

Java Web & JDBC & AJAX



목차

- 웹 응용 프로그램 개요
- Servlet 프로그래밍
- JSP 프로그래밍
- 고급 JDBC 프로그래밍
- Filter
- Java Bean
- MVC 개발 패턴
- EL
- JSTL
- AJAX 프로그래밍
- CVS를 이용한 공동 개발
- 웹 응용 프로그램 개발 실습

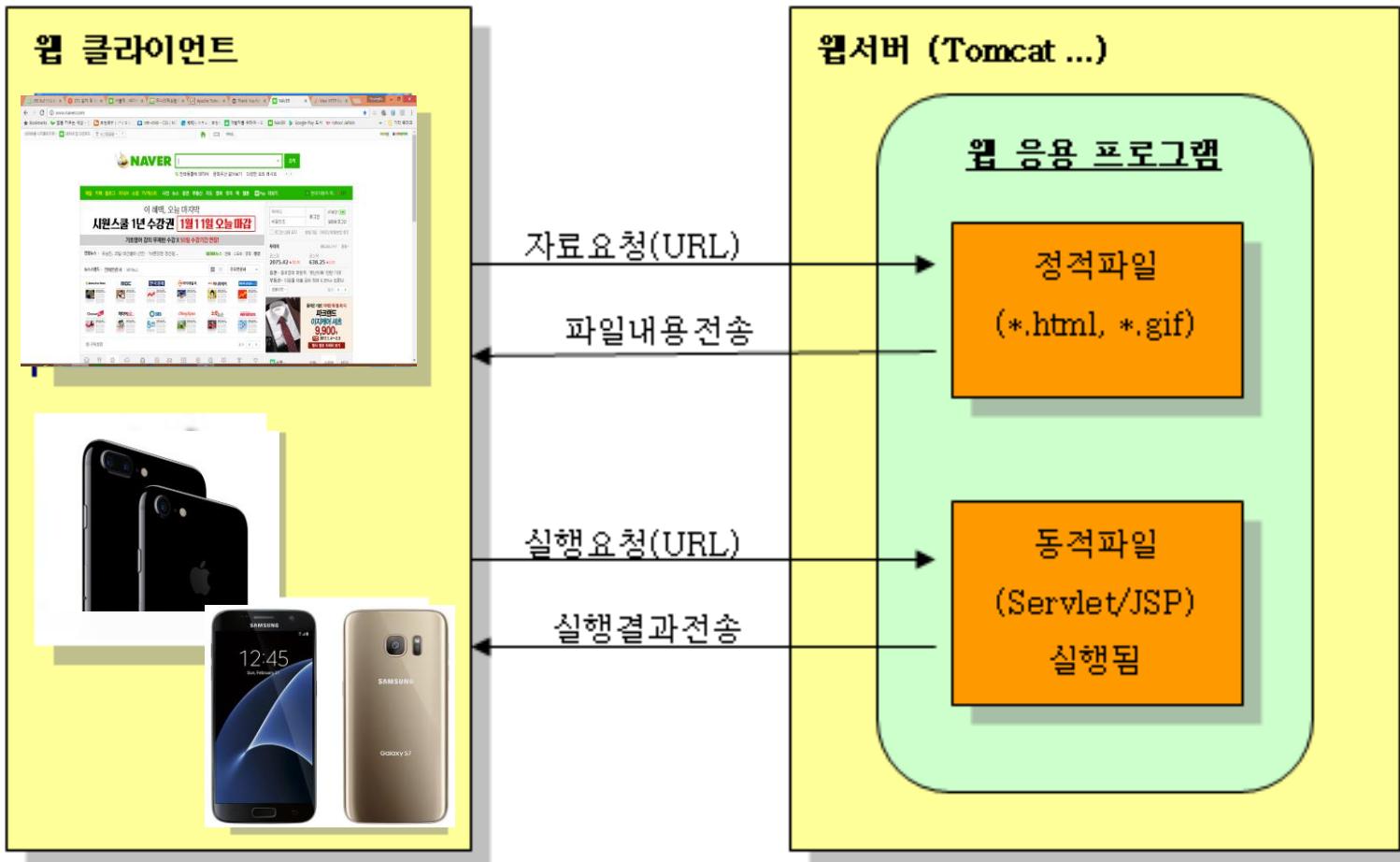
웹 응용 프로그램 개요

웹(Web)

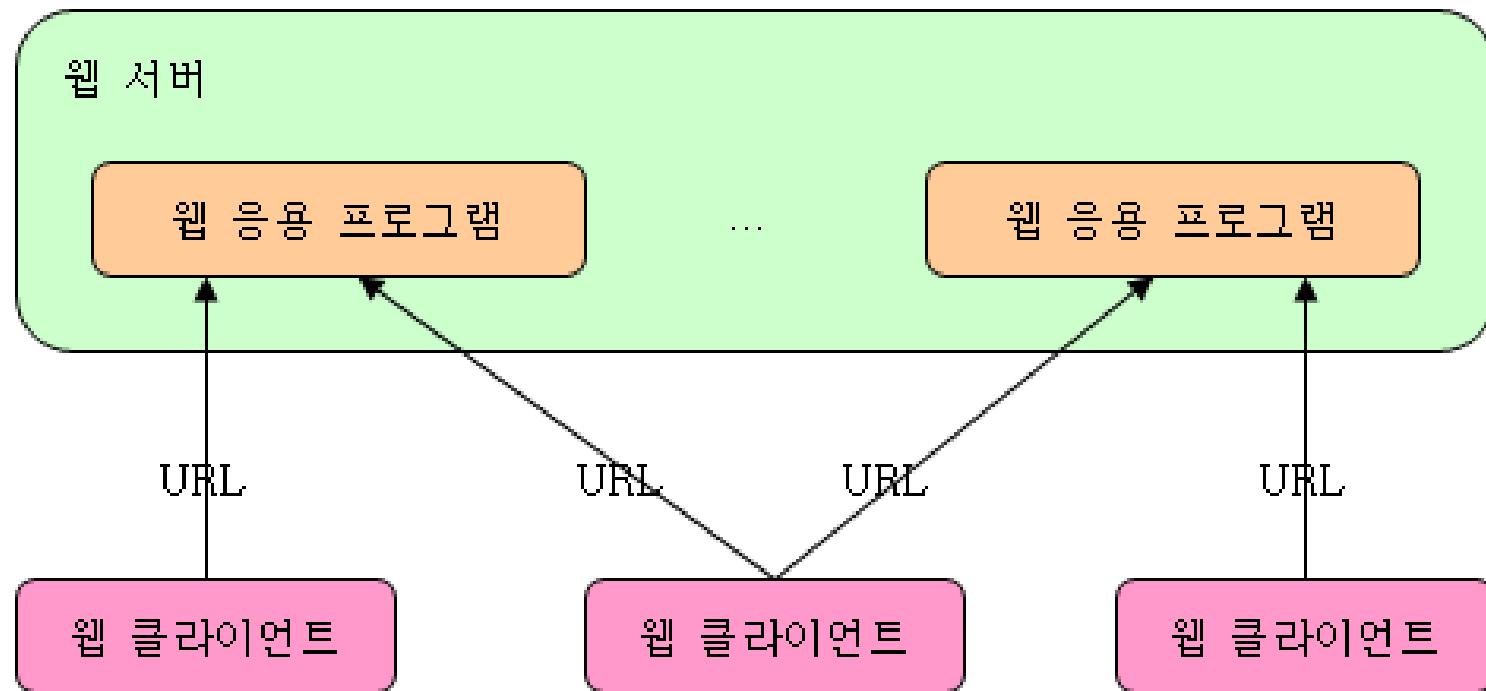
- 여러 컴퓨터에 분산되어 있는 자료를 인터넷을 통해서 쉽게 이용할 수 있도록 지원하는 정보 시스템
- 웹 서버와 웹 클라이언트로 구성
 - 웹 서버
 - 자료를 웹 클라이언트에게 제공하는 역할
 - IIS, Apache, Tomcat, Jeus, Weblogic, Websphere
 - 웹 클라이언트
 - 웹 서버의 자료를 받아 다양한 형태로 보여주는 역할
 - 웹 브라우저, 휴대폰, 스마트폰, HTTP를 사용하는 모든 장치

웹 응용 프로그램

- 웹 서버에서 실행되는 응용 프로그램이다.
- 클라이언트가 요청하는 자료를 가지고 있다.
- 정적파일과 동적파일로 구성된다.
 - 정적파일
 - HTML 파일 및 멀티미디어 파일
 - 동적파일
 - 실행이 필요한 코드를 가지고 있는 파일
 - Servlet, JSP, ASP, PHP, ASPX 파일



- 하나의 웹 서버에는 여러 개의 웹 응용 프로그램들이 존재할 수 있다.



웹 사이트

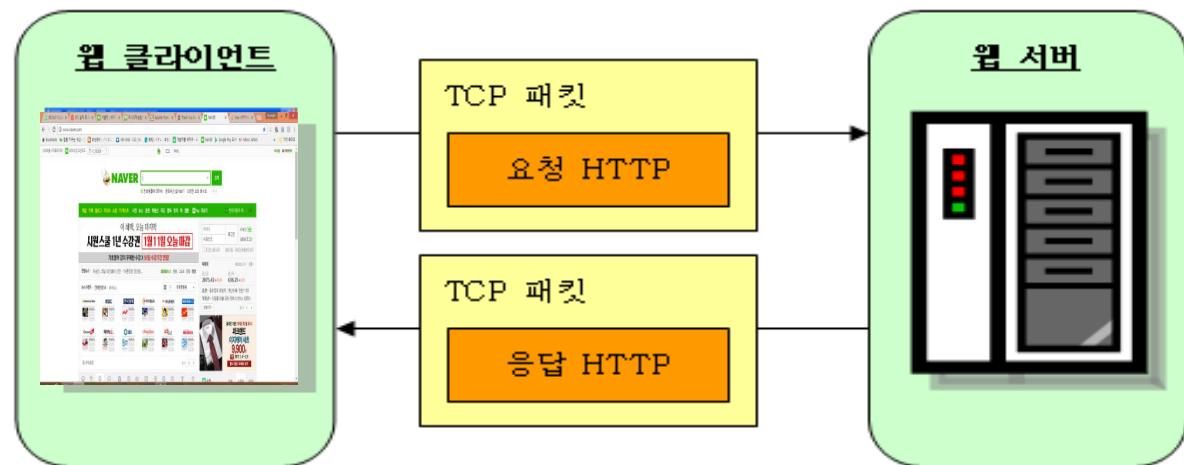
- 웹 사이트는 웹 서버의 위치와 관련된 용어이고, 웹 응용 프로그램은 웹 사이트에서 이용 가능한 기능과 관련된 용어라고 볼 수 있다.
- 하나의 웹 응용 프로그램으로 구성될 수도 있고 여러 개의 웹 응용 프로그램으로 구성될 수도 있다.
- 은행 웹 사이트를 예로 들어보자. 웹 사용자는 검색 사이트를 이용해서 은행의 웹 사이트를 주소를 찾는다. 은행의 웹 사이트 주소는 곧 은행의 웹 서버의 주소이다. 웹 사용자는 은행의 홈 페이지를 방문하고, 원하는 업무를 보기 위해 메뉴를 클릭하거나, 이미지를 클릭한다. 은행의 업무는 예금 관련 업무, 카드 관련 업무, 투자 관련 업무 등 여러가지가 있다. 메뉴를 클릭하면 해당 업무에 해당하는 웹 응용 프로그램이 실행된다. 각각의 업무는 각각의 웹 응용 프로그램으로 개발된다.

URL

- Uniform Resource Locator
- 웹 클라이언트가 웹 응용 프로그램의 자료를 요청하기 위해 사용
- [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 Transfer 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



요청 HTTP 구조

[요청방식]	[요청 자료의 경로]	[HTTP 버전]
헤더명: 헤더값		
헤더명: 헤더값		
:		
전달 데이터 (옵션)		



```
GET /website1/default.aspx HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg
Accept-Language: ko
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: 127.0.0.1:80
Connection: Keep-Alive
```

응답 HTTP 구조

[HTTP버전] [응답코드] [간단한 설명]

헤더명: 헤더값

헤더명: 헤더값

:

전달 데이터 (옵션)

HTTP/1.1 200 OK

Server: ASP.NET Development Server/8.0.0.0

Date: Tue, 25 Apr 2006 08:48:29 GMT

X-AspNet-Version: 2.0.50727

Cache-Control: private

Content-Type: text/html; charset=utf-8

Content-Length: 21552

Connection: Close

```
<html>
<head >
    <title>website1</title>
</head>
<body>
    ~
</body>
</html>
```

웹 응용 프로그램 폴더 구조

- 웹 응용 프로그램

- 정적 및 JSP 파일

- 폴더

- 정적 및 JSP 파일

- WEB-INF

- classes

- *.class

- lib

- *.jar

- web.xml

클라이언트에서 URL로 접근 가능

클라이언트에서 URL로 접근 불가

Deployment Descriptor(DD)

웹 응용 프로그램 생성 및 테스트

- JDK 설치
- Tomcat 설치
- Eclipse 설치
- 웹 응용 프로그램 생성

Tomcat 8.X 설치하기

- <http://tomcat.apache.org/download-80.cgi>에서 32-bit/64-bit Windows zip을 다운 받는다.
- 설치 폴더를 c:\java\Tomcat 8로 바꾸어서 압축을 푼다.

Eclipse Neon.2 설치

- <http://www.eclipse.org/downloads/eclipse-packages/> 에서 eclipse-jee-neon-2-win32.zip 를 다운 받는다.
- 다운 받은 eclipse-jee-neon-2-win32.zip 파일을 적당한 곳에 압축을 푸다.
- 편의상 C:\java에 두고 그 아래에 압축을 풀었으며, 압축이 풀린 폴더명을 eclipse로 수정 했다면, 다음과 같이 (C:\java\eclipse) 이클립스가 설치 되어 있다.

Tomcat과 Eclipse와의 연동 -환경설정

1. Preferences > Server > Runtime Environments
2. Add... > Apache Tomcat v8.5
3. Tomcat installation directory Browse...
4. 설치한 Tomcat폴더를 찾아 추가한다

Tomcat과 Eclipse와의 연동 -서버추가

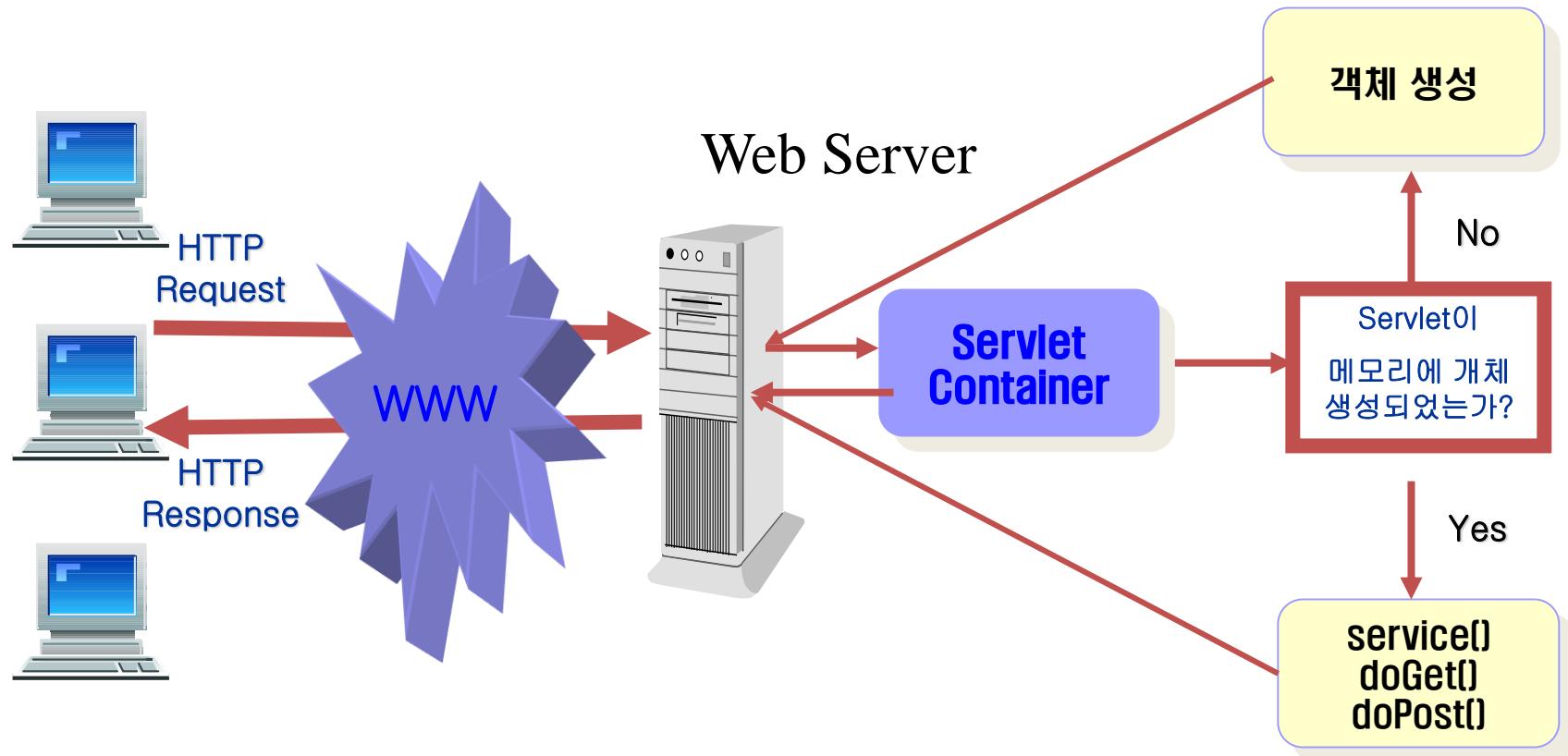
1. File > New > Other
2. Server 폴더 > Server
3. Add and Remove... -> 프로젝트 등록
4. 서버 시작

Servlet 프로그래밍

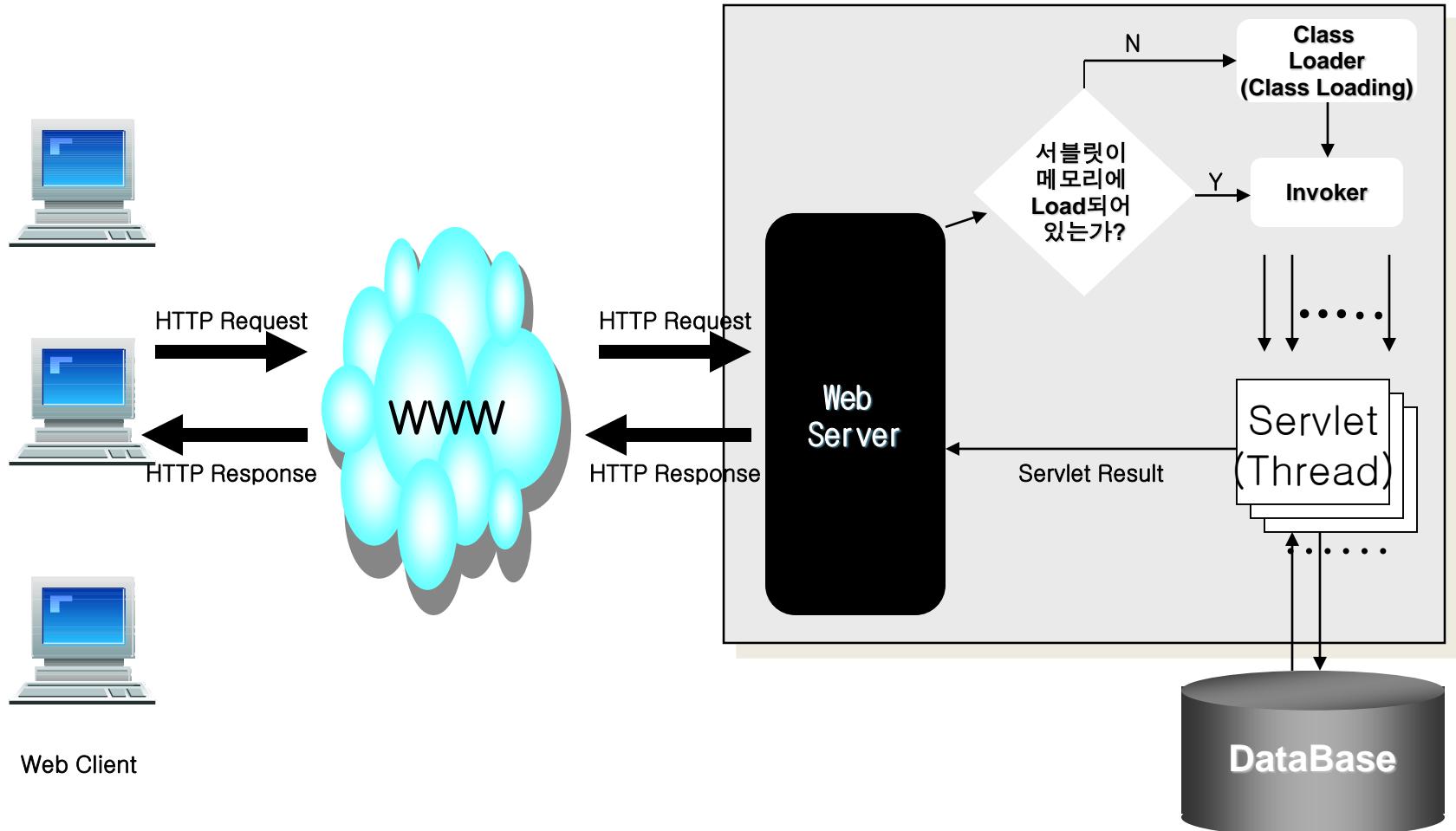
Servlet 소개

- 동적 파일이다.
- 하나의 Java Class이다.
- 서버에서 객체 생성후 사용된다.
- URL로 클라이언트가 실행 요청한다.
- 특정 메소드를 호출하여 응답을 생성한다.

