

목차

- 서블릿 프로그래밍
- JSP 프로그래밍
- Servlet Filter
- Java Bean
- MVC 개발 패턴
- EL
- JSTL

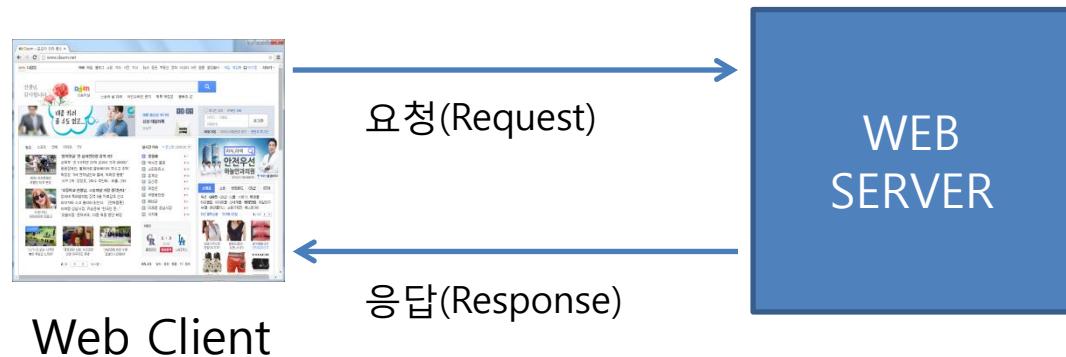
서블릿 프로그래밍

URL

- Uniform Resource Locator
- 웹 클라이언트가 웹 서버 쪽의 자료를 요청하기 위해 사용하는 표기
- [기본 응용 프로그램 실행]
 - http:// 웹서버명.도메인[:포트번호]
 - http:// 웹서버명.도메인[:포트번호]/[폴더명.../]요청파일명
- [다른 응용 프로그램 실행 URL]
 - http:// 웹서버명.도메인[:포트번호]/웹응용프로그램명
 - http:// 웹서버명.도메인[:포트번호]/웹응용프로그램명/[폴더명.../]요청파일명

HTTP

- 웹 서버와 웹 클라이언트는 소켓(Socket) 통신을 한다. 즉 TCP/IP 프로토콜로 데이터를 전달한다.
- HTTP(HyperText Transper Protocol)는 웹 클라이언트와 웹 서버가 통신할 때 사용하는 메시지의 표시 규약이다.
- HTTP는 전송 프로토콜인 TCP(Transmission Control Protocol)에 의해 웹 클라이언트와 웹 서버에 전송된다.
- 특징
 - Connectionless
 - Stateless



HTTP Protocol

요청 데이터 구조

[GET/POST] [요청하는 자료의 경로] [HTTP버전]

요청 헤더 : 헤더 값
요청 헤더 : 헤더 값
요청 헤더 : 헤더 값

.....

한 줄 공백

[POST방식의 경우 서버로 전달하는 데이터]

NAME=홍길동

ADDR=서울

Request Headers:

host: localhost:8080

connection: keep-alive

accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36

accept-encoding: gzip, deflate, sdch

accept-language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4

[요청헤더 예]

응답 데이터 구조

[HTTP버전] [응답코드] [코드에 대한 설명]

응답 헤더 : 헤더 값
응답 헤더 : 헤더 값
응답 헤더 : 헤더 값

.....

한 줄 공백

[웹브라우저로 전달하는 HTML]

<HTML>

<HEAD> </HEAD>

.....

</HTML>

Response Headers:

HTTP/1.1 200 OK

Content-type: text/html

Content-length: 1234

Cache-Control: no-cache

.....

<HTML>

<HEAD> </HEAD>

.....

</HTML>

[응답헤더 예]

개발환경 설정

- 자바8(JDK1.8) 설치

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

- Eclipse 4.4.2(Luna) 설치

<http://www.eclipse.org/downloads/>

- Tomcat 8 설치

<https://tomcat.apache.org/download-80.cgi>

- 이클립스 웹 프로젝트 생성

- Servlet/JSP 및 Tomcat 버전

Servlet/JSP Spec	Apache Tomcat version
3.0/2.2	7.0.x
2.5/2.1	6.0.x
2.4/2.0	5.5.x (archived)
2.3/1.2	4.1.x (archived)
2.2/1.1	3.3.x (archived)

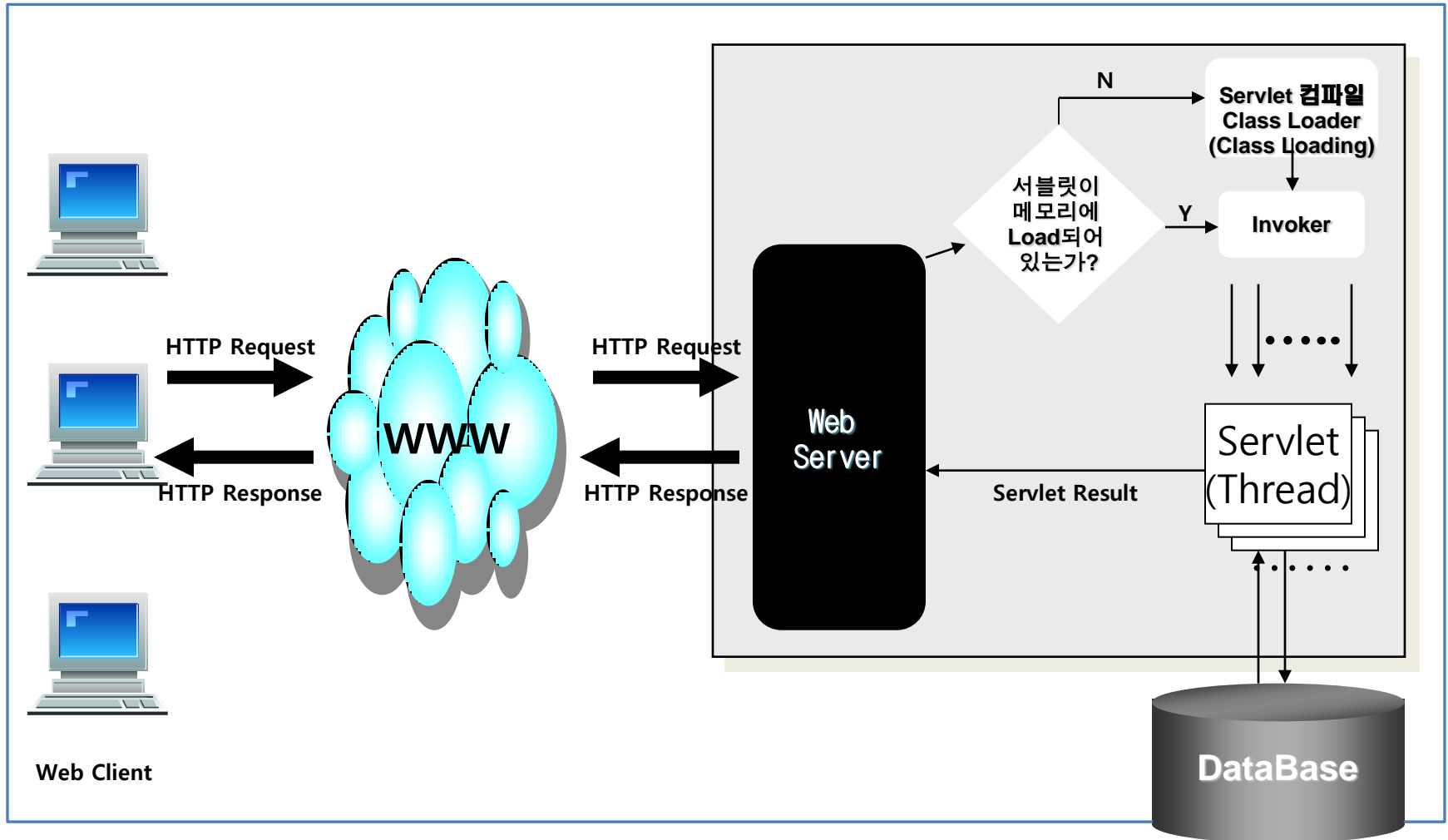
Servlet 소개

- 자바기반 웹 응용서버에서 HTTP의 요청/응답을 지원하는 자바기반 동적 파일이다.
- Web Application Server(WAS)에서 기동되는 Java Class 이다.
- 서버에서 객체 생성 후 멀티쓰레드로 동작한다.
- URL로 클라이언트가 실행 요청한다.
- 서블릿의 특정 메소드(service, doGet, doPost)를 호출하여 응답을 생성한다.

Servlet 특징

- 자바기반이므로 이식성이 좋다.
- 멀티 쓰레드로 동작하므로 수행성능이 좋다.
- TYPE에 대해 안전하며 고수준의 API를 지원한다.
- JDBC를 통해 데이터베이스의 데이터 처리 가능.
- 자바웹의 근본 기술이다.
- JSP는 서블릿으로 변환되어 WAS에서 멀티 쓰레드로 동작한다.

Servlet 실행순서



Servlet 실행 및 개발 환경

- 실행 환경
 - Java Runtime Environment
 - JDK 또는 JRE 가 설치된 서버
 - 웹 서버 + WAS라고 하는 Servlet Container(Servlet Engine)
 - Tomcat
 - Application Server (Jboss, WebLogic, JEUS, WebSphere...)
- 개발 환경
 - 실행 환경
 - J2EE API
 - 컴파일시 필요
 - 임포트 필요 : javax.servlet.*, javax.servlet.http.*
 - 개발 도구
 - Eclipse

Servlet Example

- **extends HttpServlet**
- **service() 메서드 오버라이딩**
- 실습을 위해 이클립스에서 Dynamic Web Project를 생성하세요.
- 생성시 generate web.xml deployment descriptor 체크해 주세요(web.xml 생성)

```
//HelloWorld.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
//@WebServlet("/HelloWorld"); //Tomcat7, Servlet3.0이상
public class HelloWorld extends HttpServlet{
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        //한글 쓸 경우 res.setContentType("text/html; charset=euc-kr");
        PrintWriter out = res.getWriter();
        out.println("<html> <head> <title>Hello World </title> </head>");
        out.println("<body> <BIG>OJC Hello World </BIG>");
        out.println("</body> </html>");    }    }
```

서블릿 URL 매핑

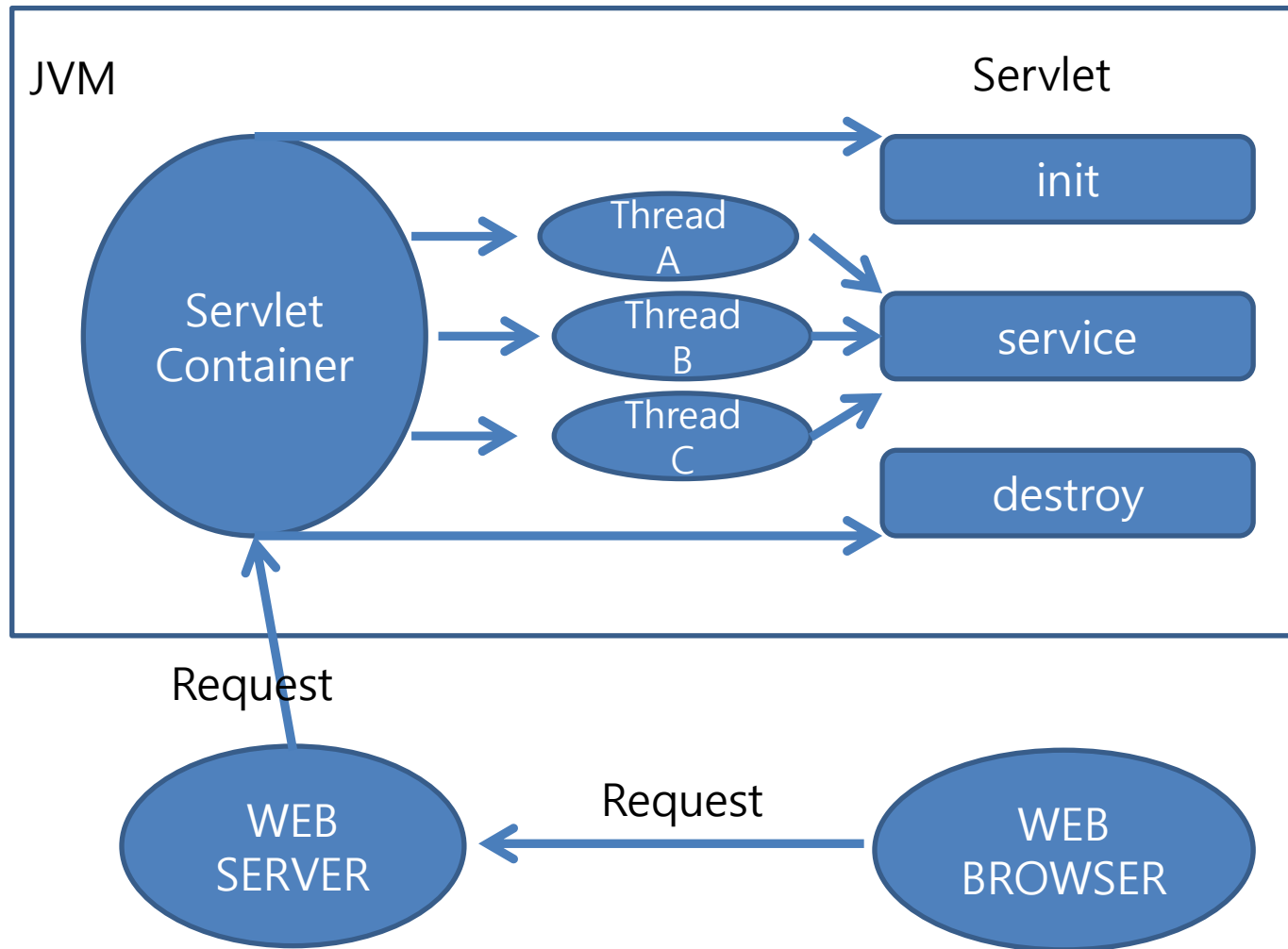
- web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>
```

Servlet의 요청 처리 메서드

- protected void **service**(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {}
 - 모든 요청 방식을 처리할 경우 오버라이딩
- protected void **doGet**(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {}
 - GET 요청 방식만 처리할 경우 오버라이딩
- protected void **doPost**(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {}
 - POST 요청 방식만 처리할 경우 오버라이딩

Servlet의 LifeCycle



Servlet의 LifeCycle

- `init()` 메소드가 호출되어 초기화가 이루어 진다.
 - 서블릿이 처음 만들어질 때 오직 한번만 호출됨
 - `public void init() throws ServletException {`
Initialization code...
`}`
- Client의 요청에 대해 `service()` 메소드가 호출.
 - 클라이언트 요청을 수행하는 메인 메소드
 - 사용자의 서블릿에서 `service()` 메소드를 오버라이드 하지 않으면 부모 서블릿의 `service()` 메소드가 사용자의 요청타입을 확인하며 타입에 맞는 메소드를 호출한다. (GET:doGet, POST:doPost...)
 - `public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException{ ... }`
- `destroy()` 메소드가 호출되어 소멸된다.
 - DB접속을 종료, 백그라운드 스레드를 종료, cookie나 hit count등을 하드디스크에 쓰는 일을 하며 `destroy()` 메소드가 호출되면 서블릿은 가비지 컬렉션을 해도 좋다고 표시된다.
- 마지막으로 서블릿은 JVM의 GC에 의해 가비지 컬렉션 된다.

초기화 정보 얻기

- init() 메소드에서 서블릿 개체 초기화
- 초기화 정보의 위치
 - web.xml

