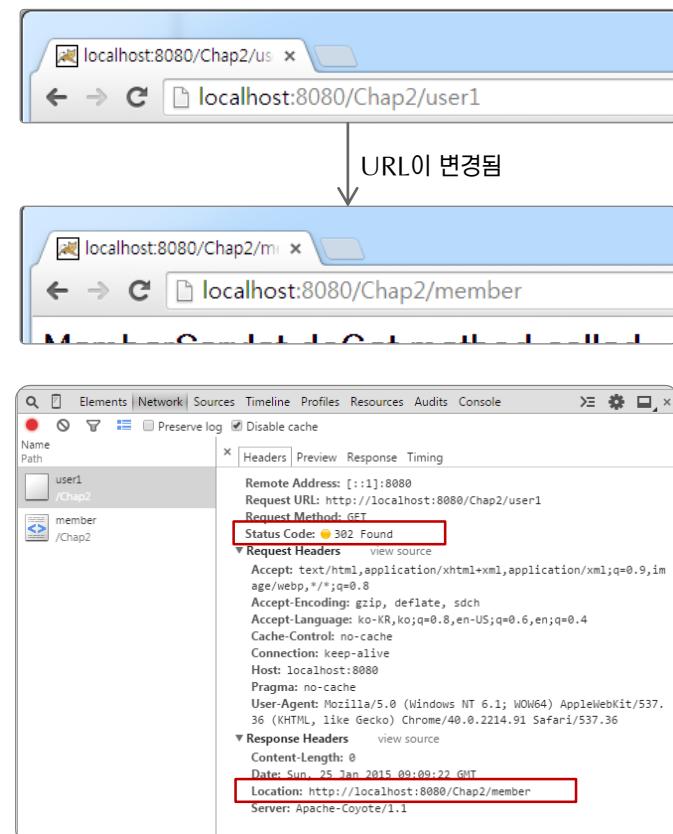
Request Dispatch 방식

2.4 요청 위임하기 (2/3) – Redirect 방식

- ✓ 요청을 받은 서블릿은 다른 URL에서 처리해야 할 대상인 경우 Response 객체의 sendRedirect()를 호출합니다.
- ✓ sendRedirect() 메소드는 HTTP 응답 메시지에 상태코드 30x와 Location 헤더에 다른 URL을 설정합니다.
- ✓ 브라우저는 응답을 받은 후 상태코드가 30x임을 확인하고 Location 헤더에 설정된 URL로 다시 요청합니다.
- ✓ 사용자도 브라우저에 변경된 URL을 확인할 수 있습니다.

```
@WebServlet("/user1")
public class RedirectUserServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.sendRedirect("member");
    }
}
```

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



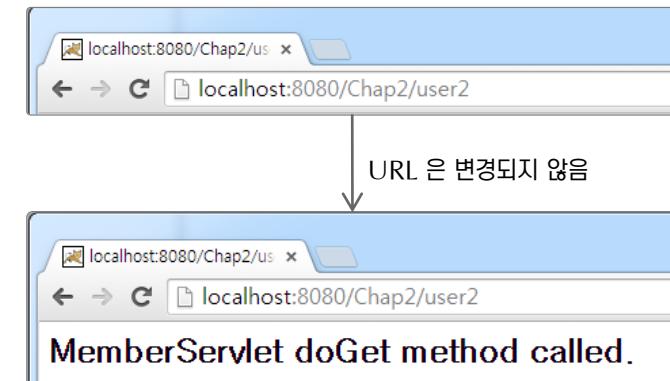
브라우저에서는 무슨 일이 일어났을까?

2.4 요청 위임하기 (3/3) – RequestDispatch 방식

- ✓ RequestDispatch 방식은 요청을 다른 방향으로 위임하는 작업이 서버 내에서 일어납니다.
- ✓ 요청을 받은 서블릿은 해당 요청이 다른 컴포넌트에서 처리해야 할 대상이라면 dispatch forward를 호출합니다.
- ✓ 브라우저의 URL은 바뀌지 않으므로 사용자는 Dispatch 된 응답인지를 알 수 없습니다.
- ✓ 클라이언트의 요청을 받은 서블릿이 View를 구성하기 위해 JSP로 요청을 위임할 때 가장 많이 사용됩니다.

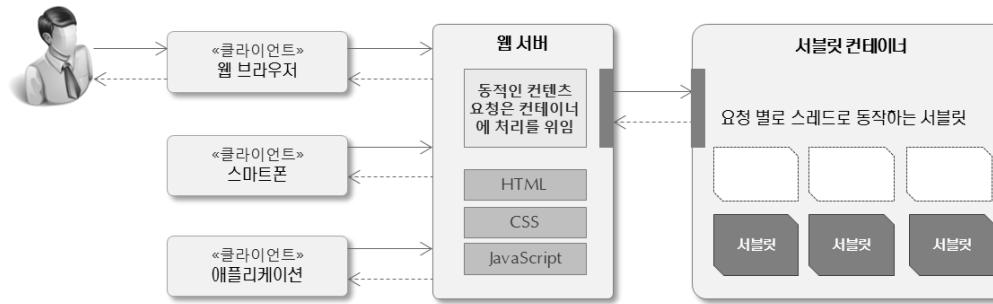
```
@WebServlet("/user2")
public class DispatchUserServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        RequestDispatcher dispatcher = req.getRequestDispatcher("member");
        dispatcher.forward(req, resp);
    }
}

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



2.5 요약

- ✓ 서블릿은 JAVA 언어로 웹 페이지를 생성할 수 있는 서버 프로그램으로써 HttpServlet 클래스를 상속받아 구현합니다.
- ✓ 서블릿 컨테이너는 서블릿이 동작하는 실행환경으로 서블릿의 생명주기를 관리합니다.
- ✓ HttpServletRequest 객체는 클라이언트의 요청정보를 관리하고, HttpServletResponse 객체는 응답정보를 관리합니다.
- ✓ 요청을 위임하는 두 가지 방법으로 Redirect, Request Dispatch 방식이 있습니다.





3. JSP (Java Server Pages) 이해

-
- 3.1 JSP 소개
 - 3.2 JSP 지시자
 - 3.3 JSP 스크립팅 요소
 - 3.4 Expression Language
 - 3.5 JSP 표준액션
 - 3.6 JSP 내장객체
 - 3.7 요약

3.1 JSP 소개 [1/2] – 개요

- ✓ JSP는 동적으로 웹 페이지를 생성하여 클라이언트에 전달하기 위한 언어로 웹 컨테이너에서 구동됩니다.
- ✓ 서블릿이 Java에 HTML을 포함하는 것과 달리 JSP는 HTML 문서에 Java 코드를 넣는 방식으로 개발합니다.
- ✓ JSP는 JSP 컨테이너에 의해 자동으로 서블릿으로 변환되어 클라이언트 요청을 처리합니다.
- ✓ JSP와 서블릿을 사용하여 개발하는 방식에 따라 모델1 방식과 모델2 방식으로 구분합니다.

```
public class 서블릿 extends HttpServlet {  
  
    protected void doGet(HttpServletRequest req,  
                         HttpServletResponse resp) ... {  
  
        ...  
        ...  
        ...  
        ...  
    }  
}
```

Servlet(Java 코드)에 포함된 HTML 문서

```
<html>  
  <head>...</head>  
  <body>  
    ...  
    <%  
      ...  
    %>  
    <br />  
    ...  
  </body>  
</html>
```

Java 코드

JSP(HTML 문서)에 포함된 Java 코드

3.1 JSP 소개 [2/2] – 구성요소

- ✓ JSP 문서는 템플릿 텍스트(*)와 동적 컨텐츠 생성을 담당하는 지시자, 액션, 스크립팅 요소를 포함합니다.
- ✓ 지시자(Directives)는 JSP가 서블릿으로 변환되는 단계에서 필요한 전역 정보를 제공합니다.
- ✓ 액션은 요청을 처리하는 단계에서 사용되며 JSP에서 제공하는 표준액션과 커스텀 액션이 있습니다.
- ✓ 스크립팅 요소는 컨텐츠 생성에 영향을 주는 요소로 템플릿 요소(예, HTML) 및 액션 사이에서 사용합니다.



* 템플릿 텍스트 : JSP 변환기에 의해 해석되지 않는 모든 텍스트 (예, HTML 요소 등)

3.2 JSP 지시자 (1/4) – 지시자의 종류

- ✓ 지시자(Directives)는 컨테이너에 의해 JSP를 서블릿으로 변환하는 시점에 필요한 정보를 제공합니다.
- ✓ page 지시자는 페이지 관련 환경정보를 제공하며 문자 인코딩, 컨텐츠 타입 등의 속성을 정의합니다.
- ✓ taglib 지시자는 JSP 페이지 내에서 태그 라이브러리를 사용하기 위해 선언합니다.
- ✓ include 지시자는 현재 페이지에 다른 페이지를 포함시킬 때 사용합니다.

page 지시자

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
```

JSP 페이지가 UTF-8로 인코딩 되었음을 컨테이너에 알려주고, 응답 처리 시 브라우저가 html 라는 것을 알 수 있도록 MIME 타입을 text/html로 설정함

taglib 지시자

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

JSP 페이지 안에서 JSTL의 core tag library를 접두어 c를 사용할 수 있도록 선언함

include 지시자

```
<%@include file="copyright.jspf" %>
```

JSP 페이지에 copyright 문구를 정의하고 있는 서브문서를 포함시킴, copyright.jspf 의 확장자를 통해 이 문서가 단편적인 JSP 문서임을 알 수 있음

3.2 JSP 지시자 (2/4) – page 지시자

✓ page지시자의 속성

속성	설명
contentType	MIME 타입과 문자 인코딩 방식을 설정 (예, contentType="text/html; charset=UTF-8")
pageEncoding	JSP 문자 인코딩 방식을 설정 (디폴트 ISO-8859-1)
isErrorPage	오류 처리용 페이지 여부. true면 내장 예외객체(exception)를 사용할 수 있음 (디폴트 false)
errorCode	이 페이지에서 잡지 못한 예외사항을 보낼 오류 페이지 URL을 정의
import	생성될 서블릿 클래스에 추가될 자바 import문을 정의
isELIgnored	페이지를 서블릿으로 전환할 때 EL 표현식을 무시할 것인지를 결정
session	내장 session 객체를 가질지 여부 (디폴트 true)
autoFlush	자동적으로 버퍼링된 출력 결과를 비울지 여부 (디폴트 true)
buffer	응답객체에 값을 출력할 때 버퍼링을 사용할지 여부 (디폴트 값은 8kb)
language	스크립트릿에서 사용할 언어를 지정 (디폴트 java)

3.2 JSP 지시자 (3/4) – taglib 지시자

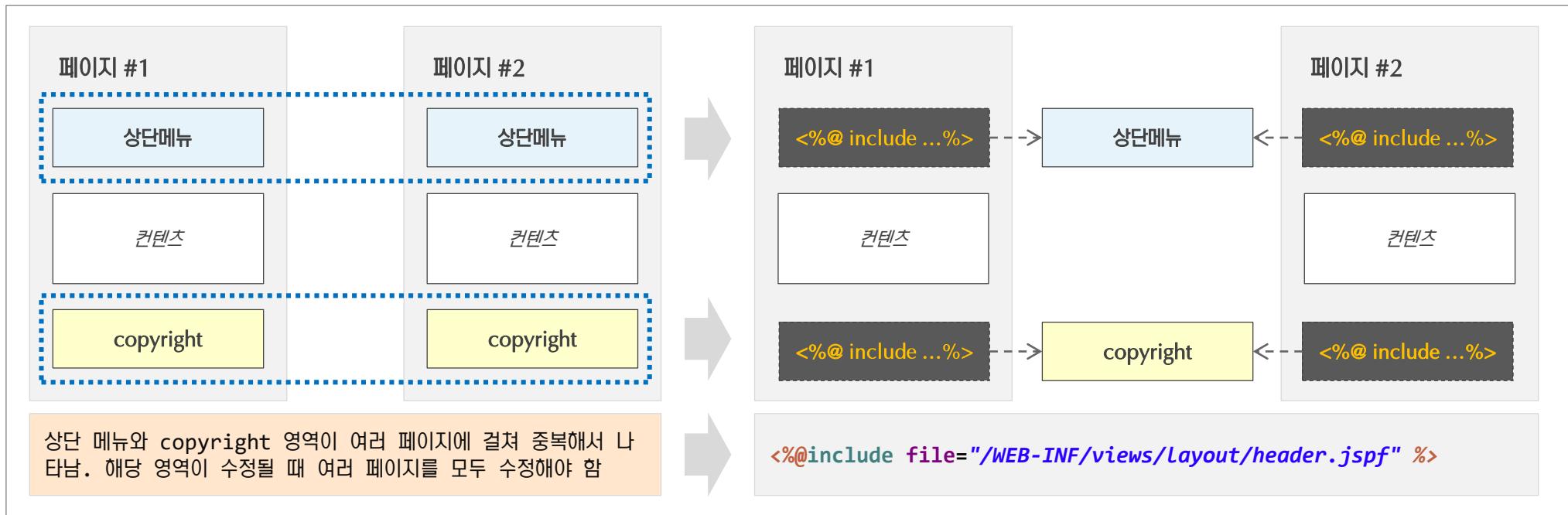
- ✓ 태그 라이브러리란 기본적인 JSP 내장 기능을 확장하기 위해 정의한 커스텀 태그들의 집합입니다.
- ✓ taglib 지시자는 커스텀 태그를 JSP에서 사용하기 위한 정보를 제공합니다.
- ✓ uri 속성에는 TLD 파일에 정의한 URI를 세팅합니다.
- ✓ prefix 속성에는 사용할 커스텀 태그들의 네임스페이스를 지정합니다.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> → JSTL의 Core Tag Library 를 네임스페이스 c로 접근하도록 함  
<%@ taglib prefix="mytag" tagdir="/WEB-INF/tags" %> → tags 폴더 하위에 있는 Tag 파일들을 네임스페이스 mytag로 접근하도록 함  
<%@ taglib prefix="namoo" uri="http://namoosori.com/tag" %> → 커스텀 태그 라이브러리를 네임스페이스 namoo로 접근함
```

*TLD 파일 : Tag Library Descriptor, 커스텀 태그에 대한 정보를 제공하고, 태그가 유효한 것인지를 검증하는데 사용하는 XML

3.2 JSP 지시자 (4/4) – include 지시자

- ✓ include 지시자는 지정한 파일을 JSP파일에 삽입할 때 사용합니다.
- ✓ include 된 파일들은 내용이 그대로 복사되어 포함되고 이에 대한 서블릿 코드는 하나만 생성됩니다.
- ✓ 중첩하여 사용할 수 있습니다. 즉, 포함되는 JSP 내부에서 또 다른 파일을 include 할 수 있습니다.
- ✓ 상단 메뉴나 하단 문구 등과 같은 공통적인 부분을 분리하여 관리하기 위해 주로 사용합니다.



include 지시자를 사용한 페이지 구성

3.3 JSP 스크립팅 요소 (1/4) – 스크립트릿

- ✓ 스크립트릿(Scriptlet)은 JSP 페이지에 자바 코드를 삽입하기 위해 사용합니다.
- ✓ page 지시자에 선언된 language에 해당하는 언어로 작성합니다. 기본적으로 java 언어를 사용합니다.
- ✓ 외부 클래스를 사용할 때는 page 지시자에 import 속성을 지정하거나 패키지를 포함한 클래스 이름을 사용합니다.
- ✓ request, response, out, session, config 같은 JSP 내장 객체를 사용할 수 있습니다.

```
<%
    User loginUser = (User) session.getAttribute("loginUser");
    if (loginUser == null) {
%>
    <a href="${ctx}/user/Login.do">로그인</a>
<%
    } else {
%>
    <b><% out.print(loginUser.getName()); %></b> 님!! 환영합니다.
<%
    }
%>
```

3.3 JSP 스크립팅 요소 (2/4) – 표현식

- ✓ 표현식(Expression)은 JSP 페이지에 출력할 내용을 나타내기 위해 사용합니다.
- ✓ 표현식으로 작성한 내용은 컨테이너가 `out.print()` 메소드 호출로 변환합니다.
- ✓ 표현식은 문장의 마지막에 세미콜론(;)은 붙이지 않습니다.
- ✓ 반환 값이 없는 경우, 즉 리턴 타입이 `void`인 경우 표현식을 사용할 수 없습니다.



3.3 JSP 스크립팅 요소 (3/4) – 선언문

- ✓ 선언문(Declarations)는 JSP 페이지에서 사용할 변수나 메소드를 선언하기 위해 사용합니다.
- ✓ 선언문은 JSP 컨테이너가 JSP를 서블릿 코드로 변환하는 시점에 인스턴스 변수, 메소드로 변환합니다.
- ✓ 선언문으로 작성한 변수는 서블릿의 인스턴스 변수로 변환합니다. 서블릿은 멀티 스레드 환경에서 동작하므로 동기화 문제가 발생하지 않도록 선언문을 신중하게 사용해야 합니다.

```
<%! int counter = 0; %>  
  
<%! int getNextCounter() {  
    return ++counter;  
} %>
```

선언문으로 변수와 메소드를 선언함

```
public final class cookbookList_jsp extends  
org.apache.jasper.runtime.HttpJspBase  
implements org.apache.jasper.runtime.JspSourceDependent {  
  
    int counter = 0;  
    int getNextCounter() {  
        return ++counter;  
    }  
}
```

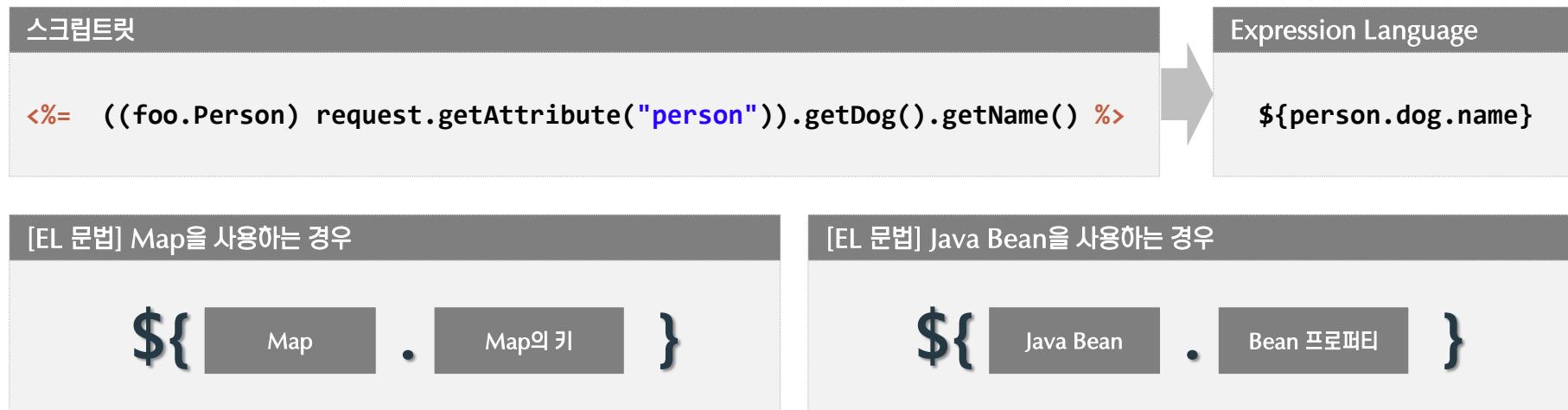
JSP 컨테이너에 의해 서블릿으로 변환된 코드

3.3 JSP 스크립팅 요소 [4/4] – 기타

- ✓ HTML 주석은 <!-- Comment --> 와 같이 작성하여 클라이언트에 제공되는 문서에 포함됩니다.
- ✓ HTML 주석은 브라우저의 [소스보기]를 통해 주석 내용을 확인할 수 있습니다.
- ✓ 반면, JSP 주석은 <%-- Comment -->와 같이 작성하여 서블릿으로 변환될 때 자바 소스 코드 주석으로 변환됩니다.
- ✓ 자바 소스코드의 주석은 클라이언트의 응답에 포함되지 않으므로 사용자가 [소스보기]를 통해 확인할 수 없습니다.

3.4 Expression Language [1/6] – 개요

- ✓ EL(Expression Language)은 JSP에서 스크립트릿이나 표현식으로 했던 작업을 간단하게 처리할 수 있습니다.
- ✓ EL 표현식에서 도트 연산자 왼쪽은 반드시 java.util.Map 객체 또는 Java Bean 객체여야 합니다.
- ✓ EL 표현식에서 도트 연산자 오른쪽은 반드시 맵의 키이거나 Bean 프로퍼티여야 합니다.
- ✓ 오른쪽에 오는 값은 식별자로서 일반적인 자바 명명 규칙을 따라야 합니다.



3.4 Expression Language [2/6] – [] 연산자

- ✓ EL에는 Dot 표기법 외에 [] 연산자를 사용하여 객체의 값에 접근할 수 있습니다.
- ✓ [] 연산자의 왼편에는 java.util.Map, Java Bean, 배열 또는 리스트가 올 수 있습니다.
- ✓ [] 연산자 안의 값이 문자열인 경우, 이것은 맵의 키가 될 수도 있고, Bean 프로퍼티나 리스트 및 배열의 인덱스가 될 수 있습니다. 배열과 리스트인 경우, 문자로 된 인덱스 값은 숫자로 변경하여 처리합니다.

[] 연산자를 이용한 객체 프로퍼티 접근

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

Dot 표기법을 이용한 객체 프로퍼티 접근

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

리스트나 배열 요소에 접근