Servlet life cycle(1)



Servlet life cycle(2)



Servlet 샘플

- //HelloWorld.java
- package com.oraclejava;
- extends HttpServlet
- doGet() 메서드 오버라이딩
- import java.io.IOException;
- import java.io.PrintWriter;
- import javax.servlet.ServletException;
- import javax.servlet.annotation.WebServlet;
- import javax.servlet.http.HttpServlet;
- import javax.servlet.http.HttpServletRequest;
- import javax.servlet.http.HttpServletResponse;
- ```
/**
```
- \* Servlet implementation class HelloWorld
- \*/
- @WebServlet("/HelloWorld")
- public class HelloWorld extends HttpServlet {
- private static final long serialVersionUID = 1L;
- ```
/**
```
- * @see HttpServlet#HttpServlet()
- */
- public HelloWorld() {
- super();
- }

Servlet 샘플

```
• /**
•      * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
•      */
•      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
•          response.setContentType("text/html; charset=utf-8");
•
•          PrintWriter out = response.getWriter();
•
•          out.println("<html>");
•          out.println("<h1>Hello World 안녕하세요</h1>");
•          out.println("</html>");
•
•      }
• }
```

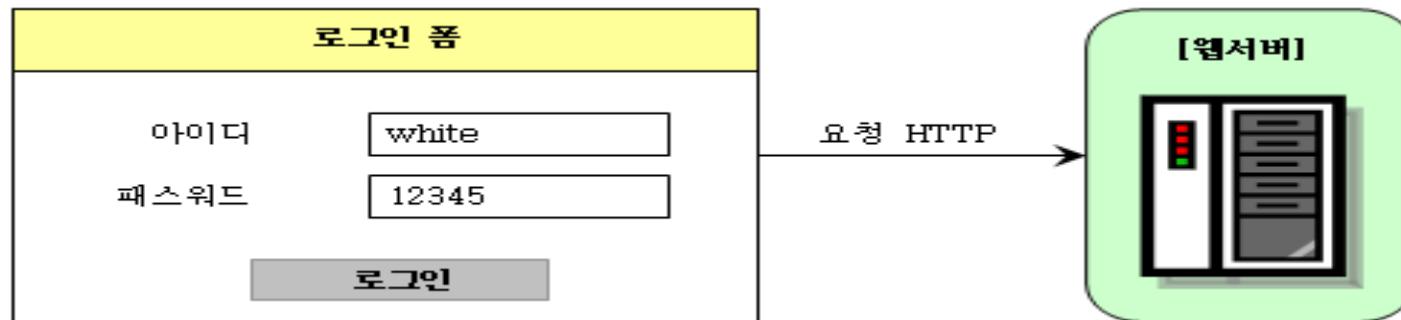
요청 방식

- 웹 클라이언트가 웹 서버로 자료를 요청할 때, 사용하는 방식
- GET 방식
 - 요청 HTTP 시작라인의 URL에 입력 데이터가 포함된다.
 - 요청 HTTP의 body가 없다.
 - 전송하는 자료량에 한계가 있다.
 - 주소 표시줄에 표시되므로 보안성이 Post 방식보다 약하다.
 - 문자만 전송가능하다.

```
<a href="http://localhost:8080/login?id=white&password=12345" > ~ </a>  
<form method="get" action="http://localhost:8080/login"> ~ </form>
```

- POST 방식
 - 요청 HTTP body에 입력 데이터가 포함된다.
 - body의 데이터는 스트림으로 전송된다.
 - 스트림 형태로 입력 데이터가 전송되므로 전송 용량에 제한을 받지 않는다.
 - 문자 및 바이너리(파일)를 전송할 수 있다.

```
<form method="post" action="http://localhost:8080/login"> ~ </form>
```



GET /JavaWebProject/login?id=white&password=12345 HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg

Accept-Language: ko

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0

Host: 127.0.0.1:80

Connection: Keep-Alive

POST /JavaWebProject/login HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg

Accept-Language: ko

Accept-Encoding: gzip, deflate

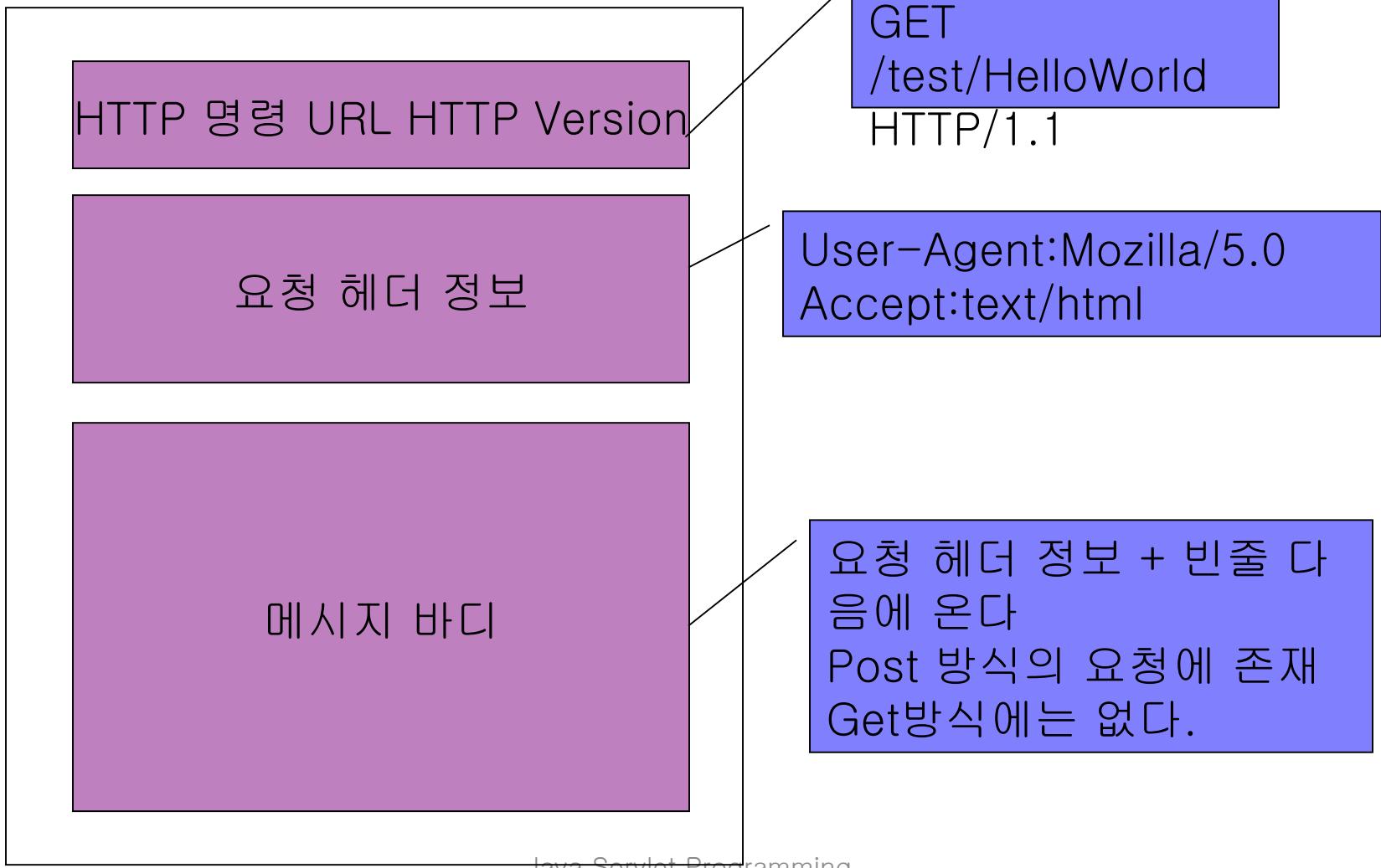
User-Agent: Mozilla/4.0

Host: 127.0.0.1:80

Connection: Keep-Alive

id=white&password=12345

요청 구조



요청 구조(예)

GET /test>HelloWorld HTTP/1.1

(/test>HelloWorld을 요청, 프로토콜 정보)

User-Agent: Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36

(클라이언트 환경정보)

Accept:

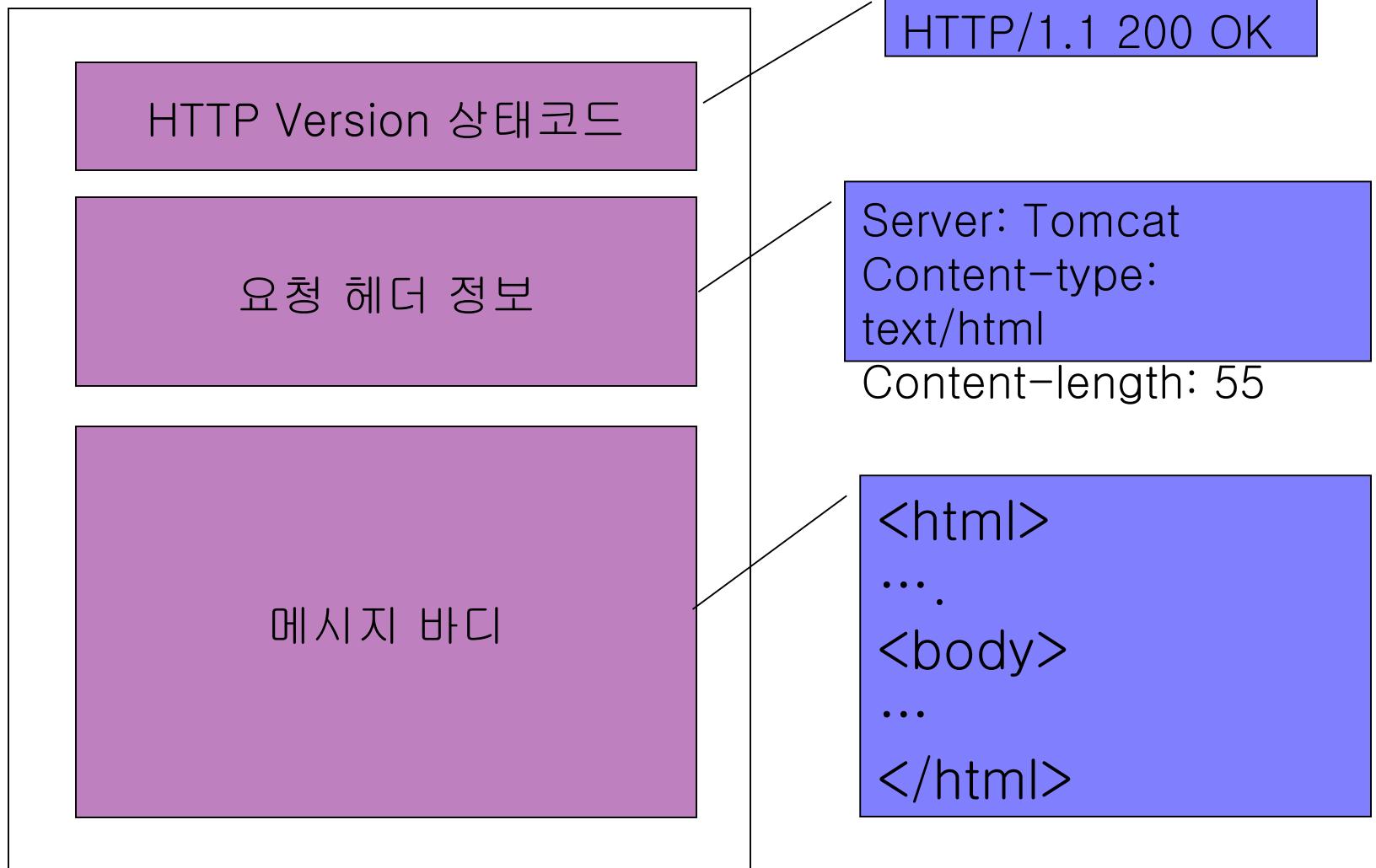
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8₩

(받아들이고자 하는 MIME 유형)

빈 줄

(공백으로 헤더의 끝을 암시).

응답 구조



응답 구조

HTTP/1.1 200 OK

Date: Wed, 11 Jan 2017 06:14:38 GMT

Server: Tomcat

Content-type: text/html

Content-length: 55

Last-modified: Wed, 11 Jan 2017 06:14:38 GMT

빈 줄

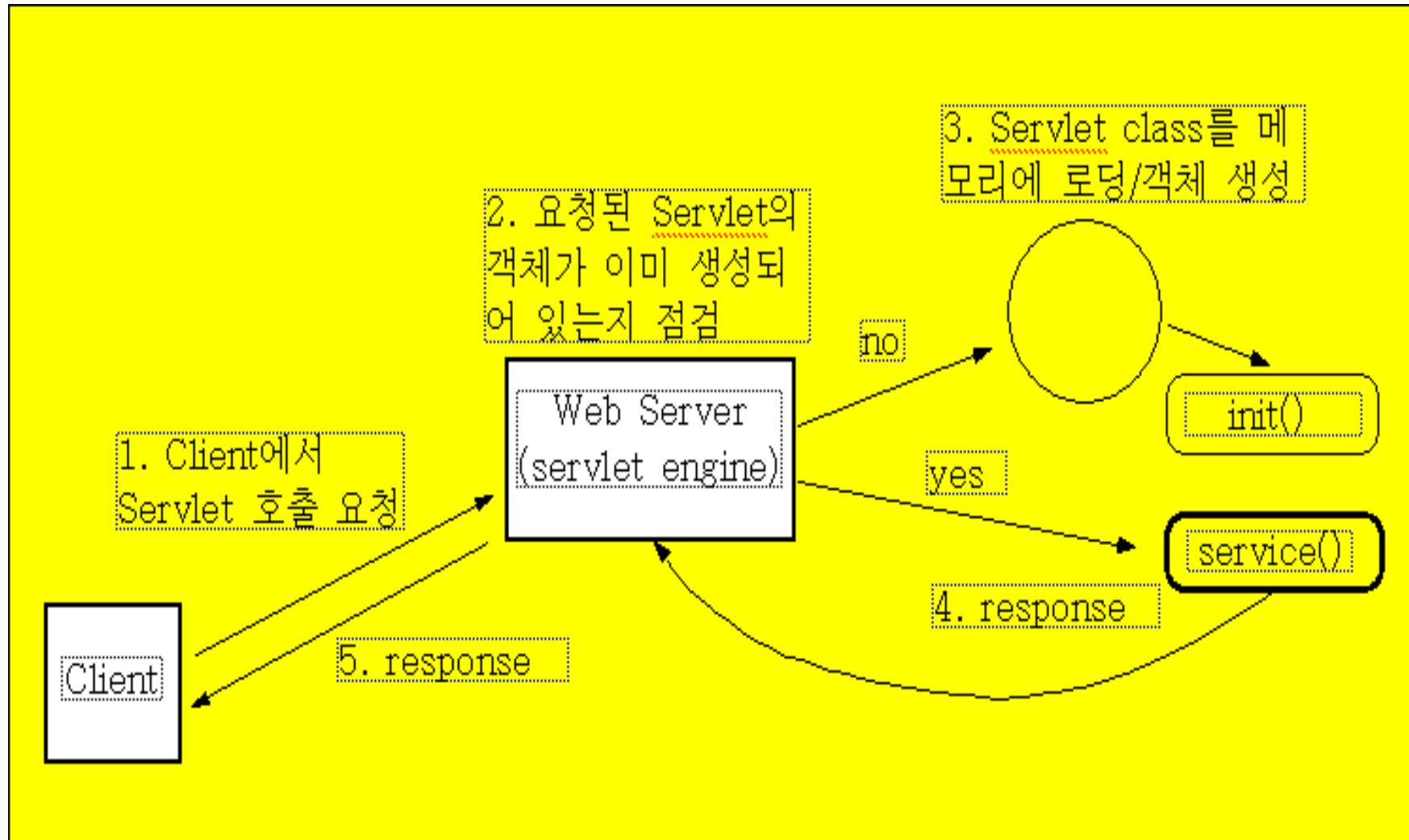
(공백으로 헤더의 끝을 암시).

요청이 성공인 경우 요청자료, 거절시 거절된 이유등의 메시지

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



초기화 정보 얻기

- init() 메소드에서 서블릿 객체 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 {
- @Override
- public void init(ServletConfig config) throws ServletException {
- super.init(config);
- System.out.println("당신의 나이는 " +
- config.getInitParameter("age"));
- }
- }

초기화 정보 얻기 (web.xml)

- <servlet>
- <servlet-name>InitParameterServlet </servlet-name>
- <servlet-class>com.oraclejava.InitParameterServlet </servlet-class>
-
- <!-- 초기화 파라미터 -->
- <init-param>
- <param-name>age</param-name>
- <param-value>100</param-value>
- </init-param>
-
- <load-on-startup>1</load-on-startup>
- </servlet>

요청 정보 얻기

- HttpServletRequest 개체
 - String **getParameter(String name)**
 - String[] **getParameterValues(String name)**
 - String **getHeader(String name)**
 - String **getRemoteAddr()**
 - void **setCharacterEncoding(String env)**

<form>와 입력 양식

- 다양한 GUI(Graphic User Interface) 입력 양식 태그를 담고 있는 컨테이너 태그
- <form> 태그가 포함할 수 있는 입력 양식 태그
 - <input>, <select>, <textarea>

```
<form name="양식명" method="get|post"  
      [enctype="multipart/form-data"]>  
  <input ~/>  
  <select ~>~</select>  
  <textarea ~>~</textarea>  
</form>
```

```
<input name="파라메터 명"
       type="text|password|checkbox|radio|
              button|submit|reset|file|hidden|image
       [value="""]
       [size="""]
       [maxlength="""]
       [checked="checked"]
       [alt="""] />
```

```
<select name="파라메터 명" [size="""]
        [multiple="multiple"] [alt="""]>
<option [selected="selected"] [value="""]>
    항목문자열
</option>
:
</select>
```

```
<textarea name="파라메터 명"
          [rows="""]
          [cols="""]></textarea>
```

응답 생성

- HttpServletResponse 개체
 - void **setContentType**(String type)
 - void **addHeader**(String name, String value)
 - PrintWriter **getWriter()**
 - ServletOutputStream **getOutputStream()**

Cookies

Cookies

■ 쿠키란?

- 서버가 클라이언트에게 보내는 작은 정보의 조각, 하나의 쿠키에는 쿠키가 어디에서 왔는지 또 언제 까지 유효한지 등이 표시
- 원래 NetScape에서 제안했으나, 현재는 인터넷의 한 표준 예) 응답헤더:

Set-Cookie : NAME=VALUE;expires=DATE;
path=PATH;domain=DOMAIN;secure

■ 단점

- 사용자가 쿠키를 받아들이지 않을 수도 있다.
- 브라우저가 쿠키를 지원하지 않을 수도 있다.

Cookies

■ 제한사항

- 보통 하나의 쿠키에 담을 수 있는 Byte 수는 4k이며, 하나의 브라우저가 도메인에 대해 저장할 수 있는 쿠키의 개수(보통 20개) 역시 한계가 있다.
- 웹 서버가 쿠키를 반응 헤더로서 웹 브라우저에 전달하면, 웹 브라우저는 쿠키를 메모리에 기억한 후, 차후 연결 시에 요청 헤더로서 그 쿠키 값을 서버에 전달한다.

Cookies

■ 쿠키의 만료일

- 다음과 같은 Set-Cookie 헤더가 브라우저에 전송되었다고 할 때 Set-Cookie:

NAME=OracleJava;path=/;expires=Wednesdays,
11-Jan-2017 23:00:00

- 만료일시가 명시적으로 지정되어 있으면 쿠키는 영속적인 저장매체에 저장된다(IE인 경우 Temporary Internet Files 디렉토리에 개별적 파일로 저장, NetScape인 경우 cookies.txt file에 저장)

Cookies

- 만료일시가 명시적으로 기술되어 있지 않으면 쿠키는 브라우저가 사용하는 메모리에 저장되며 브라우저가 종료되면 사라진다
- 두 경우 모두 Client가 서버에게 보내는 Request Header에는 다음과 같이 포함되어 전송된다. (Cookie : NAME = OracleJava)
- 목적
 - 웹 서버와 웹 브라우저 사이의 HTTP 연결들을 서로 연결시켜 주는데 있다.
- 용도
 - 사용자 설정, 간단한 로그인 기능, 데이터 수집 등

Cookies

Constructor Summary

Cookie(java.lang.String name, java.lang.String value)

Constructs a cookie with a specified name and value.