```
<servlet>
  <servlet-name>~</servlet-name>
  <servlet-class>~</servlet-class>
  <init-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </init-param>
</servlet>
```

- String **getInitParameter**(String name)

초기화 정보 얻기 (InitParameterServlet)

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

public class InitParameterServlet extends HttpServlet {
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
    {
        res.setContentType("text/plain;charset=euc-kr");
        PrintWriter out = res.getWriter();
        out.println("초기 파라미터들....");
        Enumeration e = getInitParameterNames(); //파라미터의 이름을 가지고옴
        while(e.hasMoreElements()) {
            String name = (String) e.nextElement();
            out.println(name + " ---> " + getInitParameter(name)); //이름을가지고값을
        }
    }
}
```

초기화 정보 얻기 (web.xml)

```
<servlet>
  <servlet-name>InitParameterServlet</servlet-name>
  <servlet-class>InitParameterServlet</servlet-class>
  <init-param>
    <param-name>tomcat ver</param-name>
    <param-value>8.0</param-value>
  </init-param>
  <init-param>
    <param-name>name</param-name>
    <param-value>apache tomcat</param-value>
  <init-param>
  </servlet>
<servlet-mapping>
  <servlet-name>InitParameterServlet</servlet-name>
  <url-pattern>/InitParameterServlet</url-pattern>
</servlet-mapping>
```

초기화 정보 얻기 (InitParameterServlet)

//@WebServlet 어노테이션의 속성 이용

```
@WebServlet(urlPatterns = "/init", initParams = {  
    @WebInitParam(name = "name", value = "tomcat"),  
    @WebInitParam(name = "version", value = "8.0") })
```

```
public class InitParameterServlet extends HttpServlet {  
    protected void service(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain;charset=euc-kr");  
        PrintWriter out = res.getWriter();  
        out.println("초기파라미터들....");  
        Enumeration e = getInitParameterNames();  
        while (e.hasMoreElements()) {  
            String name = (String) e.nextElement();  
            out.println(name + " ---> " + getInitParameter(name));  
        }  
    }  
}
```

요청/응답(HttpServletRequest/HttpServletResponse)

- HttpServletRequest 개체
 - String **getParameter**(String name)
 - String[] **getParameterValues**(String name)
 - String **getHeader**(String name)
 - String **getRemoteAddr**()
 - void **setCharacterEncoding**(String env)
- HttpServletResponse 개체
 - void **setContentType**(String type)
 - void **addHeader**(String name, String value)
 - PrintWriter **getWriter**()
 - ServletOutputStream **getOutputStream**()

Request Header 출력하기

```
@WebServlet("/ReqTest")
```

```
public class ReqTest extends HttpServlet {
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
response.setContentType("text/plain");
```

```
    PrintWriter out = response.getWriter();
```

```
    out.println("Request Headers:");
```

```
    Enumeration names = request.getHeaderNames();
```

```
    while (names.hasMoreElements()) {
```

```
        String name = (String) names.nextElement();
```

```
        Enumeration values = request.getHeaders(name);
```

```
        if (values != null) {
```

```
            while (values.hasMoreElements()) {
```

```
                String value = (String) values.nextElement();
```

```
                out.println(name + ": " + value);
```

```
            }
```

```
        }
```

```
    } }
```

Request Headers: host: localhost:8080 connection:
keep-alive accept:
text/html,application/xhtml+xml,application/xml;q
=0.9,image/webp,*/*;q=0.8 user-agent:
Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2272.118 Safari/537.36 accept-
encoding: gzip, deflate, sdch accept-language:
ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4

쿠키(Cookies)

쿠키(Cookie)

- 웹 서버가 클라이언트에게 보내는 텍스트 형태의 작은 정보의 조각
- 쿠키를 저장한 도메인, 만료 일시, 쿠키의 이름 등이 저장되어 있다.
- 자바 서블릿은 HTTP 쿠키를 지원하며 쿠키는 웹 서버의 응답헤더로 클라이언트로 전송된 후 브라우저는 쿠키의 만료일시가 기록된 것에 대해 디스크에 저장한다. 추후 다시 해당 사이트를 방문할 때 브라우저에 저장된 쿠키를 요청헤더과 함께 서버로 전달한다.
- 응답헤더 예
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2015 21:03:38 GMT
Server: Apache/1.3.9 (UNIX)
Set-Cookie: name=userinfo; expires=Friday, 04-Feb-07 22:03:38 GMT; path=/; domain=ojc.asia
Connection: close Content-Type: text/html
- 요청헤더 예
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: ojc.server:8080
Accept: image/gif, */* **Cookie: name=userinfo**

쿠키(Cookie)

■ 쿠키의 만료일

- 쿠키의 만료일시가 명시적으로 지정되어 있으면 쿠키는 영속적인 저장매체에 저장된다(IE인 경우 Temporary Internet Files 디렉토리에 개별적 파일로 저장, NetScape인 경우 cookies.txt file에 저장)
- 만료일시가 명시적으로 기술되어 있지 않으면 쿠키는 브라우저가 사용하는 메모리에 저장되어 브라우저가 종료되면 사라진다
- 두 경우 모두 Client가 서버에게 보내는 Request Header에는 다음과 같이 포함되어 전송된다. (Cookie : NAME = XXX)

■ 쿠키 목적 및 용도

- HTTP의 특징인 Stateless(상태를 보관하지 않음)를 보완하기 위한 방법으로 HTTP 연결 시 상태를 보관하는데 목적이 있다. 방금 방문한 사람이 그 사람인 것을 알도록 해주는 것
- 웹사이트 로그인, 장바구니, 방문횟수 카운트 등에 이용된다.

쿠키(Cookie) API

Method Summary

java.lang.Object	clone() 쿠키의 복사본을 돌려준다
java.lang.String	getComment() 주석이 있으면 주석을, 없으면 Null을 돌려준다
java.lang.String	getDomain() 쿠키에 대한 도메인을 돌려준다
int	getMaxAge() 쿠키의 최대 유효시간을 초단위로 돌려준다
java.lang.String	getName() 쿠키의 이름을 돌려준다
java.lang.String	getPath() Client가 쿠키를 돌려줄 서버의 경로를 돌려준다
boolean	getSecure() 브라우저가 쿠키를 보안 프로토콜로 보내야하면 true, 아니면 false를 돌려준다.

쿠키(Cookie) API

java.lang.String	getValue() 쿠키의 값을 돌려준다.
String	getVersion() 쿠키가 만족하는 Protocol의 버전을 돌려준다.
void	setComment (java.lang.String purpose) 쿠키의 목적을 설명하는 주석을 설정한다
void	setDomain (java.lang.String pattern)
void	setMaxAge (int expiry) 초단위로 쿠키의 최대시간을 기술, 이 값 이전에 브라우저를 종료시 HDD에 쿠키값을 보관한다, 음수값은 브라우저를 종료시 쿠키삭제, 0인경우 쿠키를 즉시 삭제
void	setPath (java.lang.String uri) Client가 쿠키를 돌려줄 서버의 경로를 설정
void	setSecure (boolean flag) 브라우저가 쿠키를 Https나 SSL와 같은 보안 프로토콜로 보내야 하는지를 결정한다
void	setValue (java.lang.String newValue) 쿠키가 만들어진후 쿠키에 새로운 Value를 setting,
void	setVersion (int v) 쿠키의 버전 Setting, 새롭게 만들어지는 쿠키는 기본적으로 0

쿠키(Cookie) 생성 및 읽기

- 서블릿 응답(HttpServletResponse)
 - addCookie(Cookie cookie)
 - 쿠키를 굽는다.
- 서블릿 요청(HttpServletRequest)
 - Cookie[] getCookies()
 - 클라이언트 웹 브라우저가 전송한 쿠키들을 읽어낸다.

쿠키 예제

```
@WebServlet("/CookieTest")
public class CookieTest extends HttpServlet {
    private static int count = 0;
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=euc-kr");
        PrintWriter out = response.getWriter();
        Cookie[] requestCookies = request.getCookies();
        out.write("<html> <head> </head> <body>");
        out.write("<h1> 쿠키 예제 </h1>");
        if (requestCookies != null) {
            out.write("<h3> Request Cookies:</h3>");
            for (Cookie c : requestCookies) {
                out.write("Name=" + c.getName() + ", Value=" + c.getValue()
                    + ", Comment=" + c.getComment() + ", Domain="
                    + c.getDomain() + ", MaxAge=" + c.getMaxAge()
                    + ", Path=" + c.getPath() + ", Version="
                    + c.getVersion());
            }
        }
    }
}
```

쿠키 예제

```
        out.write("<br>");  
    }  
}
```

```
count++;
```

```
Cookie counterCookie = new Cookie("OJC"+count, String.valueOf(count));
```

```
//쿠키에 대한 설명 추가
```

```
counterCookie.setComment("OJC Cookie Counter"+count);
```

```
// 하루짜리 쿠키
```

```
counterCookie.setMaxAge(24 * 60 * 60);
```

```
// 아래 주소로 다시 들어올때 쿠키를 가지고 온다.
```

```
counterCookie.setPath("/ojcweb/CookieTest");
```

```
// 쿠키를 굽는다.
```

```
response.addCookie(counterCookie);
```

```
out.write("Cookie Write OK..." + count);
```

```
out.write("</body> </html>");
```

```
}
```

```
}
```



Session Tracking

Session Tracking

- 세션이란? 웹 서버와 웹 클라이언트의 연결을 의미하며 웹 서버와 웹 클라이언트가 상태를 유지하기 위한 기술이다.
- 서블릿의 경우 세션을 수립하기 위해서는 `HttpServletRequest`의 `getSession()` 메소드를 불러줘야 하지만 JSP는 JSP 페이지 접속과 동시에 자동으로 세션 객체가 생성되어 세션ID가 부여된다.
- 세션은 쿠키 혹은 URL 리라이팅(rewriting)을 적절히 사용하여 구현되며 쿠키기반인 경우 `Session`의 ID는 웹브라우저의 쿠키에 저장된다. 서버에서 새로운 세션객체가 생성되면 세션ID라고 하는 고유한 식별자가 서버에 의해 생성되어 부여되고, 또한 쿠키 등을 통해 Client에 해당 ID가 주어진다.
- 서블릿 등이 웹 서버에 접속할 때 `Session ID`에 해당되는 세션객체를 얻은 후 객체에 담긴 정보를 뽑아낸다. (예를 들면 장바구니 정보, 사용자 로그인 정보)
- 웹 브라우저가 Cookie를 받아들이지 않게 설정되었다면 URL 재작성에 기반한 Session Tracking을 이용해야 한다.

Session Tracking(HttpSession 인터페이스)

- boolean isNew()
 - 현재 세션이 새로운 세션인지의 여부를 알려준다.
 - 서버에 의해 세션이 만들어졌으나 Client가 인식하지 못해 참여하지 않았다면 true
 - 서버가 단지 쿠키기반의 세션만 지원하고, Client가 완전히 쿠키의 사용을 금한다면, HttpServletRequest.getSession Method는 항상 새로운 세션을 Return한다.
- String getId()
 - Session ID를 리턴.
- long getCreationTime()
 - Session이 생성된 시각, 단위는 1970년 자정을 기준으로 밀리세컨드 개수(long type)
- long getLastAccessedTime()
 - 마지막으로 세션을 사용한 시각
 - 단위는 단위는 1970년 자정을 기준으로 흐른 밀리 세컨드의 개수(long type)

Session Tracking(HttpSession 인터페이스)

- `putValue(String name, Object value)`
 - 세션객체에 데이터를 추가, 이미 존재시 새 값으로 대체된다.
- `Object getValue(String name)`
 - 이 세션에서 유지되는 데이터의 이름과 값을 검색
- `removeValue(String name)`
 - 이름이 name인 세션 데이터 객체가 삭제
- 위의 3가지 메소드는 Servlet2.2 부터 Deprecate됨
`setAttribute(java.lang.String, java.lang.Object)`, `getAttribute(java.lang.String)`로 대체됨
- `String[] getValueName()`
 - 이 세션에서 유지되고 있는 value의 name을 String의 배열로 return.
- `public int getMaxInactiveInterval()`
 - 세션을 유지하는 최대단위(초)를 Return
- `public void setMaxInactiveInterval(int seconds)`
 - 세션을 유지하는 최대단위(초)를 설정
 - 이 시간을 넘기면 서버는 세션을 종료한다.
- `public void invalidate()`
 - 세션을 제거한다. 세션에 관련된 모든 객체를 해제하다.

Session Tracking

(HttpServletRequest 인터페이스, 메소드)

- HttpSession getSession()
 - 이 요청에 연관된 현재 세션을 돌려주고 없다면 세션을 새로 만들어 돌려준다.
- HttpSession getSession(boolean create)
 - 이 요청에 연관된 현재 세션을 돌려주며 사용자가 유효한 세션을 가지고 있지 않다면 Create 인자가 true면 새로운 세션을 돌려주며, False이면 null을 돌려줌
- boolean isRequestedSessionIdValid()
 - 세션이 유효하면 true
- boolean isRequestedSessionIdFromCookie()
 - 세션ID가 Cookie를 통해 전달된 것이면 true
- boolean isRequestedSessionIdFromUrl()
 - 세션ID가 URL를 통해 전달된 것이면 true

간단한 Session 인증예제(SessionTest.java)

/* 클라이언트의 요청시 이미 세션 Setting되어 있으면, 인증된 형태를 보여주는 HTML을 , 아닌경우 인증창이 있는 HTML을 보여준다 , SessionTest.java를 먼저 실행*/

@WebServlet("/SessionTest")

```
public class SessionTest extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException,  
        ServletException {  
        res.setContentType("text/html; charset=euc-kr");  
        PrintWriter out = res.getWriter();  
        String szID="";  
        String szPasswd= "";  
  
        HttpSession session = req.getSession(false);  
        // 이미 세션이 Setting되어 있는지 확인,인증되어 있다면 인증박스가 없는 html 표시, 아니  
        // 면 인증창이 있는 html 표시  
        if (session != null) {  
            szID = (String) session.getAttribute("id");
```

간단한 Session 인증예제(SessionTest.java)

```
szPasswd =(String) session.getAttribute("passwd");

out.println("<html> <head> <title>");
out.println("Session Test Program");
out.println("</title> </head>");
out.println("<body> <table width=500> <tr>");
out.println("<form method=get action=/ojcweb/SessionExpire name=logout >");

out.println("Id : "+szID + "   Passwd : "+szPasswd);
out.println("<input type=submit name=Submit  
value=LogOut> </tr> <tr> </form> <hr>");
out.println("[02/28]<a href=/ojcweb/SessionNewsDetail>");
out.println("신규 아파트에 지하철까지 개통, 투자가치 부각되는 암사동</a>");
out.println("</tr> </table>");
out.println("</body>");
out.println("</html>");
}
```

간단한 Session 인증예제(SessionTest.java)

```
else {
    out.println("<html> <head>");
    out.println("<title>Session Test Program</title> </head>");
    out.println("<body>");
    out.println("<table width=500> <tr>");
    out.println("<form method=post action=/ojcweb/SessionIsOk name=login >");
    out.println("<input type=text size=10 maxlength=40 name=id>");
    out.println("PassWord : ");
    out.println("<input type=password size=10 maxlength=40 name=passwd>");
    out.println("<input type=submit name=Submit value=OK>");
    out.println("</form> </tr> <hr>");
    out.println("<tr>");
    out.println("[02/28]<a href=/ojcweb/SessionNewsDetail>신규 아파트에 지하철까지 개통,");
    out.println("투자가치 부각되는 암사동</a>");
    out.println("</td> </tr> </table> </body> </html>");
}
}
```