```
// In Servlet
String[] courses = {"JSP/Servlet", "jQuery", "SpringMVC"};
request.setAttribute("courses", courses);

// EL in JSP
${courses[0]} // JSP/Servlet 이 출력됩니다.
${courses["1"]} // 문자열인 인덱스 값이 숫자로 바뀌어 courses[1]의 결과인 jQuery 가 출력됩니다.
```

3.4 Expression Language [3/6] – EL 내장 객체 [1/3]

- ✓ EL 내장 객체는 JSP 페이지의 EL 표현식에서 사용할 수 있는 객체입니다.
- ✓ EL 내장 객체에는 생존 범위 속성 맵, 요청 파라미터 맵, 요청 헤더 맵, 쿠키 맵, 컨텍스트 초기화 파라미터 맵이 있습니다.
- ✓ pageContext는 EL 내장 객체 중 유일하게 맵이 아닌 Java Bean입니다.

구분	타입	내장 객체명
생존범위 속성	Map	pageScope, requestScope, sessionScope, applicationScope
요청 파라미터	Map	param, paramValues
요청 헤더	Map	header, headerValues
컨텍스트 초기화 파라미터	Map	initParam
pageContext 객체 참조	Java Bean	pageContext

출처 : Bryan Basham, Kathy Sierra, Bert Bates, 「Head First Servlets and JSP」 : O'Reilly Media, 2008, p.415

3.4 Expression Language [4/6] – EL 내장 객체 [2/3]

- ✓ requestScope 는 request 생존범위에 묶여 있는 속성(Attributes)에 대한 단순한 맵입니다.
- ✓ EL에서는 request 객체에 직접 접근할 수 없고 pageContext를 통해서 접근할 수 있습니다.
- ✓ 이름 충돌이 발생하는 경우 생존범위 내장 객체를 통해 명시적으로 값을 조회해야 합니다.

EL에서 request 객체 접근

```
// request의 method 프로퍼티 출력  
Method is : ${pageContext.request.method}
```

생존범위 내장객체를 통한 명시적인 속성 접근

```
// In servlet  
request.setAttribute("foo.person", p);  
  
// Case #1 : 예외  
${foo.person.name} // foo라는 속성은 존재하지 않음  
  
// Case #2 : request 생존범위 내장객체에서 []연산자를 통해 속성 접근  
${requestScope["foo.person"].name}
```

3.4 Expression Language [5/6] – EL 내장 객체 [3/3]

- ✓ JSP에서 쿠키 값을 출력하기 위해서는 스크립트릿을 사용하거나 EL 내장객체인 cookie를 사용합니다.
- ✓ initParam EL 내장객체를 통해 컨텍스트 초기화 파라미터 값을 출력할 수 있습니다.



3.4 Expression Language [6/6] – 연산자

✓ 산술연산자

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

✓ 논리연산자

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

✓ 관계연산자

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

✓ 그 밖에 연산자들

- empty
- true
- false
- null

3.5 JSP 표준액션 (1/4) – 소개

- ✓ JSP 표준액션은 스크립팅 요소를 사용하지 않고 XML 문법으로 표현할 수 있는 방법을 제공합니다.
- ✓ Java Bean 객체를 다룰 때는 <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty> 표준액션을 사용합니다.
- ✓ <jsp:include> 표준액션을 통해 정적 또는 동적인 자원을 현재 페이지에 포함시킬 수 있습니다.

스크립트릿을 통한 Java Bean 접근

```
<html>
<body>
<%
    com.namoo.Course course = (com.namoo.Course) request.getAttribute("course");
%>
Person is : <%= course.getName() %>
</body>
</html>
```



HTML 안에 포함된 Java 코드는
HTML 구조를 한눈에 파악하기 어렵게
합니다.



JSP 표준액션을 사용한 Java Bean 접근

```
<html>
<body>
    <jsp:useBean id="course" class="com.namoo.Course" scope="request" />
    <jsp:getProperty name="course" property="name" />
</body>
</html>
```

3.5 JSP 표준액션 (2/4) – Bean 표준액션 (1/2)

- ✓ JSP 표준액션에는 Java Bean 객체를 다룰 수 있는 3가지 표준액션이 존재합니다.
- ✓ <jsp:useBean> 은 지정한 생존범위에 속성 Bean을 선언하고 초기화하는 표준액션입니다.
- ✓ <jsp:getProperty> 은 속성 Bean 객체의 프로퍼티를 읽어오는 표준액션입니다.
- ✓ <jsp:setProperty> 은 속성 Bean 객체의 프로퍼티에 값을 설정하는 표준액션입니다.

<jsp:useBean> 표준액션

```
<jsp:useBean id = "course"
    class = "com.namoo.Course"
    scope = "request" />
```

course 식별자를 가진 Course 타입의 빈 객체를 request 생존범위의 속성으로 선언합니다. 생존 범위를 생략하는 경우 생존범위는 page가 됩니다. 기존에 동일한 이름의 빈이 있으면 해당 빈이 사용되고, 없으면 새로운 빈 객체가 생성됩니다.

<jsp:getProperty> 표준액션

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

course라는 빈 객체의 name 프로퍼티 값을 출력합니다.

<jsp:setProperty> 표준액션

```
<jsp:setProperty name="course" property="name"
    value="JSP&Servlet" />
```

course라는 빈 객체의 name 프로퍼티 값에 “JSP&Servlet”을 설정합니다.

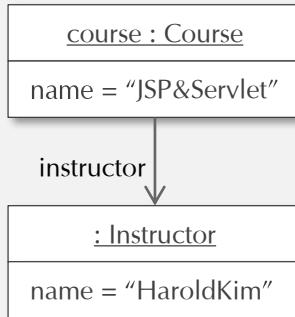
3.5 JSP 표준액션 (3/4) – Bean 표준액션 (2/2)

- ✓ <jsp:useBean> 표준액션은 몸체를 가질 수 있습니다.
- ✓ 몸체의 내용은 <jsp:useBean> 액션을 통해 Bean 객체가 새로 생성되는 경우에만 동작합니다.
- ✓ 예를 들어, 새로운 Bean 객체를 만들면서 디폴트 값을 설정할 경우 몸체에 <jsp:setProperty> 를 정의하면 됩니다.
- ✓ Bean 표준액션은 중첩된 객체의 프로퍼티의 값을 출력할 수 없는 단점이 있습니다. 이러한 경우에는 표현식이나 EL을 사용해서 처리합니다.

<jsp:getProperty> 표준액션

```
<jsp:useBean id="course" class="com.namoo.Course" scope="request">
    <jsp:setProperty name="course" property="name" value="JSP&Servlet" />
</jsp:useBean>
```

request 생존범위



[Bean 표준 액션의 단점] 중첩된 객체의 프로퍼티에 접근할 수 없다.

```
<jsp:useBean id="course" class="com.namoo.Course" scope="request" />
<jsp:getProperty name="course" property="instructor" />
```

→ instructor 객체에는 접근할 수 있지만 instructor 객체의 name 프로퍼티에 접근할 방법이 없습니다.

[해결방안 #1] 표현식 사용하기

```
<%= course.getInstructor().getName() %>
```

[해결방안 #2] EL 사용하기

```
${course.instructor.name}
```

3.5 JSP 표준액션 (4/4) – include 표준 액션

- ✓ <jsp:include>는 현재 페이지에 다른 웹 자원을 포함시킬 때 사용하는 표준액션입니다.
- ✓ include 지시자를 사용하면 서블릿으로 변환되는 단계에 소스 그대로 미리 포함되지만, include 표준액션은 요청을 처리하는 시점에 실행된 결과가 포함됩니다.
- ✓ <jsp:include>의 몸체에 <jsp:param>을 선언하면 포함되는 페이지에 파라미터를 전달할 수 있습니다.

<jsp:include> 표준액션 사용하기

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>namoosori</title>
</head>
<body>

<jsp:include page="header.jsp">
    <jsp:param name="courseName" value="JSP&Servlet" />
</jsp:include>
```

JSP&Servlet을 소개합니다.

```
...
</body>
</html>
```

header.jsp

```
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<br/>
<h2>
    Namoosori Course –
    <span style="color:darkblue">
        ${param.courseName}
    </span>
</h2>
```



Namoosori Course – JSP&Servlet

JSP&Servlet을 소개합니다. ...

3.6 JSP 내장 객체 (1/3) – 생존범위와 내장 객체

- ✓ 웹 애플리케이션에는 요청, 세션 그리고 하나의 웹 애플리케이션을 나타내는 컨텍스트 생존범위가 있습니다.
- ✓ 추가적으로 JSP 페이지에는 해당 페이지 내에서만 유효한 page 생존범위가 있습니다.
- ✓ JSP에는 스크립팅 요소를 통해 생존범위 객체에 접근할 수 있는 네 개의 내장 객체가 제공됩니다.
- ✓ JSP 내장 객체를 통하여 각각의 생존범위에 속성을 설정하거나 조회할 수 있습니다.

생존범위	JSP 내장 객체	JSP 코드 예제	서블릿 코드 예제
Context	application	<code>application.setAttribute("obj", someObj);</code>	<code>getServletContext().setAttribute("obj", SomeObj);</code>
Session	session	<code>session.setAttribute("obj", someObj);</code>	<code>request.getSession().setAttribute("obj", someObj);</code>
Request	request	<code>request.setAttribute("obj", someObj);</code>	<code>request.setAttribute("obj", someObj);</code>
Page	pageContext	<code>pageContext.setAttribute("obj", someObj);</code>	제공되지 않음

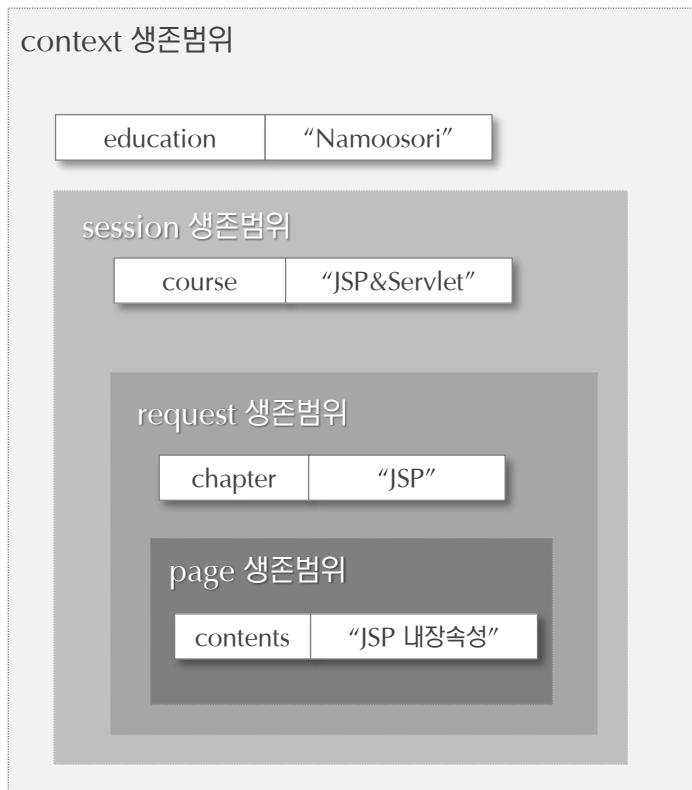
3.6 JSP 내장객체 (2/3) – 내장객체의 메소드

- ✓ 생존범위 내장객체에는 객체의 속성을 설정하거나 조회할 수 있는 동일한 메소드가 제공됩니다.
- ✓ 생존범위에 속성 값을 설정하거나 조회할 때는 `setAttribute` 및 `getAttribute` 메소드를 사용합니다.
- ✓ `getAttributeNames` 메소드는 해당 생존범위 객체가 가진 모든 속성의 이름을 반환합니다.
- ✓ `removeAttribute` 메소드는 해당하는 속성을 생존범위에서 삭제합니다.

메소드 시그니처	설명
<code>public Object getAttribute(String key)</code>	key값으로 등록된 속성을 Object로 리턴하고 해당 key값이 존재하지 않을 경우 null을 리턴
<code>public void setAttribute(String key, Object value)</code>	속성 값을 key값으로 등록
<code>public Enumeration getAttributeNames()</code>	해당 생존범위의 모든 속성들의 이름을 Enumeration으로 리턴
<code>public void removeAttributte(String key)</code>	key 값에 해당하는 속성을 생존범위에서 삭제

3.6 JSP 내장객체 (3/3) – pageContext 객체

- ✓ pageContext는 page 범위 속성 뿐만 아니라, 다른 생존범위 속성에도 접근할 수 있는 기능을 제공합니다.
- ✓ getAttribute 메소드의 두 번째 인자로 생존범위를 명시하면 해당 생존범위 속성 값을 조회할 수 있습니다.
- ✓ setAttribute 메소드의 세 번째 인자로 특정 생존범위에 해당하는 속성 값을 설정할 수 있습니다.
- ✓ findAttribute 메소드는 가장 작은 생존범위 부터 넓은 범위 순으로 속성을 검색합니다.



page 생존범위 속성 값 조회

```
<%= pageContext.getAttribute("contents") %>
```

request 생존범위 속성 값 조회

```
<%= pageContext.getAttribute("chapter",  
    pageContext.REQUEST_SCOPE) %>
```

session 생존범위의 속성 값 설정

```
<%= pageContext.setAttribute("course", "SpringMVC",  
    pageContext.SESSION_SCOPE) %>
```

생존범위를 알 수 없는 속성 찾기

```
<%= pageContext.findAttribute("education") %>
```

3.7 요약

- ✓ JSP는 HTML문서에 JAVA코드를 포함한 형태입니다. 컨테이너는 JSP를 서블릿으로 변환합니다.
- ✓ 지시자는 JSP가 서블릿으로 변환되는 시점에 컨테이너에 제공하는 정보로써 page, taglib, include 지시자가 있습니다.
- ✓ EL은 스크립틀릿이나 표현식으로 했던 작업을 간단하게 처리합니다. \${Bean.Property} 형식으로 사용합니다.
- ✓ JSP에는 내장객체가 제공되고 이 객체를 통해서 세션, 요청, 페이지, 애플리케이션 컨텍스트 생존범위에 접근할 수 있습니다.



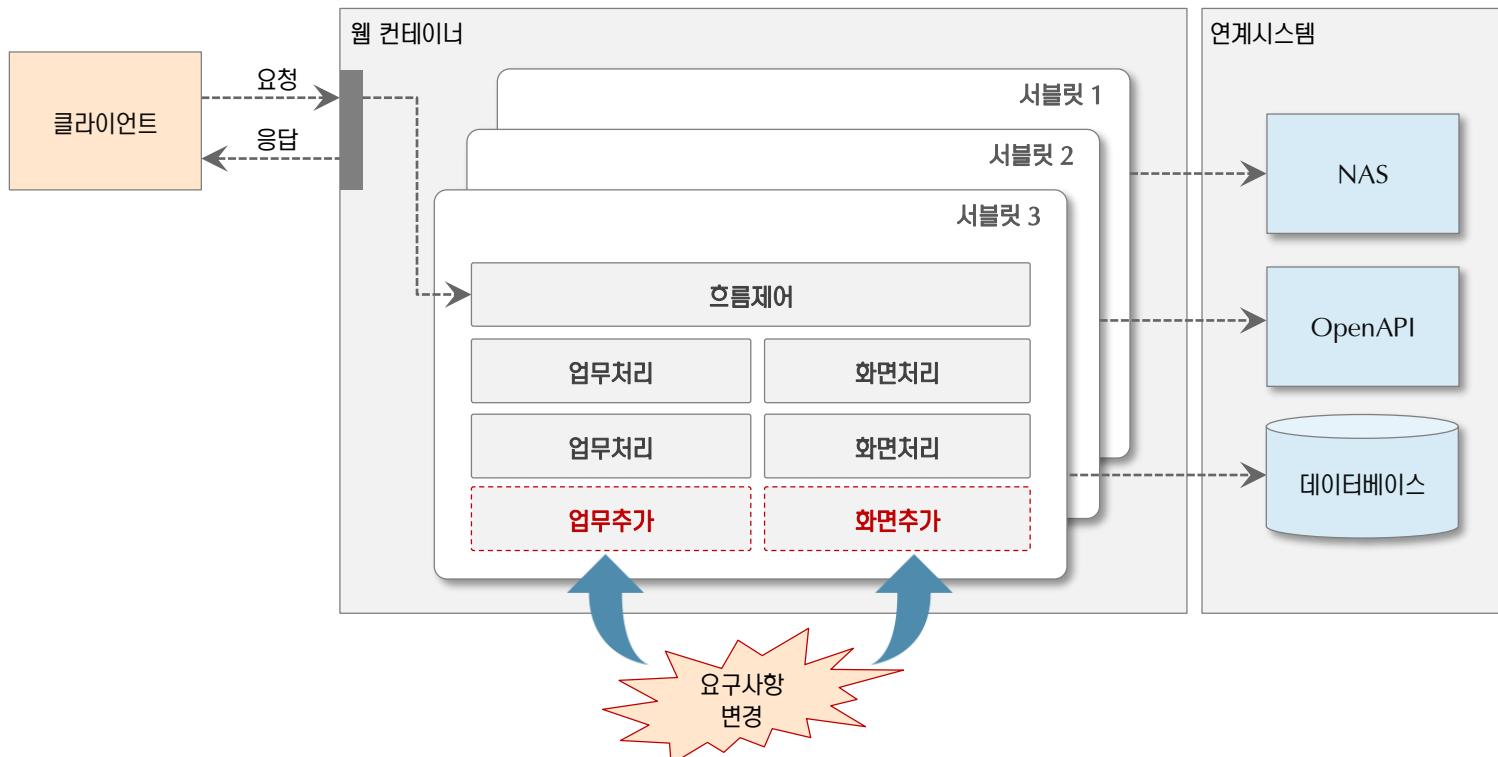


4. Java 웹 프로그래밍 I

-
- 4.1 웹 프로그래밍 개요
 - 4.2 세션과 쿠키
 - 4.3 요리책 만들기 #1
 - 4.4 서블릿 필터
 - 4.5 JSTL
 - 4.6 요약

4.1 웹 프로그래밍 개요 (1/6) – 요구사항의 변화와 웹 애플리케이션 (1/3)

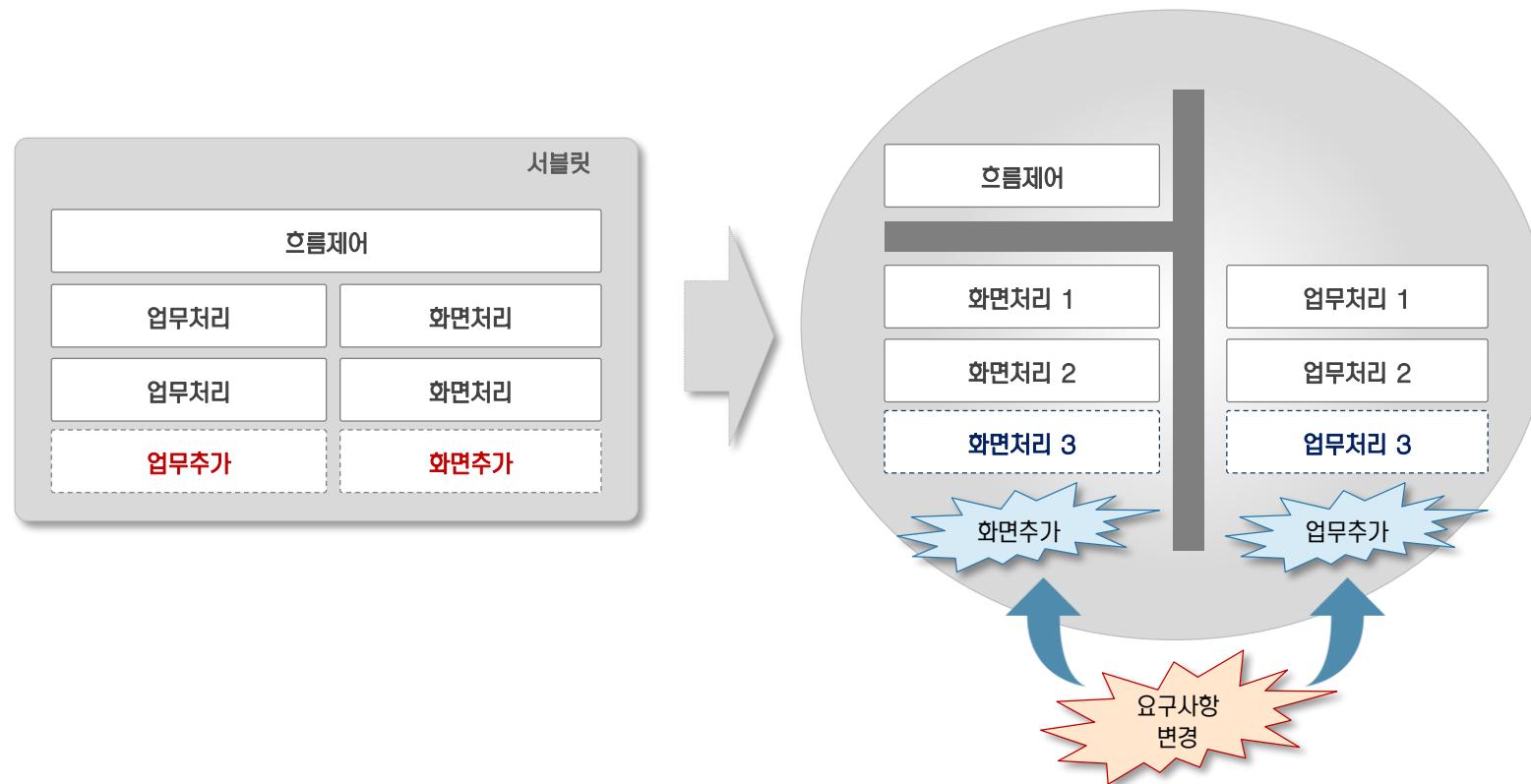
- ✓ 하나의 웹 컴포넌트(JSP 또는 서블릿)에서 하나의 요청에 대한 모든 것을 처리하는 웹 애플리케이션을 생각해 봅니다.
- ✓ 요청에 따른 흐름을 통제하는 로직과 요청한 내용을 처리하는 업무 로직 그리고 사용자에게 보여줄 결과를 생성하는 화면처리 과정이 하나의 웹 컴포넌트에 혼재되어 있습니다.
- ✓ 이러한 구조는 업무복잡도가 증가하거나 요구사항이 변경되는 경우, 프로그램 관리가 어렵습니다.