Method Summary

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

Cookies

String	getValue() 쿠키의 값을 돌려준다.
int	getVersion() 쿠키가 만족하는 Protocol의 버전을 돌려준다
void	setComment (java.lang.String purpose) 쿠키의 목적을 설명하는 주석을 설정한다
void	setDomain (java.lang.String pattern) 도메인 한정 패턴을 기술, 점으로 시작, 최소한 2개의 점을 포함 예) .oraclejava.co.kr이라고 했다면 www.oraclejava.co.kr은 유효하나, www.mail.oraclejava.co.kr은 유효하지 않다.
void	setMaxAge(int expiry) 조단위로 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

Cookies

- HttpServletResponse 인터페이스의 메소드
 - addCookie(Cookie cookie)
 - 웹 브라우저에 전송할 쿠키, 여러 개를 전송 할 수 있다.
- HttpServletRequest 인터페이스의 메소드
 - Cookie[] getCookies()
 - 웹 브라우저가 전송한 쿠키들.

CookieTest.java(1)

```
/* 클라이언트의 요청시 이미 쿠키가 Setting되어 있으면, 인증된 형태를 보여주는
   HTML을 , 아닌경우 인증창이 있는 HTML을 보여준다 */
import ...;
...
@WebServlet("/CookieTest")
public class CookieTest extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException,
        ServletException {
        res.setContentType("text/html; charset=utf-8");
        PrintWriter out = res.getWriter();
        String ID = " ";
        String Passwd = " ";
        String n = null;
        // 이미 Cookie가 Setitng되어 있는지 확인
        if (req.getCookies() != null) {
            Cookie[] cookies = req.getCookies();
            for (int i = 0; i < cookies.length; i++) {
                Cookie thisCookie = cookies[i];
                n = thisCookie.getName();
                if (n.equals("id")) ID = thisCookie.getValue();
                if (n.equals("passwd")) Passwd = thisCookie.getValue();
            }
        }
    }
}
```

CookieTest.java(2)

```
// 이미 쿠키가 Client에 Setting되어 있는지 확인, 인증되어 있다면 인증박스가 없는 html 표시, 아니면 인증창이
// 있는 html 표시
if ((ID.equals("oraclejava")) && (Passwd.equals("oraclejava"))) {
    out.println("<html><head><title>Cookie Test Program</title></head>");
    out.println("<body><table width="500"><tr>");
    out.println("<form method="get" action="/test/Expire" name="logout">");
    out.println("Id : " + ID + " Passwd : " + Passwd);
    out.println("<input type="submit" name="Submit"
value="LogOut"></tr><tr></form><hr>");
    out.println("This is Cookie sample program<p>");
    out.println("[02/28]<a href="/test/NewsDetail">아이폰 차기 모델 몸체 </a>");
    out.println("</tr></table></body></html>");
} else {
    out.println("<html><head><title>Cookie Test Program</title></head>");
    out.println("<body><table width="500"><tr><form method="post"
action="/test/isOk" name="login">");
    out.println("<input type="text" size="10" maxlength="40"
name="id"> PassWord : ");
    out.println("<input type="password" size="10" maxlength="40"
name="passwd">");
    out.println("<input type="submit" name="Submit"
value="OK"></form></tr><hr>");
    out.println("This is Cookie sample program<p></tr><tr>");
    out.println("[02/28]<a href="/test/NewsDetail"> 아이폰 차기 모델 몸체 </a>");
    out.println("</td></tr></table></body></html>");
}

```

Java Servlet Programming

IsOk.java(1)

```
/* cookieTest.java에서 넘어 오는 ID와 passwd를 parameter로 받아 인증된 사용자인지의 여부를 판단
 */
import ...;
...
@WebServlet("/IsOk")
public class IsOk extends HttpServlet {
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException,
        ServletException {
        res.setContentType("text/html; charset=utf-8");
        PrintWriter out = res.getWriter();
        String ID = null;
        String Passwd = null;
        if (req.getParameter("id") != null) ID = req.getParameter("id");
        if (req.getParameter("passwd") != null) Passwd = req.getParameter("passwd");

        // 인증이 성공이라면, 흔히 이부분의 인증을 DB의 사용자 Table등의 ID/PassWd를
        // 비교하여 인증이 성공인지의 여부를 판단한다.
        if ((ID.equals("oraclejava")) && (Passwd.equals("oraclejava"))) {
            Cookie idCookie = new Cookie("id", "oraclejava");
            Cookie passwdCookie = new Cookie("passwd", "oraclejava");
        }
    }
}
```

isOk.java(2)

```
res.addCookie(idCookie);
res.addCookie(passwdCookie);

out.println("<html><head>");
res.setHeader("Refresh", "3; URL=http://localhost:8080/test/CookieTest");
out.println("Cookie 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=http://localhost:8080/test/CookieTest");
    out.println("ID PassWord not correct...<br/>");
    out.println("3초후면 첫페이지로 이동합니다.");
    out.println("</html></head>");

}

}
```

Expire.java

```
/*CookieTest에서 LogOut을 Click시 쿠키를 해제한다 */
import ...;

...
@WebServlet("/Expire")
public class Expire extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
        IOException, ServletException {
        Cookie[] cookies = req.getCookies();
        for (int i = 0; i < cookies.length; i++) {
            Cookie thisCookie = cookies[i];
            thisCookie.setMaxAge(0);
            res.addCookie(thisCookie);
        }
        res.sendRedirect("/test/CookieTest");
    }
}
```

NewsDetail.java(1)

```
/* 사용자가 CookieTest에서 뉴스를 보고자 할때 이미 인증이 되어 있는지의 여부를 판단한 후 인증된 사용자만 뉴스를 보여준다*/
import ...;
...
@WebServlet("/NewsDetail")
public class NewsDetail extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException,
        ServletException {
        res.setContentType("text/html; charset=utf-8");
        PrintWriter out = res.getWriter();
        String ID="";
        String Passwd="";
        String n=null;
        // Cookie Setting 여부를 파악
        if (req.getCookies() != null) {
            Cookie[] cookies=req.getCookies();
            for(int i=0;i<cookies.length;i++) {
                Cookie thisCookie = cookies[i];
                n=thisCookie.getName();
                if (n.equals("id"))      ID = thisCookie.getValue();
                if (n.equals("passwd"))  Passwd = thisCookie.getValue();
            } //for
        } //if
```

NewsDetail.java(2)

```
//이미 인증이 되어 있는 경우라면
if ((ID.equals("oraclejava")) &&
    (Passwd.equals("oraclejava"))) {
    out.println("<html><head><title>Cookie Test Program</title>");
    out.println("</head><body><table><tr><td><center>");
    out.println("<b> 아이폰 차기 모델 몸체는 유리? " +
               " </b><hr> ");
    out.println(" 입력일 : 2017/01/11 <hr></center> ");
    out.println(" 아이폰 차기 모델의 몸체의 소재는 유리일 가능성 높아. " +
               " 기존의 알루미늄이 아닌 유리 판넬과 스테인레스스틸 테두리 디자인 <br> ");
    out.println(" </body></html> ");
}

else {
    res.setHeader(" Refresh ", " 3; URL=http://localhost:8080/test/CookieTest
" );
    out.println(" 첫페이지로 가서 인증을 해주시기 바랍니다.<br> ");
    out.println(" 3초후면 첫페이지로 이동합니다. " );
}

}//method
} //class
```

Session Tracking

Session Tracking

- 목 적
 - 특정 웹 브라우저 사용자와 웹 서버와의 HTTP 연결들을 하나로 이어 준다.
- 세션은 쿠키 혹은 URL 리라이팅 (rewriting) 을 적절히 사용하여 구현된다.
 - Java Webserver의 경우 쿠키들이 실패할 경우 URL 재작성을 사용하여 변경하며, 서버가 정지시 디스크등에 세션객체를 출력하는 기능을 제공한다

Session Tracking

- 서블릿을 지원하는 모든 웹서버는 최소한 쿠키기반의 Session Tracking을 지원해야하며, 여기에서 Session ID등은 Client의 지속적인 Cookie안에 저장된다. 즉 서버에서 새로운 세션객체가 생성되면 세션ID라고 하는 고유한 식별자가 서버에 의해 생성되어 부여되고, 또한 쿠키등을 통해 Client에 해당 ID가 주어진다.
- 서블릿등이 서버에 접속시 Session Id에 해당되는 세션객체를 얻은후 거기에 담긴 정보를 뽑아낸다.

Session Tracking

- 브라우저가 Cookie를 받아들이지 않게 설정되었다면 URL재작성에 기반한 Session Tracking을 지원한다.
- 하나의 사용자가 여러 자원들 (HTML, 서블릿,JSP,JavaBean등)을 사용하는 하나의 사이트를 방문할경우 각 자원들마다 세션이 생기는것이 아니라 사이트 자체에 대한 하나의 세션ID와 세션이 만들어 지는 것이다.

Session Tracking

- 처음 로그인 시에만 사용자 ID와 암호 등을 검증하고 세션 ID를 발급 하여, 이후에는 발급된 세션 ID 만 검증함으로써 기존 HTTP 인증 방법보다 효율적이다.
- HTML 폼 hidden 속성을 사용하는 것보다 세션 데이터를 서버측에서 관리 함으로서 교환되는 자료의 양이 적어 보다 효율적이다.

Session Tracking(HttpServletRequest 인터페이스)

- boolean isNew()
 - 세션이 새로운 것인지의 여부를 알려줌
 - 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.

Session Tracking(HttpSession 인터페이스)

- String getId()
 - 세션 식별자를 리턴.
- long getCreationTime()
 - 세션이 생성된 시각, 단위는 1970년 자정을 기준으로 흐른 밀리초의 개수(long type)
- long getLastAccessedTime()
 - 사용자가 마지막으로 세션을 사용한 시각, 오래된 세션을 닫는 등 세션 관리에 사용될 수 있다.
 - 단위는 단위는 1970년 자정을 기준으로 흐른 밀리초의 개수(long type)

Session Tracking(HttpSession 인터페이스)

- `putValue(String name, Object value)`
 - 세션 객체에 데이터를 추가, 이미 존재시 새값으로 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

Session Tracking(HttpSession 인터페이스)

- `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 해제 한다.

Javax.servlet.http. HttpSessionBindingListener Interface

- 몇몇 객체가 세션에 바인딩 되거나, 해제될 때 어떤 행위 등을 하고자 할 때 사용
- valueBound(HttpSessionBindingEvent event) Method
 - 리스너가 세션에 바인딩 될 때 호출됨
- valueUnbound(HttpSessionBindingEvent event) Method
 - 리스너가 세션으로부터 해제 시 호출된다

Javax.servlet.http.HttpSessionBindingEvent Class

- getName Method를 이용해 바인딩된 객체에서의 이름에 접근할수 있도록 해줌
- public String HttpSessionBindingEvent.getName()
 - 바인딩된 객체에서의 이름에 접근할수 있도록 해줌
- public String HttpSessionBindingEvent.getSession()
 - 리스너가 바인딩된 HttpSession 객체에 접근

Session Tracking

(HttpServletRequest 인터페이스와 메소드)

- HttpSession getSession()
 - 이 요청에 연관된 현재 세션을 돌려줌
 - 없으면 새로 만듬
- HttpSession getSession(boolean create)
 - 이 요청에 연관된 현재 세션을 돌려줌
 - 사용자가 유효한 세션을 가지고 있지 않다면 Create 인자가 true면 새로운 세션을 돌려주며, False이면 null을 돌려줌

Session Tracking (HttpServletRequest 인터페이스와 메소드)

- boolean isRequestedSessionIdValid()
 - 세션이 유효하면 true Return
- boolean isRequestedSessionIdFromCookie()
 - 이 세션에 연관된 세션ID가 Cookie를 통해 전달된것이면 true
- boolean isRequestedSessionIdFromUrl()
 - 이 세션에 연관된 세션ID가 URL를 통해 전달된것이면 true

간단한 Session 인증 예제(1)

```
/* 간단한 세션인증 예제 입니다. */
import ...;

...
@WebServlet("/SessionAuth")
public final class SessionAuth extends HttpServlet
{
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        sendPage(req, res, req.getSession(false));
    }

    protected void doPost(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException {
        //login button을 누른경우
        if (req.getParameter("login") != null) {
            HttpSession session = req.getSession(true);
            String name = req.getParameter("name");
        }
    }
}
```

간단한 Session 인증 예제(2)

```
//이름이 비어있는경우 Anonymous로 Setting
if (name == null || name.length() == 0) {
    name = "Anonymous";
}
session.setAttribute("name", name);
sendPage(req, res, session);
}

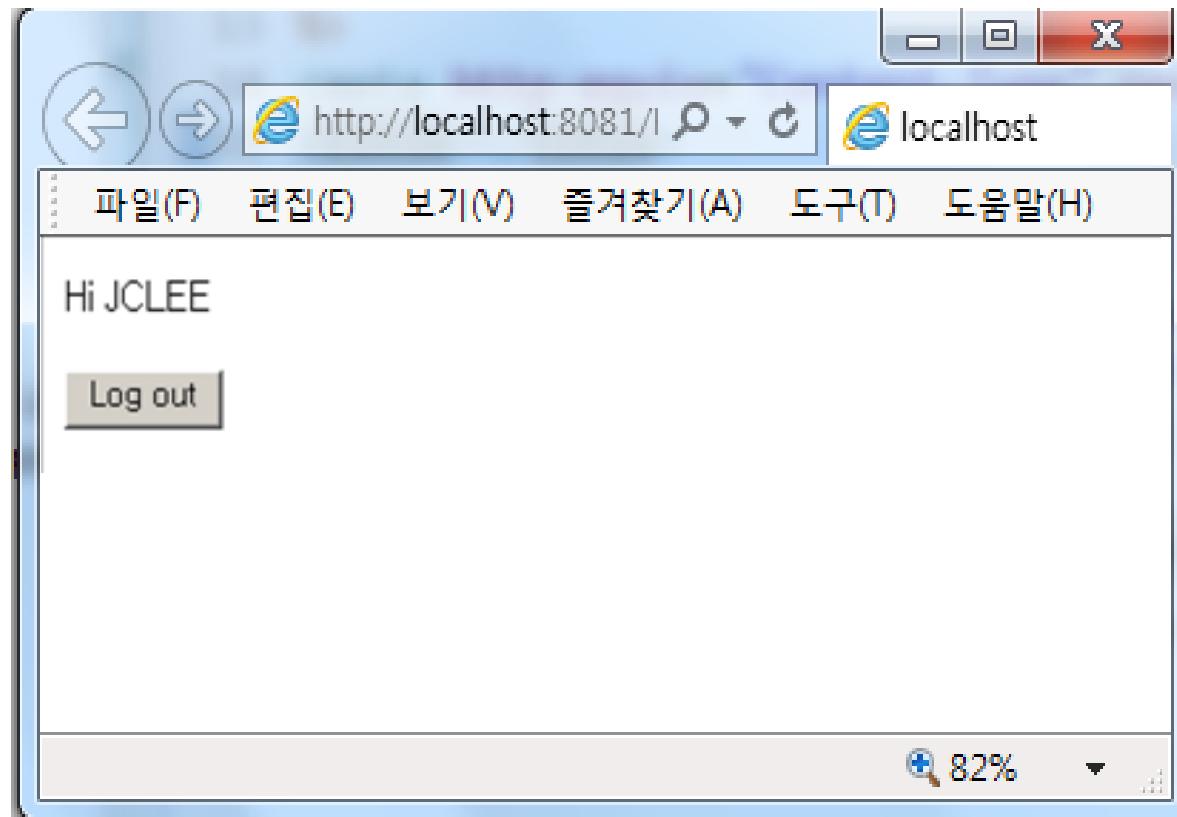
//LogOut을 누른경우
else {
    HttpSession session = req.getSession(false);
    if (session != null) {
        session.invalidate();
    }
    sendPage(req, res, null);
}

//session이 생성된경우와 생성되지 않은 경우 다르게 html구성 Method
private void sendPage(HttpServletRequest req, HttpServletResponse res,
    HttpSession session) throws ServletException, IOException {
```

간단한 Session 인증 예제(3)

```
res.setContentType("text/html;charset=utf-8");
res.setHeader("pragma", "no-cache");
PrintWriter out = res.getWriter();
    out.println("<html>");
    if (session == null) {
        out.println("<form method=post>Please enter your name:<br>" +
                    "<input type=text name=name>" +
                    "<input type=submit name=login value=Log in></form>");
    } else {
        out.println("Hi " + session.getAttribute("name") +
                    "<p><form method=post>" +
                    "<input type=submit name=logout value=Log out></form></p>");
    }
    out.println("</html>");
    out.close();
}
```

실행결과



URL Rewrite

- 사용자가 웹브라우저에 쿠키를 사용하게 하지 못했을 때, 쿠키를 이용한 세션트래킹은 불가능, 이 때 URL Rewriting을 이용한다.

Explorer : 도구/인터넷옵션 메뉴에서 개인정보 부분.... "쿠키사용하지않음"으로 설정

<Form> Tag을 기술하는 부분에 인코딩된 SessionID를 포함하는 Action URL을 설정해야 한다. 이 때 HttpServletResponse의 encodeURL 함수를 사용한다.

- Session ID의 노출이라는 보안상의 위험

URL Rewrite

- `public class HttpServletResponse.encodeUrl(String Url)`
지정된 URL이 세션ID를 포함하도록 암호화한후 새로운 URL을 Return, 암호화가 필요없거나 지원안한다면 URL은 변경되지 않는다. 서블릿에서 만들어 지는 모든 URL은 이 메소드를 통해 수행되어야 한다.
예) `res.encodeUrl(req.getRequestURI())`
- `public class HttpServletResponse.encodeRedirectUrl(String Url)`
지정된 URL이 세션ID를 포함하도록 암호화한후 새로운 URL을 Return, 암호화가 필요없거나 지원안한다면 URL은 변경되지 않는다. `HttpServletResponse`의 `sendRedirect()` 메소드로 Return되는 모든 URL은 이 메소드로 수행되어야 한다.

```
res.sendRedirect(res.encodeRedirectUrl  
("/servlet/URLRedirect"))
```

URL Rewrite(URLRewriting.html)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>Insert title here</title>
</head>
<body><center>
    <font size="+1" color="blue"><b>Login to the Personal Information Server</b></font>
    <hr>
    <form method="post" action="URLRewriting">
        <table border="0" align="center">
            <tr><td>Your ID </td>
                <td><input type="text" name="USER"></td>
            </tr>
            <tr><td>Password </td>
                <td><input type="password" name="PASSWORD"></td>
            </tr>
            <tr><td colspan="2"><input type="submit" value="Login"></td></tr>
        </table>
    </form>
</center>
</body>
</html>
```

URL Rewrite(URLRewriting.java)

```
//Other Page를 누를때 URL이 어떻게 넘어 가는지 Test하는 예제  
//웹브라우저에 쿠키를 받게 Setting되어 있다면 Cookie를 통해서 Session ID등이  
//전송되나 쿠키를 안받게 Setting되어 있다면 URL뒤에 sesison ID등이 붙어서  
넘어 가도록 되어있다.  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.util.*;  
  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;  
  
@WebServlet("/URLRewriting")  
public class URLRewriting extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        doPost(req,res);  
    }  
}
```

URL Rewrite(URLRewriting.java)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) throws  
    ServletException, IOException {  
  
    res.setContentType("text/html; charset=euc-kr");  
    PrintWriter out = res.getWriter();  
    //세션을 생성, 없으면 새로만듬  
    HttpSession sess = req.getSession();  
    String user=req.getParameter("USER");  
    String pWord=req.getParameter("PASSWORD");  
  
    //세션에 데이터 Put  
    sess.setAttribute(user,pWord);  
  
    out.println("<body><head><title>URLRewriting Servlet</title></head><body>");  
    out.println("<font size=+1 color=blue><b>URLRewriting Servlet</b></font>");  
    out.println("<hr>");
```

URL Rewrite(URLRewriting.java)

```
out.println("Session ID : " + sess.getId()+"<br/>");  
out.println("Creation Time : "+ new Date(sess.getCreationTime())+"<br/>");  
out.println("Last Accessed Time : "+  
           new Date(sess.getLastAccessedTime())+"<br/>");  
  
out.println("isRequestedSessionIdFromUrl : "+  
           req.isRequestedSessionIdFromUrl()+"<br/>");  
  
out.println("isRequestedSessionIdFromCookie : "+  
           req.isRequestedSessionIdFromCookie()+"<br/>");  
out.println("<hr><p/>");  
out.println("<b>Login Information</b><br/>");  
out.println("Your id : "+ user+"<br/>");  
out.println("Your password : "+pWord+"<br/>");  
out.println("<p/><br/>");  
  
//일반적인 URL 링크  
out.println("1. <a href='OtherPage'>OtherPage(일반적인 URL링크)</a><br/>");
```

URL Rewrite(URLRewriting.java)

```
//encodeURL() URL에 SessionID를 덧붙여전송
//(쿠키를 안받게 브라우저를 Setting시 URL뒤에 session id를 붙여넘김)
out.println("2. <a
href='"+res.encodeURL("OtherPage")+"'>OtherPage(encodedUrl)</a><br/>");

//encodeRedirectURL() 괄호안에 sendRedirect(Url)이나 절대URL경로를 기술
//(쿠키를 안받게 브라우저를 Setting시 URL뒤에 session id를 붙여넘김)
out.println("3. <a href='"+ + res.encodeURL("OtherPage") + "'>Other
Page(encodedRdirectUrl)</a><br/>");

//session.getId() method로 URL뒤에 임의로 Session ID를 붙임
out.println("4. <a href='OtherPage'$sessionid$"+sess.getId()+"'
>Other Page(일반적인 URL링크+session ID)</a><br/>");

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

}//doPost

}//class
```

URL Rewrite(OtherPage.java)

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/OtherPage")
public class OtherPage extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