간단한 Session 인증예제(SessionIsOk.java)

```
/* SessionTest에서 OK Click시 넘어 오는 ID와 passwd를 parameter로 받아  
인증된 사용자인지의 여부를 판단 */
```

```
@WebServlet("/SessionIsOk")
```

```
public class SessionIsOk extends HttpServlet {  
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException,  
        ServletException {  
        res.setContentType("text/html; charset=euc-kr");  
        PrintWriter out = res.getWriter();  
        String szID=null;  
        String szPasswd=null;  
        if (req.getParameter("id") != null)        szID = req.getParameter("id");  
        if (req.getParameter("passwd") != null)    szPasswd = req.getParameter("passwd");  
  
        // 인증이 성공이라면, 흔히 이부분의 인증을 DB의 사용자 Table등의 ID/PassWd를  
        // 비교하여 인증이 성공인지의 여부를 판단한다. 아이디, 비번 모두 ojc로 ...  
        if ((szID.equals("ojc")) &&
```

간단한 Session 인증예제(SessionIsOk.java)

```
(szPasswd.equals("ojc"))) {  
    HttpSession session = req.getSession(true);  
    session.setAttribute("id", szID);  
    session.setAttribute("passwd", szPasswd);  
    out.println("<html> <head>");  
    res.setHeader("Refresh","3 URL=/ojcweb/SessionTest");  
    out.println("Session Setting OK!! <br>");  
    out.println("3초후면 첫페이지로 이동합니다.");  
    out.println("</head> </html>");  
}  
// Id와 PassWord가 맞지 않는 경우  
else {  
    out.println("<html> <head>");  
    res.setHeader("Refresh","3; URL=/ojcweb/SessionTest");  
    out.println("ID PassWord not correct...<br>");  
    out.println("3초후면 첫페이지로 이동합니다.");  
    out.println("</html> </head>");  
}  
}  
}
```


간단한 Session 인증예제 (SessionExpire.java)

```
/* LogOut을 Click시 세션을 해제한다 */
```

```
@WebServlet("/SessionExpire")
```

```
public class SessionExpire extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws IOException, ServletException {  
        HttpSession session = req.getSession(false);  
        if (session != null) session.invalidate(); //로그아웃  
  
        res.setContentType("text/html; charset=euc-kr");  
        res.sendRedirect("/ojcweb/SessionTest");  
    }  
}
```

간단한 Session 인증예제 (SessionNewsDetail.java)

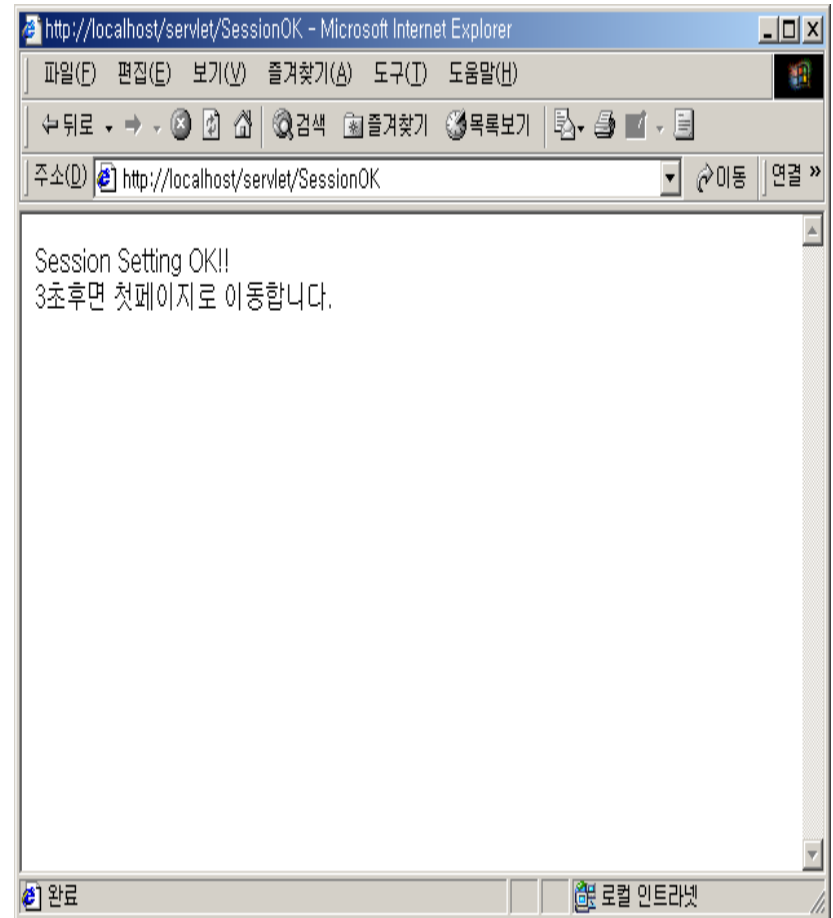
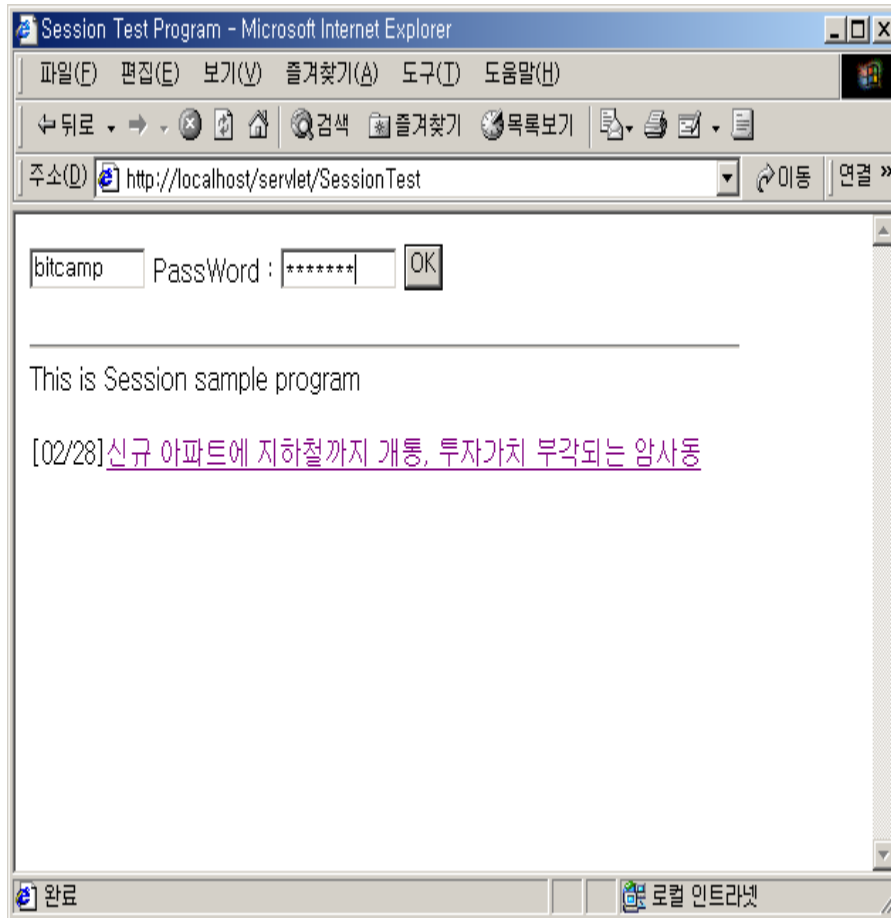
/* 사용자가 뉴스를 보고자 할 때 이미 인증이 되어 있는지의 여부를 판단한 후 인증된 사용자만 뉴스를 보여준다*/

```
@WebServlet("/SessionNewsDetail")
public class SessionNewsDetail extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException {
        res.setContentType("text/html; charset=euc-kr");
        PrintWriter out = res.getWriter();

        HttpSession session = req.getSession(false);

        if (session != null) {
            String id = (String) session.getAttribute("id");
            // 이미 인증이 되어 있는 경우라면
            if (id != null) {
                out.println("<b>신규 아파트에 지하철까지 개통, 투자가치 부각되는 암사동");
                out.println("</b><hr>");
                out.println("입력일 : 2015/02/28 <hr></center>");
            }
        }
        else{
            res.setHeader("Refresh", "3; URL=/ojcweb/SessionTest");
            out.println("로그인 부터 하세요.");
        }
    }
}
```

간단한 Session 인증예제(실행화면)



JDBC 프로그래밍

JDBC 개요

- 자바 프로그램 --> java.sql 패키지 --> JDBC드라이버 --> DB
- 정 의
 - 관계형 DBMS에 SQL 문장을 실행시키기 위한 자바 API 표준 인터페이스
- java.sql 표준 패키지 및 JDBC드라이버의 표준을 정한다.
- 목 적
 - DBMS 종류에 관계없이 자바 프로그램에서 사용하는 표준 DB API구현
- java.sql 패키지
 - SQL문장을 JDBC 드라이버에 전달하고, 그 결과를 반환하기 위한 클래스가 제공된다.
 - 이 패키지의 클래스는 내부적으로 JDBC 드라이버에 작업을 넘겨주어 그 결과를 자바 프로그램에 제공한다. java.sql 패키지의 대부분의 메소드에서 SQLException(java.lang.Exception의 하위 클래스)예외가 발생할 수 있다.
- JDBC 드라이버
 - java.sql 패키지로부터 요청된 데이터베이스 질의어(SQL) 문장을 접근하는 DBMS에 적절한 형태로 전달하고, 그 결과를 돌려준다.
 - java.sql 패키지 안에 들어 있는 JDBC API는 단지 몇 개의 견고한 클래스를 포함하고 있다. API의 상당부분이 구현 없이 행위만을 기술하는 데이터베이스 중립적인 인터페이스 클래스로 구성되어 있고, 실제구현은 DB Vender 들에 의해 구현.

JDBC 개요

- 사용할 수 있는 질의어의 범위
 - 사용된 JDBC 드라이버 및 DBMS의 능력(DB Vendor)에 의해 대부분 결정된다.
 - 데이터베이스가 지원하기만 하면 SQL이 아닌 임의의 문법도 가능하다
 - 사용된 JDBC 드라이버에 의해 DBMS의 SQL 처리능력이 어느 정도 제한 되거나 확장될 수 있다.
- JDBC 질의어 문법 표준
 - JDBC 표준을 따르는 모든 JDBC 드라이버는 표준 ANSI SQL99를 지원한다.

JDBC 연결순서

- 첫 번째
 - 데이터베이스와 접속하기 위해 애플리케이션의 JVM안으로 특정 드라이버 클래스를 적재.
 - 오라클 Thin driver : `Class.forName("oracle.jdbc.driver.OracleDriver");`
 - MS-SQL : `Class.forName(com.microsoft.sqlserver.jdbc.SQLServerDriver);`
 - 드라이버가 메모리에 적재될 때, `java.sql.DriverManager` 클래스를 사용해서 이 드라이버를 사용 가능한 드라이버로 등록.
- 두 번째 : `DriverManager` 클래스를 이용하여 URL 형태로 주어진 데이터 베이스에 대한 접속을 요청.
 - `Connection con = DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.27:1521:XXX", "scott", "tiger");`
 - JDBC URL은 드라이버 고유의 방식으로 개별적인 데이터베이스를 식별.
오라클 : `[jdbc:oracle:thin:@DB호스트명:port:sid]`
MS-SQL : `jdbc:sqlserver://DBHOST 명 :1433; databaseName= XXXDB;user=test; password=test`

JDBC 연결순서

- 세 번째 : SQL 질의어 실행
 - `java.sql.Statement`, `PreparedStatement` 클래스를 사용한다.
 - `Statement` 클래스는 직접 인스턴스화 되지 않고 `Connection` 클래스의 `createStatement()` 메소드에 의해 생성.
 - ☞ `Statement stmt = con.createStatement();`
 - `Statement`의 `executeUpdate()`를 실행하여 `INSERT`, `UPDATE`, `DELETE`등을 실행하거나, `executeQuery()` 메소드를 사용하여 데이터를 포함하는 `java.sql.ResultSet`을 리턴받는다.
 - ☞ `ResultSet rs = stmt.executeQuery("SELECT * FROM EMP");`
- 마지막 : 연결 Close
 - `connection.close();`
 - `Connection`은 메모리와 CPU 자원을 점유하여 사용하므로 상당한 `Overhead`를 가져온다. 따라서 사용 후 반드시 `Connection`을 닫아 주는 것이 좋다.

JDBC(ResultSet)

- ResultSet 객체
 - “SELECT” 질의어의 결과 레코드셋을 받을 때 사용
 - next() 메소드를 이용하여 한 레코드씩 다음행으로 이동.
 - 테이블의 칼럼 데이터타입에 따른 메소드(getXXX)를 호출해서 데이터값을 리턴 받는다.
 - ☞ getObject(), getString() 메소드를 사용해서 열(column)값을 얻는다.

DBConnectionTest.java

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

```
@WebServlet("/db")
```

```
public class DBConnectionTest extends HttpServlet {
```

```
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
```

```
        Connection con = null;
```

```
        Statement stmt = null;
```

```
        ResultSet rs = null;
```

```
        res.setContentType("text/html; charset=euc-kr");
```

```
        PrintWriter out = res.getWriter();
```

```
        try {
```

```
            // 오라클 드라이버를 Load 한다
```

```
            Class.forName("oracle.jdbc.driver.OracleDriver");
```

DBConnectionTest.java

```
// 데이터 베이스에 접속을 한다.
con = DriverManager.getConnection(
    "jdbc:oracle:thin:@192.168.0.27:1521:XXX", "scott", "tiger");

// Statement object를 생성한다.
stmt = con.createStatement();
// ResultSet을 얻기위해 SQL query를 실행한다.
rs = stmt.executeQuery("SELECT EMPNO, ENAME FROM EMP");

// 결과 출력    out.println("<HTML> <HEAD> <TITLE>Employees</TITLE> </HEAD>");
out.println("<BODY>");
out.println("<UL>");
while(rs.next()) {
    out.println("<LI>" + rs.getString("empno") + " " + rs.getString("ename"));
    //rs.getString(1), rs.getString(2)등 숫자도 가능
}
```

DBConnectionTest.java

```
        out.println("</UL>");
        out.println("</BODY> </HTML>");
    }
    catch(ClassNotFoundException e) {
        out.println("Couldn't load database driver: " + e.getMessage());
    }
    catch(SQLException e) {
        out.println("SQLException caught: " + e.getMessage());
    }
    finally {
        // 데이터베이스 연결을 종료.
        try {
            if (con != null) con.close();
        }
        catch (SQLException ignored) { }
    }
}
```

JDBC(executeUpdate)

- 데이터 베이스 갱신
 - SQL 문의 DML문장인 UPDATE, INSERT, DELETE문장을 실행. 실행 후 WHERE절에 의해 영향을 받은 레코드 건수를 리턴 (INSERT,UPDATE,DELETE된 건수를 리턴)
 - `public int executeUpdate(String sql) throws SQLException`
 - `int count = stmt.executeUpdate("DELETE FROM EMP WHERE EMPNO=7369");`

Insert 예제(Insert.java)

```
/* JDBC Insert 예제 */
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

@WebServlet("/insert")
public class Insert extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html; charset=euc-kr");
        PrintWriter out = res.getWriter();

        out.println("<html> <head> <title>JDBC Insert예제 </title> </head> <body>");
        out.println("<form name=myForm action=/web/insert method=post>");
        out.println("  이름 : <input type=text name=name> <br>");
        out.println("  전화번호 : <input type=text name=phone> <br>");
        out.println("  성별   :남 <input type=radio name=sex value='M'>");
```

Insert 예제(Insert.java)

```
        out.println("    여 <input type=radio name=sex  
value='F'> <br>");  
        out.println("나이 : <input type=text name=age size=3>");  
        out.println("<input type=submit value='저장'>");  
        out.println("</form> </body> </html>");  
    }  
  
    public void doPost(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        Connection con = null;  
        Statement stmt = null;  
        ResultSet rs = null;  
        String szName = null;  
        String szPhone = null;  
        String szSex = null;  
        int iNum = 0;  
        int iAge = 0;  
        String Query = null;
```

Insert 예제(Insert.java)

```
res.setContentType("text/plain; charset=euc-kr");
PrintWriter out = res.getWriter();

try {
    // 오라클 드라이버를 Load 한다
    Class.forName("oracle.jdbc.driver.OracleDriver");

    // 데이터 베이스에 접속을 한다.
    con =
    DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.27:1521:XXX",
    "scott", "tiger");
    con.setAutoCommit(false);

    //Form의 파라미터를 받음
    if (req.getParameter("name") != null) szName = req.getParameter("name");
    if (req.getParameter("phone") != null) szPhone = req.getParameter("phone");
    if (req.getParameter("sex") != null) szSex = req.getParameter("sex");
    if (req.getParameter("age") != null) iAge =
    Integer.parseInt(req.getParameter("age"));
```


Insert 예제(Insert.java)

```
// 순번값을 가져옴
stmt = con.createStatement();
rs = stmt.executeQuery("SELECT max(num)+1 FROM employees ");
rs.next();
iNum = rs.getInt(1);
rs.close();

//입력 Query 생성
Query = "insert into employees (num, name, phone, sex, age) values (" +
iNum + "," + szName + "," + szPhone + "," +
      + szSex + "," + iAge + ")";

stmt.executeUpdate(Query);
con.commit();
stmt.close();
con.close();

res.sendRedirect("/web/db"); //web은 해당 프로젝트의 컨텍스트명(사이트명)
}
```

Insert 예제(Insert.java)

```
catch(ClassNotFoundException e) {
    out.println("Couldn't load database driver: " +
e.getMessage());
}
catch(SQLException e) {
    out.println("SQLException caught: " + e.getMessage());
}
finally {
    // 언제나 데이터 베이스 연결을 종료한다.
    try {        if (con != null) con.close(); }
catch (SQLException ignored) { }
}
}
```

PreparedStatement Interface

- SQL 문이 빠른 수행을 위해 DB에 의해 전 처리된다.
- 오라클에서 해당 SQL문장은 상수값이 바뀌면서 반복되더라도 Soft Parsing 한다. (성능향상)
- 위치표시자를 위해 '?' 사용한다.
- 문자열에 " ' "등이 있는 경우 DB에 저장시 에러발생
→ 이를 극복하기 위해 PreparedStatement를 사용하면 효과적
- **clearParameters()**
이전에 정의된 파라미터 값을 제거
- **setXXX()**
물음표로 표시된 표시위치자에 실제 값을 할당하기 위해 사용한다.