4.1 웹 프로그래밍 개요 (2/6) – 요구사항의 변화와 웹 애플리케이션 (2/3)

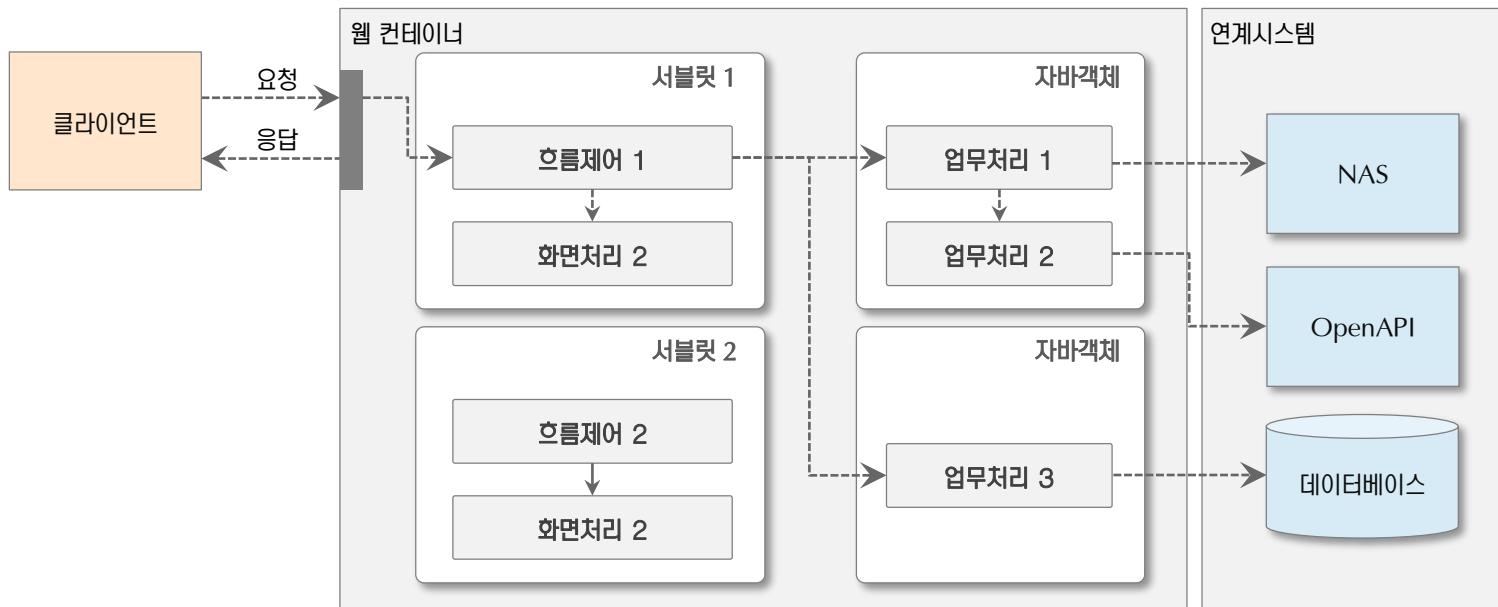
✓ 애플리케이션은 변경되는 부분과 변하지 않는 부분을 분리하여, 확장에 열려있고 변경에는 닫힌 구조로 설계되어야 합니다.

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



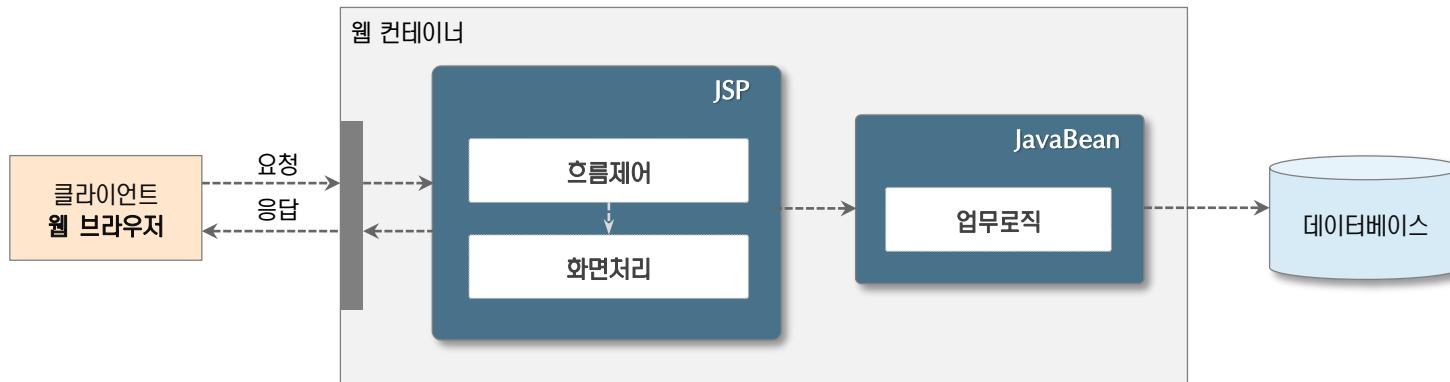
4.1 웹 프로그래밍 개요 (3/6) – 요구사항의 변화와 웹 애플리케이션 (3/3)

- ✓ 애플리케이션 내에서 관심사가 서로 다른 로직들은 서로 분리되어야 합니다.
- ✓ 서블릿은 요청에 대한 흐름통제와 화면처리와 관련된 기능을 담당합니다.
- ✓ 업무를 처리하는 로직은 관련된 것들끼리 묶어 별도의 자바객체로 분리합니다.
- ✓ 이제, 요구사항이 변화하더라도 관련된 부분만 변경되고 다른 모듈에는 영향을 주지 않습니다.



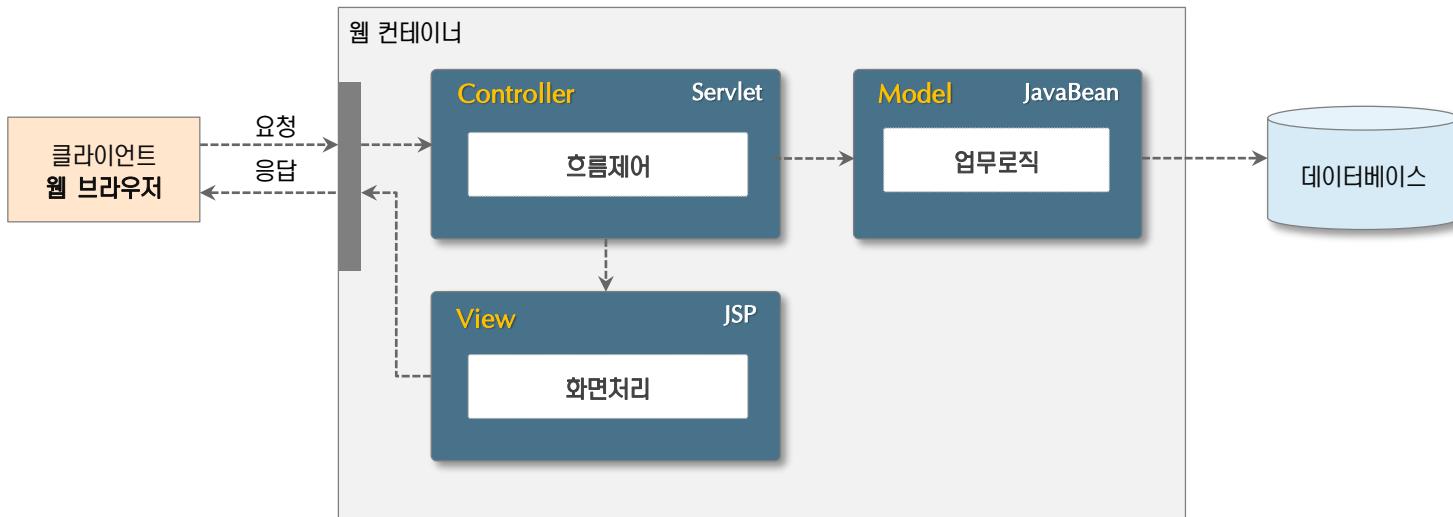
4.1 웹 프로그래밍 개요 (4/6) – 모델 1 개발방식

- ✓ 모델 1 개발방식은 규모가 작은 웹 애플리케이션을 개발할 때 적합하며 모델 2 개발방식에 비해 구조가 단순합니다.
- ✓ 그러나, 흐름 제어로직과 화면 처리가 JSP안에 혼재되어 있어 애플리케이션이 복잡해질수록 유지보수가 어려워집니다.
- ✓ HTML 코드와 Java코드가 섞여 있으므로 디자이너와 개발자간에 원활한 의사소통을 할 수 없습니다.
- ✓ 사용자의 증가하는 요구사항에 대응하기 어렵습니다.



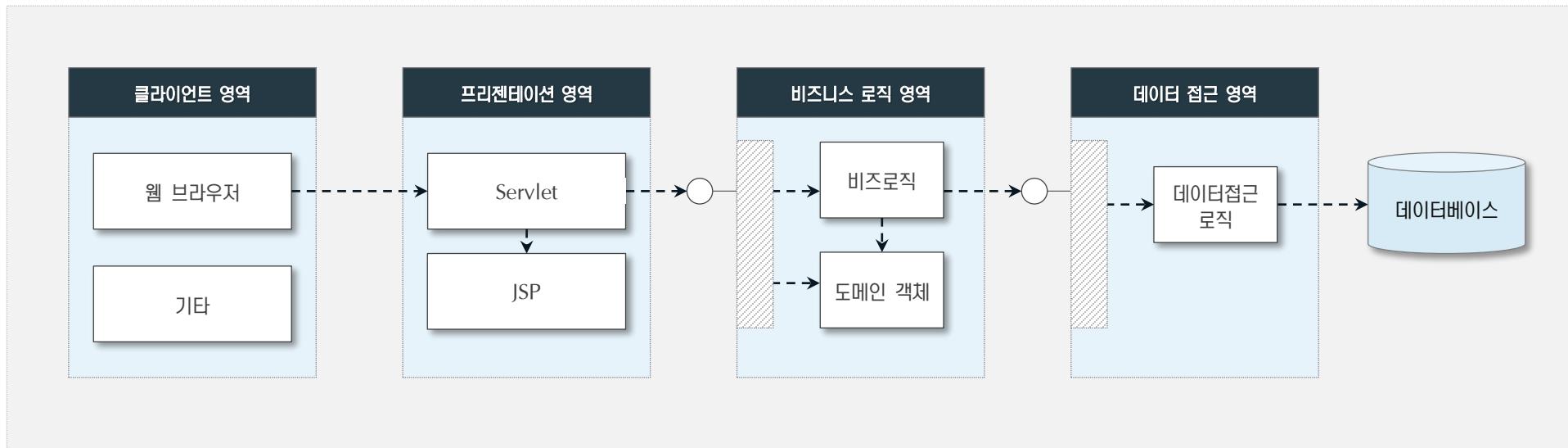
4.1 웹 프로그래밍 개요 (5/6) – 모델 2 개발방식 (MVC 모델)

- ✓ 모델 2 개발방식은 모델 1의 문제점인 흐름제어 로직과 화면처리를 서로 분리하는 것으로 MVC 모델이라고도 합니다.
- ✓ 서블릿은 흐름제어를 담당하고 JSP는 화면처리를 담당하므로 디자이너와 개발자의 작업을 분리할 수 있습니다.
- ✓ 그러나, 개발 초기에 아키텍처 설계를 위한 시간이 소요되어 개발 기간이 늘어날 수 있습니다.
- ✓ 모델 2 방식은 모델 1에 비해 구조가 복잡하여 개발자들의 이해가 필요합니다.



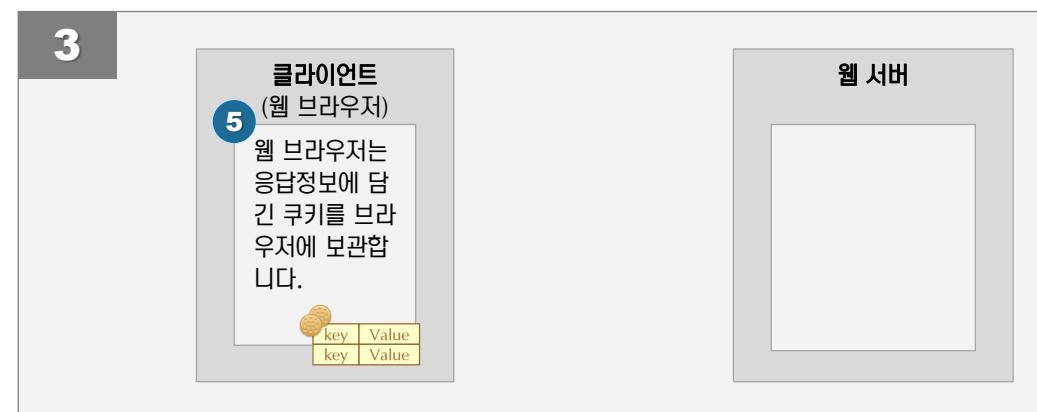
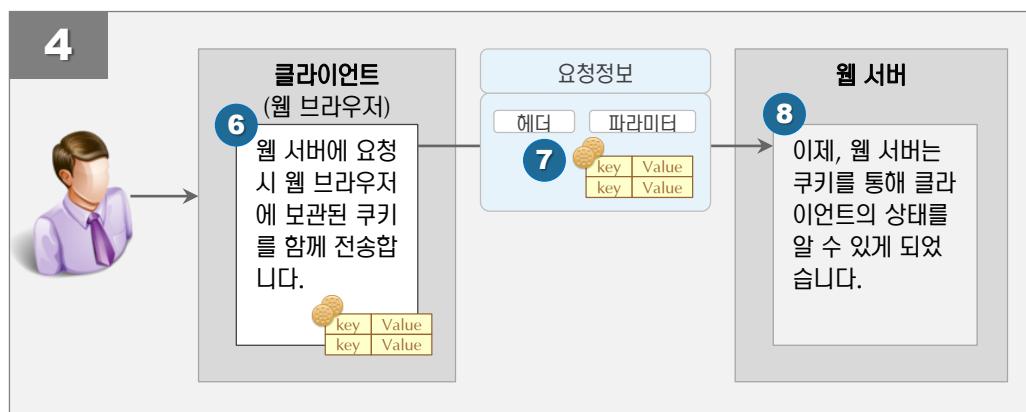
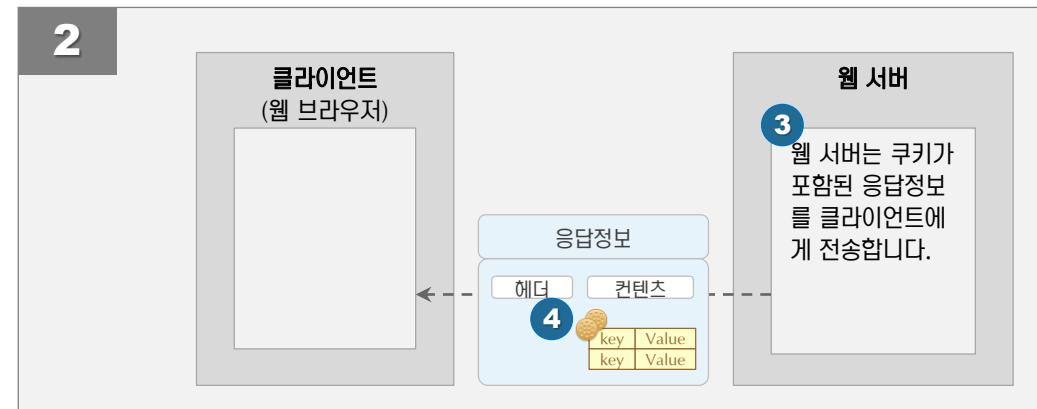
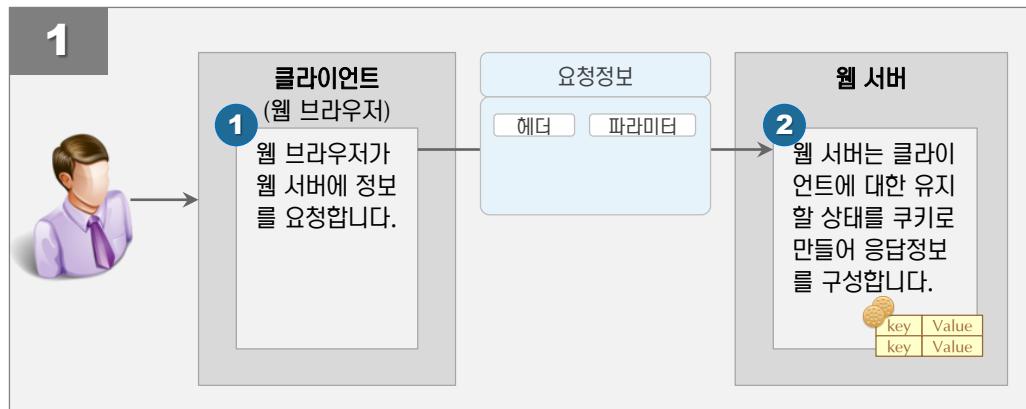
4.1 웹 프로그래밍 개요 (6/6) – 웹 애플리케이션 아키텍처

- ✓ 웹은 단순한 정보만을 보여주는 정적인 웹 페이지에서 동적인 내용을 구성하는 웹 애플리케이션으로 발전하였습니다.
- ✓ 엔터프라이즈 애플리케이션은 클라이언트-서버 환경에서 웹 기반 환경으로 패러다임이 전환되었습니다.
- ✓ 웹 애플리케이션은 클라이언트의 요청에 따라 정보를 가공하고 저장하며 다양한 형태로 보여줍니다.
- ✓ 다양한 요구사항으로 인해 웹 애플리케이션이 복잡해지는 것을 방지하려면 웹 애플리케이션 아키텍처가 필요합니다.