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

        //세션생성 있으면 그세션을 넘기고, 없으면 Null을 넘김
        HttpSession sess = req.getSession(false);
        out.println("<html><head><title>OtherPage Servlet</title></head><body>");
        out.println("<font size=“+1” color=“blue”><b>OtherPage Servlet</b></font><hr/>");
```

URL Rewrite(OtherPage.java)

```
if(sess==null) {  
    //세션이 성립되어 있지 않은 경우  
    out.println("<font size=-1 color=blue>먼저 LogIn을 하세요(세션이 성립되어 있지  
    않습니 다.)</font><br>");  
} else {  
    out.println("Session ID : "+sess.getId()+"<b/r>");  
    String key[] = sess.getValueNames();  
    out.println(key[0]+"!!! Welcome to my site.");  
    out.println("<br>session # : "+key.length+"<br/>");  
}  
out.println("isRequestedSessionIdFromUrl : "+  
req.isRequestedSessionIdFromUrl()+"<br/>");  
out.println("isRequestedSessionIdFromCookie :  
"+req.isRequestedSessionIdFromCookie()+"<br/>");  
}  
}
```

URL Rewrite 실행 예

The image displays two side-by-side screenshots of a Microsoft Internet Explorer browser window. Both windows have the URL `http://localhost:8081/OracleWeb/URLRewriting.html` displayed in the address bar.

Left Window (Login Page):

The title bar says "Insert title here". The menu bar includes "파일(F)", "편집(E)", "보기(V)", "즐겨찾기(A)", "도구(T)", and "도움말(H)". The content area shows a form titled "Login to the Personal Information Server". It contains fields for "Your ID" and "Password", and a "Login" button.

Right Window (Servlet Response):

The title bar says "URLRewriting Servlet". The menu bar includes "파일(F)", "편집(E)", "보기(V)", "즐겨찾기(A)", "도구(T)", and "도움말(H)". The content area displays session information and login details.

Session Information:

- Session ID : E37BAD94513866DAC14A02246EF463CE
- Creation Time : 2015-01-08
- Last Accessed Time : 2015-01-08
- isRequestedSessionIdFromUrl : false
- isRequestedSessionIdFromCookie : true

Login Information:

Your id : oralcejava
Your password : 1111

List of URL Rewriting Techniques:

1. [OtherPage\(일반적인 URL 링크\)](#)
2. [OtherPage\(encodeUrl\)](#)
3. [Other Page\(encodedRedirectUrl\)](#)
4. [Other Page\(일반적인 URL 링크+session ID\)](#)

JDBC 프로그래밍

- 자바 프로그램 <-->java.sql 패키지 <--> JDBC드라이버 <-->데이터베이스
- 정의
 - 관계형 DBMS에 SQL 문장을 실행시키기 위한 자바 API
- java.sql 표준 패키지 및 JDBC드라이버의 표준을 정한다.

- 목 적
 - DBMS 종류에 독립적인 자바 프로그램
- java.sql 패키지
 - SQL문장을 JDBC 드라이버에 전달하고, 그 결과를 반환하기 위한 클래스가 제공된다.
 - 이 패키지의 클래스는 내부적으로 JDBC 드라이버에 작업을 넘겨 주어 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.

- 사용할 수 있는 질의어의 범위
 - 사용된 JDBC 드라이버 및 DBMS의 능력에 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 데이터베이스 질의어 문법으로 ANSI SQL92 Entry Level(SQL-2)을 지원한다.
- JDBC 드라이버 리스트 :
 - <http://java.sun.com/products/jdbc/jdbc.drivers.html>

JDBC Driver Type

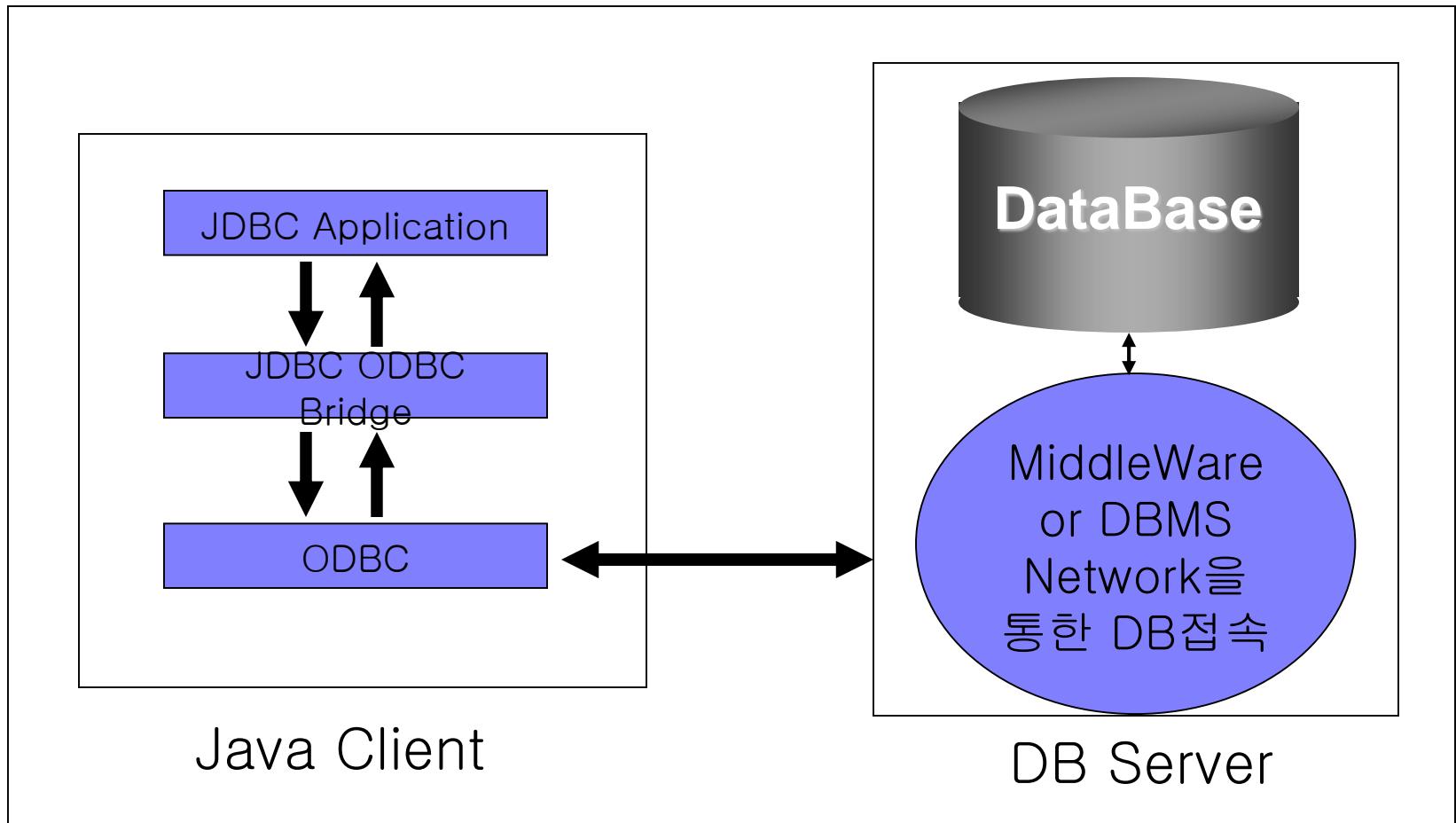
Type1: JDBC-ODBC Bridge

- JDBC 함수호출을 ODBC 함수호출로 전환하기 때문에 만약 예전에 사용하던 환경이 ODBC를 이용해 구축되어 있다면 굳이 다른 드라이버를 사용하지 않고서도 시스템을 확장해 나갈수가 있다
- ODBC와 DB 간에 소켓을 사용하기 때문에 방화벽을 통과하지 못하기 때문에 익스트라넷 환경에서 사용할 수가 없다. 또한 ODBC 접속부분에서 100% 자바코드를 사용하지 않기 때문에 애플릿으로도 연결할 수가 없다.
(주로 *.mdb, SQL Server등을 Access시 사용)

Type1: JDBC–ODBC Bridge

- 주의할점은, JDBC를 사용하는 Application과 JDBC와 브릿지로 연결된 ODBC드라이버는 하나의 시스템에 같이 존재해야한다
- 장점 : ODBC 드라이버가 풍부하기 때문에, 거의 대부분의 데이터베이스 시스템에서 사용할 수 있으며, JDBC–ODBC 드라이버를 사용하는 클라이언트에 미리 ODBC드라이버가 설치되어 있는 경우에 매우 유용하게 사용할 수 있다.
- 단점 : JDBC–ODBC 브릿지를 사용하는 시스템에는 반드시, 해당 데이터베이스에 연결하기 위한, ODBC 드라이버가 설치되어야한다. 결국 애플릿에서 JDBC를 사용하여 프로그래밍하는 경우에 많은 문제가 되는데 애플릿을 이용해서, JDBC–ODBC를 사용할 경우에, 애플릿을 다운 받은 클라이언트에 미리 해당 ODBC 드라이버가 설치되어 있어야 하기 때문에 배포등에 많은 문제가 생기게 된다.

Type1: JDBC-ODBC Bridge



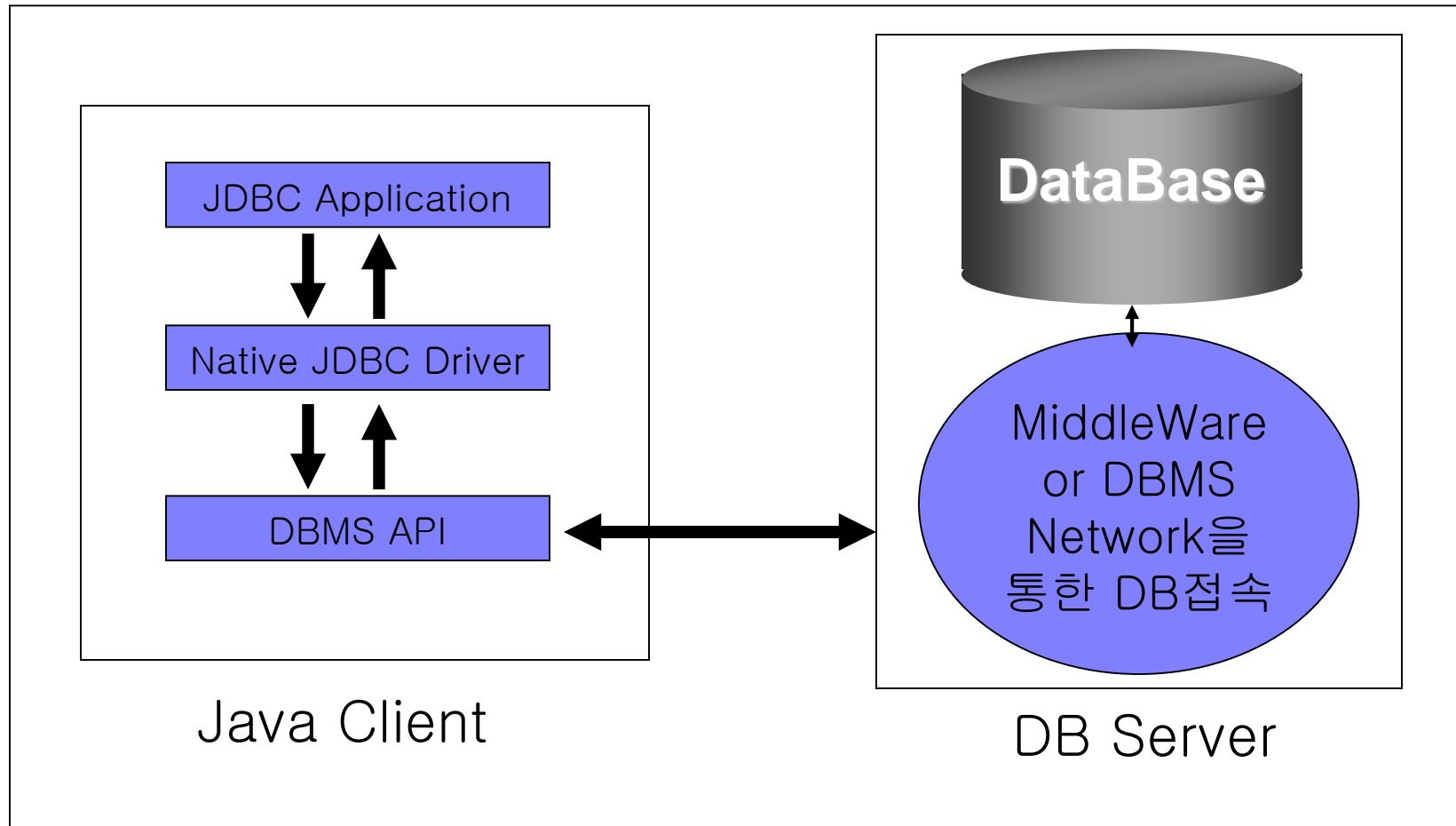
Type2:Native-API partly-Java Driver

- 네이티브-API란 벤더에서 제공하는 라이브러리를 이용해 DB를 엑세스한다는 의미이다. 그러므로 벤더에서 제공되는 2진 파일과 인터페이스하기 위해 자바는 부분적으로 Java Native Method를 이용하게 된다. (partly-Java) 이것은 클라이언트 쪽에 DB 회사에서 제공하는 라이브러리를 가지고 있어야 한다. 이또한 Applet에서는 작동을 하지 않는다.
- DB에 따라서 Networking S/W를 설치할 필요가 있다.

Type2:Native-API partly-Java Driver

- DBMS와 연결되는 부분은 C/C++ 과 같은 Native Code로 작성되어 있고, 이 Native 부분과 연결하여, Java로 Wrapping 해서 JDBC 드라이버를 구현한다.
- 개념적으로 생각해보면, Type 1의 ODBC 부분이 각 vendor에서 제공하는 Native library로 바뀌었다고 생각하면 됨
- JDBC드라이버를 사용하고자 하는 각각의 클라이언트에, DBMS Vendor의 데이터베이스 라이브러리가 로드되어야 하기 때문에, 인터넷이나 CS환경에서는 사용하기 적합하지 않다. 또한 Type 3,4 드라이버에 비해서 낮은 성능을 제공한다.(Oracle의 SQL*Net)

Type2:Native-API partly-Java Driver



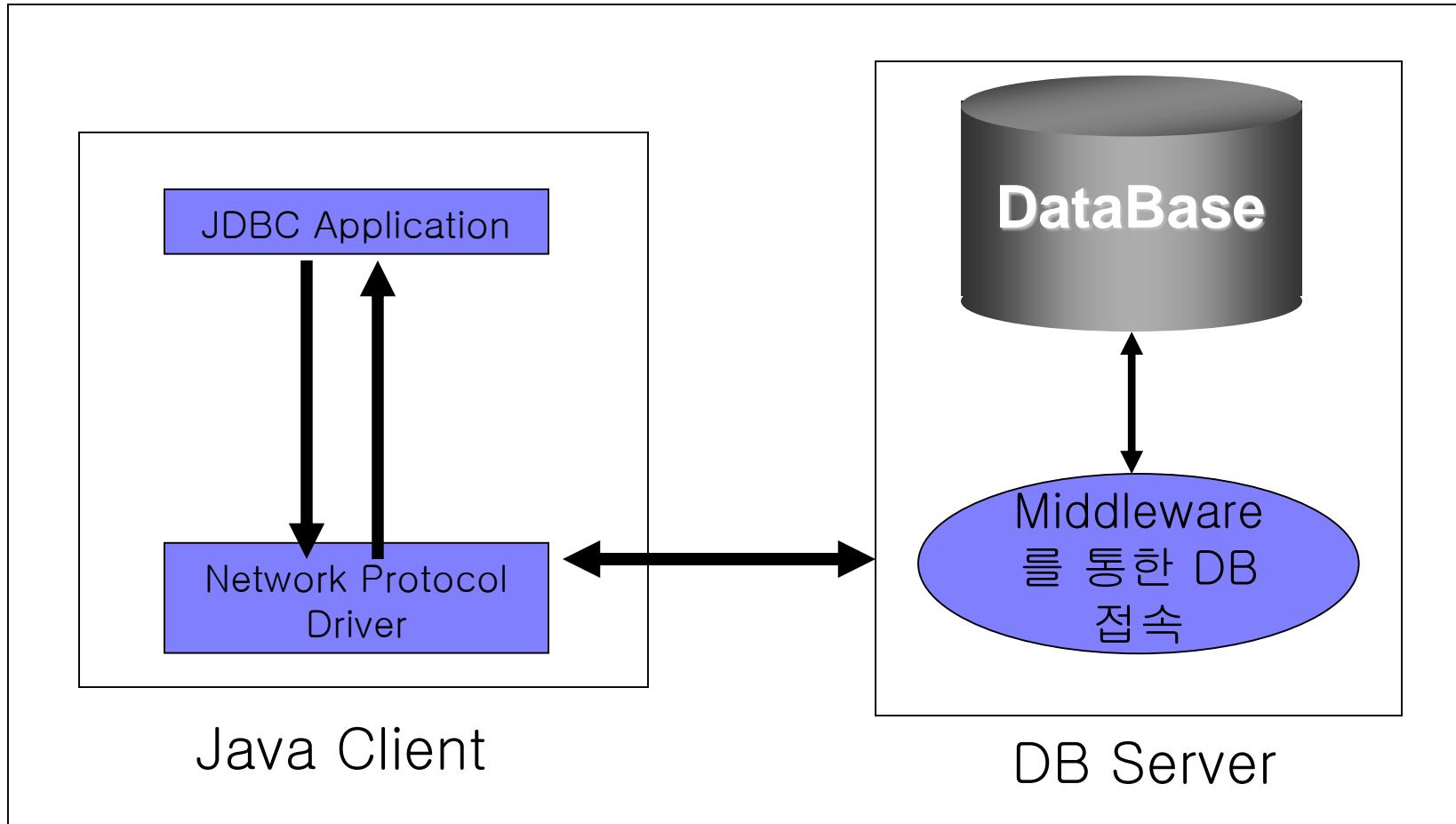
Type3: JDBC-Net pure Java Driver

- JDBC의 호출을 DBMS에 비종속적인 네트워크 프로토콜로 바꾼 후 다시 서버에 의해 원하는 DBMS의 프로토콜로 해석.
- 서버에 존재하는 네트워크 미들웨어에 의해 자바로 만들어진 모든 클라이언트 프로그램이 다른 여러 데이터베이스에 연결되는 3-tier 아키텍처.
- JDBC API 표준에 의하여 만들어 졌기 때문에 DBMS의 종류에 상관없이 사용할 수 있다.
4가지 Type 중에서 가장 융통성이 뛰어남.

Type3: JDBC–Net pure Java Driver

- JDBC 드라이버를 통해서 Database Command를 내리면 JDBC 드라이버는 그 Command를 변경하지 않고 그대로 Middleware (Database Access Server)로 전송한다. Middleware 에서는 데이터베이스에 맞는 명령어로 변환하여, 데이터베이스에 Command를 전송 한 후 그 결과 값을 받아서 다시 JDBC 드라이버에게로 전송
- 장점 :데이터베이스 접근을 Middleware를 통하기 때문에, 각각의 Client에 Database 별도로 Native Lib 또는 ODBC 드라이버를 설치할 필요가 없다.
- 단점 :중간에 Middleware 서버를 구현해야 하며, Middleware에서 연결하고자 하는 데이터베이스 vendor 별로 연결 부분을 제작해줘야 한다

Type3: network-protocol driver



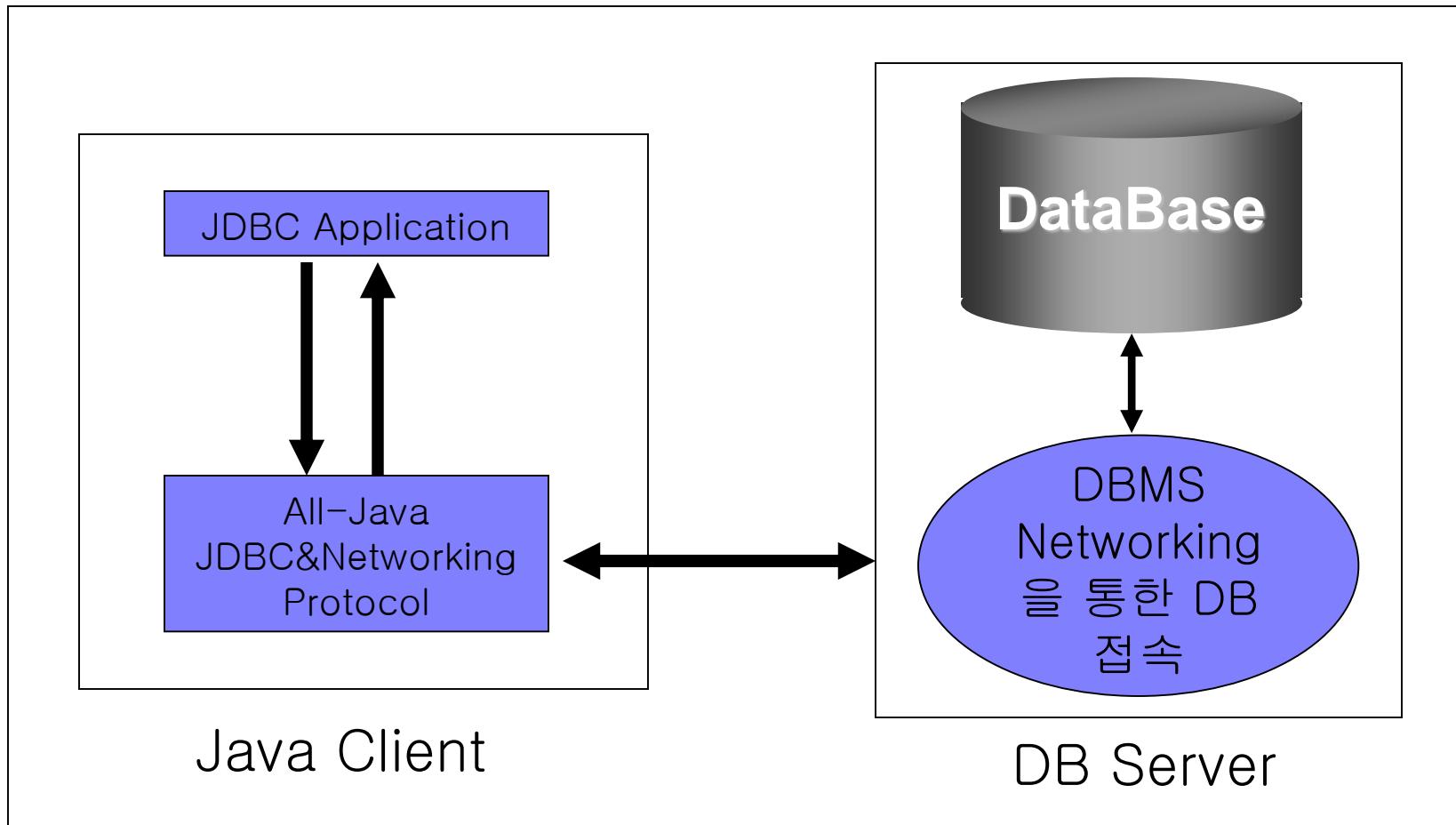
Type4:Native-protocol All Java Driver

- JDBC API는 DBMS가 사용하는 전용 프로토콜을 이용해 직접 인터페이스를 한다. 클라이언트에서 DBMS 서버로 직접 엑세스.
- 개별적인 Database 업체에서 지원하고 있는 JDBC driver로 JDBC문을 직접 특정 데이터 베이스 Protocol로 변환.
- Type 3와는 다르다. JDBC API 표준을 기준으로 만들었다기 보다는 DBMS Vendor가 표준을 기준으로 기능을 추가 또는 삭제하여 독자적인 형태로 만든 type이다. 따라서 특정 DBMS에 의존적인 반면에 해당 DBMS만이 제공하는 기능들을 사용할 수 있다는 장점.

Type4:Native-protocol pure Java Driver

- 가장 많이 사용하고, 가장 좋은 성능을 내는 JDBC Type으로 Type 4 는 100% Java 로 구현된 JDBC 드라이버로 네트워크를 통해서 각각의 DBMS로 직접 request를 전송하는 방법
- 대부분의 DBMS vendor들이 Type 4 드라이버를 무료로 제공하고 있으며, DBMS vendor site에서 대부분 무료로 다운 받을 수 있다.
- 특별하게 Driver나 Lib, Middleware등을 설치할 필요가 없기 때문에 배포등이 매우 용이하다.
- 단점 : 각각의 데이터베이스마다 다른 JDBC 드라이버를 사용해야한다

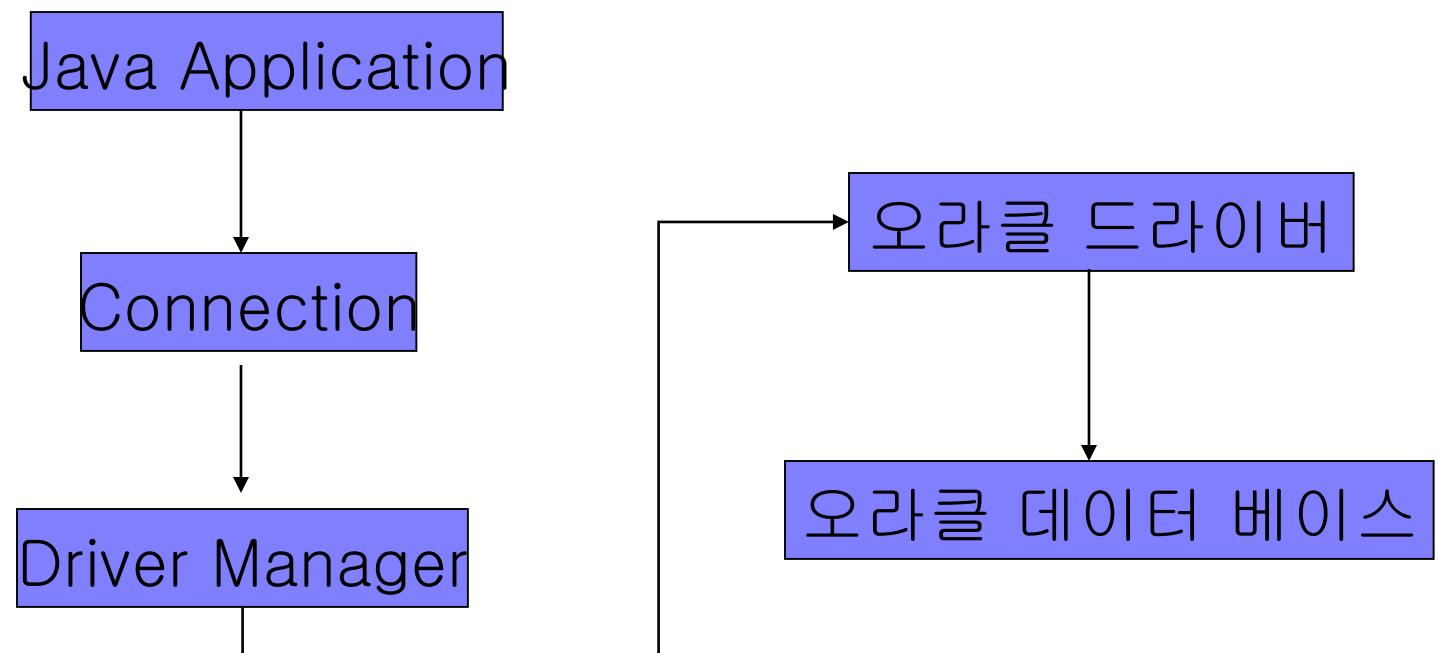
Type4:Native-protocol pure Java Driver



JDBC 구조

- JDBC 드라이버
 - `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 vendor 들에 의해 구현.

- 자바와 데이터 베이스 연결도.



JDBC 연결순서

- 첫 번째
 - 데이터베이스와의 접속을 오픈 하기 위해 애플리케이션의 JVM안으로 특정 드라이버 클래스를 적재.
 - 오라클 Thin dirver :
 - `Class.forName("oracle.jdbc.driver.OracleDriver");`
 - 드라이버가 메모리에 적재될 때,
`java.sql.DriverManager` 클래스를 사용해서
이 드라이버를 사용 가능한 드라이버로 등록.

- 두 번째 : DriverManager클래스를 이용하여 URL형태로 주어진 데이터 베이스에 대한 접속을 요청.
 - Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@dbhost:port:sid", "계정명", "비번");
 - JDBC URL은 드라이버 고유의 방식으로 개별적인 데이터베이스를 식별.
-  오라클:
[jdbc:oracle:thin:@dbhost:port:sid]

- 세 번째 : SQL 질의어 실행
 - java.sql.Statement 클래스를 사용한다.
 - Statement 클래스는 직접 인스턴스화되지 않고 Connection 클래스의 createStatement() 메소드에 의해 생성.
 - ☞ Statement stmt = con.createStatement();
 - Statement의 executeQuery() 메소드를 사용하여 데이터를 포함하는 java.sql.ResultSet을 리턴받는다.
 - ☞ ResultSet rs = stmt.executeQuery("SELECT * FROM CUSTOMERS");

- 마지막 : Close the Connection
 - connection.close();
 - Connection은 상당한 Overhead를 가져온다.
따라서 최적화된 상태를 유지하기 위해서는 반드시 Connection을 닫아 주어야 한다.

JDBC를 연결한 서블릿 예제

DBConnectionTest.java

- ResultSet 객체
 - 한번에 한 행씩 리턴하는 질의어의 결과
 - next() 메소드를 이용하여 한 행씩 이동.
 - DB의 데이터타입에 따른 메소드를 호출해서 데이터를 리턴 받는다.
- ☞ getObject(), getString() 메소드를 사용해서 열(column)값을 얻는다.

DBConnectionTest.java