예제(Insert2.java)

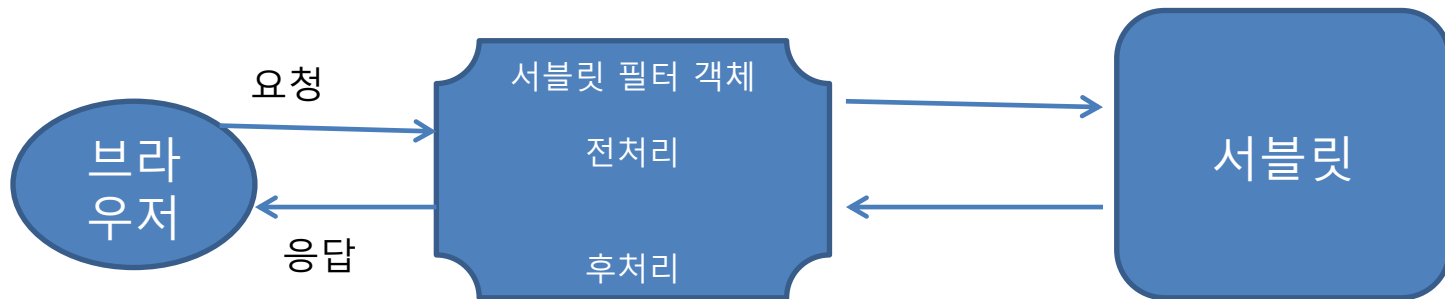
/* 앞의 Insert.java에서 입력 Query 생성부분을 아래와 같이 바꾸면 된다. */

```
//입력 Query 생성
Query = "insert into employees (num, name, phone, sex, age)
values (?, ?, ?, ?, ?)";
PreparedStatement pstmt = con.prepareStatement(Query);
pstmt.setInt(1, iNum);
pstmt.setString(2, szName);
pstmt.setString(3, szPhone);
pstmt.setString(4, szSex);
pstmt.setInt(5, iAge);

pstmt.executeUpdate();
con.commit();
con.close();
```

Servlet Filter 소개

- 서블릿 스펙 2.3 이후
- 클라이언트 요청을 처리하기 전에 전처리, 응답을 보내기 전에 후처리 된다.
- 스프링의 경우 스프링 인터셉터, AOP기능을 이용하면 필터 역할을 대체할 수 있다.



- 서블릿 필터로 가능한 작업
 - 로깅 및 감사 기능
 - 응용 프로그램의 실행 시각 측정 가능
 - 인증 여부 확인 기능
 - 복호화 및 암호화 기능
 - 문자셋 변환 기능

Servlet Filter Life Cycle

- javax.servlet.Filter 인터페이스를 구현
- 웹응용프로그램이 처음 실행될 때 개체 생성되고 init() 실행
- 클라이언트 요청이 있을 때마다 doFilter() 실행
- 웹응용프로그램이 종료될 때 destroy() 실행



Filter 클래스 작성

- javax.servlet.Filter 인터페이스를 구현
- Init(...), doFilter(...), destroy() 오버라이딩

```
package filter;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
public class FilterTest implements Filter {
    private FilterConfig fc;
    public void init(FilterConfig config) throws ServletException {
        this.fc = config;
    }
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException,
    IOException {
        HttpServletRequest httpReq = (HttpServletRequest) req;
        String ip = httpReq.getRemoteHost();
        System.out.println("IP Address is " + ip);
        chain.doFilter(req, resp);
    }
    public void destroy() {        // TODO Auto-generated method stub    }
}
```

Filter 등록

- /WEB-INF/web.xml
 - 필터 등록
 - 필터 매핑(url-pattern 이나 servlet 기술)

```
<filter>  
  <filter-name>myfilter</filter-name>  
  <filter-class>filter.FilterTest</filter-class>  
</filter>  
<filter-mapping>  
  <filter-name>myfilter</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>
```


Filter 등록

- @WebFilter 어노테이션 사용하기
- /WEB-INF/web.xml에 필터매핑 할 필요 없다.

```
@WebFilter(filterName = "myFilter",
           urlPatterns = {"/*"},
           initParams = {
               @WebInitParam(name = "addr", value = "seoul")})
public class FilterTest implements Filter {
    .....
    .....
}
```

서블릿 리스너(Servlet Listener)

- Spring Framework의 ContextLoaderListener 같은 리스너를 구현해 보자.
- 서블릿 컨텍스트, 즉 웹사이트가 다운되거나 시작되는 경우에 어떤 일을 하고자 할 때 ServletContextListener를 구현하면 된다.
- contextDestroyed는 컨텍스트가 더 이상의 웹요청을 받을 수 없을 때, 즉 톰캣 등이 종료될 때 호출되는 이벤트이고 contextInitialized 메소드는 톰캣 등이 시작됨 으로서 컨텍스트가 요청을 받아들일 수 있을때 호출되는 이벤트이다.
- 결국 서블릿 컨텍스트가 초기화 되거나 소멸될 때 어떤 일을 하고 싶을 때 이용하면 된다.

서블릿 리스너(MyListener.java)

```
package java1;

import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;
import javax.servlet.*;

public class MyListener implements ServletContextListener {
    private ServletContext ctx = null;

    //Tomcat Shutdown 시키면 아래 메시지 확인
    public void contextDestroyed(ServletContextEvent event) {
        System.out.println(">>> onjweb Webapp destroyed!!");
        this.ctx = null;
    }

    //Tomcat시작시 아래 메시지 출력
    public void contextInitialized(ServletContextEvent event) {
        this.ctx = event.getServletContext();
        System.out.println(">>> onjweb Webapp start!!");
    }
}
```

서블릿 리스너(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"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <listener>
    <listener-class>java1.MyListener</listener-class>
  </listener>
</web-app>
```

어노테이션 기반으로 작성하려면
앞페이지 MyListener클래스에
@WebServletContextListener
어노테이션을 기술하고 web.xml에
<listener> 설정을 빼면 된다.

서블릿 리스너(실행)

- 먼저 톰캣을 시작하면 contextInitialized 메소드가 호출된다.
- Tomcat을 그냥 꺼버리면 contextDestroyed에서 정의한 메시지가 출력안되니 , 테스트를 위해MyListener.java를 수정해 보라.

11월 09, 2014 3:54:16 오전 org.apache.catalina.core.ApplicationContext log
INFO: No Spring WebApplicationInitializer types detected on classpath

>>> **onjweb Webapp start!!**

11월 09, 2014 3:54:16 오전 org.apache.coyote.AbstractProtocol start

.....

INFO: Server startup in 1155 ms

----- 시작 후 여기서 자바파일 수정!!

11월 09, 2014 3:54:46 오전 org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/onjweb] has started

>>> **onjweb Webapp destroyed!!**

11월 09, 2014 3:54:47 오전 org.apache.catalina.core.ApplicationContext log
INFO: No Spring WebApplicationInitializer types detected on classpath

>>> **onjweb Webapp start!!**

.....

JSP 프로그래밍

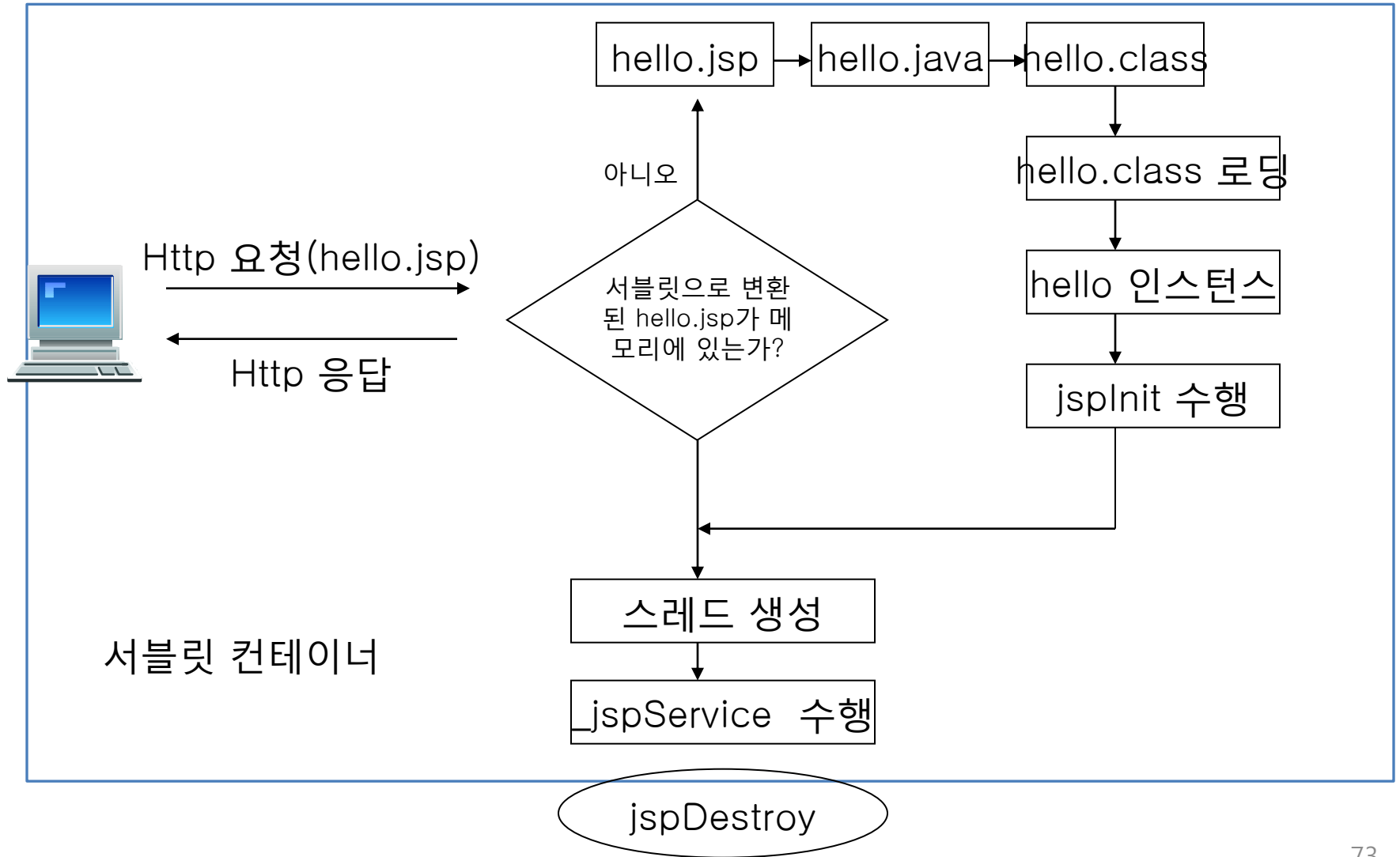
JSP 소개

- Java Server Page
- 서블릿과 JSP 차이
 - 서블릿
 - 자바 언어 중심적 동적 파일
 - HTML 응답을 생성하기 보다는 요청 흐름을 제어하는 역할
 - JSP
 - 태그 중심적 동적 파일
 - 클라이언트의 프리젠테이션이 목적, 데이터를 보여주는 VIEWER 용도
- 확장자가 **jsp** 이다.
- 서블릿 파일로 변환 후 컴파일 된 후 객체 생성후 멀티 스레드로 실행된다.
- URL로서 클라이언트가 실행 요청한다.

JSP 실행 원리

- JSP에서 서블릿으로 변환 된 후 초기화(init) → 서비스(service) → 소멸(destroy)의 단계를 거친다.
- .jsp파일을 .java로 변환 후 .class 파일로 컴파일하는 번역단계 (translation phase)가 필요하며 이 번역단계는 HTTP요청마다 매번 실행되는 것이 아니라 처음 또는 JSP파일이 변경되는 경우에 실행된다.
- 번역단계를 마친 후 초기화(jspInit) → 서비스(_jspService) →소멸(jspDestroy)과정을 거친다.
- 서블릿으로 변환된 JSP가 메모리에 로드되면 HTTP 요청시마다 생성되는 독립적인 서블릿 Thread에서 서비스 메소드(_jspService)가 실행되어 Client의 요청에 응답한다.

JSP 라이프사이클



JSP Hello world!

- HTML 폼에서 데이터를 입력 받아서 Web Server의 first.jsp에 GET 방식으로 데이터 전송, first.jsp를 first.java로 변환 first.class 로 컴파일 후 객체 생성하여 Thread로 만들어 클라이언트의 요청 처리한다.
- Web Server의 first.jsp에서 데이터를 처리하고 브라우저에 전송 하여 결과를 출력한다.

first.html

```
<html><head><title>JSP First Page </title></head>
<body>
<p>How many times?</p>
<form method=get action=/javaweb/jsp/first.jsp>
<input type=text size=2 name=loopcount>
<input type=submit>
</form></body></html>
```

JSP Hello world!

first.jsp