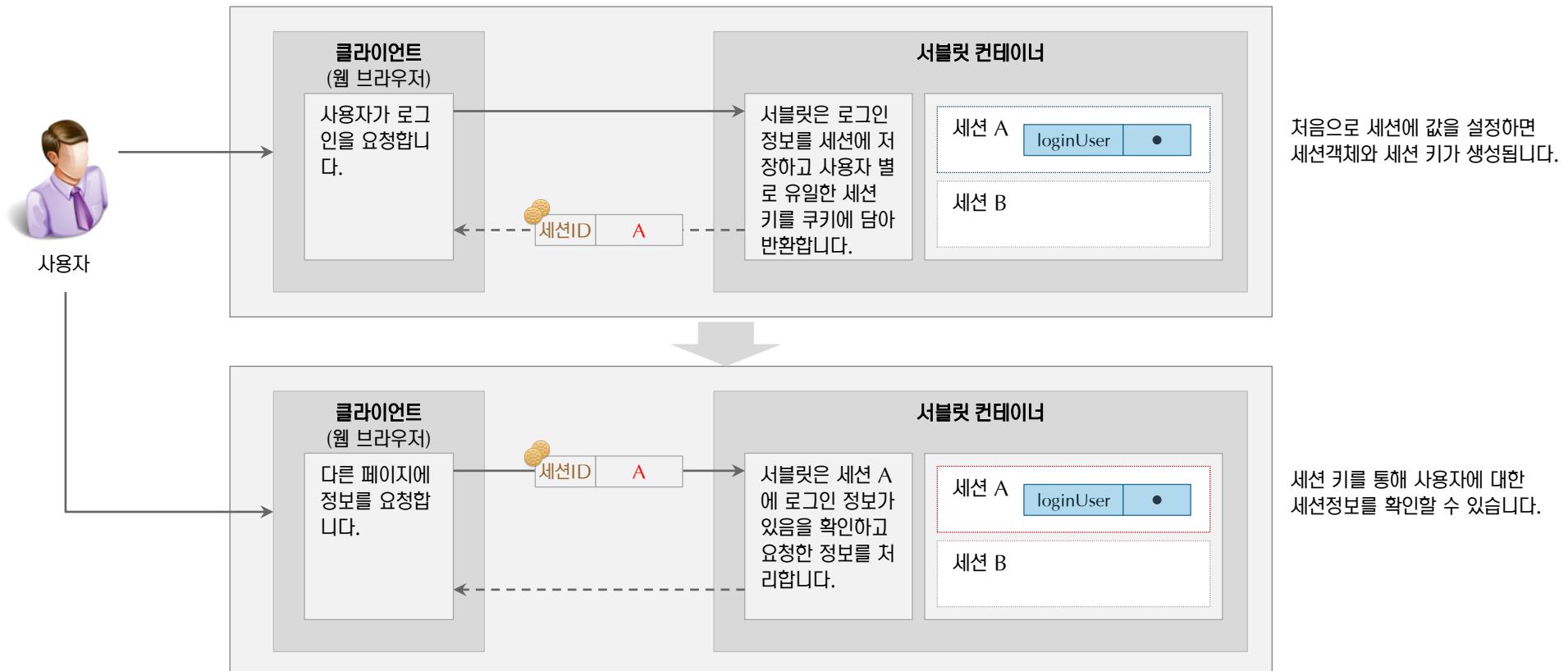
4.2 세션과 쿠키 (1/6) – Connectionless & Stateless

- ✓ HTTP 프로토콜은 비-연결지향으로 웹 서버가 클라이언트의 상태를 유지할 수 없습니다.
- ✓ 쿠키는 웹 서버에서 웹 브라우저로 전송되는 작은 데이터 조각으로 클라이언트의 유지해야 할 상태나 활동정보를 저장하기 위하여 설계되었습니다.
- ✓ 웹 서버로부터 쿠키가 전송되면 웹 브라우저는 저장해 두었다가 서버 요청 시 다시 전달하여 클라이언트 상태를 알게 합니다.



4.2 세션과 쿠키 (2/6) – Session 개요

- ✓ 쿠키가 클라이언트 상태를 유지하는 방법을 제공하지만 해당 정보는 웹 브라우저에 저장되므로 보안상 매우 취약합니다.
- ✓ 세션은 사용자에 대한 상태를 서버에 저장할 수 있는 방법을 제공하며 서블릿 컨테이너에 의해 관리됩니다.
- ✓ 세션은 사용자 식별을 위해 서버에서 유일한 키를 세션 키로 생성하여 쿠키에 담아 웹 브라우저로 전송합니다.
- ✓ 웹 브라우저는 세션 키가 담긴 쿠키를 요청마다 서버로 전달하므로 서버는 세션 키로 사용자 세션정보를 찾을 수 있습니다.



4.2 세션과 쿠키 (3/6) – HttpSession 객체

- ✓ 세션 객체는 HttpSession 타입으로 request 객체의 getSession() 메소드를 호출하면 생성됩니다.
- ✓ getSession() 메소드는 기존 세션이 존재하면 그대로 반환하고, 없는 경우에만 새로운 세션을 생성하여 반환합니다.
- ✓ 세션 객체의 isNew() 메소드를 호출하면 세션 객체가 이번에 신규로 생성되었는지 여부를 확인할 수 있습니다.
- ✓ 생성된 세션 ID는 응답객체에 쿠키로 저장되어 웹 브라우저로 전달됩니다.

getSession() → 세션이 없으면 새로운 세션을 생성하여 반환합니다.

```
HttpSession session = req.getSession();

if (session.isNew()) {
    System.out.println("This is new session.");
} else {
    System.out.println("This is previous session.");
}

System.out.println("Session ID : " + session.getId());
```

[기존 세션이 있으면] getSession(false) → 기존 세션을 반환합니다.

```
HttpSession session = req.getSession(false);
System.out.println("This session may not be null.");
```

[기존 세션이 없으면] getSession(false) → null 을 반환합니다.

```
HttpSession session = req.getSession(false);
System.out.println("This session may be null.");
```

4.2 세션과 쿠키 (4/6) – URL 재작성

- ✓ 서블릿 컨테이너는 쿠키에 담겨있는 세션ID를 이용하여 사용자를 식별하고 세션정보를 관리합니다.
- ✓ 웹 브라우저가 쿠키를 사용하지 않도록 설정되어 있는 경우에는 세션ID를 URL 뒤에 붙이는 방법을 사용합니다.
- ✓ 웹 브라우저는 서버 요청 시 쿠키 대신에 URL 파라미터를 통해 세션ID를 서버에 제공합니다.
- ✓ 이 방법을 ‘URL 재작성’이라 하며 response 객체에는 URL 재작성을 위한 유ти리티 메소드가 포함되어 있습니다.

encodeURL(URL) : URL 문자열 뒤에 세션 ID를 추가하여 반환합니다.

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

HttpSession session = request.getSession();

out.println("<html><body>");

// URL 뒤에 세션 ID를 추가함
out.println("<a href='" + response.encodeURL("/signup.do") + "'>Sign up</a>");
out.println("</body></html>");
```

sendRedirect(URL) : 리다이렉트를 위한 목적으로 URL을 재작성 할 때 사용합니다.

```
String redirectURL = response.encodeRedirectURL("/signup.do");
response.sendRedirect(redirectURL);
```

4.2 세션과 쿠키 (5/6) – 세션 관리

✓ 세션이 종료되는 경우에는 세 가지가 있으며 이 경우 세션 속성들은 삭제됩니다.

- 서블릿에서 세션 객체의 invalidate() 메소드를 호출한 경우 (예, 로그아웃)
- web.xml에 설정된 세션 타임아웃 시간 초과된 경우
- 서블릿 컨테이너가 다운된 경우

세션 종료하기

```
HttpSession session = request.getSession(false);
if (session != null) {
    session.invalidate();
}
```

세션 타임아웃 설정 (web.xml)

```
<session-config>
    <!-- 분 단위 -->
    <session-timeout>15</session-timeout>
</session-config>
```

메소드	설명	언제 사용할까요?
getCreationTime()	세션 생성시간을 Time 객체로 리턴	세션이 얼마나 오래 되었는지 알고 싶을 때
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 때. 세션이 invalidate() 되기까지 시간이 얼마나 남았는지 알기 위하여 사용
invalidate()	세션 종료. 현재 세션에 저장된 모든 세션 속성을 제거하는(unbind)작업이 포함됨	클라이언트가 비활성화이거나, 세션 작업이 완료되어 강제로 세션을 종료할 때. invalidate()는 세션 ID가 더 이상 존재하지 않으니, 관련 속성을 세션 객체에서 제거하라는 의미

출처 : Bryan Basham, Kathy Sierra, Bert Bates, 「Head First Servlets and JSP」 : O'Reilly Media, 2008, p.276

4.2 세션과 쿠키 (6/6) – Cookie 객체

- ✓ 쿠키는 서버와 클라이언트간에 주고 받는 조그마한 데이터(이름/값의 문자열 쌍)입니다.
- ✓ 서버는 클라이언트로 쿠키를 보내고, 이후 클라이언트는 매번 요청에 이 값을 전송합니다.
- ✓ request 객체를 통해 클라이언트에서 전송된 쿠키 데이터를 조회할 수 있습니다.
- ✓ response 객체를 통해 클라이언트에 전송할 쿠키 데이터를 설정할 수 있습니다.

쿠키 객체 생성

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

쿠키가 클라이언트에 얼마나 오랫동안 살아 있을지 설정

```
// 초 단위, -1을 설정하면 브라우저 종료 시 쿠키삭제  
cookie.setMaxAge(30 * 60);
```

response 객체에 클라이언트로 보낼 쿠키 설정

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

request 객체로 부터 클라이언트에서 전송된 쿠키 조회

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

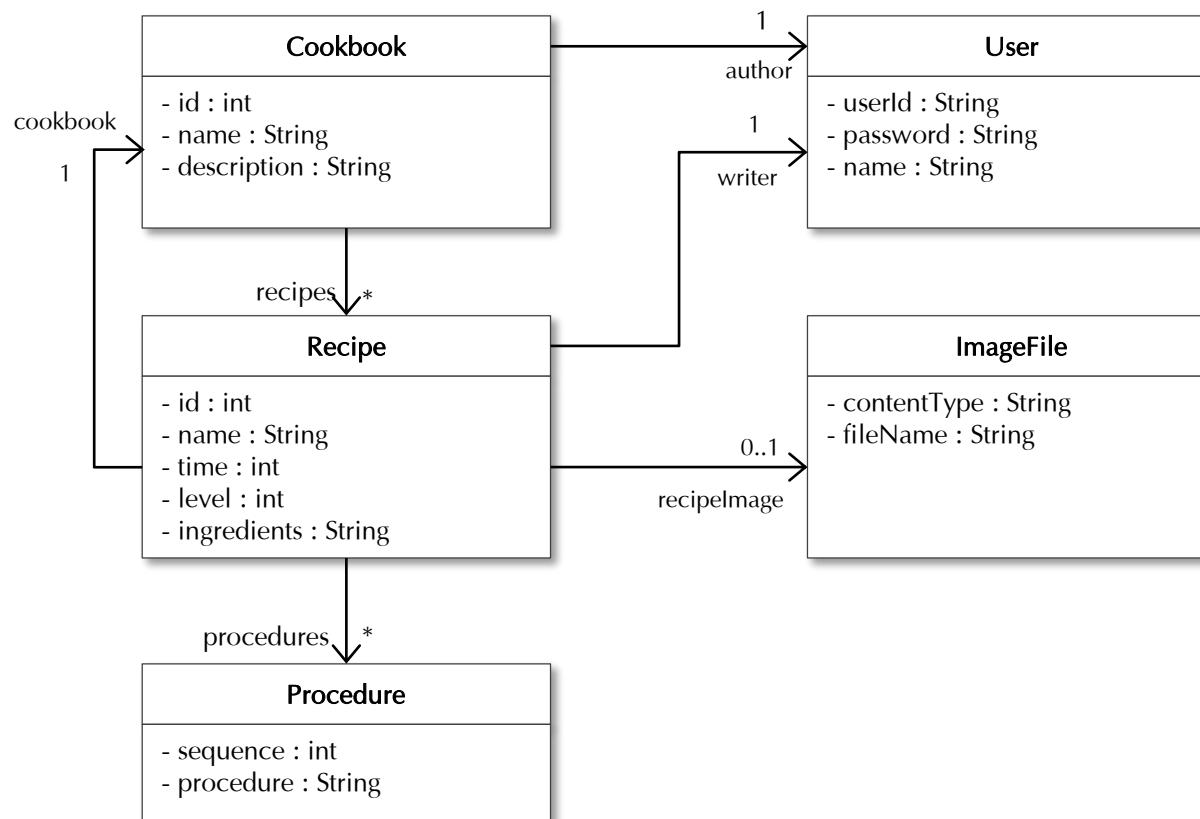
4.3 요리책 만들기 #1 [1/12] – 요구사항

- ✓ 요리 매니아들을 위하여 레시피를 서로 공유할 수 있는 서비스를 만들어 봅니다. 서비스의 이름은 [요리조리]입니다.
- ✓ [요리조리] 요리책 서비스는 요리책 및 레시피를 등록하고 조회할 수 있는 기능을 제공해야 합니다.
- ✓ 사용자 등록을 하면 [요리조리] 회원이 되며, 회원만 요리책 및 레시피를 등록할 수 있습니다.
- ✓ 비 회원이라도 요리책 및 레시피 조회기능은 사용할 수 있습니다.



4.3 요리책 만들기 #1 [2/12] – 도메인 객체 모델

- ✓ 요리조리 요리책 서비스의 요구사항을 분석하여 도메인 객체를 식별합니다.
- ✓ Cookbook은 요리책의 기본 정보와 여러 개의 레시피 목록을 갖고 있습니다.
- ✓ Recipe는 레시피를 나타내며 레시피에 대한 간단한 정보와 조리절차 그리고 조리사진을 관리합니다.
- ✓ 요리책 서비스의 도메인 객체들을 클래스 다이어그램을 통해 표현하면 아래와 같습니다.



4.3 요리책 만들기 #1 [3/12] – 데이터 모델

- ✓ 요리조리 요리책 서비스는 데이터를 저장하기 위하여 관계형 데이터베이스를 사용합니다.
- ✓ 도메인 모델을 통하여 엔티티를 식별하고 테이블을 정의합니다.
- ✓ 요리책 서비스의 테이블을 ERD로 표현하면 다음과 같습니다.

COOKBOOK (요리책)			
번호	컬럼명	속성명	데이터타입
1	BOOK_ID (PK)	요리책ID	INTEGER
2	BOOK_NAME	요리책명	VARCHAR(20)
3	BOOK_DESC	설명	VARCHAR(200)
4	AUTHOR_ID	등록자ID	VARCHAR(30)
5	AUTHOR_NAME	등록자명	VARCHAR(30)



RECIPE (조리법)			
번호	컬럼명	속성명	데이터타입
1	RECIPE_ID (PK)	레시피ID	INTEGER
2	BOOK_ID (FK)	요리책ID	INTEGER
3	RECIPE_NAME	레시피명	VARCHAR(20)
4	RECIPE_TIME	조리시간	INTEGER
5	LEVEL	난이도	INTEGER
6	INGREDIENTS	재료	VARCHAR(300)
7	IMG_CONT_TYPE	사진파일형식	VARCHAR(20)
8	IMG_FILE_NAME	사진파일명	VARCHAR(20)
9	WRITER_ID	등록자ID	VARCHAR(30)
10	WRITER_NAME	등록자명	VARCHAR(30)

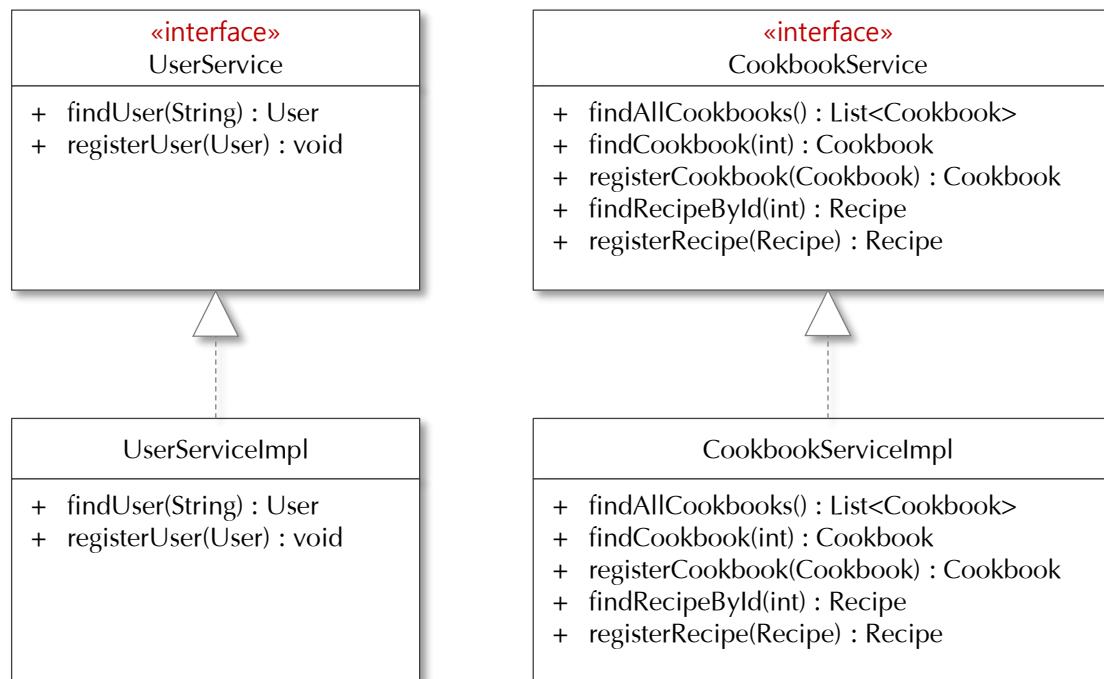


USER (사용자)			
번호	컬럼명	속성명	데이터타입
1	USER_ID (PK)	사용자ID	VARCHAR(30)
2	PASSWORD	비밀번호	VARCHAR(30)
3	USER_NAME	사용자명	VARCHAR(30)

RECIPE_PROCEDURE (조리절차)			
번호	컬럼명	속성명	데이터타입
1	RECIPE_ID (PK)	레시피ID	INTEGER
2	SEQ_NUM (PK)	조리절차순번	INTEGER
3	PROCEDURE	조리절차설명	VARCHAR(200)

4.3 요리책 만들기 #1 [4/12] – 서비스 인터페이스

- ✓ 요구사항을 분석하여 사용자 서비스와 요리책 서비스 인터페이스를 식별하였습니다.
- ✓ 사용자 관리 서비스는 사용자를 등록하고, 사용자를 조회하는 오퍼레이션을 제공합니다.
- ✓ 요리책 관리 서비스는 요리책 및 레시피를 등록하고 조회하는 오퍼레이션을 제공합니다.
- ✓ 서비스 인터페이스 및 구현 클래스를 클래스 다이어그램으로 나타내면 아래와 같습니다.



4.3 요리책 만들기 #1 [5/12] – 화면 Navigation

- ✓ [요리조리] 요리책 서비스에서 제공하는 화면은 다음과 같습니다.
- ✓ 요리책 등록 및 레시피 등록화면은 로그인이 필요한 화면입니다. 로그인 되지 않은 사용자가 화면을 요청하면 로그인 화면으로 리다이렉트 합니다.

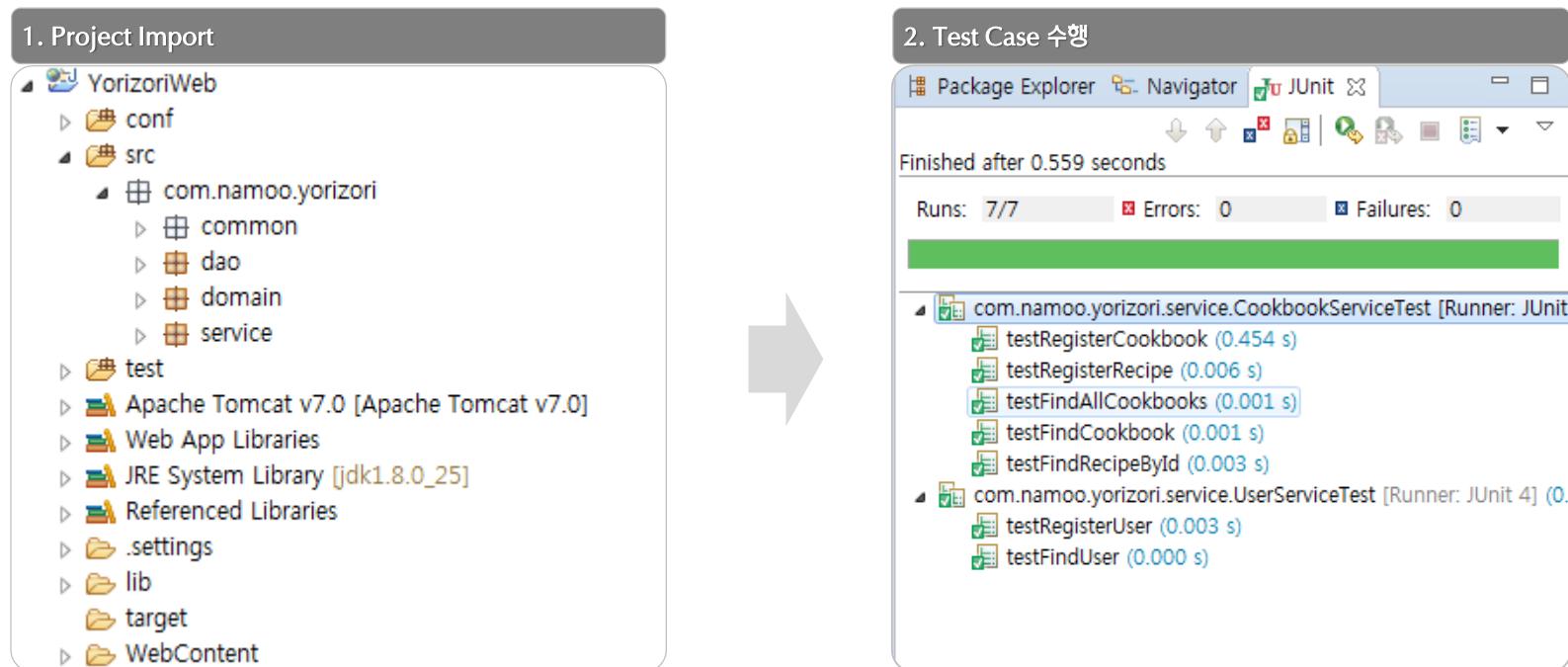


4.3 요리책 만들기 #1 [6/12] – 실습준비

소스
코드

YorizoriWeb_Empty.zip

- ✓ 이번 과정은 JSP&Servlet을 사용하여 프리젠테이션 영역을 구현하는 것이 실습의 목표입니다.
- ✓ 비즈니스 로직 및 데이터 접근 영역은 이미 구현된 프로젝트로 제공합니다.
- ✓ 예제소스 프로젝트를 이클립스에 Import하고 테스트 케이스를 수행하여 에러가 없는지 확인합니다.
- ✓ 실습 화면에 사용하는 html은 html 폴더에 포함되어 있습니다.



4.3 요리책 만들기 #1 [7/12] – [실습] 사용자 등록

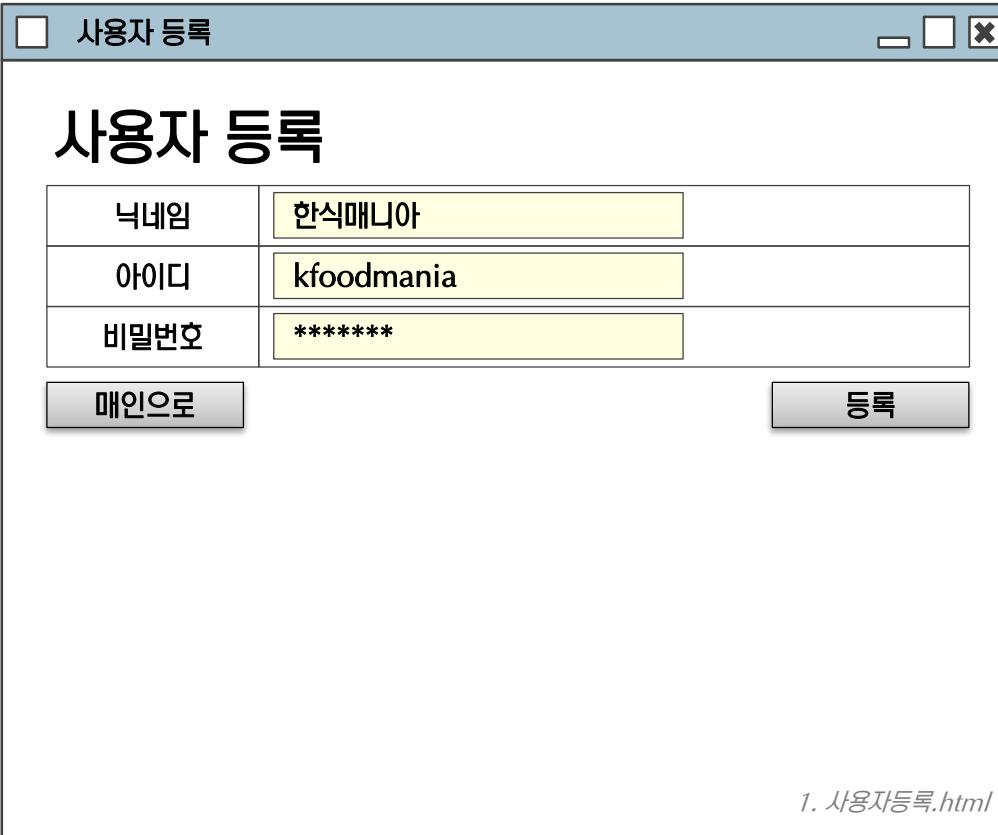
- ✓ 화면 URL : <http://localhost:8080/yorizori/user/signup.do>
- ✓ 화면 URL로 접속하면 사용자 등록 화면을 보여줍니다.
- ✓ 사용자 정보를 입력하고 [등록] 버튼을 클릭하면 사용자 등록 후 로그인 화면(/user/login.do)으로 리다이렉트 합니다.
- ✓ [메인으로] 버튼을 클릭하면 Context Root 로 이동합니다.

사용자 등록

닉네임	한식매니아
아이디	kfoodmania
비밀번호	*****

메인으로 **등록**

1. 사용자등록.html/



URL 맵핑	HTTP 메소드	설명
/user/signup.do	GET	사용자등록 화면 조회
/user/register.do	POST	사용자 등록 요청
서블릿 클래스명	com.namoo.yorizori.web.controller.UserFormController	
JSP 파일	/WEB-INF/views/userForm.jsp	

[실습] JSP 파일로 서블릿 매핑