```
import ...;  
...  
@WebServlet("/DBConnectionTest")  
public class DBConnectionTest extends HttpServlet {  
    ...  
    public void doGet(HttpServletRequest req,  
                      HttpServletResponse res) throws ServletException,  
                      IOException {  
        Connection con = null;  
        Statement stmt = null;  
        ResultSet rs = null;
```

DBConnectionTest.java

```
res.setContentType("text/html; charset=utf-8");
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:@192.168.1.104:1521:XE", "scott", "tiger");

    // Statement object를 생성한다.
    stmt = con.createStatement();
```

DBConnectionTest.java

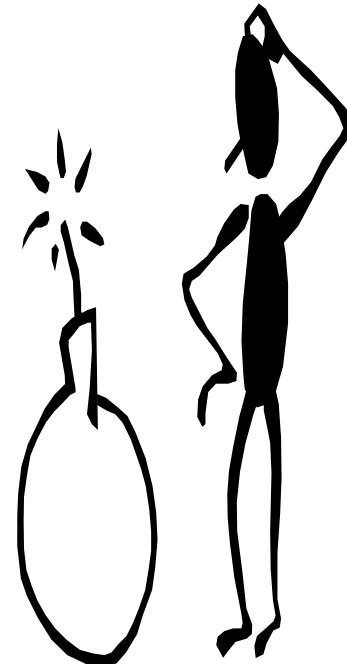
```
// ResultSet을 얻기위해 SQL query를 실행한다.  
rs = stmt.executeQuery("SELECT empno, ename FROM EMP");  
  
// 결과 출력  
  
out.println("<HTML><HEAD><TITLE>Phonebook</TITLE></HEAD>");  
out.println("<BODY>");  
out.println("<UL>");  
while(rs.next()) {  
    out.println("<li>" + rs.getString("empno") + " " +  
    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 {  
    // 언제나 데이터 베이스 연결을 종료한다.  
    try {  
        if (con != null) con.close();  
    }
```

DBConnectionTest.java

```
}catch (SQLException ignored) { }  
}  
}
```



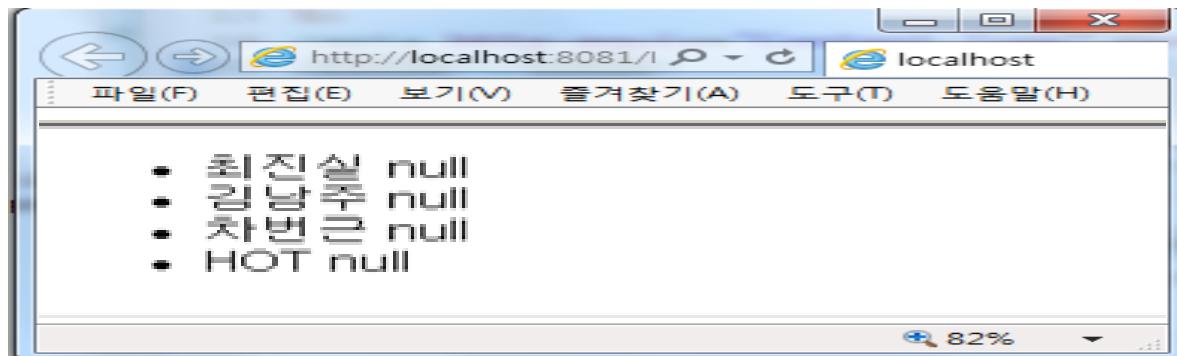
- 데이터 베이스 갱신
 - SQL 문의 UPDATE, INSERT, DELETE문장을 수행.
 - public int executeUpdate(String sql) throws SQLException
 - int count = stmt.executeUpdate("DELETE FROM DB WHERE NUM=1"); 이 함수는 질의 문장의 수행으로 변경된 행의 개수를 리턴.

NULL 필드 다루기

- JDBC에서 getInt()와 같은 객체를 리턴 하지 않는 경우는 error가 발생한다.
- 따라서 객체를 반환하는 getObject()메소드를 사용하거나, 이전에 조건검사를 해야 한다.
- getString()이나 , getInt()로 받은후 wasNull을 이용하여 Null인지 확인

String의 Null Test

- Employees Table 의 name 컬럼을 null로 update
SQL>update employees set phone=null; 또는
SQL>update employees set phone='';
다시 ConnectionTest를 실행
(getString()으로 받을때 받은값이 null이라면
null이라고 찍힘)



Integer의 Null Test

- Employees Table에 컬럼을 추가

SQL>alter table employees add (age number(2));

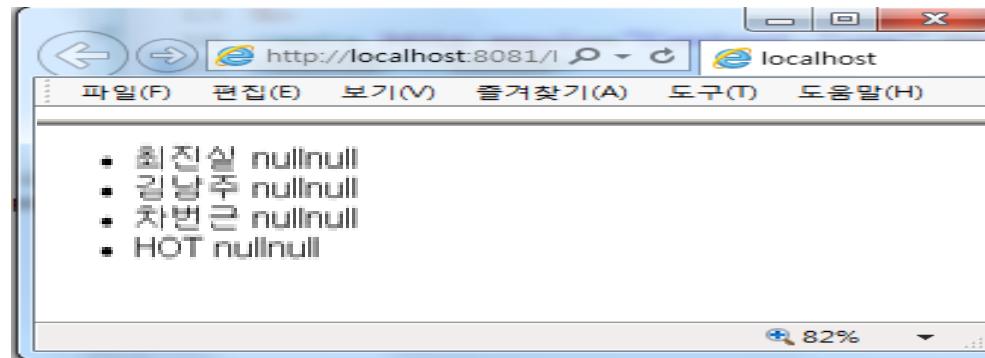
ConnectionTest의 소스를 age Column도 출력할 수 있도록 수정하여 (아래처럼)

경우1 : out.println("" + rs.getString("name") + " " +
rs.getString("phone") + rs.getString("age"));

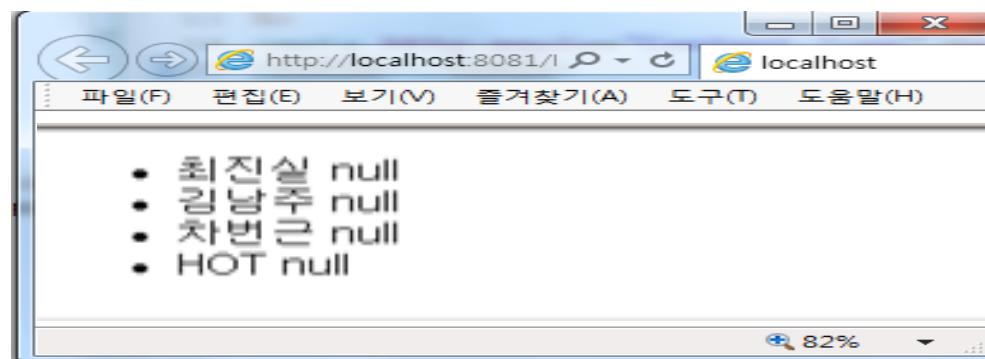
경우2 : out.println("" + rs.getString("name") + " " +
rs.getString("phone") + rs.getInt("age"));

Integer의 Null Test 결과

- getString("age")로 받는 경우



- getInt("age")로 받는 경우



Null 의 처리

- wasNull() 함수로 사전에 확인

ConnectionTest.java의 결과 출력 부분수정

```
String name, phone;  
while(rs.next()) {  
    out.println("<li>");  
    name = rs.getString("name");  
    if (!rs.wasNull()) out.println(name);  
    else out.println("<이름입력안됨!>");  
    phone = rs.getString("phone");  
    if (!rs.wasNull()) out.println(phone);  
    else out.println("<전화번호입력안됨!>");  
}
```

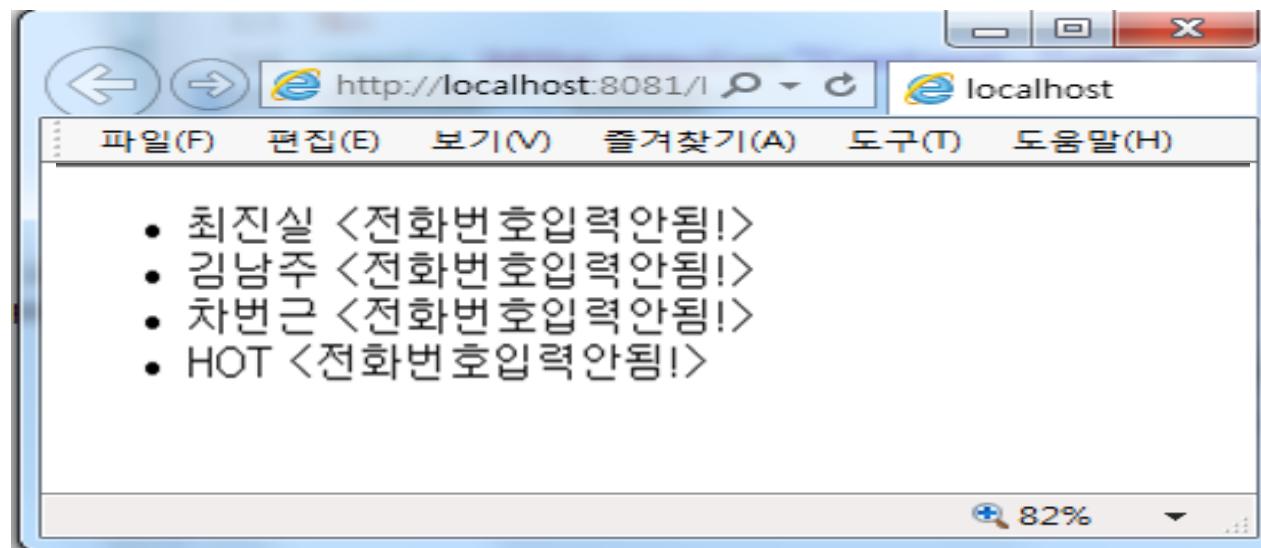
Null 의 처리

- `getObject()` 함수로 처리

ConnectionTest.java의 결과 출력 부분수정

```
Object name,phone;  
while(rs.next()) {  
    out.println("<li>");  
    name = rs.getObject("name");  
    if (name != null) out.println(name.toString());  
    else out.println("<이름입력안됨!>");  
    phone = rs.getObject("phone");  
    if (phone != null) out.println(phone.toString());  
    else out.println("<전화번호입력안됨!>");  
}
```

Null 의 처리 결과



ResultSet 인터페이스와 데이터 검색문

ResultSet 인터페이스와 데이터 검색문

- ResultSet 객체는 SQL 문장의 실행 결과(테이블 형태)를 담고 있다
- ResultSet은 SQL문내의 조건들을 만족시키는 모든 열들을 포함하고, 현재 열에서 다양한 행들로 접근 할 수 있는 `getXXX` 메소드 집합을 통해서 그 열들 내에 있는 데이터에 접근을 제공한다.
- 실행결과 테이블의 행 검색
 - 한 행씩 차례로 읽어 들인다.
 - `boolean next()` 메소드를 호출하여 다음 행으로 넘어갈 수 있다.
 - 더 이상 남아 있는 행이 없으면 `next()`는 `false`를 반환한다.

- 실행결과 테이블의 열 검색 메소드
 - 현재 검색되고 있는 행 내에서의 열은 임의의 순서로 읽어 들일 수 있다. 그러나, 각 열은 왼쪽에서 오른쪽으로 최대 한번씩만 읽어 들이는 것이 안전하다.
- XXX getXXX(String columnName)
 - 열 이름 columnName의 값을 데이터베이스 자료형으로부터 자바 자료형 XXX로 변환하여 읽어 들인다.
- XXX getXXX(String columnIndex)
 - 열의 인덱스 columnIndex(1부터 시작)로 열의 값을 읽어 들일 수도 있으며, 더 효율적이다.
- int findColumn(String columnName)
 - 이름 columnName을 갖는 열의 인덱스 반환, 여러 번 참조하는 열은 인덱스로 바꾸어서 검색하면 효율적이다.

- `getXXX` 메소드를 통하여 SQL 자료형으로부터 몇몇 자바 자료형으로 변환하여 읽어들일 수 있으나, 다음과 같이 가장 대응되는 자바 자료형으로 읽어 들이는 것이 바람직하다.
- ResultSet으로부터 데이터를 얻는 메소드

ResultSet으로부터 데이터를 얻는 메소드

SQL 데이터 타입	호출 함수	getObject() 리턴 타입
CHAR	String getString()	String
VARCHAR	String getString()	String
NUMERIC	BigDecimal getBigDecimal()	java.math. BigDecimal

ResultSet으로부터 데이터를 얻는 메소드

SQL 데이터 타입	호출 함수	getObject() 리턴 타입
DECIMAL	BigDecimal getBigDecimal()	java.math.BigDecimal
BIT	boolean getBoolean()	Boolean
INTEGER	int getInt()	Integer

ResultSet으로부터 데이터를 얻는 메소드

SQL 데이터	호출 함수	GetObject() 리턴 타입
---------	-------	-------------------

BIGINT	long getLong()	Long
--------	----------------	------

FLOAT	double	Double
-------	--------	--------

	getDouble()	
--	-------------	--

DOUBLE	double	Double
--------	--------	--------

	getDouble()	
--	-------------	--

ResultSet으로부터 데이터를 얻는 메소드

SQL 데이터 타입	호출 함수	GetObject() 리턴 타입
------------	-------	-------------------

BINARY	byte[] getBytes()	byte[]
DATE	Date getDate()	java.sql.Date
TIME	Time getTime()	java.sql.Time

ResultSet 기타메소드

- first() : Moves the cursor to the first row in this ResultSet object.
- next() : Moves the cursor down one row from its current position.
- previous() : Moves the cursor to the previous row in this ResultSet object.
- last() : Moves the cursor to the last row in this ResultSet object.
- isLast() : Indicates whether the cursor is on the last row of this ResultSet object.

SQL NULL값 처리

- 데이터베이스의 값이 SQL NULL이면, getXXX 메소드의 반환 자료형에 따라 다음과 같이 반환한다.
 - 객체일 경우 : null 반환
 - 기본 자료형일 경우 : false, 0, 0.0 반환
- boolean wasNull()
 - getXXX 메소드를 호출한 직후에 호출하면, 본래의 데이터베이스 값이 SQL NULL 이면 true.

- close()
- ResultSet 객체를 만들어낸 Statement 객체가 close 되거나, 재실행 되거나, 다수 결과를 만드는 ResultSet에서 다음 결과를 가져오면 자동적으로 호출된다. 필요한 경우에는 명시적으로 호출한다.

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=utf-8");
        PrintWriter out = res.getWriter();

        out.println("<html><head><title>JDBC Insert 예제</title></head><body>");
        out.println("<form name="myForm" action="/test/Insert""
                  "method="post">");
        out.println("이름 : <input type="text" name="name"><br>");
```