```
<% @ page language="java"%>
<html><head><title>First JSP</title></head>
<body>
    <p>numtomes : <%= request.getParameter("loopcount") %>
        <br><br>
        <%
            int loopcount =
Integer.parseInt(request.getParameter("loopcount"));
            for (int i=0; i < loopcount; i++) {
                out.println("Hello... <br>");
            }
        %>
    </p>
</body>
</html>
```

JSP 구성 요소

- Static Content
 - HTML, XML, WML
- Dynamic Content
 - 지시자(Directive): @page, @include, @taglib
 - 스크립팅 요소(자바 코드 기술)
 - 선언문(Declarations)
 - 스크립틀릿(Scriptlet)
 - 표현식(Expressions)
 - JSP 액션
 - EL(Expression Language)
 - JSTL
- 주석문
 - 태그 주석문
 - JSP 주석문

지시문(Directive)

- 지시자는 JSP가 JSP 컨테이너에게 어떠한 메시지를 보내기 위한 것
- 클래스선언, 구현할 메소드, Content-Type등의 값을 설정
- 클라이언트에 어떠한 출력도 내보내지 않는다.
- 모든 지시자들은 JSP 파일 전체에 대한 범위를 가진다.
- 지시자는 태그 안에서 @로 시작한다.
- page, include, taglib 지시자 등이 있다.
 - @page
 - 실행에 관련된 정보를 JSP 컨테이너에게 제공토록 지시한다.
 - 한번 이상 기술할 수 있고, 일반적으로 JSP 상단에 기술한다.
 - @include
 - 외부 파일에 있는 내용을 복사하도록 지시한다.
 - 한번 이상 기술할 수 있고, 기술 위치는 상관없다.
 - @taglib
 - 태그 라이브러리를 사용토록 지시한다.
 - 한번 이상 기술할 수 있고, 일반적으로 JSP 상단에 기술한다.

지시문(@Page)

- @page 지시문
 - <%@page
 - [language="java"] //페이지 개발언어
 - contentType="{text/plain|text/html|text/xml}" //컨텐츠의 마임타입
 - [pageEncoding="{iso-8859-1|euc-kr}" //현재 JSP페이지 인코딩
 - [import="java.util.*, java.sql.*"] //자바 импорт 정의
 - [isErrorPage="true|false"] //에러페이지인지의 여부(기본은 false)
 - [info="페이지정보"] //서블릿 getServletInfo() 메시지로 출력되는것
 - [session="true"|"false"] //세션을 사용할건지의 여부(기본은 true)
 - [isELIgnored="true"|"false"] //EL을 무시할건지의 여부(기본은 false)
 - %>
- Page 전체에 영향을 미치는 중요한 특성을 정의
- JSP 페이지의 여러 속성과 페이지가 가지는 기능을 정의
- 하나의 JSP 파일에서 여러 개의 page지시자의 사용은 가능하나, 동일한 특성이 여러 번 나와서는 안 된다.(import는 예외)

지시문(@include)

- @include 지시문
 - `<%@include file="포함할 파일의 URL" %>`
 - 포함되는 파일이 포함하는 파일과 정적으로 하나의 파일로 결합되어 컴파일 된다.(<jsp:include>는 동적으로 런타임으로 포함되어 실행된다.)

예) `<%@ include file="/javaweb/jsp/header.jsp">`

지시문(@taglib)

- @taglib 지시문

- <%@taglib

- uri="tagLibraryURI"

- prefix="tagPrefix" %>

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

- <%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>

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

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

- <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

선언문(Declarations)

- 멤버 필드와 메서드를 선언하며 번역된 서블릿의 `_jspService()` 메소드 밖에 코드가 생성된다.
- `<%!`
 멤버필드선언;
 메서드 선언;
 `%>`
- 예약된 메서드
 - `public void jspInit() {}` `//서블릿/JSP 초기화`
 - `public void jspDestroy() {}` `//서블릿/JSP가 소멸될 때 호출`

jspInit(), jspDestroy()

- javax.servlet.jsp.JspPage 인터페이스에 정의
- jspInit()
 - 해당 페이지가 최초로 load될 때 호출된다.
 - 페이지에 서비스를 수행하기 전에 초기화할 루틴을 삽입한다.
 - Optional Method
 - `<%! ~ %>` declaration 태그에 정의한다.
- jspDestroy()
 - 해당 페이지의 instance가 사라지는 시점에 호출된다.
 - 해당 페이지가 변경되어 새로이 load될 필요가 있을 때 이미 load된 instance는 destroy가 호출되고 새로운 instance의 init이 호출된다.
 - Optional Method
 - `<%! ~ %>` declaration 태그에 정의한다.

JSP로 만든 Counter

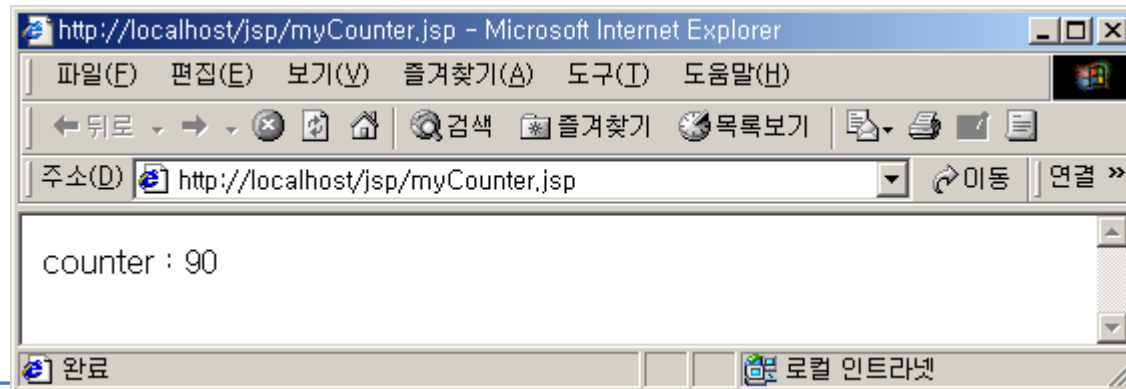
```
<!-- myCounter.jsp 영구적인 Counter 제작 ->
<% @ page import="java.io.*" contentType="text/html; charset=euc-kr" %>
<%!
    int count;
    public void jspInit() {
        try {
            FileReader fr = new FileReader("c:\\program
files\\allaire\\jrun\\jsp\\myCounter.dat");
            BufferedReader br = new BufferedReader(fr);
            String initial = br.readLine();
            count = Integer.parseInt(initial);
            return;
        }
        catch (FileNotFoundException ignored) {}
        catch (IOException ignored) {}
        catch (NumberFormatException ignored) {}
    }
}
```

JSP로 만든 Counter

```
String initial = getInitParameter("initial"); //초기 파라미터에서 확인
try {
    count = Integer.parseInt(initial);
    return;
}
catch (NumberFormatException ignored) { }
count=0;
}
public void jspDestroy() { //JSP Instance가 소멸시 카운터 값을 파일에 저장
    saveState();
}
public void saveState() {
    try {    FileWriter fw = new FileWriter("c:\\program
files\\allaire\\jrun\\jsp\\myCounter.dat");
        String initial = Integer.toString(count);
        fw.write(initial, 0, initial.length());
```

JSP로 만든 Counter

```
fw.close();  
    return;  
}  
catch (IOException e) {  
}  
}  
%>  
<% count++; %>  
counter : <%= count %>  
<% saveState(); %> //현재의 counter 값을 파일에 저장
```



스크립틀릿(Scriptlet)

- 응답을 생성하기 위해 실행할 자바 코드가 작성되는 부분
- <% 자바코드; %>

표현식(Expressions)

- 특정 위치에 문자열로 출력하고 할 때 사용
- `<%=단일값%>` -> `<% out.println(단일값); %>`
- `<%=변수|메서드호출|수식%>`

객체 사용 범위

- JSP 실행 과정에서 개체를 저장하거나 사용할 위치
- 스코프 종류
 - page : 해당 JSP 파일 안에서만 사용 가능
 - request : 응답이 출력되기 전까지 사용 가능
 - session : 세션이 유지되는 동안 동일한 클라이언트에서 사용 가능
 - application : 모든 클라이언트 웹페이지에서 사용 가능

JSP 내장 객체

내장 개체 참조 변수	타입
page	java.lang.Object
request	javax.servlet.http.HttpServletRequest
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
out	javax.servlet.jsp.JspWriter
exception	java.lang.Throwable
response	javax.servlet.http.HttpServletResponse
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

request 객체

- 클라이언트에서 서버로 보내는 요청을 담고있는 객체
- 요청범위를 가지며 javax.servlet.http.HttpServletRequest 클래스의 한 인스턴스 이다.
- 서블릿의 service 메소드의 아규먼트인 HttpServletRequest와 동일하게 사용된다.
- JSP 컨테이너는 이 객체를 _jspService() 메소드의 한 인자로서 JSP에게 넘겨준다.

request 객체의 메소드

- `getAttribute()` : 속성값을 반환한다.
- `getAttributeNames()` : 모든 속성들의 이름을 배열로 반환
- `getParameter()` : HTML문서의 form객체의 값을 반환
- `getHeader(name)` : 지정된 name의 header값을 반환
- `getCookies()` : Client로 전송된 Cookie 값을 반환
- `getSession()` : 현재 세션객체를 반환
- `getRemoteHost()` : Client의 HostName을 반환
- `getRemoteAddr()` : Client의 IP 주소를 반환
- `getContentType()` : MIME 값을 반환

response 객체

- `javax.servlet.http.HttpServletResponse` 클래스의 한 인스턴스.
- JSP에 의해 생성된 클라이언트에 보내질 응답을 캡슐화 한 것.
- 서블릿의 `service` 메소드의 인자인 `HttpServletResponse`와 동일하게 사용된다.
- 이 객체는 컨테이너에 의해 생성되어 `_jspService()` 메소드의 한 인자로서 JSP에게 전달되며 JSP는 그 메소드 안에서 적절하게 response 객체를 수정한다.
- JSP에서는 클라이언트로의 출력 스트림에 버퍼링이 적용되므로 일단 어떠한 출력이 있는 후에 HTTP상태코드나 응답헤더를 설정해도 오류가 생기지 않는다.
- 주로 JSP 페이지의 쿠키설정, 헤더설정, 버퍼설정, 문자셋, MIME Type을 변경 시 사용

response 객체 메소드

- `getCharacterEncoding()` : 속성값을 반환한다.
- `flushBuffer()` : 버퍼의 내용을 Client에 전송한다.
- `getBufferSize()` : 출력버퍼의 크기를 정수로 반환한다.(byte 단위)
- `setBufferSize(a)` : 출력버퍼의 크기를 a로 설정(byte 단위)
- `addCookies(a, value)` : Client에 새로운 쿠키정보를 전송
- `sendRedirect(url)` : 브라우저에 Redirect 응답을 보낸다.
- `addHeader(a, value)` : 이름이 a이고 값이 value인 헤더를 추가한다.
- `setHeader(a, value)` : 이름이 a인 헤더의 값을 value로 지정한다.
- `containsHeader(a)` : 이름이 a인 헤더를 포함하는지 확인한다.

response 객체 예제

response.jsp

```
<% @ page contentType="text/html; charset=euc-kr" %>
<html><head><title>Response 객체 예제</title></head><body>
    <h3>Response 객체를 이용한 예제</h3>
    Buffer의 크기를 설정 : 32K로 설정
<%
    response.setBufferSize(32000);
%>
<p>    현재 버퍼의 크기는 :
<%=    response.getBufferSize()/1000 %> K
<p>3초후 다음 메인페이지로 이동합니다...
<%
    //아래의 주석부분을 해제후 실행하세요~
    //$response.setHeader("refresh", "3; URL='http://www.daum.net'");
    //아래의 코드는 요청된 리소스가 새위치로 이동되었으며, 그위치는
Location Header에 포함된다는것을 알린다.
    /*response.setStatus(response.SC_MOVED_TEMPORARILY);
    /*response.setHeader("Location", "http:// www.daum.net ");

    //^response.sendRedirect("http:// www.daum.net ");
%></body></html>
```

pageContext 객체

- 페이지 범위를 가지며, javax.servlet.jsp.PageContext 클래스의 한 인스턴스이다.
- 해당 JSP Page의 페이지문맥을 캡슐화 한것
- 다른 명시적 객체에 접근하는 여러 가지 편리한 함수제공(예를 들면 setAttribute를 이용하여 scope[page, request, session, application) 내에서 작동하는 객체에 접근 할 수 있고, removeAttribute 메소드를 이용해 객체를 삭제 할 수도 있다.)

예) HttpSession session = pageContext.getSession();

JspWriter out = pageContext.getOut();

ServletRequest request = pageContext.getRequest();

pageContext 객체

- pageContext객체는 현재 페이지의 제어권을 다른 페이지로 넘기는 방법을 제공한다. 잠시 줄 수도 있고 영구히 줄 수도 있다.<jsp:include> <jsp:forward>등에 해당되는 include, forward등의 메소드가 있다.
- JSP 컨테이너에 의해 실행 되기 전 자동으로 서블릿의 javax.servlet.jsp.PageContext 객체로 변환되어 해석된다.
- 내부객체를 얻는 메소드
 - getPage(), getServletConfig()
 - getRequest(), getResponse()
 - getOut(),getSession()
 - getServletContext():Application객체
 - getException() : exception객체

pageContext 객체 메소드

- Scope에 따라 속성과 관계되는 메소드

Abstract java.lang.Object.[getAttribute](#)(java.lang.String name)

- “name”이라는 객체를 반환, 없을 경우 null반환

Abstract void [removeAttribute](#)(java.lang.String name)

- “name”이라는 이름의 객체를 삭제

Abstract void [setAttribute](#)(java.lang.String name, java.lang.Object attribute)

- attribute에 해당하는 “name”이라는 이름의 객체생성

pageContext 객체 예제

pageContext.jsp

```
<% @ page contentType="text/html; charset=euc-kr" %>
<html><head><title>Response 객체 예제</title></head><body>
    <h3>pageContext 객체를 이용한 예제</h3>
<%
    //pageContext.getOut().println("Hello~");
    //pageContext.getOut().println("<br>");
    //pageContext.include("date.jsp");

    //아래의 경우에는 에러난다. 같은 문맥이 아니므로...
    //pageContext.forward("http://www.daum.net");