```
<!-- JSP 페이지로 서블릿 등록하기 -->
<servlet>
  <description>사용자 등록화면</description>
  <servlet-name>userRegisterForm</servlet-name>
  <jsp-file>/WEB-INF/views/userForm.jsp</jsp-file>
</servlet>
<servlet-mapping>
  <servlet-name>userRegisterForm</servlet-name>
  <url-pattern>/user/signup.do</url-pattern>
</servlet-mapping>
```



서블릿에서 특별한 처리를 하지 않고 JSP 파일만 하는 경우 서블릿 클래스를 만들지 않고 JSP 파일을 바로 서블릿 매핑할 수도 있습니다. (결국 JSP도 서블릿이니까요...)

4.3 요리책 만들기 #1 [8/12] – [실습] 로그인

- ✓ 화면 URL : <http://localhost:8080/yorizori/user/login.do>
- ✓ 로그인 정보를 입력하고 로그인 버튼을 클릭하면 등록된 사용자 여부를 확인하여 결과를 보여줍니다.
- ✓ 로그인 정보가 일치하면 Context Root로 이동하고 일치하지 않으면 예러 페이지를 출력합니다.
- ✓ [아이디 저장하기]를 체크하고 [로그인]을 클릭, 다시 로그인 화면을 조회하면 저장된 아이디를 보여줍니다.

로그인

아이디	kfoodmania	<input type="checkbox"/> 아이디 저장하기
비밀번호	*****	
사용자 등록		로그인

2. login.html

URL 맵핑	HTTP 메소드	설명
/user/login.do	GET	로그인 화면 조회
/user/login.do	POST	로그인 요청
서블릿 클래스명	com.namoo.yorizori.web.controller.LoginController	
JSP 파일	/WEB-INF/views/loginForm.jsp	

[실습] 로그인 화면 조회 시

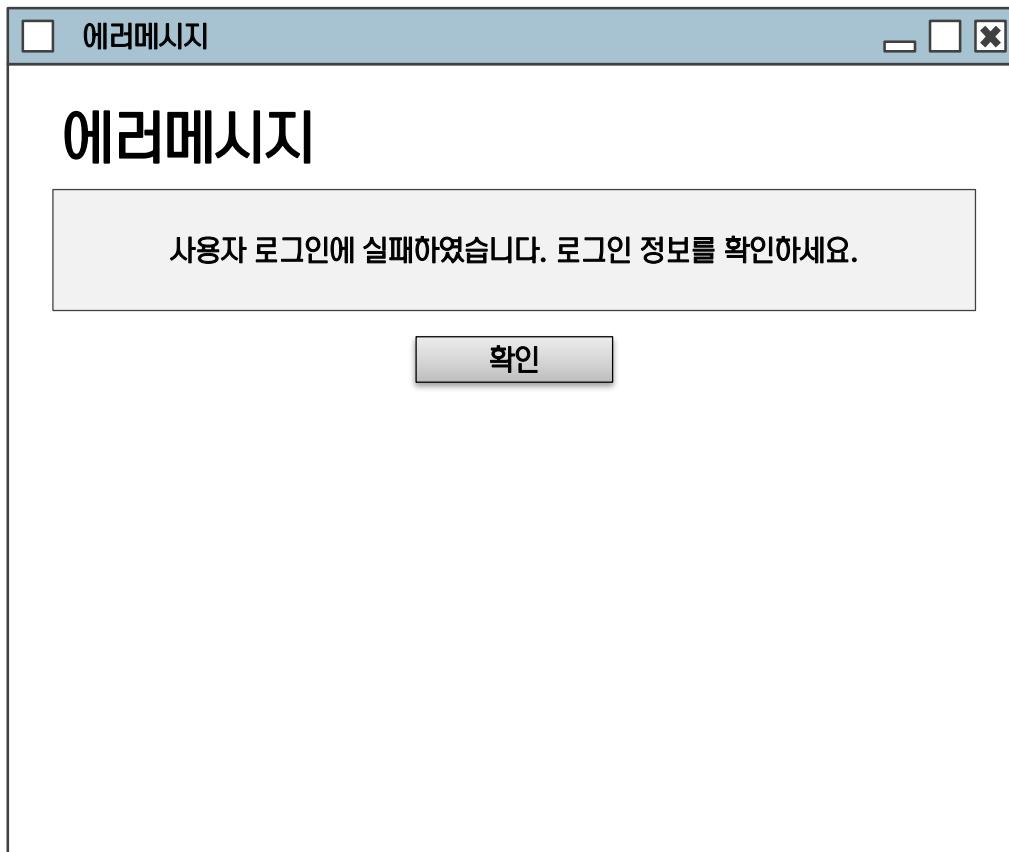
1. 서블릿에서 쿠키에 사용자 ID 가 들어 있는 경우 request의 속성으로 추가합니다.
2. JSP에서는 request 속성에 사용자 ID 값이 존재하면 그 값을 입력필드의 기본값으로 설정하고 아이디 저장하기 체크박스를 체크합니다.

[실습] 로그인 요청 시

1. 로그인 요청을 받은 서블릿은 UserService의 findUser() 메소드를 호출하여 사용자 정보를 조회합니다.
2. 조회된 사용자 정보가 입력한 정보와 일치하면 세션의 속성으로 추가합니다. 로그인 정보가 불일치 한 경우 YzRuntimeException 예외를 발생시킵니다.
3. [아이디 저장하기] 체크박스가 눌린 경우, 쿠키 객체를 생성하여 사용자 ID 값을 세팅한 후 response 객체에 쿠키를 추가합니다.

4.3 요리책 만들기 #1 [9/12] – [실습] 공통 에러페이지

- ✓ 서블릿 컨테이너는 서블릿에서 발생하는 에러를 공통으로 처리할 수 있는 방법을 제공합니다.
- ✓ 에러 처리는 공통 에러 페이지 작성과 어떤 에러에 대해 에러페이지로 처리할지를 정의하는 부분으로 나뉘어집니다.
- ✓ 에러의 유형에는 특정 예외가 발생한 경우 또는 특정 응답코드가 발생한 경우가 있습니다.
- ✓ web.xml에 특정 예외(YzRuntimeException)가 발생한 경우 에러페이지를 보여주도록 처리해 봅시다.



JSP 파일	/WEB-INF/views/common/error.jsp (step0에 포함되어 있음)
web.xml	/WEB-INF/web.xml

[설명] 에러페이지 (error.jsp)

에러 페이지는 JSP의 page 지시자의 isErrorPage 속성을 true로 설정합니다. 이 속성이 true로 설정된 에러페이지에서는 exception 내장 객체를 통해 예외 객체에 접근할 수 있습니다.

[확인] 버튼 처리

YzRuntimeException은 요리책 서비스에서 정의한 사용자 정의 예외 클래스입니다. 이 예외 클래스에는 redirectURL 속성이 있는데, 이 값이 설정되어 있으면 [확인] 버튼을 클릭할 때 해당 URL로 이동하도록 처리합니다. redirectURL 값이 설정되어 있지 않으면 이전화면으로 이동하도록 자바스크립트의 history.back() 함수를 호출합니다.

[실습] web.xml