Insert 예제(Insert.java)

```
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 name = null;  
    int iNum = 0;  
    String Query = null;
```

Insert 예제(Insert.java)

```
res.setContentType("text/html; charset=utf-8");
req.setCharacterEncoding("utf-8"); //한글 처리

PrintWriter out = res.getWriter();

try {
    Class.forName("oracle.jdbc.driver.OracleDriver");

    con =
DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.153:1521:XE",
                            "scott", "tiger");
    con.setAutoCommit(false);

    if (req.getParameter("name") != null) name =
req.getParameter("name");
```

Insert 예제(Insert.java)

```
stmt = con.createStatement();

rs = stmt.executeQuery("SELECT max(empno)+1 FROM emp");
rs.next();
    iNum = rs.getInt(1);
rs.close();

Query = "insert into emp (empno, ename) values (" + iNum + "," + name +
        ")";
stmt.executeUpdate(Query);
con.commit();
stmt.close();
con.close();

res.sendRedirect("/test/Insert");
}
```

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 {  
    try {  
        if (con != null) con.close();  
    } catch (SQLException ignored) {}  
}  
}  
}
```

PreparedStatement Interface

PreparedStatement Interface

- SQL 문이 빠른 수행을 위해 DB에 의해 전처리된다.
- 일반적으로 반복적으로 수행하는 경우 용이함
- 위치표시자를 위해 ‘?’ 사용
- 문자열등에 ‘등이 있는 경우 DB에 저장시 에러
→ 이를 극복하기 위해 PreparedStatement를 사용하면 효과적

PreparedStatement Interface

- **clearParameters()**

- 이전에 정의된 파라미터 값을 제거

- **setXXX()**

- 율음표로 표시된 표시위치자 각각에 실제값을 할당하기 위해 사용

예제(Insert2.java)

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

```
//입력 Query 생성
Query = "insert into emp (empno, ename) values (?,?)";
PreparedStatement pstmt = con.prepareStatement(Query);
pstmt.setInt(1,iNum);
pstmt.setString(2,name);

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

ResultSetMetaData 인터페이스

ResultSetMetaData 인터페이스

- 목 적
 - ResultSet의 형태를 미리 알 수 없거나 다양한 ResultSet을 처리하는데 사용
 -  메타 데이터란? 실제 Data는 아니지만 Data의 설명을 위한 정보 데이터로서 컬럼이 름, type, Max size값 들이 이에 해당한다.
- ResultSet의 각 열에 대한 자료형 및 기타 정보를 얻기 위한 메소드 제공

ResultSetMetaData 인터페이스

- ResultSet 인터페이스의
ResultSetMetaData rsmd = rs.getMetaData()
- 열의 개수
 - int getColumnCount()
- 열에 대한 정보
 - String getColumnName(int column)
 - String getColumnLabel(int column)
 - 열의 제목을 출력하는 데 적절한 제목

ResultSetMetaData 인터페이스

- int getColumnDisplaySize(int column)
 - 열을 문자열로 출력하기 위한 최대 문자 수
- int getColumnType(int column)
 - 열의 SQL Type을 돌려줌
- String getColumnTypeName(int column)
- int isNullable(int column)
 - SQL NULL값을 설정할 수 있는지 여부.

ResultSetMetaData 인터페이스

- boolean isCurrency(int column)
- boolean isSigned(int column)
- int getPrecision(int column)
- boolean isReadOnly(int column)
- boolean isDefinitelyWritable(int column)
- boolean isWritable(int column)
- boolean isSearchable(int column)
- boolean isAutoIncrement(int column)

ResultSetMetaData 인터페이스

- 기타 정보
 - String getTableName(int column)
 - 열 column이 속하는 테이블의 이름
 - String getCatalogName(int column)
 - String getSchemaName(int column)

ResultSetMetaData 예제 (MetaData.java)

```
import java.sql.*;  
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
@WebServlet("/MetaData")  
public class MetaData extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        Connection con = null;  
        Statement stmt = null;  
        ResultSet rs = null;  
        ResultSetMetaData rsmd = null;  
        res.setContentType("text/html; charset=utf-8");  
        PrintWriter out = res.getWriter();  
        StringBuffer buffer = new StringBuffer();  
        try { // 오라클 드라이버를 Load 한다  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
            // 데이터 베이스에 접속을 한다.
```

ResultSetMetaData 예제(MetaData.java)

```
con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
"scott", "tiger");
// Statement object를 생성한다.
stmt = con.createStatement();
// ResultSet을 얻기위해 SQL query를 실행한다.
if (stmt.execute("SELECT * FROM EMPLOYEES")) {
    //ResultSet이 있는경우, SQL문장이 하나이상의 결과셋이 있으면 True
    rs = stmt.getResultSet();           //갱신된 Count를 얻으면 false
    rsmd = rs.getMetaData();
    int numCols = rsmd.getColumnCount();
    // 결과 출력
    buffer.append("<html><head><title>ResultSetMetaData 예제
</title></head><body><table><tr>");
    for (int i=1;i <= numCols ; i++ )
    {   buffer.append("<th>" + rsmd.getColumnLabel(i) + "</th>");
    }
}
```

ResultSetMetaData 예제 (MetaData.java)

```
buffer.append("</tr>\n");

    while(rs.next()) {
        buffer.append("<th>");
        for (int i=1; i <= numCols ;i++ ) {
            buffer.append("<td>");
            Object obj = rs.getObject(i);
            if (obj != null) buffer.append(obj.toString());
            else          buffer.append(" ");
            buffer.append("</td>");
        }
        buffer.append("</tr>\n");
    }
    buffer.append("</table></body></html>");
}
out.println(buffer.toString());
}
```

ResultSetMetaData 예제 (MetaData.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) { }  
}  
}  
}
```

실행결과

JAVA Servlet - Oracle11 Hyungku

localhost:7080/test/MetaData

JAVA Servlet - Oracle11g JDBC Test

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	salary	commission_pct	manager_id	department_id
100	Steven	King	SKING	515.123.4567	2003-06-17 00:00:00.0	AD_PRES	24000		90	
101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-21 00:00:00.0	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	2001-01-13 00:00:00.0	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03 00:00:00.0	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	2007-05-21 00:00:00.0	IT_PROG	6000		103	60
105	David	Austin	DAUSTIN	590.423.4569	2005-06-25 00:00:00.0	IT_PROG	4800		103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-05 00:00:00.0	IT_PROG	4800		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-07 00:00:00.0	IT_PROG	4200		103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	2002-08-17 00:00:00.0	FI_MGR	12008		101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	2002-08-16 00:00:00.0	FI_ACCOUNT	9000		108	100
110	John	Chen	JCHEN	515.124.4269	2005-09-28 00:00:00.0	FI_ACCOUNT	8200		108	100
111	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-30 00:00:00.0	FI_ACCOUNT	7700		108	100
112	Jose Manuel	Urman	JMURMAN	515.124.4469	2006-03-07 00:00:00.0	FI_ACCOUNT	7800		108	100
113	Luis	Popp	LPOPP	515.124.4567	2007-12-07 00:00:00.0	FI_ACCOUNT	6900		108	100
114	Den	Raphaely	DRAPHEAL	515.127.4561	2002-12-07 00:00:00.0	PU_MAN	11000		100	30
115	Alexander	Khoo	AKHOO	515.127.4562	2003-05-18 00:00:00.0	PU_CLERK	3100		114	30
116	Shelli	Baida	SBAIDA	515.127.4563	2005-12-24 00:00:00.0	PU_CLERK	2900		114	30
117	Sigal	Tobias	STOBIAS	515.127.4564	2005-07-24 00:00:00.0	PU_CLERK	2800		114	30
118	Guy	Himuro	GHIMURO	515.127.4565	2006-11-15 00:00:00.0	PU_CLERK	2600		114	30
119	Karen	Colmenares	KCOLMENA	515.127.4566	2007-08-10 00:00:00.0	PU_CLERK	2500		114	30
120	Matthew	Weiss	MWEISS	650.123.1234	2004-07-18 00:00:00.0	ST_MAN	8000		100	50
121	Adam	Fripp	AFRIPP	650.123.2234	2005-04-10 00:00:00.0	ST_MAN	8200		100	50
122	Payam	Kaufling	PKAUFLIN	650.123.3234	2003-05-01 00:00:00.0	ST_MAN	7900		100	50
123	Shanta	Vollman	SVOLLMAN	650.123.4234	2005-10-10 00:00:00.0	ST_MAN	6500		100	50
124	Kevin	Mourgos	KMOURGOS	650.123.5234	2007-11-16 00:00:00.0	ST_MAN	5800		100	50
125	Julia	Nayer	JNAYER	650.124.1214	2005-07-16 00:00:00.0	ST_CLERK	3200		120	50
126	Irene	Mikkilineni	IMIKKILI	650.124.1224	2006-09-28 00:00:00.0	ST_CLERK	2700		120	50
127	James	Landry	JLANDRY	650.124.1334	2007-01-14 00:00:00.0	ST_CLERK	2400		120	50
128	Steven	Markle	SMARKLE	650.124.1434	2008-03-08 00:00:00.0	ST_CLERK	2200		120	50
129	Laura	Bissot	LBISSOT	650.124.5234	2005-08-20 00:00:00.0	ST_CLERK	3300		121	50
130	Mozhe	Atkinson	MATKINSO	650.124.6234	2005-10-30 00:00:00.0	ST_CLERK	2800		121	50
131	James	Marlow	JAMRLOW	650.124.7234	2005-02-16 00:00:00.0	ST_CLERK	2500		121	50
132	TJ	Olson	TJOLSON	650.124.8234	2007-04-10 00:00:00.0	ST_CLERK	2100		121	50
133	Jason	Mallin	JMALLIN	650.127.1934	2004-06-14 00:00:00.0	ST_CLERK	3300		122	50
134	Michael	Rogers	MROGERS	650.127.1834	2006-08-26 00:00:00.0	ST_CLERK	2900		122	50
135	Ki	Gee	KGEE	650.127.1734	2007-12-12 00:00:00.0	ST_CLERK	2400		122	50

Oracle의 Stored Procedure 사용하기

CallableStatement

- 모든 DBMS들에 대한 표준방법으로 저장 프로시저 (stored procedure)를 호출하는 방법을 제공한다.
- 두가지 형태중의 하나인 escape 문법으로 작성되어진다.
즉, 결과 매개변수를 가지는 형태와 결과 매개변수가 없는 형태
- 결과 매개변수를 리턴하는 프로시저를 위한 문법은 다음과 같다.:
`{? = call procedure_name[(?, ?, ...)]}` 매개변수가 없는 저장 프로시저를 위한 문법은 다음과 같을 것이다. :
`{call procedure_name}`

CallableStatement

- CallableStatement 객체는 Connection의 prepareCall 메소드에 의해 생성된다.
아래의 예제는 저장 프로시저 getTestData의 호출을 포함하는 CallableStatement의 인스턴스를 만든다.
이것은 두 개의 인자들을 가지고 있고 결과 매개변수는 없다.:

```
CallableStatement cstmt = con.prepareCall("{call getTestData(?, ?)}");
```

CallableStatement

- CallableStatement 객체로 IN 매개변수를 넘겨주는 것은 PreparedStatement로부터 상속받은 setXXX 메소드다. 넘겨질 값의 데이터형은 사용할 setXXX 메소드를 결정한다.(setFloat는 float 값을 넘겨준다. 등등) 만약 저장 프로시저가 OUT 매개변수를 리턴한다면, 각 OUT 매개변수의 SQL형은 CallableStatement 객체를 실행할 수 있기 전에 등록되어져야 한다.(이것은 몇몇 DBMS들이 SQL형을 필요로 하기 때문에 필요하다.) SQL형을 등록하는 것은 registerOutParameter 메소드를 사용한다.

CallableStatement

- 다음의 예제는 유일한 매개변수로 INOUT 매개변수를 가지고 있는 저장 프로시저 reviseTotal이 있다는 것을 가정한다. setByte 메소드는 매개변수를 25로 설정하며, 이것은 드라이버가 SQL TINYINT로 써 데이터베이스에 전송할 것이다. 그런 다음 registerOutParameter는 SQL TINYINT로 써 매개변수를 등록한다. 저장 프로시저가 실행된 후에, 새로운 SQL TINYINT 값을 리턴하고, getByte 메소드는 이 새로운 값을 자바 byte로 검색할 것이다.
- ```
CallableStatement cstmt = con.prepareCall("{call reviseTotal(?)}");
cstmt.setByte(1, 25); cstmt.registerOutparameter(1,
java.sql.Types.TINYINT); cstmt.executeUpdate(); byte x = cstmt.getByte(1);
```

# CallableStatement 예제

```
/* Oracle Server에 작성된 Function */
```

```
create or replace function emp_name
(v_empno in number)
return varchar2
is
v_name varchar2(20) := '';
begin
 select ename into v_name
 from emp
 where empno = v_empno;
 return v_name;
exception
 when no_data_found then
 return v_name;
end;
```

# CallableStatement 예제

```
// storedprocedure.html
```

```
<html>
<head><title>Stored Procedure Call Test </title></head>
<body>
<form name="myform" action="/test/StoredProcedure"
 method="get">
사번 : <input type="text" name="empno">
<input type="submit">
</form>
</body>
</html>
```

# CallableStatement 예제

```
/* Oracle의 Stored Procedure를 JDBC로 이용하는 예제입니다.
 StoredProcedure.java*/
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
@WebServlet("StoredProcedure")
public class StoredProcedure extends HttpServlet {

 public void doGet(HttpServletRequest req, HttpServletResponse res)
 throws ServletException, IOException {
 Connection con = null;
 Statement stmt = null;

 res.setContentType("text/html; charset=euc-kr");
 PrintWriter out = res.getWriter();
 try {
 // 오라클 드라이버를 Load 한다
 Class.forName("oracle.jdbc.driver.OracleDriver");
 } catch (Exception e) {
 out.println("오라클 드라이버를 Load하는 중 오류가 발생했습니다.");
 out.println("오류 내용 : " + e.getMessage());
 }
 try {
 con = DriverManager.getConnection(
 "jdbc:oracle:thin:@127.0.0.1:1521:xe",
 "scott", "tiger");
 stmt = con.createStatement();
 String sql = "begin
 :out := myproc('Hello');
 end;";
 ResultSet rs = stmt.executeQuery(sql);
 if (rs.next())
 out.println(rs.getString(1));
 rs.close();
 stmt.close();
 con.close();
 } catch (Exception e) {
 out.println("SQL 처리 중 오류가 발생했습니다.");
 out.println("오류 내용 : " + e.getMessage());
 }
 }
}
```

# CallableStatement 예제

```
// 데이터 베이스에 접속을 한다.
con =
DriverManager.getConnection("jdbc:oracle:thin:@211.53.71.228:1521:wink",
"scott", "tiger");

// Statement object를 생성한다.
CallableStatement cstmt = con.prepareCall("{? = call emp_name(?)}");

//Stored Procedure의 input value의 setting 및 output value의 type 설정
cstmt.registerOutParameter(1, Types.NVARCHAR);
cstmt.setInt(2, req.getParameter("empno")); //Procedure의 parameter setting
cstmt.execute(); //CallableStatement 실행

String ename = cstmt.getString(1);

out.println("요청하신 " + req.getParameter("empno") + " 사번의 이름은 " + ename
+ " 입니다.");
}
```

# CallableStatement 예제

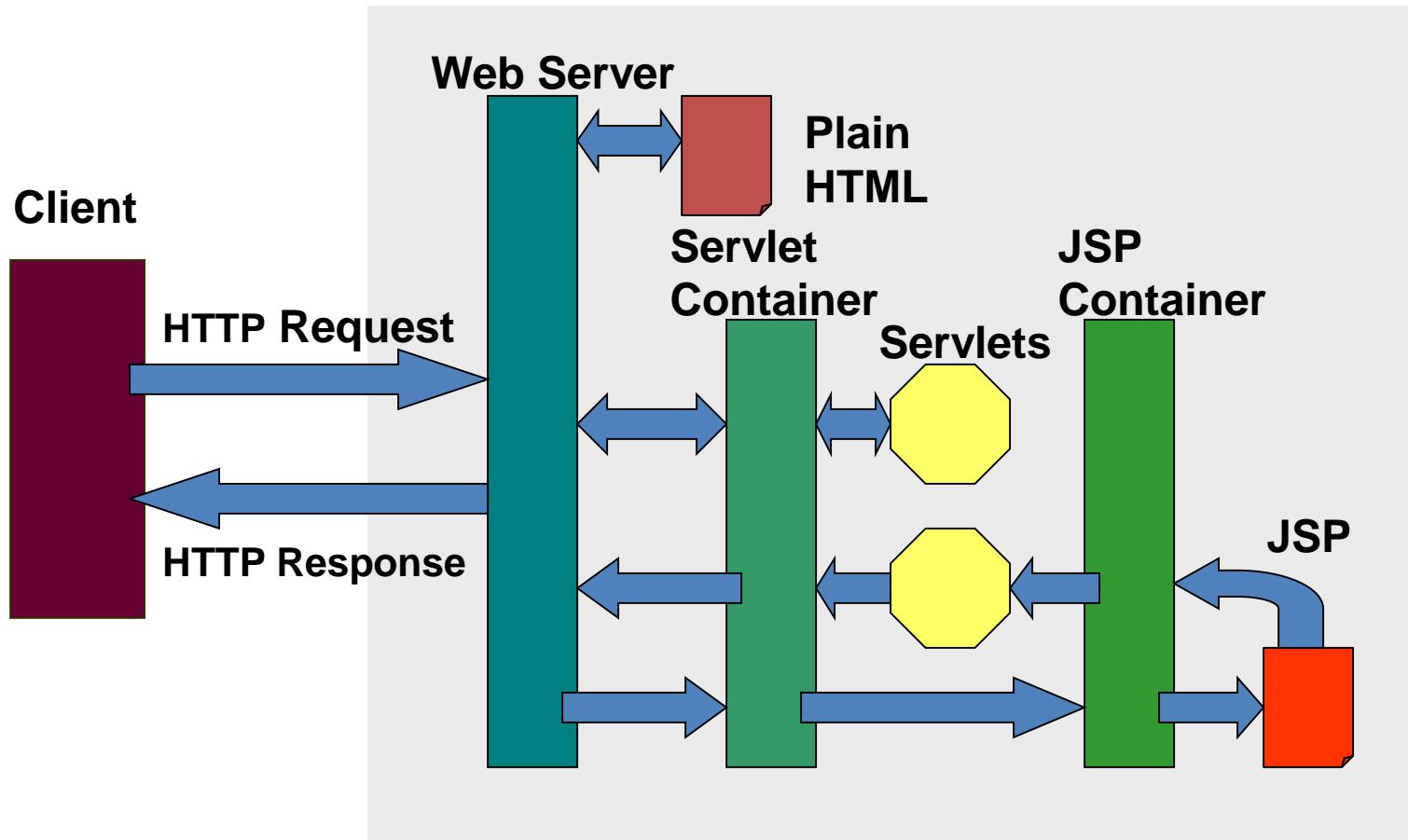
```
catch(ClassNotFoundException e) {
 out.println("Couldn't load database driver: " + e.getMessage());
}
catch(SQLException e) {
 out.println("SQLException caught: " + e.getMessage());
}
finally { // 언제나 데이터 베이스 연결을 종료한다.
 try { if (con != null) con.close(); }
 catch (SQLException ignored) { }
}
}
}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

# JSP 프로그래밍

# JSP 소개

- Java Server Page
- 서블릿과 JSP 차이
  - 서블릿
    - 자바 언어 중심적 동적 파일
    - HTML 응답을 생성하기 보다는 바이너리 응답을 생성
    - 요청 흐름을 제어하는 역할을 주로함
  - JSP
    - 태그 중심적 동적 파일
    - HTML 응답을 생성하여 클라이언트에 보내는 역할
    - 클라이언트의 프리젠테이션이 목적
- 파일 확장명이 **jsp** 이다.
- 서블릿 Class로 변환 후 컴파일 된다.
- 서버에서 객체 생성후 사용된다.
- URL로서 클라이언트가 실행 요청한다.