    //다음과 같은 경우엔 가능하다.
    pageContext.forward("first.jsp");
%></body></html>
```

session 객체

- 세션범위를 가지며 , `javax.servlet.http.HttpSession` 객체의 한 인스턴스이다. 이것은 요청을 보낸 클라이언트에 대해 생성된 세션을 대표하며 HTTP 요청에 대해서만 유효하다.
- 세션은 자동적으로 생성되므로 이 객체는 세션을 사용하지 않는 경우에도 유효하다. 유일한 예외는 page지시자의 session특성을 이용해서 session을 끈 경우인데, 그런 경우 이 객체를 참조하려고 하면 컴파일 시점에 오류가 생긴다.
- JSP의 세션객체는 JSP컨테이너에 의해 서블릿의 `javax.servlet.http.HttpSession` 객체로 변환된다. 결국 JSP의 session객체와 `HttpSession`는 같은 것이다.
- 하나의 클라이언트가 JSP 페이지가 있는 웹 서버에 페이지를 요구하면 서버는 요청에 응답하여 클라이언트에 세션을 부여한다. 각각의 클라이언트에 부여되는 세션은 다르게 할당된다.

session 객체(메소드)

- `getId()` : session ID를 돌려줌
- `getCreateTime()` : session이 생성된 시간을 돌려줌
- `getLastAccessedTime()` : session이 마지막으로 액세스된 시간을 돌려준다.
- `getMaxInactiveInterval()` : session이 유지되는 시간을 얻음(초단위)
- `setMaxInactiveInterval(time)` : session이 유지되는 시간을 설정(초단위)
- `isNew()` : 사용자의 브라우저가 세션ID를 할당받지 않았으면 `true`를 return 한다.
- `invalidate()` : session 객체를 소멸시킨다. Session에 저장되어 있던 정보는 모두 삭제된다.

application 객체

- JSP파일을 포함하고 있는 웹 애플리케이션 전체에 대한 정보를 가지고 있는 객체로 여기에서 웹 애플리케이션이라고 하는 것은 웹상에서 실행되는 프로그램으로서 JSP, 서블릿, HTML의 집합을 의미한다.
- JSP 페이지는 해당 웹 페이지의 URL에 따라서 여러 개의 Application으로 분류되고 JSP 컨테이너는 URL에 있는 첫번째 디렉토리 이름을 애플리케이션으로 사용한다.
- application 범위를 가지며, javax.servlet.ServletContext 클래스의 한 인스턴스이다.
- JSP가 실행되고 있는 문맥(서블릿 문맥)을 대표한다.
- JSP 컨테이너에 의해 자동으로 서블릿의 ServletContext 객체로 변환된다.
- session객체의 경우 각 사용자마다 하나의 세션을 공유하나, application 객체의 경우 각 서버 내에 있는 모든 JSP File들은 하나의 application 객체를 공유한다.
- application 객체의 getServerInfo() 메소드를 이용하여 컨테이너의 버전과 이름을 알 수 있다.

application 객체(메소드)

- `application.getMajorVersion()` : 서블릿 컨테이너의 메이저 버전을 구한다.
- `application.getServletContextName()` : 현재 JSP가 속한 웹 애플리케이션의 이름을 return
- `application.MimeType(filename)` : 지정한 파일의 MIME Type을 return
- `application.RealPath(path)` : 지정한 path의 로컬 파일 시스템 URL을 return
- `application.log(message)` : 로그 파일에 message를 기록

out 객체

- 페이지 범위를 가지며, `javax.servlet.jsp.JspWriter` 클래스의 한 인스턴스이다. 클라이언트에게 전달될 출력 스트림을 대표한다.
- 클라이언트에게 실제 출력을 보내는 것은 `PrintWriter`이며 `JspWriter`는 `response` 객체를 좀더 유용하게 하기위해서 `PrintWriter`에 버퍼링 기능을 추가한 것이다.
- `page` 지시자의 `buffer` 특성을 이용해서 버퍼크기를 변경도 가능하며 버퍼링 자체를 끌 수도 있다.

page 객체

- 페이지 범위를 가지며, `java.lang.Object` 클래스의 한 인스턴스이다.
- `page` 객체는 JSP 문서로부터 생성된 서블릿의 인스턴스 그 자체를 가리킨다.
- JSP 페이지 자체를 가르키는 것으로 페이지의 언어가 `java`인 경우 `page` 객체는 'this' 객체와 같은 의미를 갖는다.
- JSP가 서블릿으로 변환된 소스의 `_jspService()` 메소드를 보면 다음과 같은 코드가 보일 것 이다.

`Object page = this;`

- 거의 사용하지 않는다.

config 객체

- 페이지 범위를 가지며, javax.servlet.ServletConfig 객체의 한 인스턴스이다. 서블릿 설정을 대표한다.
- JSP페이지가 JSP 컨테이너에 의해 초기화 될때 전달 받는 객체이다.
- 서블릿 초기 환경 설정 데이터를 저장 할 때 사용하는 객체로써 JSP 웹 페이지를 변환한 서블릿을 초기화 할 때 사용되는 초기화 파라미터를 저장하고 있다. 이 객체는 서블릿 에서는 유용하게 사용되지만 JSP 에서는 별로 사용되지 않는다.
-

config 객체

- `getInitParameterNames()` : 모든 초기화 파라미터를 Enumeration 형태의 객체형식으로 가져온다.
- `getInitParameter(String name)` : 지정된 name 초기화 파라미터를 가져온다.

```
예) <%! public void jspInit() {  
    String name = config.getInitParameter("name");  
    }  
%>
```

exception 객체

- 페이지 범위를 가지며, `java.lang.Throwable` 클래스의 한 인스턴스이다.
- 에러페이지(`isErrorPage=true`)안에서만 사용이 가능하다.
- 대부분의 컨테이너들은 예외를 request객체의 한 특성으로 포함시켜(`ServletRequest.setAttribute()` 메소드를 이용) 에러페이지에게 전달하고 에러페이지에서는 `request.getAttribute(String name)` 메소드를 이용하여 예외를 뽑아내는 방식을 취하고 있다.

내장객체 예제(1) – response

```
<!-- redirection.jsp →  
<%  
response.setContentType("text/html; charset=euc-kr");  
response.setHeader("Refresh", "10; URL=http://www.daum.net") ;  
%>  
  
<html>  
<body>  
    <%! int i=0; %>  
  
    <% i++; %>  
    <h2><%= i %>번째 방문입니다. <br>  
        10초 후에 다음으로 이동합니다. ^^ <br>  
</body>  
</html>
```

JSP 표준 Action

- JSP 표준 Action?
 - 자주 사용되는 자바 코드를 표준 태그화 시킨것
 - XML 문법에 맞추어 작성
 - <시작태그/>, <시작태그>...</끝태그>
 - 속성명=“속성값” or 속성명=‘속성값’
- JSP 표준 Action 종류
 - <jsp:include>
 - <jsp:forward>
 - <jsp:param>
 - <jsp:useBean>
 - <jsp:setProperty>
 - <jsp:getProperty>

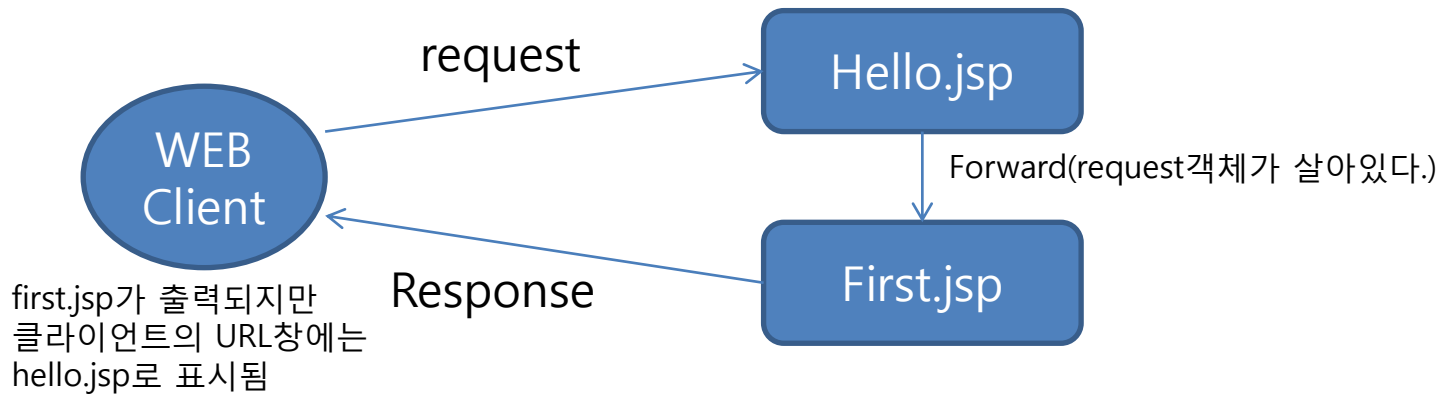
JSP 표준 Action(jsp:include)

- 런타임 중에 동적으로 파일이 포함 된다.
- `<jsp:include page="/jsp/first.jsp" flush="true">`
 - `<jsp:param name="name" value=" 홍길동"/>`
 - `<jsp:param name="age" value="<%=age%>"/>`
- `</jsp:include>`
- flush의 의미는 포함되기 전까지의 응답을 포함되기 직전 클라이언트로 전송한다는 의미. 응답이 조금이라도 전송되면 응답헤더 세팅은 불가능하다.
- 서블릿 또는 자바 빈즈에서 동일한 효과를 내는 코드

```
RequestDispatcher rd = request.getRequestDispatcher("url");
rd.include(request, response);
```

JSP 표준 Action(forward)

- `<jsp:forward page="url">`
 `[<jsp:param name="name" value="value"/>]`
• `</jsp:forward>`



//서블릿 또는 자바 빈즈에서 동일한 효과를 내는 코드
`RequestDispatcher rd = request.getRequestDispatcher("이동할url");`
`rd.forward(request, response);`

forward와 redirect 비교

- **Redirect**

- Request 객체 새로 생성
- 조금 느리다.
- 상단 URL이 바뀜
- 클라이언트에서 서버로 요청을 보내면 서버는 HTTP Protocol 응답 헤더 location 에 Redirection 될 곳의 주소를 넣어서 응답을 보냄. 클라이언트는 Location 값을 읽어 다시 그쪽으로 요청을 보냄, URL 주소도 당연히 바뀜

- **Forward**

- Request 객체 재사용
- 조금 빠르다.
- 상단 URL이 바뀌지 않음
- 컨테이너 안에서 forward 해석을 하므로 forward한 곳의 자원을 이용하여 응답을 만들어 보내게 됨, 결국 URL이 바뀌지 않게 됨

JSP 예외처리

- 예외가 발생하는 JSP에서 직접 처리하면 되지만 소스코드가 복잡해진다.
 - Try {...} catch(Exception e) {...} finally {...}
- Page 지시자의 `errorPage` 속성을 이용한 JSP에서 예외 처리방법
 - 예외가 발생하는 JSP
 - `<%@page errorPage="예외처리JSP경로"%>`
 - 예외를 처리하는 JSP
 - `<%@page isErrorPage="true"%>`
 - exception 내장개체로 예외 종류 식별
 - `<%if(exception instanceof NumberFormatException) {%>`
 - `<%} else if(exception instanceof ArithmeticException) {%>`
 - `<%}%>`

ErrorPage 예제 (numbertest.jsp)

```
<% @ page contentType="text/html; charset=euc-kr" %>
<!-- 예외가 발생했을 경우 이를 처리하는 예러 page를 만들어 본다.
      숫자가 들어가야 할 자리에 문자를 넣거나 두번째 입력항목에 0을 넣어보자. -
->

<html><head> <title> JSP Exception handling </title></head>
><body><center><h3><i>사칙연산 예제</i></h3>
<form method=Get action="calc.jsp">
  OP1: <input type="text" name="op1" > <br>
  OP2 : <input type="text" name="op2" ><br>
  <hr>
  <input type="submit" value="전  송">
  <input type="reset" value="다시 입력">
</form></center></body></html>
```

ErrorPage 예제 (calc.jsp)

```
<% @ page contentType="text/html; charset=euc-kr"
    errorPage="errorHandler.jsp"
%>
<html><head> <title> Result </title></head>
<body>
<%
    int op1= Integer.parseInt(request.getParameter("op1"));
    int op2 = Integer.parseInt(request.getParameter("op2"));
%>
<h4><hr>
덧셈 <%= op1 + op2 %> <br><br>
뺄셈 <%= op1 - op2 %> <br><br>
곱셈 <%= op1 * op2 %> <br><br>
나눗셈 <%= op1 / op2 %> <br><hr>
</h4></body></html>
```

ErrorPage 예제 (errorHandler.jsp)

```
<% @ page contentType="text/html; charset=euc-kr" isErrorPage="true" %>
<html><body><center>
<%   if(exception instanceof NumberFormatException) {   %>
        숫자를 입력하세요.
<%   }else if( exception instanceof ArithmeticException) {   %>
        0으로는 나눌 수 없습니다.
<%   }else {
%>
    <%= exception.toString() %>
<%   }   %>
</body></html>
```

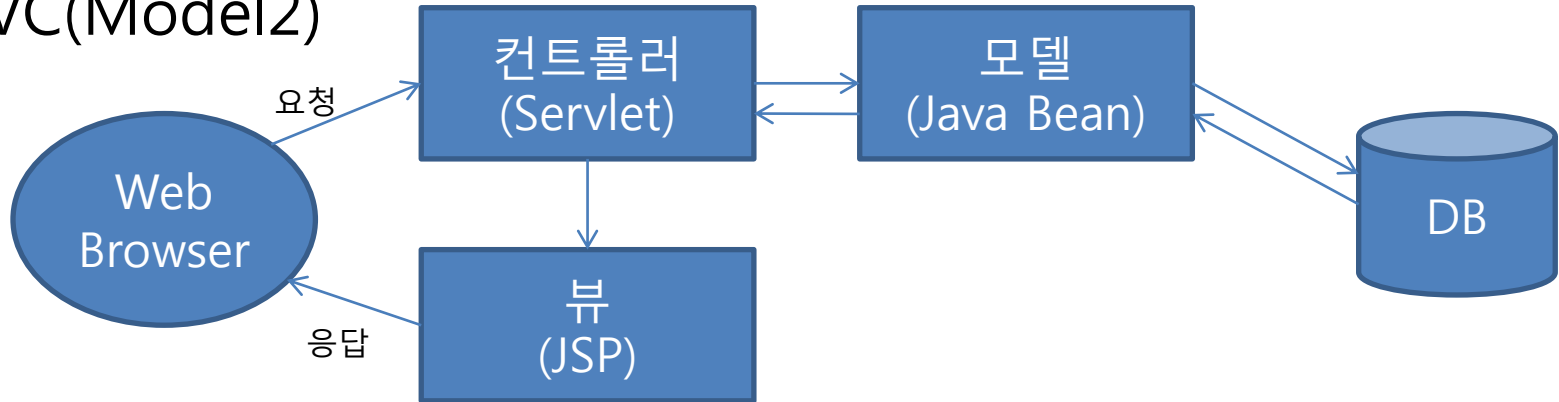
Java Bean

자바 빈 소개

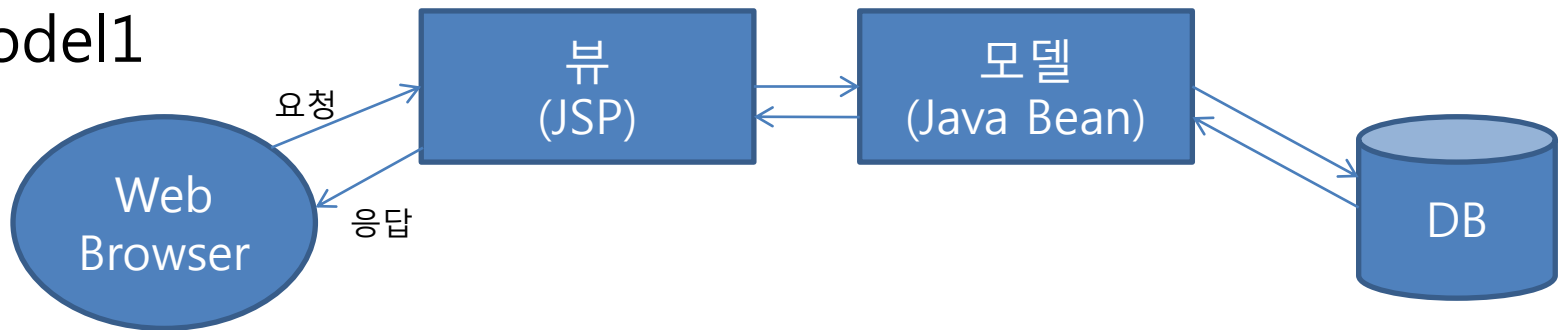
- 자바 빈(Java Bean)이란?
 - 자바빈규약을 따르는 클래스, MVC구조의 Model 영역의 클래스가 해당된다.
 - JSP 페이지의 복잡한 자바코드를 줄이고 재사용성을 높여준다.
 - 자바빈 규약
 - 빈 클래스는 객체직렬화가 가능해야 하며 클래스는 파라미터없는 기본 생성자를 가진다.
 - 멤버변수는 private으로 정의하고 public으로 getter, setter가 있어야 하며 생성자는 매개변수가 없어야 한다.
 - getter는 파라미터가 없어야 하며 setter는 하나이상의 파라미터를 가져야 한다.
 - property가 boolean형인 경우 get 대신 is를 사용한다.
 - 자바 빈 종류
 - Visual Bean : 자바 스윙의 JButton, JTextBox ➔ windows program
 - Non Visual Bean
 - 데이터 저장/DB작업/작업흐름 제어
 - JSP Bean, EJB(Enterprise Java Bean), Spring Framework등에서 사용하는 빈
- 자바 빈즈(Java Beans)?
 - 자바 빈을 설계하기 위한 스펙

Servlet, JSP에서의 자바 빈

MVC(Model2)



Model1



자바 빈 프로퍼티 설계