```
<error-page>
    <exception-type>...exception.YzRuntimeException</exception-type>
    <location>/WEB-INF/views/common/error.jsp</location>
</error-page>
```

4.3 요리책 만들기 #1 [10/12] – [실습] 요리책 등록

- ✓ 화면 URL : <http://localhost:8080/yorizori/cookbook/register.do>
- ✓ 요리책 등록 화면은 로그인한 사용자만 사용할 수 있도록 처리합니다.
- ✓ 로그인 되지 않은 사용자가 화면을 요청하면 예외를 발생시킵니다.
- ✓ 등록 양식의 등록자 부분에는 로그인 사용자의 이름과 아이디를 출력합니다.

요리책 등록

요리책명	한식 요리책
등록자	한식매니아 (kfoodmania)
설명	한식 요리를 맛있게 만드는 방법을 설명합니다.

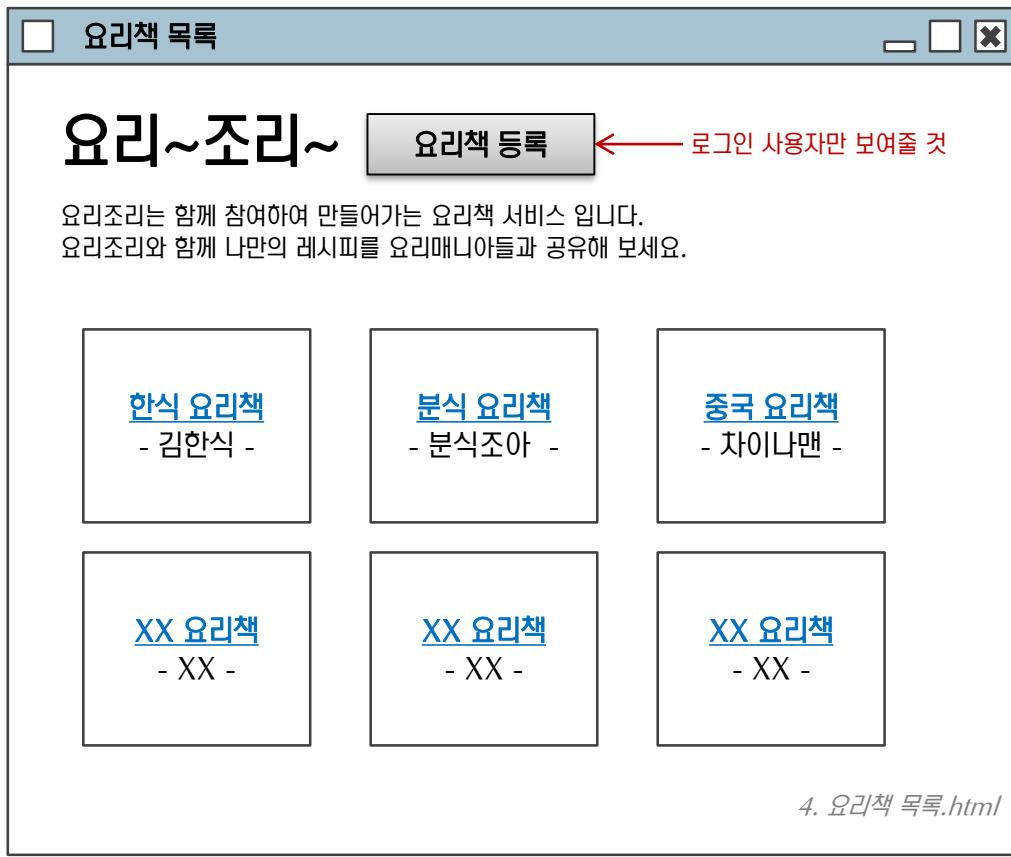
목록 **등록**

3. 요리책 등록.html/

URL 맵핑	HTTP 메소드	설명		
/cookbook/register.do	GET	요리책 등록 화면 조회		
/cookbook/register.do	POST	요리책 등록 요청		
서블릿 클래스명	com.namoo.yorizori.web.controller.CookbookFormController			
JSP 파일	/WEB-INF/views/cookbookForm.jsp			
[실습] 로그인 등록 화면 조회				
1. 요리책 등록화면은 로그인이 필요한 화면입니다. 로그인되지 않은 사용자가 화면을 요청하면 YzRuntimeException 객체를 생성하여 message에는 “로그인이 필요합니다.”로 설정하고, redirectURL에는 로그인 화면 URL(/user/login.do)을 설정하여 예외를 발생시킵니다. → 해당 내용은 공통 예러페이지에서 보여지게 됩니다.				
2. 로그인된 사용자인 경우 cookbookForm.jsp 페이지로 Request Dispatch 처리합니다.				
[실습] 요리책 등록 요청				
1. 화면에서 입력받은 정보로 Cookbook 객체를 생성합니다. Cookbook 객체의 author 속성에는 로그인 사용자 객체를 세션에서 꺼내서 세팅합니다.				
2. CookbookService의 registerCookbook() 메소드를 호출하여 요리책을 등록합니다.				
3. 등록이 완료되면 ContextRoot로 리다이렉트 처리합니다.				

4.3 요리책 만들기 #1 [11/12] – [실습] 요리책 목록 조회

- ✓ 화면 URL : <http://localhost:8080/yorizori/cookbook/list.do>
- ✓ 요리책 목록화면에서는 등록된 요리책의 목록을 조회하여 보여줍니다.
- ✓ [요리책 등록] 버튼은 사용자가 로그인 된 경우만 보여줍니다.
- ✓ 각 요리책의 이름을 클릭하면 해당 요리책의 레시피 목록화면으로 이동합니다. (URL : /recipe/list.do?cbId=요리책ID)



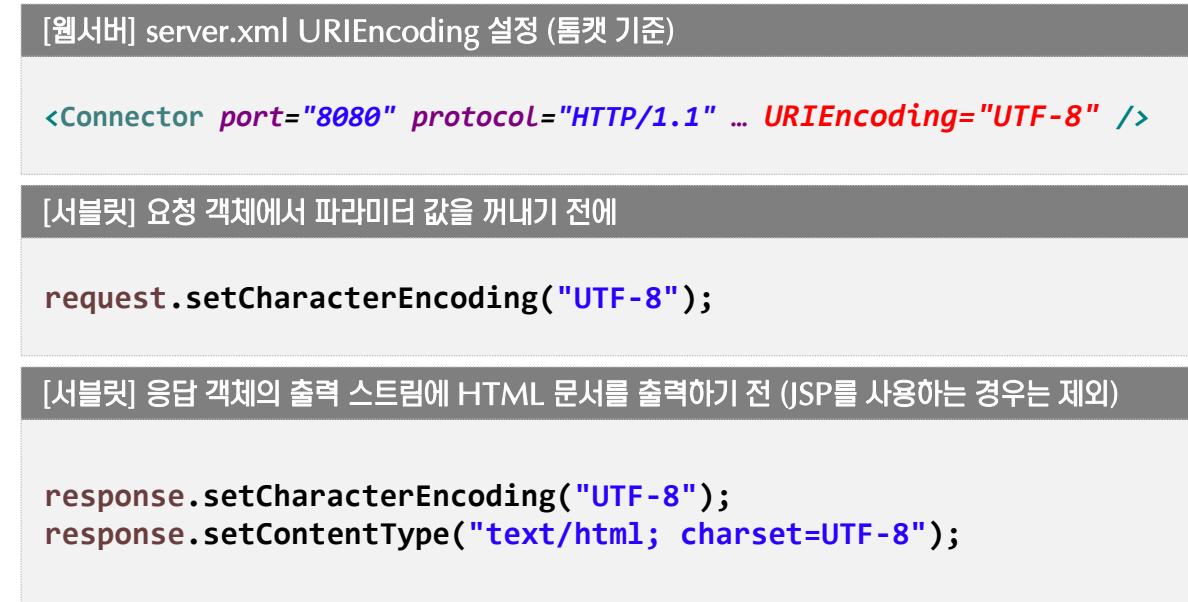
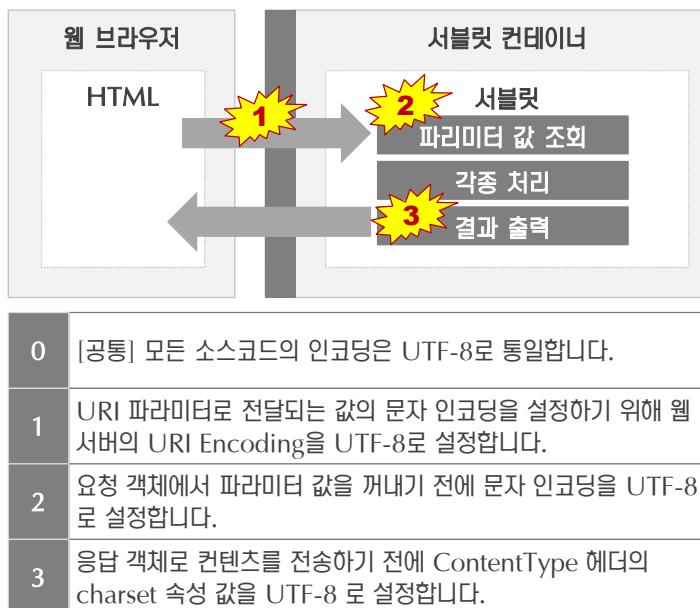
URL 맵핑	HTTP 메소드	설명
/cookbook/list.do	GET	요리책 목록 화면 조회
서블릿 클래스명	com.namoo.yorizori.web.controller.CookbookListController	
JSP 파일	/WEB-INF/views/cookbookList.jsp	

[실습] 요리책 목록 화면 조회

- 요리책 목록 화면 조회를 요청하면 CookbookService의 findAllCookbooks() 메소드를 호출하여 요리책 리스트를 request 속성에 추가합니다.
- cookbookList.jsp에서 스크립트릿을 사용하여 요청 객체에 담긴 요리책 목록 리스트를 반복하면서 html을 구성합니다. (단, JSTL은 사용하지 않습니다.)
- 요리책 등록 버튼은 세션 객체의 속성에 로그인 정보가 존재하는 경우에만 보여지도록 처리합니다.

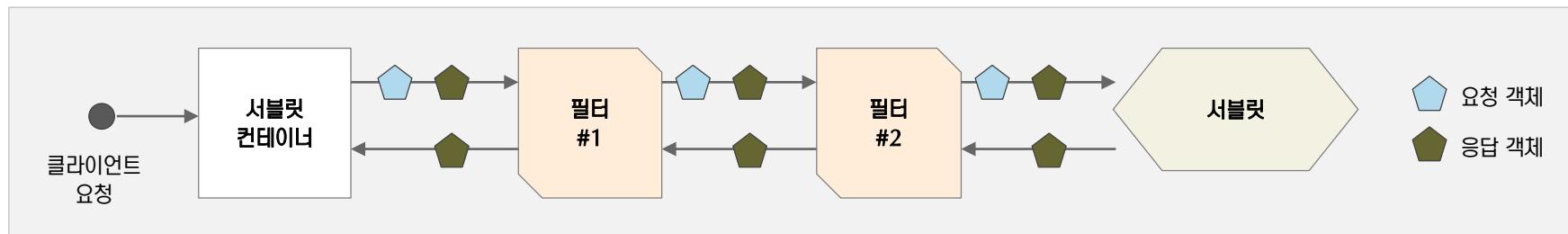
4.3 요리책 만들기 #1 [12/12] – 한글 인코딩 문제

- ✓ 웹 프로그래밍을 하다 보면 한글 깨짐 현상이 자주 발생합니다. 몇 가지 문제상황에 대한 해결방법을 알아 봅니다.
- ✓ URI 파라미터로 전달되는 값이 있을 때 문자 웹 서버에서 문자 인코딩 설정이 필요합니다.
- ✓ 서블릿에서 요청 객체의 파라미터 값을 꺼내기 전에 문자 인코딩 설정이 필요합니다.
- ✓ 서블릿에서 응답 객체의 출력스트림에 컨텐츠를 전송하기 전 문자 인코딩 설정이 필요합니다.



4.4 서블릿필터 (1/3) – Servlet Filter 개요

- ✓ 서블릿 필터는 ‘서블릿으로 요청을 넘기기 전’과 ‘응답을 웹 서버로 전송하기 전’에 가로채어 어떤 처리를 할 수 있습니다.
- ✓ 서블릿 컨테이너는 DD(web.xml)에 선언한 정보를 바탕으로 필터를 언제 실행할지 결정합니다.
- ✓ 서블릿 필터는 javax.servlet.Filter 인터페이스를 구현하여 작성합니다.
- ✓ 예를 들어 요청객체 및 응답객체에 대한 문자 인코딩 설정을 위하여 필터를 활용할 수 있습니다.



Filter 인터페이스의 메소드	설명
init(FilterConfig)	init() 메소드는 필터가 서블릿 컨테이너에 등록될 때 최초 한번 초기화를 위하여 호출됩니다. 필터 설정 초기화 파라미터는 FilterConfig 객체를 통해 조회할 수 있습니다.
doFilter(ServletRequest, ServletResponse, FilterChain)	doFilter() 메소드는 필터에서 수행하는 로직을 구현합니다. 후속 필터(또는 서블릿)으로 처리를 위임하기 위해 FilterChain 객체의 doFilter() 메소드를 반드시 호출해 주어야 합니다.
destroy()	destroy() 메소드는 필터가 서블릿 컨테이너에서 소멸할 때 한번 호출됩니다.

4.4 서블릿필터 [2/3] – [실습] Filter를 이용한 문자 인코딩 처리 [1/2]

- ✓ 서블릿 별로 반복적으로 처리해야 하는 문자 인코딩 설정을 서블릿 필터를 구현하여 처리해봅니다.
- ✓ 문자 인코딩 필터는 요청객체 및 응답객체에 문자 인코딩 타입을 설정합니다.
- ✓ 문자 인코딩 필터는 브라우저가 컨텐츠에 대한 MIME 타입을 알 수 있도록 ContentType 헤더를 설정합니다.
- ✓ 문자 인코딩 필터는 CharacterEncodingFilter 로 명명합니다.



1. init() 메소드는 필터 초기화 파라미터에서 encoding 값을 조회하여 필터의 인스턴스 변수 encoding에 할당합니다. init()은 서블릿 컨테이너에서 최초 한번만 호출됩니다.
2. doFilter() 메소드는 요청 객체와 응답 객체의 문자인코딩을 encoding 값으로 설정합니다. 그리고 브라우저를 위한 MIME 타입을 설정합니다.

문자 인코딩 필터 구현 (com.namoo.yorizori.web.filter.CharacterEncodingFilter)

```
public class CharacterEncodingFilter implements Filter {  
    private String encoding;  
  
    @Override  
    public void init(FilterConfig config) throws ServletException {  
        this.encoding = config.getInitParameter("encoding");  
    }  
  
    @Override  
    public void doFilter(ServletRequest req, ServletResponse resp,  
        FilterChain chain) throws IOException, ServletException {  
        req.setCharacterEncoding(encoding);  
        resp.setCharacterEncoding(encoding);  
        resp.setContentType("text/html;charset="+encoding);  
  
        chain.doFilter(req, resp);  
    }  
  
    @Override  
    public void destroy() {  
    }  
}
```

4.4 서블릿필터 [3/3] – [실습] Filter를 이용한 문자 인코딩 처리 [2/2]

- ✓ 앞서 구현한 CharacterEncodingFilter 를 DD 파일(web.xml)에 등록합니다.
- ✓ 문자 인코딩이 변경될 때 소스코드를 변경하지 않도록 인코딩은 필터 초기화 파라미터에 설정합니다.
- ✓ 필터는 요청 URL이 *.do 끝나는 모든 요청에 대하여 수행하도록 필터를 매핑합니다.
- ✓ 참고로, 서블릿 3.0 부터는 @WebFilter 어노테이션을 통한 필터 등록 및 매핑을 지원합니다.



- 1 문자 인코딩 필터 구현
- 2 필터 등록 및 필터 매핑

- i
1. <filter> 요소를 사용하여 필터를 등록합니다. <init-param> 요소에 필터 초기화 파라미터를 정의합니다. 파라미터의 이름은 encoding, 값을 UTF-8로 설정합니다.
 2. <filter-mapping> 요소를 통해 문자 인코딩 필터를 매핑합니다. url-pattern은 *.do 로 설정합니다.

필터 등록 및 필터 매핑 (web.xml)

```
<filter>
    <description>문자인코딩 필터</description>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>
        com.namoo.yorizori.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

4.5 JSTL [1/8] – JSTL 개요 [1/2]

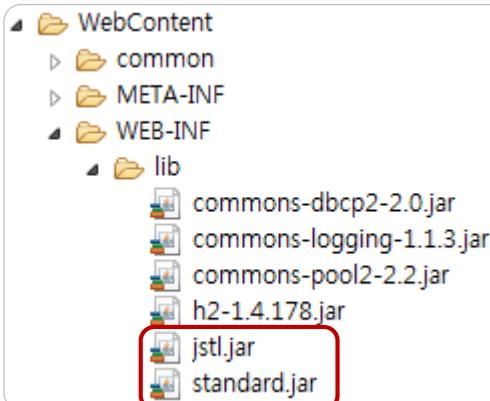
- ✓ JSP를 사용하여 프리젠테이션 영역을 개발하는 많은 개발자는 Java와 같은 프로그래밍 언어에 익숙하지 않습니다.
- ✓ 그러나, JSP 페이지에서 동적인 데이터를 처리하기 위해서는 스크립트릿에 프로그래밍 언어를 사용해야 합니다.
- ✓ JSTL은 JSP 페이지에서 스크립트릿을 사용하지 않고 액션을 통해 간단하게 처리할 수 있는 방법을 제공합니다.
- ✓ JSTL에는 다양한 액션이 있으며, EL과 함께 사용하여 코드를 간결하게 작성할 수 있습니다.



4.5 JSTL [2/8] – JSTL 개요 [2/2]

- ✓ JSTL은 JSP 표준 태그 라이브러리입니다.
- ✓ JSTL은 스크립트릿 대신에 반복문, 조건문, 포맷팅 및 XML 처리 등을 HTML 태그처럼 사용할 수 있도록 지원합니다.
- ✓ JSTL을 사용하려면 jstl.jar와 standard.jar 를 WEB-INF/lib 아래에 설치해야 합니다.
- ✓ JSP 페이지 상단에 taglib 지시자로 사용하려는 태그 라이브러리를 선언합니다.

1. 웹 프로젝트에 JSTL 설치



Jstl.jar, standard.jar 파일은 어디에서 찾을 수 있나요?

인터넷에서 다운로드 받아서 설치하거나 툴캣을 사용 중이라면 툴캣의 예제 프로젝트에 포함된 것을 사용하면 됩니다.

툴캣 설치경로의 webapps/examples/WEB-INF/lib 폴더를 확인하세요.

2. JSP 페이지 상단에 taglib 지시자로 사용하려는 태그라이브러리 선언

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

3. JSP 페이지에서 JSTL 사용하기

```
<c:choose>
  <c:when test="\${loginUser == null}">
    <a href="\${ctx}/user/Login.do">로그인</a> /
    <a href="\${ctx}/user/signup.do">회원가입</a>
  </c:when>

  <c:otherwise>
    <b>\${loginUser.name}</b> 님!! 환영합니다.
    [<a href="\${ctx}/user/logout.do">로그아웃</a>]
  </c:otherwise>

</c:choose>
```

4.5 JSTL (3/8) – <c:out> 액션

- ✓ <c:out> 액션은 value 속성에 지정된 표현식을 평가하여 그 결과를 JspWriter를 통해 출력합니다.
- ✓ value 속성에는 출력하려는 표현식을 적습니다. 생략할 경우 빈 문자열을 출력합니다.
- ✓ escapeXml 속성은 표현식의 결과에 <, >, &, ', " 문자가 포함된 경우 escape 할지 여부를 설정합니다.(디폴트 true)
- ✓ default 속성 또는 액션태그 Body에 문자열을 설정하면, value 의 결과가 null인 경우 기본 값으로 사용됩니다.

1. JSP 상단에 core 태그라이브러리 사용을 위한 taglib 지시자 선언

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

In Servlet

```
String html = "<span style=\"color:red\">escapeXml attribute</span>";  
request.setAttribute("html", html);  
request.setAttribute("someValue", null);
```

3-1. <c:out>

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

웹브라우저 출력결과

```
<span  
style="color:red">escapeXml  
attribute</span>
```

3-2. escapeXml 속성을 true로 설정한 경우

```
<c:out value="${html}" escapeXml="false" />
```

웹브라우저 출력결과

```
escapeXml attribute
```

2-1. default 속성으로 디폴트 값 설정하기

```
<c:out value="${someValue}" default="Empty"/>
```

2-2. 액션태그 Body에 디폴트 값 설정하기

```
<c:out value="${someValue}">Empty Body</c:out>
```

웹브라우저 출력결과

```
Empty Body
```

4.5 JSTL [4/8] – <c:set> 액션

- ✓ <c:set> 액션은 생존범위 변수나 특정 객체의 프로퍼티에 값을 할당할 때 사용합니다.
- ✓ value 속성의 값이나 액션의 Body 컨텐츠로 값을 설정합니다.
- ✓ var 속성은 생존범위 변수를 나타내며, 변수의 생존범위는 scope 속성으로 설정합니다. (디폴트는 page 생존범위)
- ✓ 특정 객체의 프로퍼티에 값을 할당할 때는 target 속성에 객체를 설정하고 property에 프로퍼티명을 설정합니다.

[문법1] value 속성을 이용하여 생존범위 변수 값 할당

```
<c:set value="value" var="varName" [scope="{page/request/session/application}"]/>
```

[문법2] 액션의 Body 컨텐츠를 사용하여 생존범위 변수 값 할당

```
<c:set var="varName" [scope="{page/request/session/application}"]>  
body content  
</c:set>
```

[문법3] value 속성을 이용하여 대상 객체의 프로퍼티 값 할당

```
<c:set value="value" target="target" property="propertyName"/>
```

[문법4] 액션의 Body 컨텐츠를 사용하여 대상 객체의 프로퍼티 값 할당

```
<c:set target="target" property="propertyName">  
body content  
</c:set>
```

4.5 JSTL [5/8] – <c:catch> 액션

- ✓ 기본적으로 JSP 페이지는 예외가 발생하면 지정된 오류페이지를 통해 처리합니다.
- ✓ <c:catch> 액션은 JSP 페이지에서 예외가 발생할 만한 코드를 오류페이지로 넘기지 않고 직접 처리할 때 사용합니다.
- ✓ var 속성에는 발생한 예외를 담을 page 생존범위 변수를 지정합니다.
- ✓ <c:catch>와 <c:if> 액션을 함께 사용하여 Java 코드의 try~catch 와 같은 기능을 구현할 수 있습니다.

try ~ catch 구문

```
try {
    String str = null;
    out.println("Length of string : " +
                str.length());
} catch(Throwable ex) {
    out.print(ex.getMessage());
}
```

<c:catch>와 <c:if> 사용하기

```
<%@ page contentType="text/html" pageEncoding="UTF-8"
       errorPage="error.jsp" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<c:catch var="ex">
<%
    String str = null;
    out.println("Length of string : " + str.length()); // 예외 발생!!!
%>
</c:catch>

<c:if test="${ex != null}">
    예외가 발생하였습니다. ${ex.message}
</c:if>
```

4.5 JSTL [6/8] – 조건문 액션

- ✓ <c:if> 액션은 test 속성에 지정된 표현식을 평가하여 결과가 true인 경우 액션의 Body 컨텐츠를 수행합니다.
- ✓ <c:if> 액션의 var 속성은 표현식의 평가 결과인 Boolean 값을 담을 생존범위 변수를 나타내며, 변수의 생존범위는 scope 속성으로 설정합니다. (디폴트는 page 생존범위)
- ✓ <c:choose><c:when><c:otherwise> 액션을 사용하면 if, else if, else 와 같이 처리할 수 있습니다.

<c:if> 액션 사용 예

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

<c:if> 액션의 var 속성

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

...
<c:if test="${accessible && userDevice == 'iPhone'}">
    User type is member.
</c:if>
```

<c:choose>, <c:when>, <c:otherwise> 액션 사용 예

```
<c:choose>
    <c:when test="${userDevice == 'iPhone'}">
        실행 할 내용...
    </c:when>