# JSP 실행 원리



# JSP 샘플

- 확장명 hello.jsp로 텍스트 파일 생성

```
<%@ page language="java"
 contentType="text/html; charset=EUC-KR"
 pageEncoding="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=EUC-KR">
<title>Insert title here</title>
</head>
<body>
```

Hello! This is JSP.

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

# JSP 구성 요소

- Static Content
  - HTML, XML, WML
- Dynamic Content
  - 지시문(Directive): @page, @include, @taglib
  - 스크립팅 요소(코드 요소)
    - 선언(Declarations)
    - 스크립트릿(Scriptlet)
    - 표현식(Expressions)
  - JSP 액션
  - EL(Expression Language)
  - JSTL
- 주석문
  - 태그 주석문
  - JSP 주석문

# 지시문(Directive)

- 지시문의 종류
  - @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.

- @page 지시문

- <%@ page

- [language="java"]

- contentType="{text/plain|text/html|text/xml|}"

- [pageEncoding="{iso-8859-1|euc-kr}"]

- [import="java.util.\*, java.sql.\*"] %>

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

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

- @include 지시문

- <%@include file="포함할 파일의 URL" %>

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

<%@ include file="common/include/footer.jsp" %>
```

- @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" %>
```

# 선언(Declarations)

- 필드와 메서드를 선언
- <%!
  - 필드 선언;
  - 메서드 선언;
  - %>
- 예약된 메서드
  - public void jspInit() {}
  - public void jspDestroy() {}

# 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;
 }

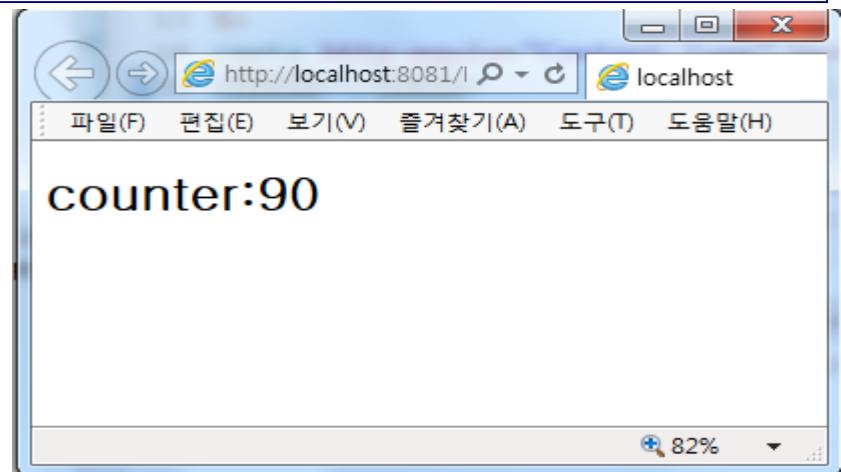
 catch (FileNotFoundException ignored) {}
 catch (IOException ignored) {}
 catch (NumberFormatException ignored) {}
```

# 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 소멸 시 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
```



# 스크립트릿(Scriptlet)

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

# 표현식(Expressions)

- 특정 위치에 문자열로 출력하고 할 때 사용
- <%=단일값%>
- <%=변수|메서드호출|계산식%>

# 객체 사용 범위

- JSP 실행 과정에서 객체를 저장하거나 사용할 위치
- 스코프 종류
  - page : 해당 JSP에서만 사용 가능
  - request : 응답이 출력되기전까지 사용 가능
  - session: 동일한 클라이언트에서 사용 가능
  - application : 모든 클라이언트에서 사용 가능

# 내장 객체

- 스크립트릿에서 객체 생성 없이 사용할 수 있는 객체

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

# request 객체

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

# request 객체의 메소드

- getAttribute() : 속성값을 반환한다.
- getAttribueNames() : 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 HttpServletResponse와 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 오라클자바 메인페이지로 이동합니다...
 <% //아래의 주석부분을 해제후 실행하세요~
 //response.setHeader("refresh", "3;URL='http://www.oraclejava.co.kr'");
 //아래의 코드는 요청된 리소스가 새위치로 이동되었으며, 그위치는
 Location Header에 포함된다는것을 알린다.
 /*response.setStatus(response.SC_MOVED_TEMPORARILY);
 /*response.setHeader("Location","http://www.oraclejava.co.kr");

 //^response.sendRedirect("http://www.oraclejava.co.kr");
 %></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(예를들면 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

- pageContext 객체는 현재 페이지의 제어권을 다른 페이지로 넘기는 방법을 제공한다. 잠시 줄 수도 있고 영구히 줄 수도 있다. <jsp:include> <jsp:forward> 등에 해당되는 include, forward 등의 메소드가 있다.
- JSP 컨테이너에 의해 실행 되기 전 자동으로 서블릿의 javax.servlet.jsp.PageContext 객체로 변환되어 해석된다.

# pageContext 객체 메소드

- 내부 객체를 얻는 메소드
  - getPage(), getServletConfig()
  - getRequest(), getResponse()
  - getOut(), getSession()
  - getServletContext(): Application 객체
  - getException() : exception 객체

# 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.js

```
p<%@ 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("
");
 //pageContext.include("date.jsp");

 //아래의 경우에는 에러난다. 같은 문맥이 아니므로...
 //pageContext.forward("http://www.oraclejava.co.kr");

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

# session 객체

- 세션 범위를 가지며, javax.servlet.http.HttpSession 객체의 한 인스턴스이다. 이것은 요청을 보낸 클라이언트에 대해 생성된 대표하며 HTTP 요청에 대해서만 유효하다.
- 세션은 자동적으로 생성되므로 이 객체는 세션을 사용하지 않는 경우에도 유효하다. 유일한 예외는 page 지시자의 session 특성을 이용해서 session을 끈 경우인데, 그런 경우 이 객체를 참조하려고 하면 컴파일 시점에 오류가 생긴다.
- JSP의 세션 객체는 JSP 컨테이너에 의해 서블릿의 javax.servlet.http.HttpSession 객체로 변환된다. 결국 JSP의 session 객체와 HttpSession은 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)

# session 객체(메소드)

- `isNew()` : 사용자의 브라우저가 세션ID를 할당받지 않았으면 `true`를 `return` 한다.
- `invalidate()` : `session` 객체를 소멸시킨다. `Session`에 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 첫 번째 디렉토리 이름을 애플리케이션으로 사용한다.
- application 범위를 가지며, javax.servlet.ServletContext 클래스의 한 인스턴스이다.
- JSP가 실행되고 있는 문맥(서블릿 문맥)을 대표한다.

# application 객체

- JSP 컨테이너에 의해 자동으로 서블릿의 ServletContext 객체로 변환된다.
- session 객체의 경우 각 사용자마다 하나의 세션을 공유하나, application 객체의 경우 각 서버 내에 있는 모든 JSP File 들은 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 PrintWriter이며 JspWriter는 response 객체를 좀더 유용하게 하기 위해서 PrintWriter에 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 문서로 부터 생성된 서블릿의 인스턴스 그 자체를 가리킨다.
- JSP 페이지 자체를 가르키는 것으로 페이지의 언어가 java인 경우 `page` 객체는 ‘`this`’ 객체와 같은 의미를 갖는다.
- JSP가 서블릿으로 변환된 소스의 `_jspService()` 메소드를 보면 다음과 같은 코드가 보일 것 이가.

`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 서블릿에서는 유용하게 사용되지만 JSP에서는 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`)만에서만 사용이 가능하다.
- 대부분의 컨테이너들은 예외를 `request` 객체의 한 특성으로 포함시켜(`ServletRequest.setAttribute()` 메소드를 이용) 에러페이지에게 전달하고 에러페이지에서는 `request.getAttribute(String name)` 메소드를 이용하여 예외를 뽑아내는 방식을 취하고 있다.

# 내장 객체 예제(1)

```
<!-- resreq.html Id와 비밀번호를 입력받아 가입완료의 표시를 함 -->
<html>
<head>
<title>Respose/Request 예제 </title>
</head>
<body>
<form name="myform" action="/jsp/user.jsp" method="post">
이름 : <input type="text" name="name">

나이 : <input type="text" name="age">

<input type="submit" value="가입">
</form>
</body>
</html>
```

# 내장 객체 예제(1)

```
<!-- age_exception.html 나이가 부정확하게 들어와 NumberFormatException 발생
시 호출됨 */
<html><head><title> Response/Request 예제 </title></head><body>
<h2>나이를 올바르게 넣으세요 </h2></body></html>
```

# 내장 객체 예제(1)

<!-- **user.jsp** name과 age를 받아 나이를 정수로 변환하고(에러발생시 다른 html호출),  
가입OK표시후 Data를 보여주는 JSP →

```
<%@ page contentType="text/html;charset=euc-kr" import="java.io.*" %>
<html><body>
<%
 request.setCharacterEncoding("euc-kr");
 String name = request.getParameter("name");
 String str_age = request.getParameter("age");

 int age=0;
 try { age = Integer.parseInt(str_age); }
 catch(NumberFormatException e) { response.sendRedirect("/age_exception.html"); }
%>
가입OK!!

이름 : <%=name%>

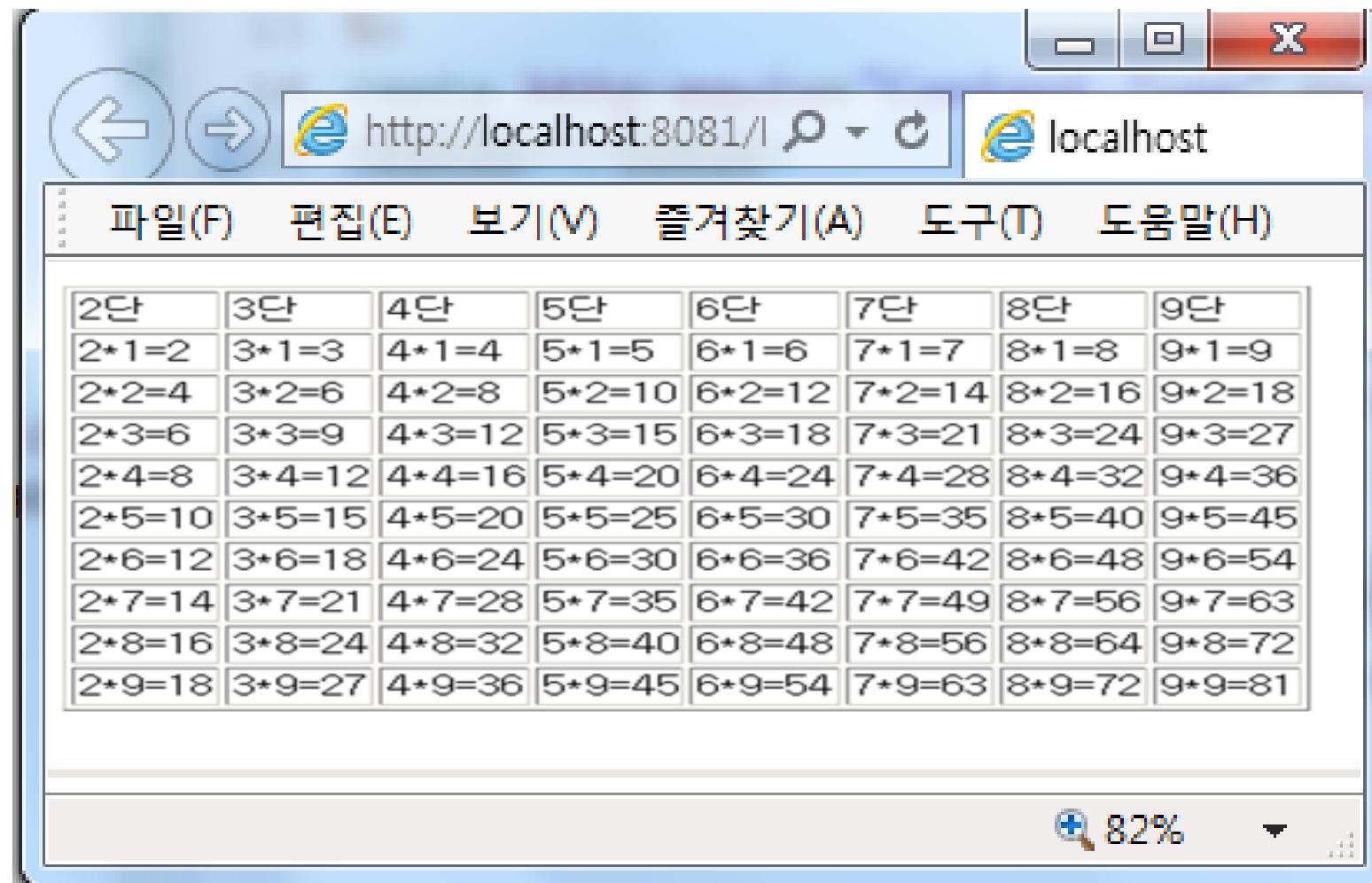
나이 : <%= age %>

</body>
</html>
```

# 내장 객체 예제(2) – 구구단

```
<!-- gugu.jsp 구구단 리스트 프로그램 -->
<%@ page contentType="text/html;charset=euc-kr" %>
<html><head><title>구구단</title></head>
<body><table border=1><tr>
<td>2단</td><td>3단</td><td>4단</td><td>5단</td>
<td>6단</td><td>7단</td><td>8단</td><td>9단</td></tr>
<%
for(int i=1;i<=9;i++) {
 out.println("<tr>");
 for(int j=2;j<=9;j++) { //단
 out.println("<td>");
 out.println(j+"*"+i+"="+i*j);
 out.println("</td>");
 }
 out.println("</tr>");
}
%>
</table></body></html>
```

# 내장 객체 예제(2) – 구구단



# 내장 객체 예제(3) – exception

```
<!-- errorCreate.jsp 일부러 예러를 발생시켜 exception 객체를 Test함 -->
<%@ page errorPage = "errorPage.jsp" contentType="text/html;charset=euc-kr" %>
<% int i = Integer.parseInt("하하하"); %>
```

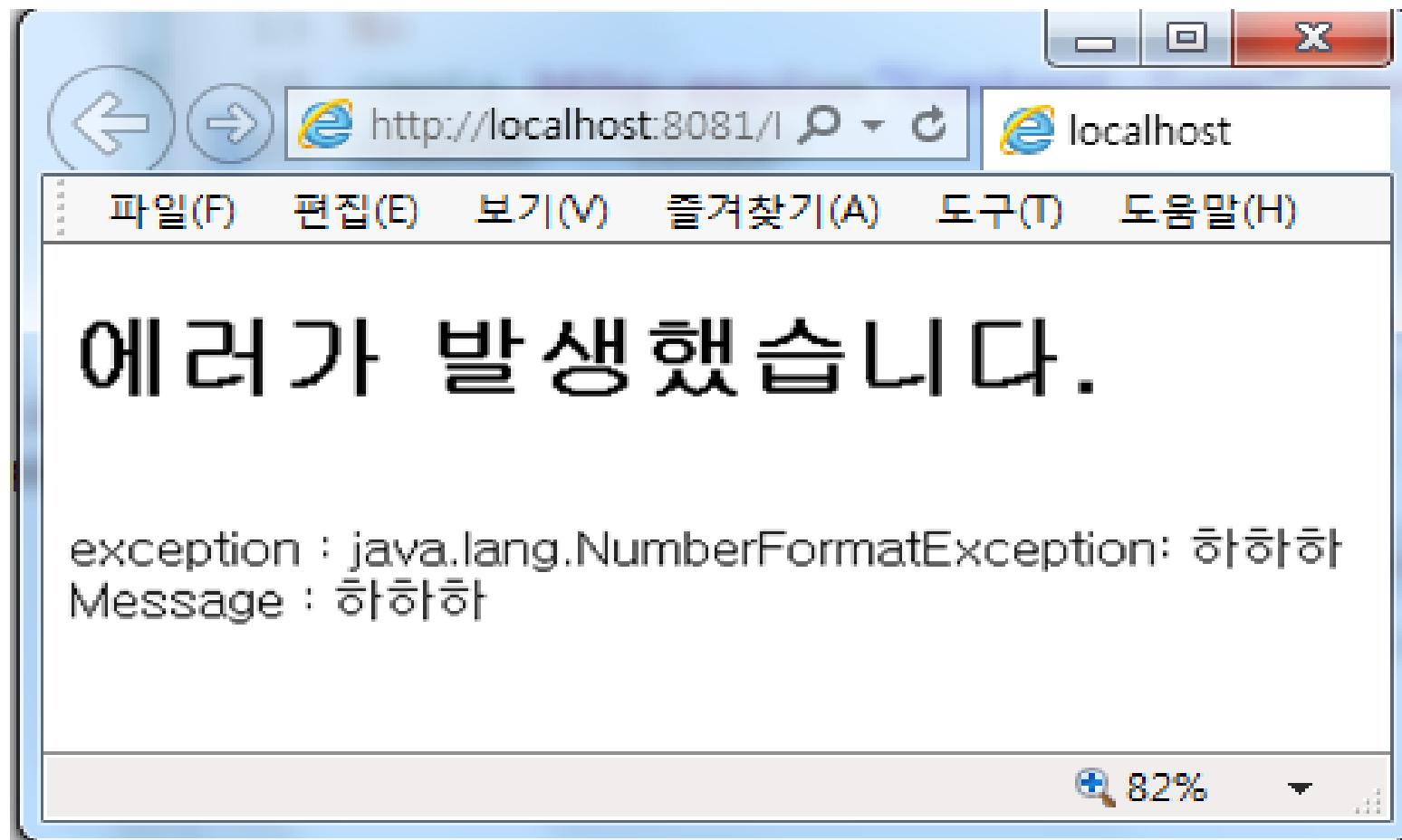
```
<!-- errorPage.jsp 예러발생시 호출되는 페이지 -->
<%@ page isErrorPage="true" contentType="text/html; charset=euc-kr"%>
<html>
<body>
<h1> 예러가 발생했습니다. </h1>

exception : <%= exception.toString() %>

Message : <%= exception.getMessage() %>

</body>
</html>
```

# 내장객체 예제(3) – exception 실행 결과



# 내장 객체 예제(4) – response

```
<!-- redirection.jsp →

<%
response.setContentType("text/html; charset=euc-kr");
//response.setHeader("Refresh", "10; URL=http://oraclejava.co.kr") ;
response.setHeader("Refresh", "10") ;
%>

<html>
<body>
 <% ! int i=0; %>
```

# 내장 객체 예제(4) – response

```
<% i++; %>
<h2><%= i %>번째 방문입니다.