```
public class BeanName {  
    private datatype property;  
    //getter  
    public datatype getProperty() {  
        return this.propertyName;  
    }  
    //datatype이 boolean인 경우  
    public datatype isProperty() {  
        return this.propertyName;  
    }  
    //setter  
    public void setProperty(datatype propertyName) {  
        this.propertyName = propertyName;  
    }  
}
```


자바 빈과 관련된 JSP 표준 액션

- JSP에서 Java Bean 객체를 생성, setter, getter 호출
- `<jsp:useBean id = "빈이름" class="자바빈클래스이름" scope="범위">`
 `<jsp:setProperty`
 `name="빈변수명"`
 `property="프로퍼티명|*"`
 `[value="value"]`
 `[param="파라미터명"]/>`
- `</jsp:usebean>`
- **`<jsp:useBean id="myBean" class="java.MyBean" scope="page" />`**
 → `java.MyBean myBean= new java.MyBean();` 과 동일.
 `<jsp:useBean id="myBean" class="java1.MyBean" scope="page" >`
 `<jsp:setProperty name=" myBean " property="name" />`
 `</jsp:useBean>`
 → `java.MyBean myBean= new java.MyBean();`
 `myBean.setName(request.getParameter("name"));`
 프로퍼티가 많을 경우에 다 써주는 것보다 property 속성값을 *로 준다.

자바 빈과 관련된 JSP 표준 액션

- <jsp:getProperty

name="빈 변수명"
property="프로퍼티명"/>

<jsp:getProperty name=" myBean " property="name" />

→ myBean.getName();

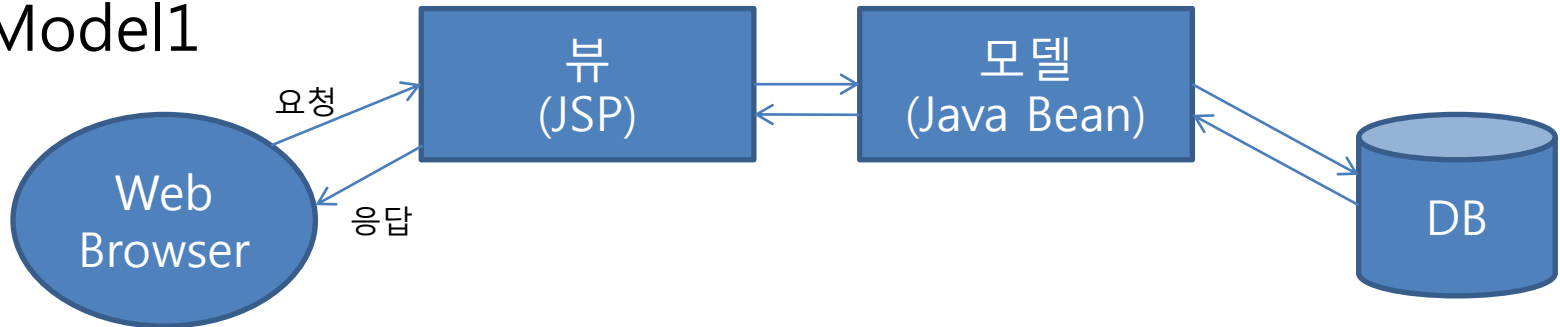
MVC 개발 패턴

MVC 소개

- Model1 방식의 문제점
 - 비즈니스 로직과 프리젠테이션 로직이 JSP에 뒤섞여 있다.
 - 개발자와 디자이너 협업의 어려움
- JSP가 View, Servlet이 Controller, Java Bean이 Model 역할을 해서 프리젠테이션과 비즈니스 로직을 명확하게 구분하는 Model2 방식을 MVC 모델이라고 한다.
- Model
 - 데이터 생산, DB삽입/삭제/저장/ 등의 비즈니스 로직 수행
 - DAO, DTO
- View : 클라이언트에게 보여줄 프리젠테이션에 집중하는 역할
- Controller
 - 클라이언트의 요청을 처리하기 위해 제어 흐름을 담당, 주로 Servlet

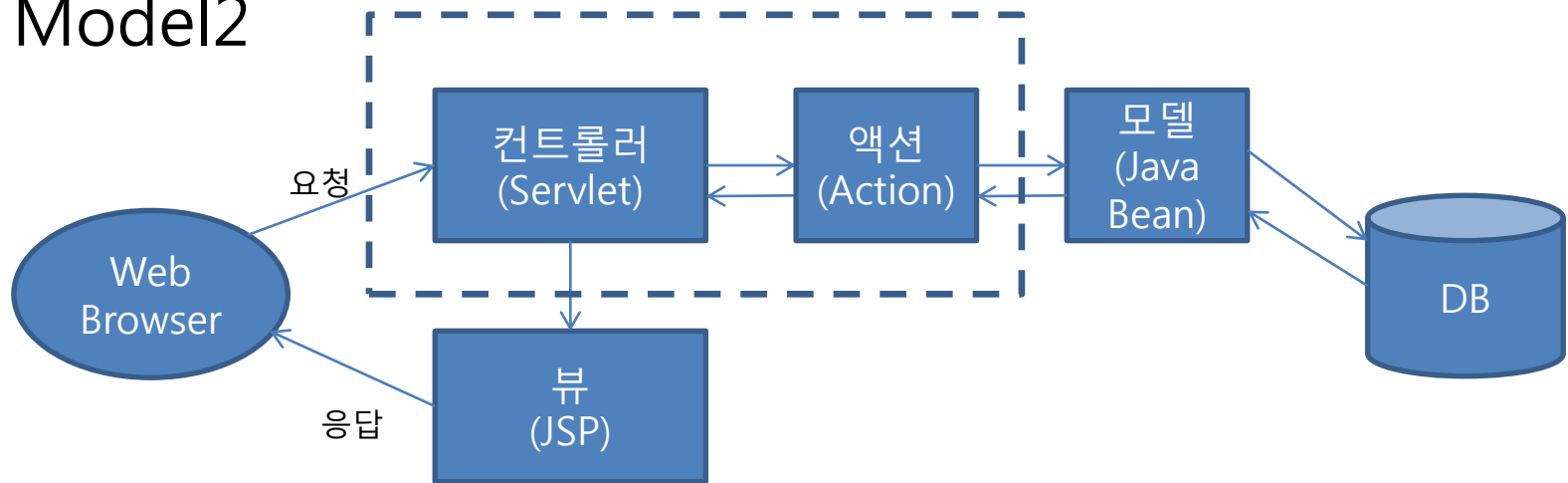
MVC 패턴 종류

- Model1



- View: JSP
- Controller: Bean
- Model: DAO
- 비즈니스 로직의 결과에 따라 뷰를 바꿔주기가 쉽지 않다. ➔ 컨트롤러와 뷰를 분리하기 힘들다. ➔ 뷰의 재 사용성이 떨어진다.
- 구조가 간단하기 때문에 쉽고, 빠르게 개발 가능
- 초기 자바 웹 응용 프로그램 구축에 널리 보급

• Model2



- 컨트롤러(Controller) : 서블릿 or 액션객체
- 모델(Model) : DAO
- 뷰(View) : 주로 JSP
- 모든 클라이언트의 요청은 서블릿이 받고 사용자의 요청에 따라 처리 객체 (Action 개체) 선택한다.
- 각 Action 개체는 Model(DAO)를 이용해서 비즈니스 로직 실행한다.
- 비즈니스 로직 수행 결과를 Action, Controller로 리턴한다.
- JSP를 뷰로 해서 forward 시킨다. JSP에서 프레젠테이션 생성함
- 컨트롤러 설계할 때 많은 주의가 필요하며 프레임워크의 필요성이 증대된다.
- MVC 골격을 프레임워크화 한 것이 스트럿츠 프레임워크(Struts Framework)이다.

EL

EL 소개

- Expression Language. JSP 2.0에 추가되었고 JSTL 및 JSF(Java Server Faces)에서 사용된다.
- JSP에서 객체(자바 빈)의 출력을 단순화 하기 위해 사용되는데, JSP에서는 모든 변수의 생성, 선언을 반드시 표시를 해주어야 되지만 EL은 그러한 과정 없이 바로 접근 가능하다.
- JSP에서 자바빈 데이터에 쉽게 접근하기 위해 정의된 표현식이며, EL 표현식은 항상 중괄호로 묶고 제일 앞에 달러(\$)기호를 붙인다.
- 결국 JSP에서 자바코드를 없애기 위해서 사용하며 JSP 실행 시 EL 인터프리터에 의해 번역되어 실행된다.
- jsp 의 네 기본객체 (page, request, session, application)가 제공하는 scope의 속성사용
- Collection 객체(배열,컬렉션)에 대한 접근 방법 제공하며 수치연산 , 관계 연산, 논리 연산자 제공한다.
- 자바클래스의 메소드에 대한 호출 가능하다.

EL 문법

- `${식}`
 - 기본적으로 request 자원에 접근
 - 식 부분에는 표현 언어가 정의한 문법에 따라 값을 표현하는 식이 온다.
(액션태그 또는 커스텀 태그의 속성 값, 표현식)
 - JSP의 스크립트 요소(스크립트릿, 표현식, 선언부)를 제외한 나머지 부분에서 사용될 수 있다.
- `#{식}`
 - Deferred Expression 이다.
 - JSP 2.1 부터 새로 지원하는 구문으로 JSF(JavaServer Faces)에서 사용되던 표현 언어 구문이다.
 - `${식}`은 표현식이 실행되는 시점에 값을 계산하고 `#{식}`은 실제로 값이 필요한 시점에 계산을 한다.
 - JSP 템플릿 텍스트에서는 사용할 수 없고, 허용되는 태그의 속성에만 위치할 수 있다.

EL 문법

- \${식}의 형태는 Servlet 2.0 이상, #{식} 형태는 Servlet 2.5 이상에서 가능하다.

- **EL 비활성화(web.xml)**

<!-- EL \${식} 비활성화 방법 -->

<jsp-config>

 <jsp-property-group> <url-pattern> /jsp/* </url-pattern> <el-ignored> true </el-ignored>

 </jsp-property-group>

</jsp-config>

<!-- EL #{식} 비활성화 방법 -->

<jsp-config> <jsp-property-group> <url-pattern> /jsp/* </url-pattern>

 <deferred-syntax-allowed-as-literal> true </deferred-syntax-allowed-as-literal>

 </jsp-property-group>

</jsp-config>

- **EL 비활성화(JSP 페이지)**

<!-- EL \${식} 비활성화 -->

<%@ page isELIgnored="true" %>

<!-- EL #{식} 비활성화 -->

<%@ page deferredSyntaxAllowedAsLiteral="true" %>

EL 문법

- **도트(.)연산자: `${first.second}`**
 - first
 - EL의 MAP형 내장 객체, 스코프 객체에 저장된 속성명(객체명)
 - page, request, session, application 순으로 찾음
 - second
 - First가 맵형 내장 개체일 경우, second은 키이름
 - First가 스코프 개체의 속성일 경우, 빈의 프로퍼티 `getSecond()`, `setSecond(...)`
 - customer객체의 name값을 출력하려면 → `$(customer.name)`
- **[] 연산자1: `${first["second"]}`**
 - first
 - EL 맵형 내장 개체
 - 스코프 객체에 저장된 속성명(객체 저장명)
 - 컬렉션의 List형 개체, 배열을 출력할때 주로 사용되며 구문을 실행 도중 null이 나오면 더 이상 진행하지 않고 출력도 하지 않는다.
 - 세션에 저장된 배열 arr의 첫번째 요소를 출력
→ `$(sessionScope.arr[0])` or `$(sessionScope.arr['0'])`
 - second
 - first이 맵 내장 개체일 경우, second은 키명
 - first이 스코프 개체의 속성일 경우, 빈의 프로퍼티
 - first이 List형 개체 또는 배열일 경우 숫자(1 또는 "1")

customer객체의 name값을 출력하려면 → `$(customer["name"])`
<% String[] array = { "1", "2", "3" }; pageContext.setAttribute("array", array); %>
<c:out value='\${array[0]}'/>
- **[] 연산자2: `${first[second]}`**
 - second
 - 스코프 객체에 저장된 속성명(객체 저장명)

EL 내장 개체

- pageContext
 - JSP의 pageContext 내장 개체
 - JSP에서 얻을 수 있는 대부분의 객체들은 이 객체를 통해 접근할 수 있다.
 - servletContext(application), session, request, response
 - `클릭`
 - EL Scope 객체
 - pageScope : page scope의 변수에 접근
 - requestScope : request scope의 변수에 접근
 - sessionScope : session scope의 변수에 접근
 - applicationScope : application scope의 변수에 접근
- `<%=session.getAttribute("name")%>` ➔ `${sessionScope.name}`
`<%=request.getParameter("name")%>` ➔ `${requestScope.name}`

맵형 개체

- 맵형 개체((key,value)의 쌍으로 저장하는 개체)
 - **param** : 요청매개변수의 기본값을 이름으로 저장
Ex) `<%=request.getParameter("name")%>` ➔ `${param.name}`
 - **paramValues** : 요청 파라미터명에 모든 값을 문자열 배열로 받음
 - **Header** : HTTP 요청 헤더의 단일 값을 조회
 - **headerValues** : HTTP 헤더의 값들을 문자열 배열로 받음
 - **Cookie** : 요청에서 쿠키 값을 조회한다.(`${cookie.이름.value}`)
 - **initParam** : web.xml등에서 설정한 컨텍스트의 초기화 파라미터를 조회

EL 예제(1) – MyBean.java

```
package java1;
public class MyBean {
    String name;
    String age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAge() {
        return age;
    }
    public void setAge(String age) {
        this.age = age;
    }
}
```

EL 예제(1) – ELTEST.jsp

```
<% @ page language="java" contentType="text/html; charset=EUC-KR"
    import="java1.MyBean"%>
<!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>EL TEST</title></head>
<body>
<%
    MyBean myBean = new MyBean();
    myBean.setName("홍길동");
    myBean.setAge("12");
    //session.setAttribute("myBean", myBean);
%>
name : ${myBean.name} <br>   age : ${myBean.age} <br>
</body></html>
```

EL 예제(2) – input.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>
<title>EL Example1</title>
</head>
<body>
    <form action="result.jsp" method="post">
        좋아하는 과일을 선택하세요<br>
        <br> 이름 : <input type="text" name=name /> <br> <input
            type="checkbox" name="fruits" value="사과">사과 <input
            type="checkbox" name="fruits" value="딸기">딸기 <input
            type="checkbox" name="fruits" value="참외">참외 <input
            type="checkbox" name="fruits" value="수박">수박 <input
            type="checkbox" name="fruits" value="귤">귤 <br> <br>
        <input type="submit"> <input type="reset">
    </form>
</body>
</html>
```


EL 예제(2) – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">
  <context-param>
    <description> 초기파라미터1 </description>
    <param-name>was</param-name>
    <param-value>tomcat</param-value>
  </context-param>

  <context-param>
    <description> 초기파라미터2 </description>
    <param-name>version</param-name>
    <param-value>8.0</param-value>
  </context-param>
</web-app>
```

EL 예제(2) – result.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%
    request.setCharacterEncoding("UTF-8");
    request.setAttribute("name2", "홍길동2");
%>
<html>
<head>
<title>표현 언어</title>
</head>
<body>
    WAS : ${ initParam.was }
    <br> VERSION : ${ initParam.version }
    <hr>
    요청 URI : ${ pageContext.request.requestURI }
    <hr>
    이름 : ${ param.name }
    <br> 이름2 : ${ requestScope.name2 }
    <hr>
    선택한 과일 : ${ paramValues.fruits[0] } ${ paramValues.fruits[1] } ${ paramValues.fruits[2] }
    ${ paramValues.fruits[3] } ${ paramValues.fruits[4] }

</body>
</html>
```