    <c:when test="${userDevice == 'Android'}">
        실행 할 내용...
    </c:when>

    <c:otherwise>
        해당 조건이 맞지 않을 때의 디폴트 실행문
    </c:otherwise>
</c:choose>
```

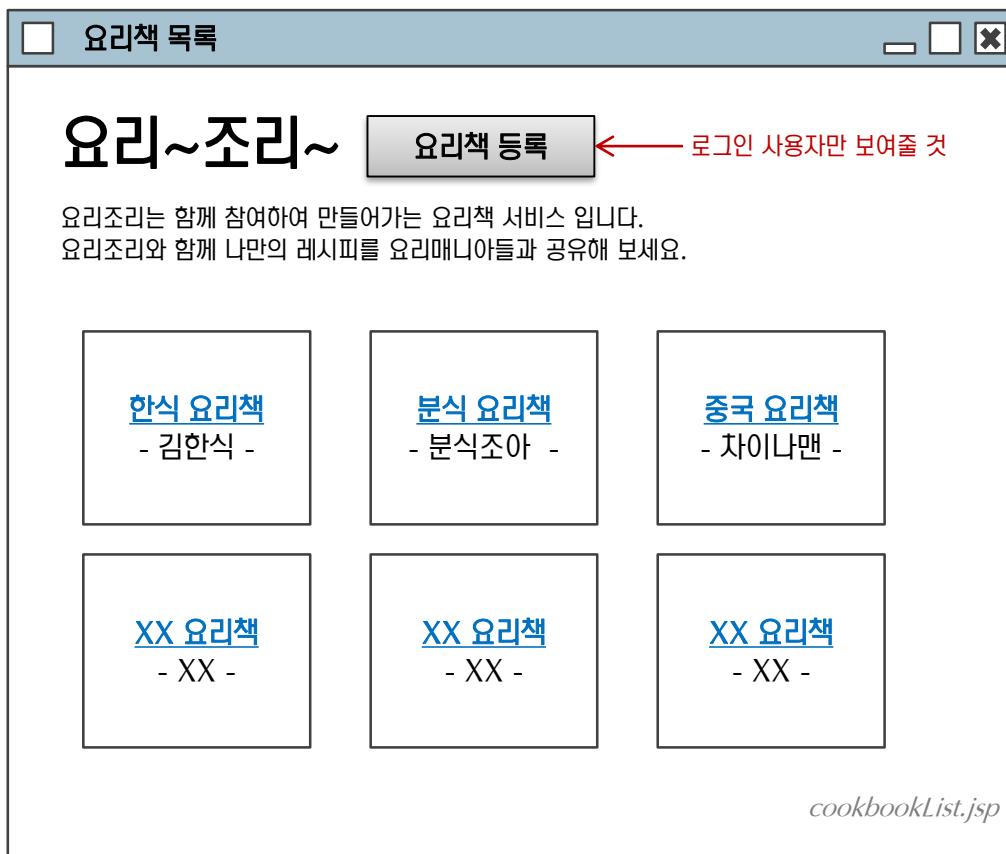
4.5 JSTL (7/8) – 반복문 액션

- ✓ <c:forEach> 액션은 컬렉션에 있는 항목들에 대하여 액션의 Body 컨텐츠를 반복하여 수행합니다.
- ✓ 컬렉션에는 Array, Collection, Map 또는 콤마로 분리된 문자열이 올 수 있습니다.
- ✓ var 속성에는 반복에 대한 현재 항목을 담는 변수를 지정하고 items 속성은 반복할 항목들을 갖는 컬렉션을 지정합니다.
- ✓ varStatus 속성에 지정한 변수를 통해 현재 반복의 상태를 알 수 있습니다.

courses	<c:forEach> 액션의 다양한 활용	실행결과																
<table border="1"><tr><td>0</td><td>Java Fundamental</td></tr><tr><td>1</td><td>Java IO</td></tr><tr><td>2</td><td>Java Socket</td></tr><tr><td>3</td><td>JSP&Servlet</td></tr><tr><td>4</td><td>Spring MVC</td></tr><tr><td>5</td><td>JavaScript</td></tr><tr><td>6</td><td>jQuery Basic</td></tr><tr><td>7</td><td>MyBatis</td></tr></table>	0	Java Fundamental	1	Java IO	2	Java Socket	3	JSP&Servlet	4	Spring MVC	5	JavaScript	6	jQuery Basic	7	MyBatis	<p>1) courses 리스트를 반복하면서 과정명 출력하기</p> <pre><c:forEach var="course" items="\${courses}"> \${course.name}
 </c:forEach></pre> <p>2) courses 리스트를 반복하면서 순번과 과정명 출력하기</p> <pre><c:forEach var="course" items="\${courses}" varStatus="varStatus"> \${varStatus.count}. \${course.name}
 </c:forEach></pre> <p>3) 짹수번째 과정명만 출력하기</p> <pre><c:forEach var="course" items="\${courses}" begin="0" end="5" step="2"> \${course.name}
 </c:forEach></pre> <p>4) 숫자 1부터 5까지 출력하기</p> <pre><c:forEach var="value" begin="1" end="5" step="1"> \${value}
 </c:forEach></pre>	1) courses 리스트를 반복하면서 과정명 출력하기 Java Fundamental Java IO Java Socket JSP&Servlet Spring MVC JavaScript jQuery Basic MyBatis 2) courses 리스트를 반복하면서 순번과 과정명 출력하기 1. Java Fundamental 2. Java IO 3. Java Socket 4. JSP&Servlet 5. Spring MVC 6. JavaScript 7. jQuery Basic 8. MyBatis 3) 짹수번째 과정명만 출력하기 Java Fundamental Java Socket Spring MVC 4) 숫자 1부터 5까지 출력하기 1 2 3 4 5
0	Java Fundamental																	
1	Java IO																	
2	Java Socket																	
3	JSP&Servlet																	
4	Spring MVC																	
5	JavaScript																	
6	jQuery Basic																	
7	MyBatis																	

4.5 JSTL [8/8] – [실습] 요리책 목록 조회 개선

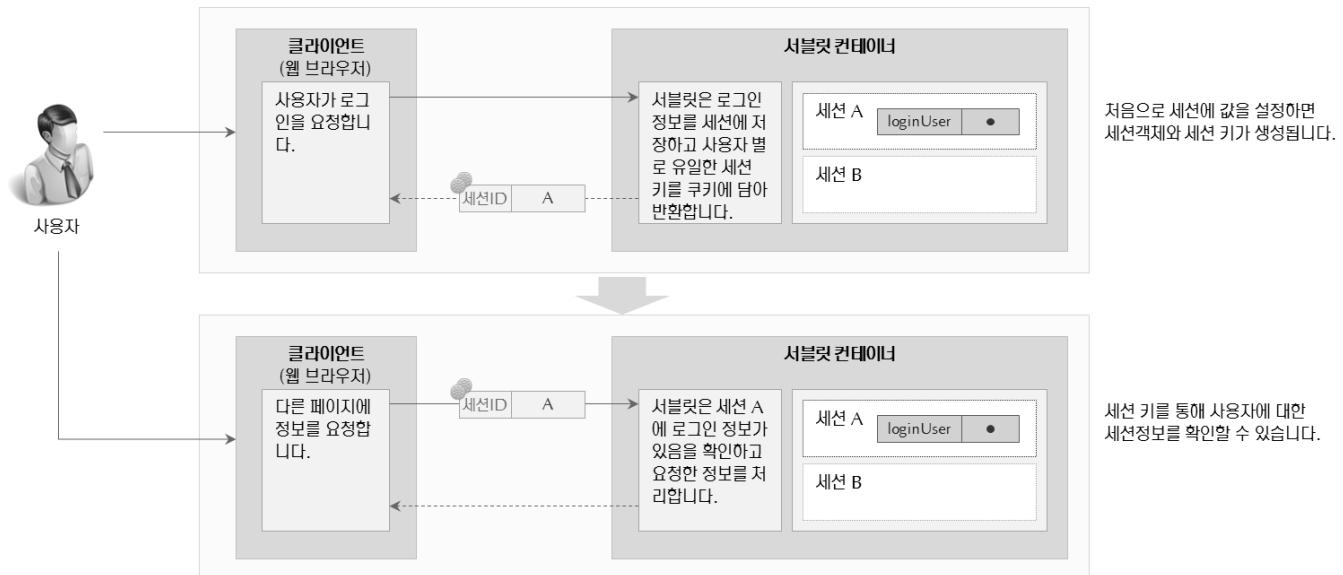
- ✓ 화면 URL : <http://localhost:8080/yorizori/cookbook/list.do>
- ✓ 요리책 목록 JSP 페이지에서 스크립트릿을 제거하고 JSTL을 사용하도록 개선합니다.
- ✓ 로그인 사용자만 보여주는 [요리책 등록] 버튼을 `<c:if>` 액션으로 변경합니다.
- ✓ 요리책 목록을 출력할 때 사용하는 반복문을 `<c:forEach>` 액션으로 변경합니다.



URL 맵핑	HTTP 메소드	설명
/cookbook/list.do	GET	요리책 목록 화면 조회
서블릿 클래스명	com.namoo.yorizori.web.controller.CookbookListController	
JSP 파일	/WEB-INF/views/ cookbookList.jsp	

4.6 요약

- ✓ 비연결지향인 HTTP프로토콜을 사용하면서 서버가 클라이언트를 식별하고 상태를 저장하기 위해서 세션을 사용합니다.
- ✓ 서블릿 컨테이너는 세션을 관리합니다. 세션은 시간이 경과, 컨테이너의 작동중지 또는 메소드 호출에 의해서 만료됩니다.
- ✓ 서블릿 필터는 요청객체와 응답객체를 요청과 응답의 중간에 가로채어 필요한 처리를 합니다.
- ✓ JSTL은 표준 태그 라이브러리로써, 반복문, 조건문, 포맷팅, XML처리 등의 기능을 제공합니다.





5. Java 웹 프로그래밍 II

-
- 5.1 파일 업로드 처리
 - 5.2 요리책 만들기 #2
 - 5.3 사용자 정의 태그
 - 5.4 이벤트 리스너
 - 5.5 JSP 동작원리
 - 5.6 요약

5.1 파일 업로드 처리 (1/3) – 업로드 폼 구성하기

- ✓ 파일업로드란 클라이언트에 있는 파일을 웹 브라우저를 통해 서버 상의 특정 경로로 업로드 하는 것을 의미합니다.
- ✓ HTML은 폼 요소를 이용하여 웹 서버로 파일을 업로드 하는 방법을 제공합니다.
- ✓ <form> 태그의 method 속성을 “post”로 enctype 속성은 “multipart/form-data”로 설정합니다.
- ✓ 파일을 선택하는 <input> 태그를 추가합니다. input 태그의 type 속성은 “file”로 설정합니다.

파일업로드 HTML 폼 구성하기

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>File upload example</title>
</head>
<body>

<form action="upload.do" method="post" enctype="multipart/form-data">
    Description : <input type="text" name="description" /> <br />
    Attachment : <input type="file" name="attachment" /> <br />

    <input type="submit" value="전송" />
</form>

</body>
</html>
```

fileupload.jsp

5.1 파일 업로드 처리 (2/3) – @MultipartConfig

- ✓ 서블릿 3.0 부터 MultiPart 형식의 파일업로드를 위한 @MultipartConfig 어노테이션을 제공합니다.
- ✓ @MultipartConfig 어노테이션을 서블릿 클래스에 선언하면 Multipart 요청을 처리할 수 있습니다.
- ✓ 필요에 따라 @MultipartConfig 어노테이션의 몇 가지 속성으로 업로드 관련 설정을 할 수 있습니다.
- ✓ 서블릿의 doPost() 메소드에서 업로드 된 파일 정보를 Part 객체를 통해 조회할 수 있습니다.

서블릿에 파일업로드 기능 추가하기

```
@MultipartConfig(  
    fileSizeThreshold = 1024 * 1024 * 1,  
    maxFileSize      = 1024 * 1024 * 10,  
    maxRequestSize   = 1024 * 1024 * 15,  
    location         = "/upload"  
)  
public class FileUploadServlet extends HttpServlet {  
    ...  
}
```

설정 값 설명 : 업로드된 파일의 크기가 1MB가 넘는 경우 /upload 경로에 임시 파일로 저장합니다. 업로드 파일은 최대 10MB를 초과할 수 없고, 요청의 최대 크기는 15MB를 초과할 수 없습니다.

속성	설명	디폴트 값
fileSizeThreshold	업로드된 파일의 크기가 이 값보다 큰 경우만 임시 폴더에 파일로 생성함. 아닌 경우 메모리 상에서 처리됨	0
maxFileSize	업로드 가능한 최대 파일크기 (바이트 수)	무제한
maxRequestSize	업로드 파일을 포함한 요청의 최대크기 (바이트 수)	무제한
location	임시로 파일을 저장할 폴더로 절대경로만 가능	""

i @MultipartConfig 어노테이션에 있는 모든 속성은 생략 가능합니다. 이러한 속성 값들은 성능에 직접적인 영향을 주는 부분이므로 웹 애플리케이션이 구동되는 시스템 환경을 잘 고려하여 설정되어야 합니다.

5.1 파일 업로드 처리 [3/3] – Part 객체 사용하기

- ✓ @MultipartConfig 어노테이션이 선언된 서블릿은 Part 객체를 통해 업로드 된 파일을 처리할 수 있습니다.
- ✓ Part 객체는 요청객체로 부터 HTML 폼에서 지정한 파일 입력 폼의 name 값으로 조회합니다.
- ✓ Part 객체의 write() 메소드를 호출하면 업로드 된 파일을 다른 경로에 저장할 수 있습니다.
- ✓ Part 객체에는 업로드 파일의 정보를 조회하는 다양한 메소드가 존재합니다.

Part 객체를 사용한 업로드 파일 저장

```
@MultipartConfig  
@WebServlet("/upload.do")  
public class FileUploadServlet extends HttpServlet {  
    @Override  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, IOException {  
  
        // 1. retrieve part object  
        Part part = req.getPart("attachment");  
  
        // 2. prepare upload folder  
        File file = new File("d:\\upload\\");  
        if (!file.exists()) {  
            file.mkdirs();  
        }  
  
        // 3. save file  
        part.write("d:\\upload\\uploadfile");  
    }  
}
```

FileUploadServlet.java

메소드	설명
getSize() : long	현재 파트의 사이즈 반환
getContentType() : String	웹 브라우저에서 전달된 contentType 반환
write(String) : void	업로드 된 파트를 디스크에 저장함
getInputStream() : InputStream	파일의 컨텐츠를 조회할 수 있는 입력 스트림을 반환
getHeader(String) : String	지정한 파트의 헤더 값을 문자열로 반환
getHeaders(String) : Collection<String>	지정한 파트의 모든 헤더 값을 Collection 으로 반환
getName() : String	현재 파트에 대한 multipart form의 이름 반환
delete()	현재 파트와 관련된 모든 임시 파일을 삭제

5.2 요리책 만들기 #2 [1/3] – 레시피 등록

- ✓ 화면 URL : <http://localhost:8080/yorizori/recipe/register.do?cbId=요리책ID>
- ✓ 레시피 등록화면은 레시피 기본정보와 조리사진을 입력 받아 등록합니다.
- ✓ 등록자에는 로그인 된 사용자의 이름을 출력합니다
- ✓ 등록이 완료되면 레시피 목록화면으로 이동합니다.

레시피 등록

레시피명	된장찌개	등록자	한식매니아
조리시간	30	분	난이도 3
재료	된장, 물, 대파, 돼지고기		
조리절차			
조리사진	된장찌개.jpg	파일찾기	
목록		등록	

5. 레시피등록.html

URL 맵핑	HTTP 메소드	설명
/recipe/register.do	GET	레시피 등록 화면 조회
/recipe/register.do	POST	레시피 등록 요청

서블릿 클래스명	com.namoo.yorizori.web.controller.RecipeFormController
JSP 파일	/WEB-INF/views/recipeForm.jsp

서블릿 컨텍스트 파라미터를 통한 조리사진 업로드 경로 관리

이미지 업로드 경로는 web.xml에서 서블릿 컨텍스트 초기화 파라미터로 설정합니다.

```
<context-param>
  <description>레시피 이미지 저장경로</description>
  <param-name>imagePath</param-name>
  <param-value>업로드 경로(절대경로)</param-value>
</context-param>
```

서블릿에서는 컨텍스트 객체의 getInitParameter()를 호출하여 파라미터 값을 조회합니다.

```
String imagePath =
    getServletContext().getInitParameter("imagePath");
```

5.2 요리책 만들기 #2 [2/3] – 레시피 목록 조회

- ✓ 화면 URL : <http://localhost:8080/yorizori/recipe/list.do?cbId=요리책ID>
- ✓ 레시피 목록 화면은 요리책에 등록된 모든 레시피를 조회하여 출력합니다.
- ✓ 레시피 ID를 입력 받아 업로드 된 레시피 이미지를 OutputStream으로 출력하는 서블릿을 구현합니다.
- ✓ 레시피 이미지는 레시피 이미지 서블릿을 통해 이미지를 요청하여 태그를 사용하여 보여줍니다.

레시피 등록

레시피 목록 레시피 등록 ← 로그인 사용자만 버튼 보여줌

[한식요리책] 한식 요리를 맛있게 만드는 방법을 설명합니다.
해당 요리책의 이름과 설명을 보여줌

된장찌개
(30분/난이도:3)

김치찌개
(20분/난이도:2)

궁중불고기
(60분/난이도:1)

6. 레시피목록.html/

URL 맵핑	HTTP 메소드	설명		
/recipe/list.do	GET	레시피 목록 화면 조회		
서블릿 클래스명	com.namoo.yorizori.web.controller.RecipeListController			
	com.namoo.yorizori.web.controller.RecipeImageController			
JSP 파일	/WEB-INF/views/recipeList.jsp			
레시피 이미지를 제공하는 서블릿 만들기				
이미지를 조회할 수 있는 서블릿 클래스를 작성합니다.				
<ul style="list-style-type: none">- 서블릿 클래스 명 : RecipeImageController- 요청 메소드 : GET- 요청 URL : {ctx}/recipe/image.do?recipId=레시피ID				
처리내용 :				
<ol style="list-style-type: none">1. 서블릿의 응답객체에 ContentType을 레시피 이미지의 contentType 으로 설정합니다.2. JavaIO를 사용하여 저장된 이미지를 서블릿의 OutputStream 객체로 출력합니다.				

5.2 요리책 만들기 #2 [3/3] – 레시피 상세 조회

- ✓ 화면 URL : <http://localhost:8080/yorizori/recipe/detail.do?recipId=레시피ID>
- ✓ 레시피 상세 화면에서는 레시피ID를 입력받아 레시피 상세정보를 조회하여 보여줍니다.

레시피 등록

레시피 목록

레시피 등록 ← 로그인 사용자만 버튼 보여줌

레시피명	된장찌개	등록자	한식매니아
조리시간	30 분	난이도	3
재료	된장, 물, 대파, 돼지고기		
조리절차	1. 물을 넣고 끓인다. 2. 끓는 물에 된장을 푼다.		
조리예			

목록

7. 레시피상세.html

URL 맵핑	HTTP 메소드	설명
/recipe/detail.do	GET	레시피 상세 화면 조회
서블릿 클래스명	com.namoo.yorizori.web.controller.RecipeDetailController	
JSP 파일	/WEB-INF/views/recipeDetail.jsp	

5.3 사용자 정의 태그 (1/4) – 태그 파일

- ✓ 태그파일은 JSP 문법을 사용하여 사용자 정의 태그를 만드는 방법을 제공합니다.
- ✓ 작성한 태그파일은 웹 애플리케이션의 WEB-INF/tags 폴더에 위치시킵니다.
- ✓ 태그파일에서 tag 지시자로 태그 요소에 대한 설정을 하고 태그 요소에 속성이 있으면 attribute 지시자를 사용합니다.
- ✓ 사용자 정의 태그를 사용하는 JSP 페이지 상단에 taglib 지시자를 선언합니다.

태그 파일 작성

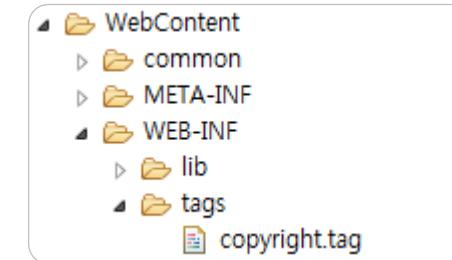
```
<%@ tag pageEncoding="UTF-8" body-content="empty" %>
<p style="clear:left">
    Copyright 2015 NamooSori(TM) all rights reserved.
</p>
```

copyright.tag

JSP에서 태그사용

```
<%@ taglib prefix="mytag" tagdir="/WEB-INF/tags" %>
...
<mytag:copyright />

</body>
</html>
```



5.3 사용자 정의 태그 (2/4) – 커스텀 태그 라이브러리 (1/2)

- ✓ 커스텀 태그 라이브러리는 Java 클래스로 사용자 정의 태그를 만드는 방법입니다.
- ✓ 커스텀 태그 클래스는 SimpleTagSupport 클래스를 상속받아 doTag() 메소드를 재정의 합니다.
- ✓ 커스텀 태그의 정보를 태그 라이브러리 템플릿(TLD) 파일로 작성하여 WEB-INF/tld 폴더에 위치시킵니다.
- ✓ 커스텀 태그를 사용하는 JSP 페이지 상단에 taglib 지시자를 선언합니다.

커스텀 태그 클래스 작성

```
public class StarTag extends SimpleTagSupport {  
  
    private Integer count;  
    private Integer max;  
  
    public void setCount(Integer count) {  
        this.count = count;  
    }  
  
    public void setMax(Integer max) {  
        this.max = max;  
    }  
  
    @Override  
    public void doTag() throws JspException, IOException {  
        StringBuilder sb = new StringBuilder(10);  
        for (int i = 0; i < max; i++) {  
            sb.append( (i < count) ? "★" : "☆");  
        }  
        getJspContext().getOut().print(sb.toString());  
    }  
}
```

StarTag.java

JSP에서 태그 사용

```
<%@ taglib prefix="namoo" uri="http://namoosori.com/tag" %>  
...  
<namoo:star count="${recipe.Level}" max="5" />  
recipeDetail.jsp
```

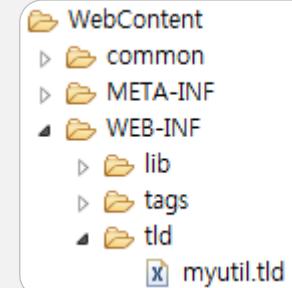
레시피 상세

레시피명	맛있는 피자	등록자	김기사
조리시간	30 분	난이도	★★★☆☆
재료	밀가루		
조리절차	1. 물에 밀가루를 넣는다. 2. 오븐에 넣고 굽는다.		
조리예			

5.3 사용자 정의 태그 (3/4) – 커스텀 태그 라이브러리 (2/2)

- ✓ 태그 라이브러리 기술자(TLD)를 작성합니다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib ...>
<description>Namoosori Tag Library</description>
<display-name>Namoo Tag</display-name>
<tlib-version>1.0</tlib-version>
<short-name>namoo</short-name>
<uri>http://namoosori.com/tag</uri>
<tag>
    <description>주어진 숫자만큼 채워진 별을 출력하는 태그</description>
    <name>star</name>
    <tag-class>com.namoo.yorizori.web.tag.StartTag</tag-class>
    <body-content>empty</body-content>
    <attribute>
        <name>count</name>
        <required>true</required>
        <rteprvalue>true</rteprvalue>
        <type>Integer</type>
    </attribute>
    <attribute>
        <name>max</name>
        <required>true</required>
        <rteprvalue>true</rteprvalue>
        <type>Integer</type>
    </attribute>
</tag>
</taglib>
```



myutil.tld

5.3 사용자 정의 태그 (4/4) – EL 함수

- ✓ 사용자 정의 태그는 아니지만 태그 라이브러리를 구현하여 EL 표현식으로 자바 메소드를 사용할 수 있습니다.
- ✓ EL 함수는 정적인 공용 메소드로 작성하여 태그 라이브러리 기술자(TLD) 파일에 정의합니다.
- ✓ EL 함수를 사용하는 JSP 상단에 taglib 지시자를 선언합니다.
- ✓ 이제, EL 표현식에서 작성한 EL 함수를 호출할 수 있습니다.

정적 메소드를 가진 클래스

```
package com.namoo.yorizori.web.el;

public class Adder {

    public static int add(int a, int b) {
        return a + b;
    }
}
```

Adder.java

[TLD] 정적 메소드를 EL 함수로 등록

```
...
<function>
    <description>두 수를 더하는 EL 함수</description>
    <name>add</name>
    <function-class>com.namoo.yorizori.web.el.Adder</function-class>
    <function-signature>int add(int, int)</function-signature>
</function>

</taglib>
```

myutil.tld

EL함수를 사용하는 JSP

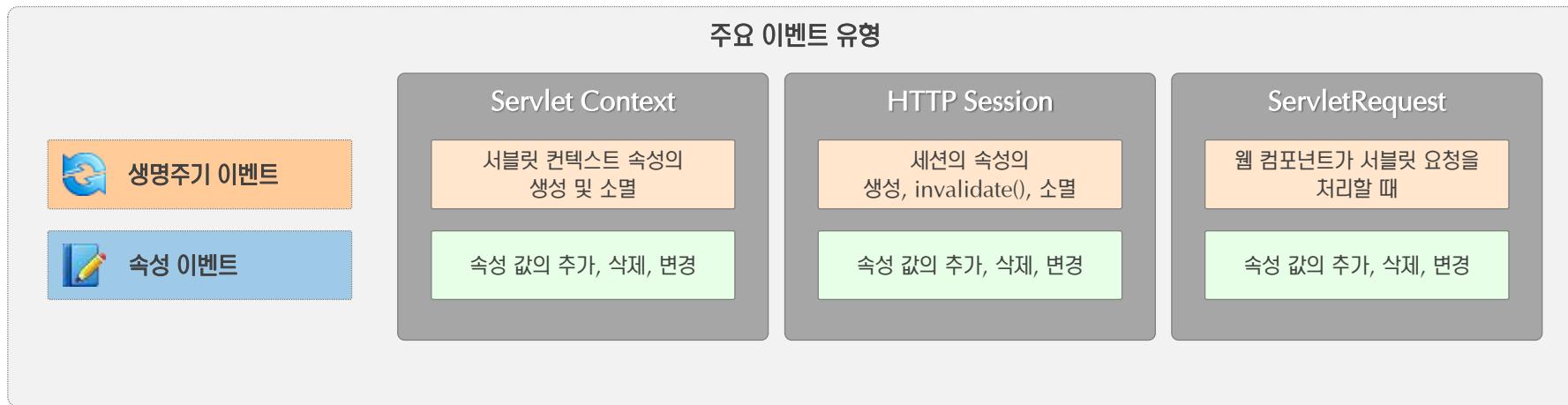
```
<%@ taglib prefix="namoo" uri="http://namoosori.com/tag" %>
...
${namoo:add(1, 3)}
```

웹 브라우저 실행결과

4

5.4 이벤트 리스너 (1/3) – 이벤트와 리스너

- ✓ 웹 애플리케이션 안에서는 생명주기 또는 속성 값의 변화 등과 관련된 다양한 이벤트가 발생합니다.
- ✓ 웹 컨테이너는 웹 애플리케이션에서 발생하는 이벤트를 통지하여 특별한 처리를 할 수 있는 이벤트 리스너를 제공합니다.
- ✓ 이벤트 리스너는 처리할 이벤트 유형에 따라 리스너 인터페이스를 구현하여 개발합니다.
- ✓ 이벤트는 범위에 따라 ServletContext, HTTP Session, HttpServletRequest로 구분하며, 이벤트의 유형에 따라 생명주기 이벤트, 속성값 변화 등으로 분류할 수 있습니다.

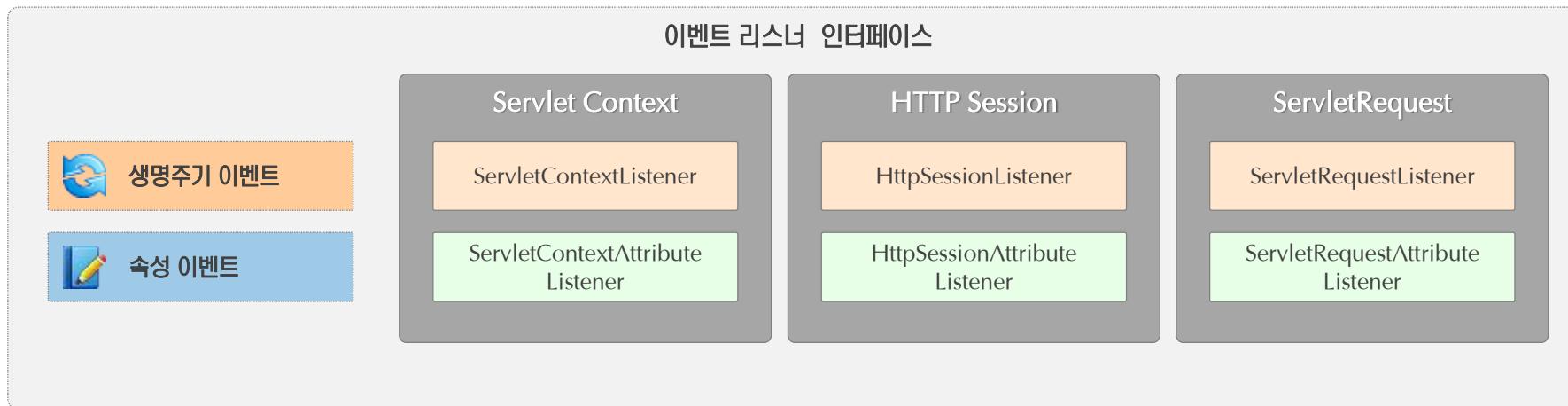


Q. 사용자가 로그인하거나 로그아웃 했을 때, 로그를 남기고 싶습니다. 그런데 문제가 있습니다. 로그인은 사용자의 서블릿 요청이므로 로그인을 담당하는 서블릿에서 로그를 남기면 될 것 같은데, 로그아웃은 사용자의 요청에 의한 로그아웃 뿐만 아니라, 세션 타임아웃이 발생하거나, 서블릿 컨테이너가 종료될 때 등 다양한 상황에서 발생합니다. 좋은 해결 방법이 있을까요?

A. 문제 상황은 이벤트 리스너를 활용하기에 적합한 상황입니다. 로그인 정보는 세션에서 관리되므로 HttpSession 속성 관련 이벤트를 처리하는 이벤트 리스너를 구현하여 로그인 정보를 담는 세션 속성의 추가 및 삭제 이벤트에 대해 로깅 처리를 하면 됩니다.

5.4 이벤트 리스너 (2/3) – 이벤트 리스너 종류

- ✓ 웹 애플리케이션에서 발생하는 이벤트의 종류에 따라 다양한 이벤트 리스너가 인터페이스로 제공됩니다.
- ✓ `ServletContext*Listener`는 서블릿 컨텍스트의 생명주기와 속성의 변화에 관한 이벤트를 처리합니다.
- ✓ `HttpSession*Listener`는 HttpSession 객체의 생명주기 및 속성의 변화에 관한 이벤트를 처리합니다.
- ✓ `ServletRequest*Listener`는 서블릿을 요청하거나, 요청객체 속성의 변화에 관한 이벤트를 처리합니다.



5.4 이벤트 리스너 [3/3] – [실습] 이벤트 리스너 활용

- ✓ 앞서 개발한 요리책 서비스에 사용자가 로그인하거나 로그아웃 한 경우 로그를 남기도록 기능을 추가해 봅시다.
- ✓ 로그인 정보는 HttpSession 속성으로 관리되므로 HttpSessionAttributeListener 인터페이스가 적합한 후보입니다.
- ✓ HttpSessionAttributeListener 인터페이스를 구현하여 UserAccessLoggingListener 클래스를 생성합니다.
- ✓ 리스너 등록은 web.xml 에 정의하거나 서블릿 3.0 부터 제공되는 @WebListener 어노테이션을 사용합니다.

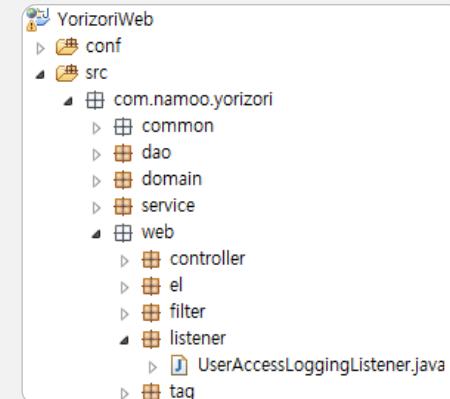
세션 속성의 추가 및 삭제 이벤트를 처리하는 이벤트 리스너 구현

```
@WebListener
public class UserAccessLoggingListener implements HttpSessionAttributeListener {
    private static final String SESSION_ATTRIBUTE_NAME = "loginUser";

    @Override
    public void attributeAdded(HttpSessionBindingEvent evt) {
        if (SESSION_ATTRIBUTE_NAME.equals(evt.getName())) {
            User user = (User) evt.getValue();
            System.out.println(user.getName() + " is login.");
        }
    }

    @Override
    public void attributeRemoved(HttpSessionBindingEvent evt) {
        if (SESSION_ATTRIBUTE_NAME.equals(evt.getName())) {
            User user = (User) evt.getValue();
            System.out.println(user.getName() + " is logout.");
        }
    }

    @Override
    public void attributeReplaced(HttpSessionBindingEvent evt) {
    }
}
```



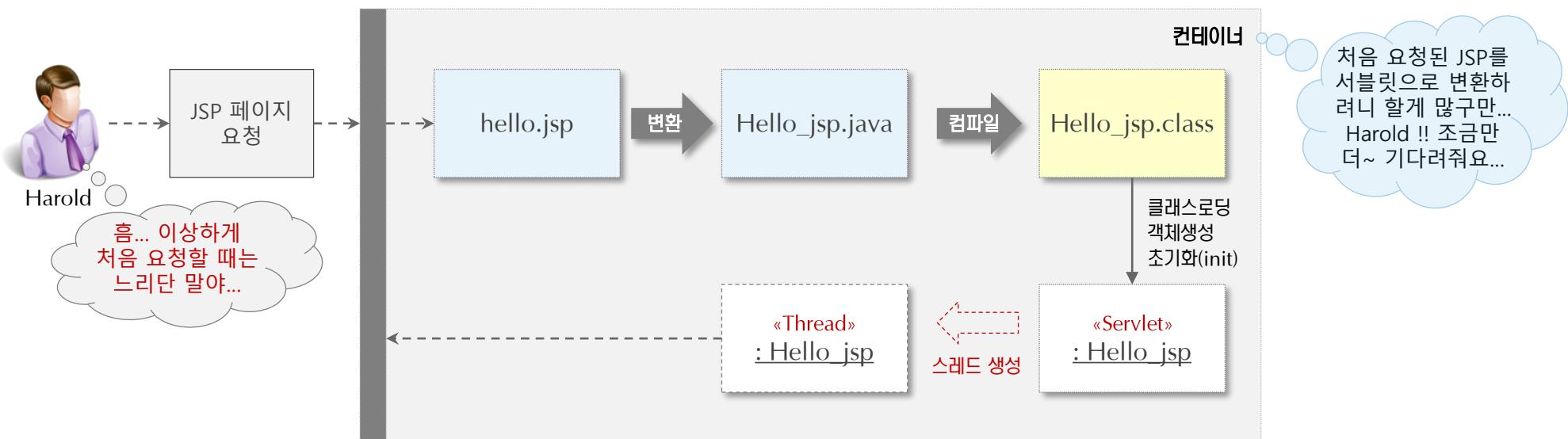
웹 브라우저 실행결과

요리조리 is login.
요리조리 is logout.
김기사 is login.

UserAccessLoggingListener.java

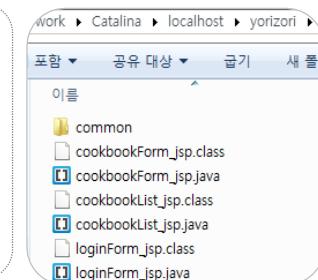
5.5 JSP 동작원리 (1/2)

- ✓ JSP는 컨테이너에 의하여 웹 애플리케이션에서 동작하는 서블릿으로 변환되어 실행됩니다.
- ✓ 컨테이너는 이러한 변환과 컴파일을 최초 요청 시 딱 한번만 처리합니다.
- ✓ JSP 요청 시점에 서블릿 변환을 처리하므로 최초 요청 시 약간의 시간이 필요하게 됩니다.
- ✓ 대부분의 컨테이너에서는 컨테이너가 구동되는 시점에 미리 변환처리를 할 수 있는 옵션을 제공합니다.



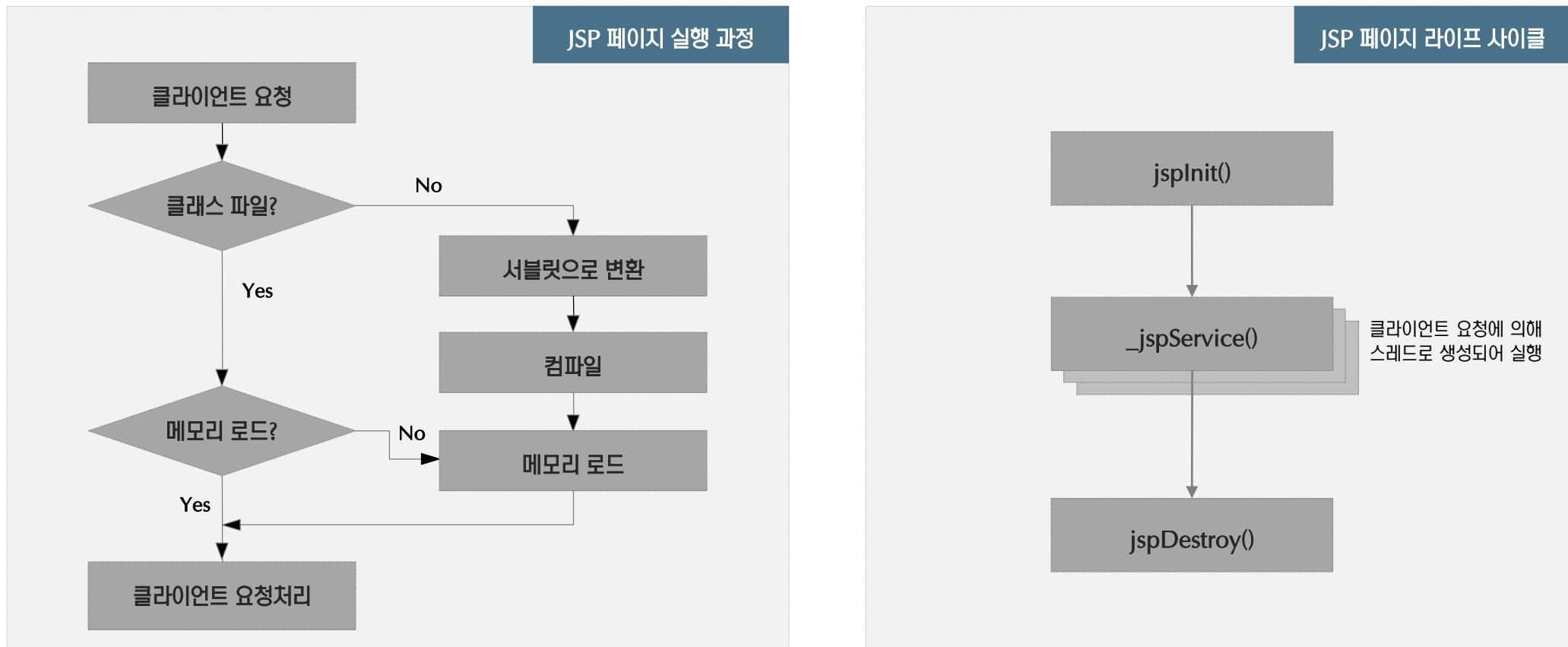
Q. 서블릿으로 변환된 자바 파일은 어디에서 확인할 수 있나요?

A. 톰캣을 기준으로 설명하면 톰캣이 설치된 위치의 work 폴더에서 확인할 수 있습니다. 혹시 변환된 서블릿 파일이 보이지 않나요? 서블릿 변환은 JSP 페이지의 최초 요청 시점에 일어납니다. (물론 서블릿 컨테이너 설정에 따라 컨테이너가 구동되는 시점에 먼저 변환할 수도 있긴 합니다.) 참고로, WTP 환경에서 실습하는 경우 work 폴더의 위치는 다음과 같습니다. ➔ {이클립스 워크스페이스경로}\.metadata\.plugins\org.eclipse.wst.server.core\tmp0



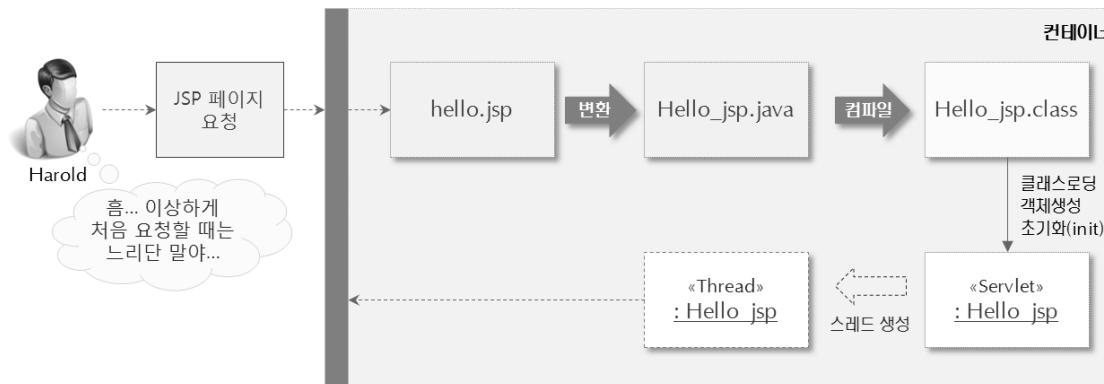
5.5 JSP 동작원리 (2/2)

- ✓ JSP가 변환된 서블릿 클래스에는 몇 가지 주요 API가 구현되어 있습니다.
- ✓ `jsplInit()` 메소드는 서블릿의 `init()` 메소드에서 호출합니다. 이 메소드는 JSP에서 선언문을 통해 재정의 할 수 있습니다.
- ✓ `jspDestory()` 메소드는 서블릿의 `destroy()` 메소드에서 호출합니다. JSP에서 선언문을 통해 재정의 할 수 있습니다.
- ✓ `_jspService()` 메소드는 서블릿의 `service()` 메소드에서 호출합니다. 각 요청마다 새로운 스레드로 실행되며 재정의 할 수 없습니다.



5.6 요약

- ✓ 서블릿은 @MultipartConfig 어노테이션을 사용해서 Multipart형식의 파일 업로드를 할 수 있습니다.
- ✓ SimpleTagSupport 클래스를 상속하고, TLD파일을 작성해서 커스텀 태그 라이브러리를 구현할 수 있습니다.
- ✓ 컨테이너는 웹 애플리케이션에서 발생하는 이벤트를 통지하고, 이 이벤트를 처리할 수 있도록 이벤트 리스너를 제공합니다.
- ✓ 컨테이너는 JSP를 서블릿으로 변환하여 실행합니다. 서블릿 변환과 컴파일은 최초 요청 시 한 번만 수행합니다.



토론

- ✓ 질문과 대답
- ✓ 토론