WAS : tomcat
VERSION : 8.0
요청 URI : ojcweb/jsp/result.jsp
이름 : 홍길동
이름2 : 홍길동2
선택한 과일 : 사과 딸기

EL 예제(3) – ELCookie.jsp

```
<% @ page language="java" contentType="text/html; charset=EUC-KR" %>
<!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>EL Cookie Test</title></head>
<body>
<%
    String key = "city";
    String value = "JEJU";
    Cookie cookie = new Cookie(key, value);
    response.addCookie(cookie);
%>
    name : ${cookie["city"].value} <br>
</body></html>
```

EL 예제 (4) – Name.java

```
package java1;
public class Name {
    private String firstName, lastName;
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

EL 예제(4) – Address.java

```
package java1;
public class Address {
    private String city;
    private int zip;
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public int getZip() {
        return zip;
    }
    public void setZip(int zip) {
        this.zip = zip;
    }
}
```

EL 예제(4) – Profile.java

```
package java1;
public class Profile {
    private Name name;
    private Address address;
    public void setName(Name name) {
        this.name = name;
    }
    public Name getName() {
        return name;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
    public Address getAddress() {
        return address;
    }
}
```

EL 예제(4) – profile.jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<% @ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html> <head> <title>EL Example(4)</title> </head>
<body>
  <!--page scope에 빈 생성 -->
  <jsp:useBean id='name' class='java1.Name'>
    <jsp:setProperty name='name' property='firstName' value='길동' />
    <jsp:setProperty name='name' property='lastName' value='홍' />
  </jsp:useBean>
  <!--page scope에 빈 생성 -->
  <jsp:useBean id='address' class='java1.Address'>
    <jsp:setProperty name='address' property='city' value='SEOUL' />
    <jsp:setProperty name='address' property='zip' value='135050' />
  </jsp:useBean>
```

jstl1.2.jar 필요

다운로드 URL

<http://download.java.net/maven/1/jstl/jars/>

<http://mvnrepository.com/artifact/javax.servlet/jstl/1.2>

EL 예제(4) – profile.jsp

```
<!--page scope에 빈 생성 -->
<jsp:useBean id='profile' class='java1.Profile'>
  <jsp:setProperty name='profile'
    property='name'
    value='${pageScope.name}'/>


  <jsp:setProperty name='profile'
    property='address'
    value='pageScope.address'/>
</jsp:useBean>
Profile :
<c:out value='${name["firstName"]}'/>
<c:out value='${pageScope.profile.name.lastName}'/>:
<p>
```


EL 예제(4) – profile.jsp

```
<table>    <tr>
    <td>First Name:</td>
    <td><c:out value='${profile["name"].firstName}'/></td>
</tr><tr>
    <td>Last Name:
    <td><c:out value='${profile.name["lastName"]}'/></td>
</tr>

    <tr>

    <td>City:
    <td><c:out value='${profile.address.city}'/></td>
</tr>
<tr>
    <td>Zip Code:
    <td><c:out value='${profile.address.zip}'/></td>
</tr>
</table>  </body></html>
```



Profile : 길동 홍:
First Name:길동
Last Name:홍
City: SEOUL
Zip Code: 135050

EL 연산자

- 산술 연산자
 - +, -, *, / and div, %, mod
- 논리 연산자
 - and, &&, or, ||, not, !
- 관계 연산자
 - ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le.
- 삼항 연산자 : A ? B : C (A가 참이면 B, 거짓이면 C가 리턴)
- Empty : 값이 NULL일 경우 true를 리턴
- () : 연산 우선순위 지정
- . : 클래스 또는 맵 형 객체에 접근
- [] : 배열 또는 리스트 형 객체에 접근

EL 연산자

`${1 + 2}`

3

`${"70" + 80}`

150, 지동 형 변환

`${null + 5}`

5 (null은 0으로 치환됨)

`${10 mod 3}`

1

`${2 < 3}`

true

`${2 gt 3}`

false

`${3.0 le 3.01}`

true

`${(6 > 3) ? 3 : 2}`

3

`${someValue == '2016'}`

True (EL의 문자비교는 == 도 가능)

`${empty someValue2}`

true (someValue2가 null이라면)

`${header["host"]}`

localhost:8080

`${header["user-agent"]}`

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; InfoPath.2)

JSTL

JSTL 소개

- JSP Standard Tag Library
 - JSP에서 사용할 수 있는 표준 태그 라이브러리
 - JCP(Java community Process)에서 스펙을 정의
- Tag Libaray?
 - 반복적인 프리젠테이션 로직을 XML 태그로 캡슐화한 것
 - JSP 컨테이너에 의해 컴파일 시 자바 코드로 변환되어 실행됨
 - JSP에서 자바 코드를 없애기 위해 사용됨

JSTL 구현체 다운로드

- 다운로드

- **Standard Taglib**

- <http://tomcat.apache.org/taglibs/standard/>

jstl-impl-1.2.jar, jstl-api-1.2.jar 두 파일을 라이브러리에 추가

- Standard-1.1: JSTL 1.1

- Servlet 2.4, JSP 2.0 이상
 - Tomcat 5 이상

- Standard-1.2 : JSTL 1.2

- Servlet 2.5, JSP 2.1 이상
 - Tomcat 7 이상

JSTL 기능별 분류

Area	Function	URI	Prefix
Core	<ul style="list-style-type: none">- 변수 지원- 흐름 제어- URL 관리	http://java.sun.com/jsp/jstl/core	c
XML	<ul style="list-style-type: none">- XML Core- 흐름 제어- XML 변환	http://java.sun.com/jsp/jstl/xml	x
Database	<ul style="list-style-type: none">- 데이터베이스	http://java.sun.com/jsp/jstl/sql	sql
Functions	<ul style="list-style-type: none">- 컬렉션- 특수문자처리- 컬렉션 길이- String 조작	http://java.sun.com/jsp/jstl/functions	fn

@ taglib 지시자

- JSP에서 JSTL을 사용하기 위해 선언
- <%@ taglib
 uri="태그정의TLD위치 및 태그URI"
 prefix="태그접두사"%>
 - <%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c"%>
 - <%@ taglib uri="<http://java.sun.com/jsp/jstl/xml>" prefix="x"%>
 - <%@ taglib uri="<http://java.sun.com/jsp/jstl/fmt>" prefix="fmt"%>
 - <%@ taglib uri="<http://java.sun.com/jsp/jstl/sql>" prefix="sql"%>
 - <%@ taglib uri="<http://java.sun.com/jsp/jstl/functions>" prefix="fn"%>

JSTL을 기능별로 분류(C 코어태그)

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

변수 지원	set	변수 설정.
	remove	변수 제거.
흐름 제어	if	조건 분기.
	choose	다중 조건을 처리.
	forEach	컬렉션이나 Map의 각 반복 처리시 사용.
	forEachTokens	구분자(Token)로 분리된 각각의 토큰을 처리할 때 사용한다.
URL 처리	import	URL을 사용하여 다른 리소스의 결과를 삽입.
	redirect	기술한 경로로 리다이렉트.
	url	URL을 재작성.
기타 태그	catch	예외를 받아낼 때 사용.
	out	출력.

JSTL CORE 태그(c:set, c:remove)

- EL 변수 생성 및 제거에 사용된다.
 - `<c:set var="변수명" value="값" [scope="영역"] />`
 - `<c:set var="변수명" [scope="영역"]>값</c:set>`
 - `<c:remove var="변수명" scope="영역"/>`
→ 영역을 지정하지 않으면 해당 이름의 변수를 모두 제거한다.
- value 속성 사용
- ```
<c:set var="ename" value="SMITH" scope="page" />
<c:set var="ename" value="<%=e.getName() %>" scope="request"/>
<c:set var="ename" value="${ e.name }${ e.job }" scope="request" />
```
- 태그의 몸체를 값으로 사용
- ```
<c:set var="ename">SMITH</c:set>
<c:set var="ename"><%= e.getName() %> <%=m.getJob() %> </c:set>
<c:set var="ename">${ e.name } ${ e.job }</c:set>
```
- 변수 제거
- ```
<c:remove var="ename"/>
```

# JSTL CORE 태그(c:if, c:choose)

- `<c:if>`
  - 프로그래밍 언어의 IF문의 역할과 동일하다.
  - `<c:if test="조건"> </c:if>` → 조건식은 참, 거짓을 리턴하는 조건문이 온다.
  - `<c:if test="조건" var="결과를 저장할 변수"> </c:if>` → 조건식 결과를 지정한 변수에 저장
- `<c:choose>`
  - 프로그래밍 언어의 switch문이다.
  - `<c:choose>`
    - `<c:when test="조건1"> ... </c:when>`
    - `<c:when test="조건1"> ... </c:when>`
    - .....
    - `<c:otherwise> ... </c:otherwise>`
  - `</c:choose>`

# JSTL CORE 태그(c:if, c:choose 예제)

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <head> <title>EL IF, CHOOSE</title> </head>
<body>
 <c:if test="${pageContext.request.method=='POST'}">Ok, Your choice is
 <c:out value="${param.enter}" />

 <c:choose>
 <c:when test="${param.enter=='1'}">JAVA
 </c:when>
 <c:when test="${param.enter=='2'}">ORACLE
 </c:when>
 <c:when test="${param.enter=='3'}">C#.NET
 </c:when>
 <c:otherwise>C Language
 </c:otherwise>
 </c:choose>
 </c:if>
 <form method="post">Enter a number between 1 and 5:
 <input type="text" name="enter" />
 <input type="submit" value="SUBMIT" />

 </form>
</body> </html>
```

# JSTL CORE 태그(c:forEach)

- 자바의 for 반복문과 유사하다.
- 배열, 컬렉션, Map의 값들을 탐색할 때 유용하다.
- `<c:forEach var="변수" items="배열|컬렉션|맵" begin="시작값" end="종료값" step="증분값"> ${ 변수 } </c:forEach>`
- 2~9단까지 구구단 출력

```
<c:forEach var="i" begin="2" end="9">
 <c:forEach var="j" begin="1" end="9">
 ${ i } * ${ j } = ${ i * j }

 </c:forEach>

</c:forEach>
```

# JSTL CORE 태그(c:forEach)

- 1부터100까지의 수중 홀수의 합 출력

```
<c:set var="sum" value="0" />
```

```
<c:forEach var="i" begin="1" end="100" step="2">
```

```
 <c:set var="sum" value="${ sum + i }" />
```

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

```
${ sum }
```

# JSTL CORE 태그(c:forEach 예제)

```
<% @ page language="java" contentType="text/html; charset=EUC-KR"
 pageEncoding="EUC-KR" import="java.util.*" %>
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!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=EUC-KR">
<title>JSTL forEach Sample</title>
</head>
<body>
<%
 ArrayList carList = new ArrayList();
 carList.add("sonata");
 carList.add("ford");
 carList.add("avante"); carList.add("grandure");
 request.setAttribute("carList", carList);
%>
```

# JSTL CORE 태그(c:forEach 예제)

```
<c:forEach var="car" items="${carList}" varStatus="loop">
 <c:if test = "${loop.first}">
 <table border="1" cellspacing="0">
 <tr bgcolor="yellow">
 <td>순번</td>
 <td>차종</td>
 </tr>
 </table>
 </c:if>
 <tr>
 <td>${loop.count}</td>
 <td>${car}</td>
 </tr>
 <c:if test = "${loop.last}">
 </table>
 </c:if>
</c:forEach>
</body>
</html>
```



# JSTL CORE 태그(c:forTokens)

- 자바 Tokenizer 클래스와 유사한 기능을 한다.
- `<c:forTokens var="변수" items="문자열" delims="델리미터" varStatus="루프상태변수">...</c:forTokens>`
- 루프상태변수는 루프의 상태를 나타내는 변수이며 first, last, count 속성 등이 있다.

[예]

```
<c:forTokens var="car" items="소나타,포드,봉고,트럭,그렌저"
delims=",">
 ${ car }
</c:forTokens>
```

# JSTL CORE 태그(c:import)

- 지정한 URL의 결과를 현 위치에 삽입하거나 변수에 저장할 때 사용
- 동일한 웹 어플리케이션 뿐 아니라 외부의 URL도 가능하다.
- var 속성이 있으면 URL의 결과를 변수에 저장하며 없는 경우 현 위치에 URL의 실행결과를 삽입한다.
- 동일한 서블릿 컨테이너의 다른 웹 응용프로그램(컨텍스트)의 URL 인 경우 context속성에 해당하는 컨텍스트 명을 기술하면 된다.

```
<c:import url="URL"
 [context="다른컨텍스트명"]
 [var="변수"]
 [scope="{page|request|session|application}"]>
 [<c:param name="name" value="value"/>]
</c:import>
```

# JSTL CORE 태그(c:redirect)

- 서블릿의 redirect처럼 지정한 페이지로 리다이렉트 시킨다.
- URL 속성 값이 슬래시("/")로 시작하는 경우 컨텍스트 경로를 포함한다.
- 다른 컨텍스트 경로로 리다이렉트 하는 경우 context 속성에 해당 컨텍스트 경로를 표시한다.
- <c:redirect 이후의 태그는 실행되지 않는다.
- <c:param> 태그를 이용해서 리다이렉트 되는 URL에 파라미터를 추가할 수 있다.

**<c:redirect url="리다이렉트 될 URL" context="context"]>**

**[<c:param name="name" value="value"/>]**

**</c:redirect>**

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

<c:redirect url="http://search.daum.net/search">

  <c:param name="w" value="blog" />

  <c:param name="q" value="오라클자바커뮤니티학원" />

</c:redirect>

# JSTL CORE 태그(c:url)

- URL을 생성해 준다.
- var, scope 속성은 생략 가능한데 var 속성이 지정되지 않으면 현재 위치에 생성한 URL을 출력하고 var 값이 지정되면 해당 변수에 URL을 할당한다.
- <c:param> 태그를 이용해서 URL에 파라미터를 추가할 수 있다.
- value 속성은 절대경로 또는 상대경로 또는 현재 JSP에 대한 상대경로를 기준으로 입력할 수 있다.

```
<c:url value="url"
 [context="context"]
 [var="varName"]
 [scope="{page|request|session|application}"]>
 [<c:param name="name" value="value"/>]
</c:url>
```

# JSTL CORE 태그(c:url 예제)

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <head> <title>URL Example</title> </head>
<body>
- 현재 컨텍스트명이 javaweb
- 클릭시 http://localhost:8080/javaweb/jsp/first.jsp 로 이동
<a href="<c:url value='/jsp/first.jsp'/>">JSP

- 클릭시 컨텍스트 루트(http://localhost:8080/javaweb/)로 이동.
<a href="<c:url value='/' />">Context Root(/)
</body>
</html>
```

# JSTL 기타 CORE 태그(c:out)

- JSP에서 JspWriter에 값을 출력할 때 사용한다.
- 출력하고자 하는 값은 value 속성에 지정한다.
- Value 속성에 값을 지정하지 않는 경우 default 속성을 이용하여 값을 출력할 수도 있다.
- escapeXml 속성이 true일 경우 아래에 해당하는 문자를 이스케이프 시퀀스로 변경한다. 생략할 경우 기본값은 true이다.
- `<` → `&lt;`, `>` → `&gt;`, `&` → `&amp;`, `'` → `&#039;`
- `"` → `&#034;`

```
<c:out value="${..}"
 [escapeXml="{true|false}"]
 [default="defaultValue"]>
 [default value]
</c:out>
```

# JSTL 기타 CORE 태그(c:catch)

- 예외발생시 발생한 오류를 EL 변수에 저장할 때 사용한다.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%!
 int num1 = 10; int num2 = 0;
%>
 <c:catch var="errorMsg">
 <%
 int res = num1 / num2;
 out.println(res);
 %>
 </c:catch>
 <c:if test="${errorMsg != null}">
 <p>${errorMsg}</p>
 </c:if>
```