 10초 후에 오라클자바로 이동합니다. ^^

 아님 10초 후에 다시 방문 합니다. --:</h2>

</body>
</html>
```

# 내장 객체 예제(5) – session

```
<html>
<head><title>Session 객체 예제</title>
<meta http-equiv="Content-Type" content="text/html;
charset=euc-kr"/></head>
<body>
<h2>Session Time 시간</h2>
Session 시간 : <%= session.getMaxInactiveInterval() %> sec.
</body>
</html>
```

# 내장 객체 예제(6) – application

```
<html>
<head><title>application 객체 예제</title>
<meta http-equiv="Content-Type" content="text/html;
charset=euc-kr"/></head>
<body>
<h2>application 객체 예제</h2>
Web Application Name : <%=

application.getServletContextName() %>
<%
 application.log("jsp application implicit object test~");
%>
</body>
</html>
```

해당 JSP 컨테이너의 default log 디렉토리에  
서 확인 할 것

# jspInit(), jspDestroy()

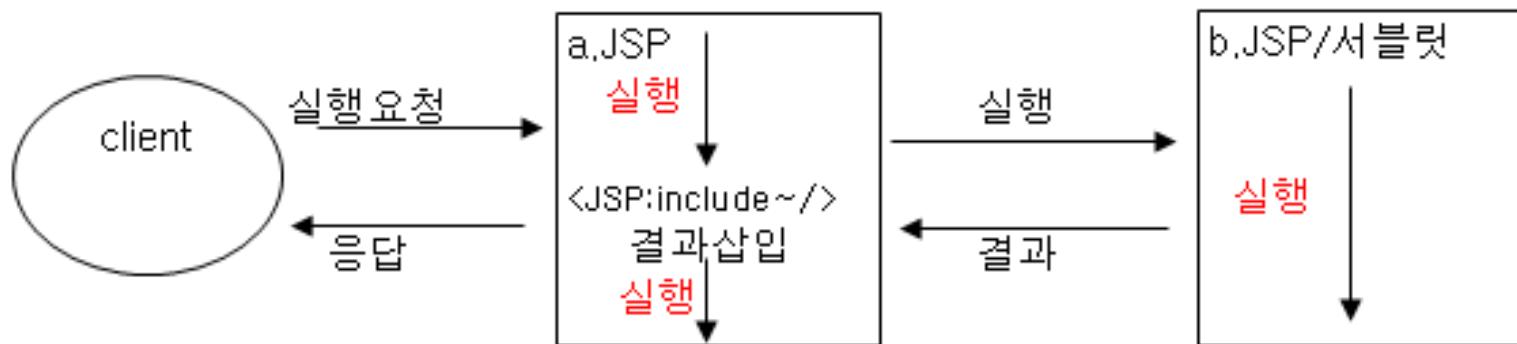
- javax.servlet.jsp.JspPage 인터페이스에 정의
- jspInit()
  - 해당 페이지가 최초로 load될 때 호출된다.
  - 페이지에 서비스를 수행하기 전에 초기화할 루틴을 삽입한다.
  - Optional Method
  - <%! ~ %> declaration 태그에 정의한다.
- jspDestroy()
  - 해당 페이지의 instance가 사라지는 시점에 호출된다.
  - 해당 페이지가 변경되어 새로이 load될 필요가 있을 때 이미 load된 instance는 destroy가 호출되고 새로운 instance의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

# JSP 표준 Action

- JSP 표준 Action?
  - 자주 사용되는 자바 코드를 표준 태그화 시킨 것
  - 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:include>
 

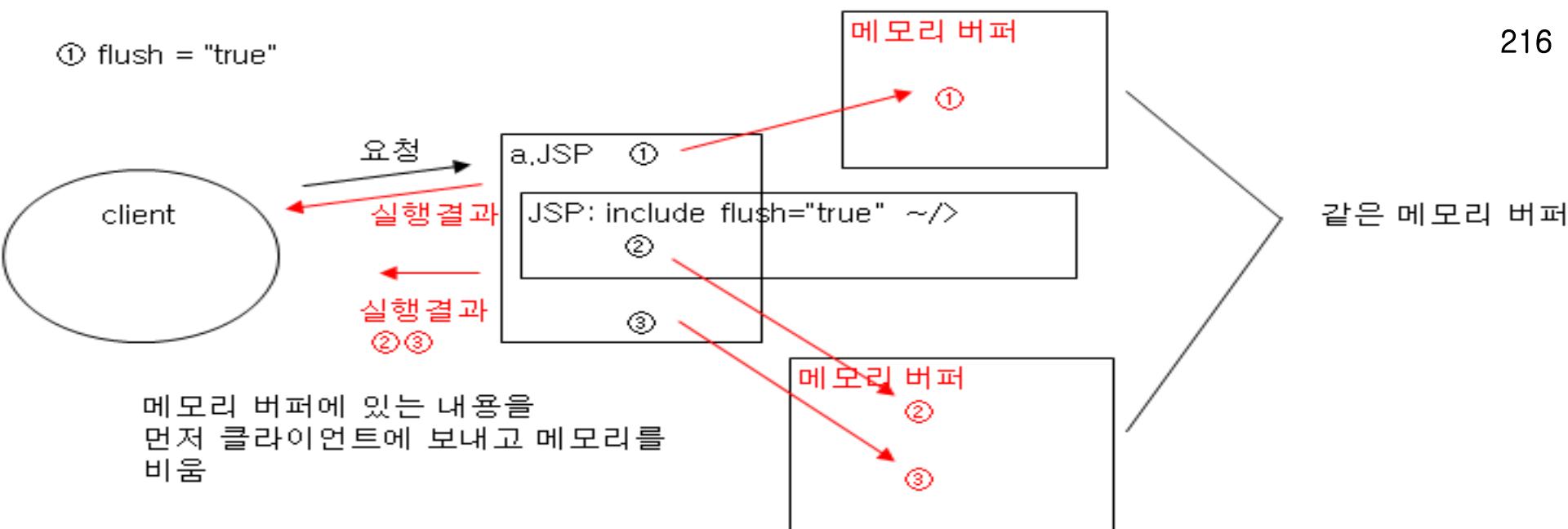
page="정적/동적 파일 url"  
   [flush="true|**false**"] >  
   [<jsp:param name="name" value="value"/>]  
 </jsp:include>



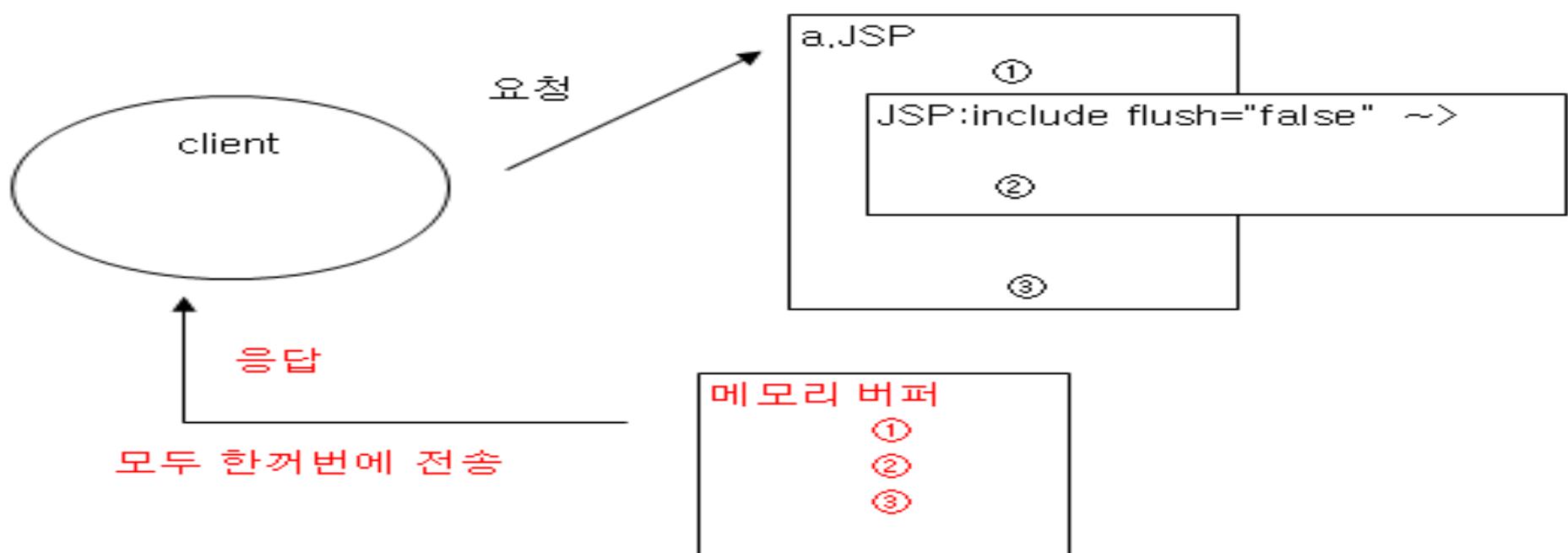
//서블릿 또는 자바 빈즈에서 동일한 효과를 내는 코드

```
RequestDispatcher rd = request.getRequestDispatcher("정적/동적파일url");
rd.include(request, response);
```

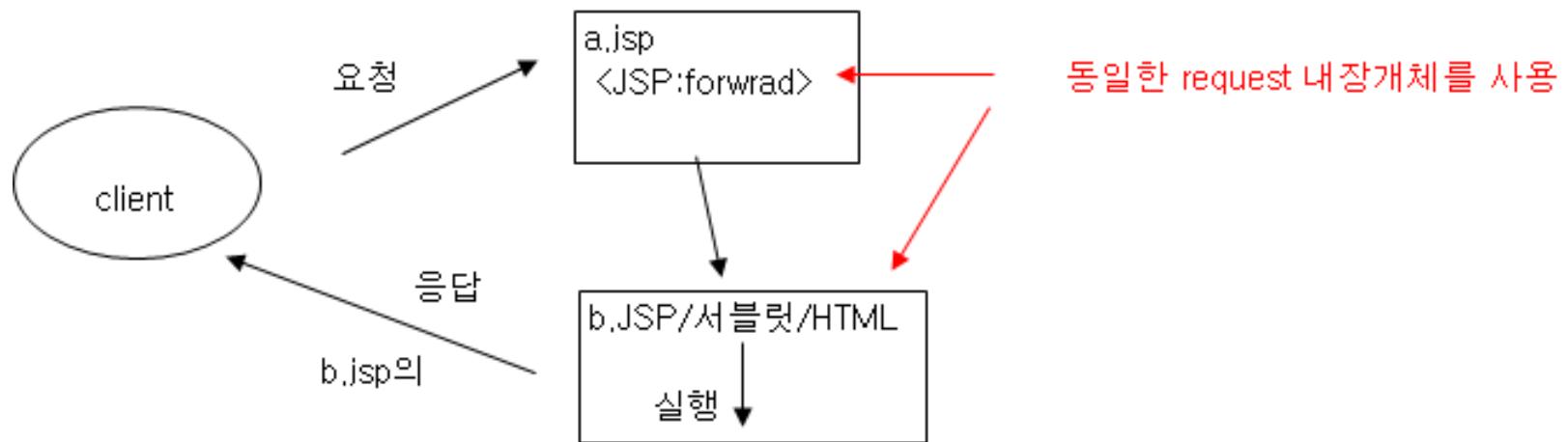
① flush = "true"



② flush = "false"



- <jsp:forward page="이동할url">  
 [<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);
```

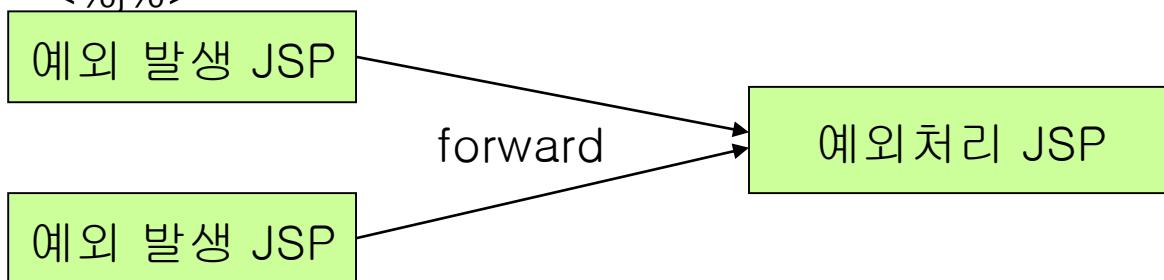
# forward와 redirect

- Forward
  - Reauest 객체 재사용
  - 조금 빠르다.
  - 상단 URL이 바꾸지 않음
  - 컨테이너 안에서 forward 해석을 하므로서 forward한 곳으로 자원을 이용하여 응답을 만들어 보내게 됨, 결국 URL이 바뀌지 않게 됨
- Redirect
  - Request 객체 새로 생성
  - 조금 느리다.
  - 상단 URL이 바뀜
  - 클라이언트에서 서버로 요청을 보내면 서버는 HTTP Protocol 응답헤더 location에 Redirection 될 곳의 주소를 넣어서 응답을 보냄. 클라이언트는 Location 값을 읽어 다시 그 쪽으로 요청을 보냄, URL주소도 당연히 바뀜

# 예외처리

- 예외 발생 JSP에서 직접 처리
  - Try {...} catch(Exception e) {...} finally {...}
- 예외 처리 JSP 이용
  - 예외 발생 JSP
    - <%@page errorPage="예외처리JSP경로"%>
  - 예외 처리 JSP
    - <%@page isErrorPage="true"%>
    - exception 내장객체로 예외 종류 식별

```
<%if(exception instanceof NumberFormatException) {}%>
<%} else if(exception instanceof NullPointerException) {}%>
<%}%>
```



# JSP page에 Error page 연결 예제

- n Error가 발생 했을 경우 해당 하는  
Error page를 생성하는 방법에 대한 간  
단한 사칙연산 예제..

# ErrorPage 예제(NumberSend.jsp)

```
<%@ page contentType="text/html; charset=euc-kr" %>
<!--
 이번 예제를 통하여 예외가 발생했을 경우 이에 해당하는
 에러 page를 만들어 본다.
 숫자가 들어가야 할 자리에 문자를 넣어 본다.
-->
<html>
<head>
<title>Exception handling</title>
</head>
<body>
<center><h3><i>
두개의 숫자를 입력하면 이에 대한 사칙 연산의 결과를 던져 주지..^^.
</i></h3>
```

# ErrorPage 예제(NumberSend.jsp)

```
<form method=Get action="ReceiveTwoNumTest.jsp">
 첫번째 인자 값 : <input type="text" name="firstNum" size=5 >

 두번째 인자 값 : <input type="text" name="secondNum" size=5 >

 <hr>
 <input type="submit" value="전 송">
 <input type="reset" value="다시입력">
</form>
</center>
</body>
</html>
```

# ErrorPage 예제(ReceiveTwoNumTest.jsp)

```
<%@
page contentType="text/html; charset=euc-kr"
errorPage="ExceptionHandler.jsp"
%>
<html>
<head> <title> Result </title>
</head>
<body>
<!-- NumberFormatException을 발생시키는 부분이다
반드시 처리해 주어야 한다. -->
<%
 int firstNum = Integer.parseInt(request.getParameter("firstNum"));
 int secondNum = nteger.parseInt(request.getParameter("secondNum"));
%>
```

# ErrorPage 예제(ReceiveTwoNumTest.jsp)

```
<h4><hr>
덧셈의 결과는 <%= firstNum+secondNum %>

뺄셈의 결과는 <%= firstNum-secondNum %>

곱셈의 결과는 <%= firstNum*secondNum %>

나눗셈의 결과는 <%= firstNum/secondNum %>
<hr>
</h4>
</body>
</html>
```

# ErrorPage 예제(ExceptionHandler.jsp)

```
<%@
page contentType="text/html; charset=euc-kr"
isErrorPage="true"
%>
<html>
<body>
<center>
<%
if(exception instanceof NumberFormatException)
{
%>
 숫자를 입력하는 영역입니다. 반드시 숫자를 입력하세요.
<%
}
else if(exception instanceof ArithmeticException)
```

# ErrorPage 예제(ExceptionHandler.jsp)

```
{
%>
0으로 나눈 결과는 불능이므로 오류입니다. 두 번째 인자 값은 0이 아닌 수
를 입력하세요.
<% }else
{
%>
<%= exception.toString() %>
<%
}
%>
</body>
</html>
```

# 클라이언트 정보 유지

# 접속간 정보 유지 기법

- HTTP 프로토콜은 무 상태 프로토콜
- 상태 유지 기법 종류
  - 숨김 필드: <input type="hidden" ~/>
  - 쿠키: Cookie
  - 내장 객체를 이용해서 객체 사용 범위에 저장
    - session : 동일한 클라이언트만 정보 유지
    - application : 모든 클라이언트가 정보 공유

# 숨김 필드

- <input type="hidden"  
name="paramName"  
value="paramValue"/>
- String varName =  
request.getParameter("paramName");

# 쿠키

- 서블릿 또는 JSP에서 생성
- 저장위치는 클라이언트 메모리 또는 HD
- 문자열만 저장 가능
- 저장형태는 이름과 값으로 저장
- 웹서버로 재접속할때 요청 HTTP 헤더에 포함
- 쿠키 생성 방법

```
String strName = "name";
String strValue = URLEncoder.encode("홍길동", "euc-kr");
Cookie cookie = new Cookie(strName, strValue);
cookie.setMaxAge(intSecond);
cookie.setPath(strURI);
cookie.setDomain(strDomain);
```

- 쿠키값 얻기

```
Cookie[] cookies = request.getCookies();
Cookie cookie = null;
if(cookies != null) {
 for(int i=0; i<cookies.length; i++) {
 String cookieName = cookies[i].getName();
 if(cookieName.equals("name")) {
 cookie = cookies[i];
 }
 }
}
if(cookie!=null) {
 String strValue = URLDecoder.decode(cookie.getValue(),
 "euc-kr");
}
```

# session 객체 이용

- 동일한 클라이언트만 정보 유지
- 객체 저장
  - HttpSession session = request.getSession(); ← JSP에서는 생략
  - session.setAttribute(strName, objValue);
- 객체 얻기
  - Object obj = session.getAttribute(strName);
  - 형변환 필요
- 동일한 session 객체를 클라이언트가 찾는 방법
  - 쿠키: JSESSIONID=xxx 이용
  - 쿠키를 허용하지 않도록 옵션이 되어 있을 경우
    - String urlRewriting = response.encodeURL(strURL);  
`<a href="<% =urlRewriting %>~>`
    - Servlet에서는 encodeURL() 메서드를 사용하기 전에 request.getSession()가 먼저 호출되어야 한다.
    - 테스팅 방법
      - IE6-인터넷옵션-개인정보-슬라이더를 최상단 위치
      - <http://127.0.0.1:8080/~>으로 테스팅(localhost(x))

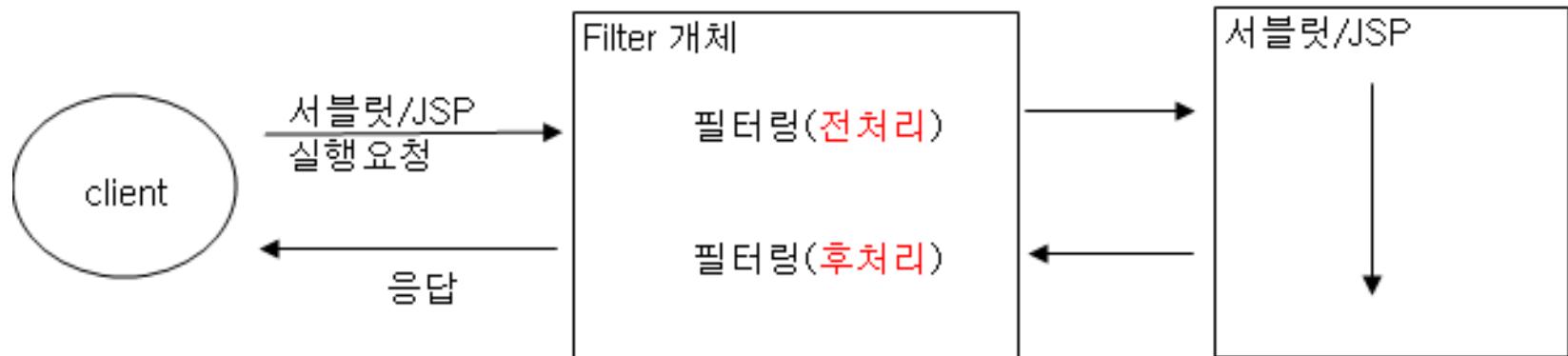
# application 객체 이용

- 모든 클라이언트와 정보 공유
- 객체 저장
  - ServletContext application = request.getSession().getServletContext();  
    ↳ JSP에서는 생략
  - application.setAttribute(strName, objValue);
- 객체 얻기
  - Object obj = application.getAttribute(strName);
  - 형변환 필요

# Filter

# Filter 소개

- 서블릿 스펙 2.3
- 요청을 처리하기 전에 전처리
- 응답을 보내기 전에 후처리



- 필터도 가능한 작업
  - 로깅 및 감사 기능
  - 인증 여부 확인 기능
  - 복호화 및 암호화 기능
  - 문자셋 변환 기능

# 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.*;
import javax.servlet.*;

public class CharacterEncodingFilter implements Filter{
 public void init(FilterConfig filterConfig)
 throws ServletException {
 }

 public void doFilter(ServletRequest request,
 ServletResponse response,
 FilterChain chain)
 throws IOException, ServletException {
 request.setCharacterEncoding("euc-kr");
 chain.doFilter(request, response);
 }

 public void destroy() {
 }
}
```

# Filter 등록

- /WEB-INF/web.xml

- 필터 등록
  - 필터 매팅

```
<filter>
 <filter-name>CharacterEncodingFilter</filter-name>
 <filter-class>filter.CharacterEncodingFilter</filter-class>
 <init-param>
 <param-name>charset</param-name>
 <param-value>euc-kr</param-value>
 </init-param>
</filter>

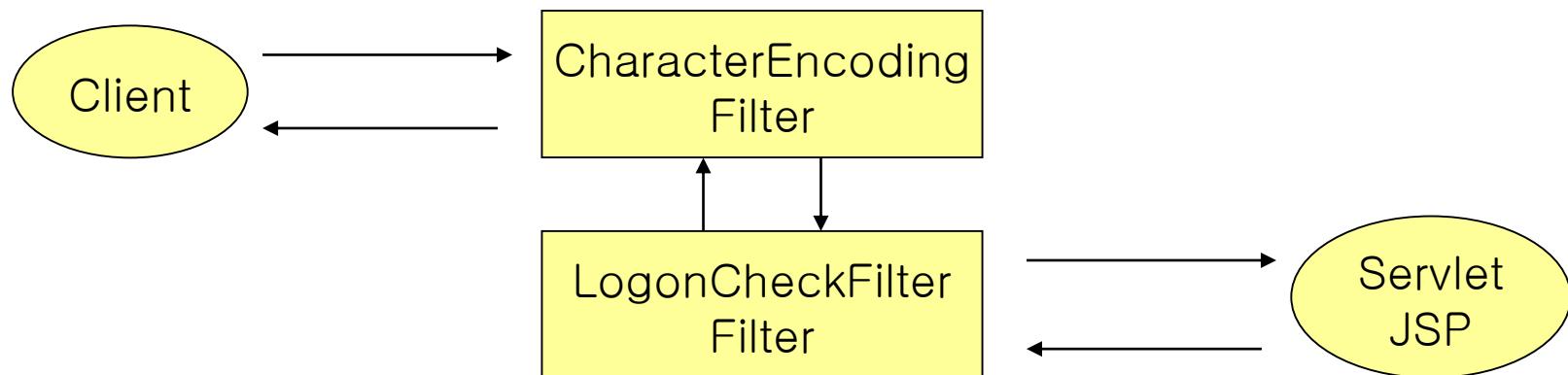
<filter-mapping>
 <filter-name>CharacterEncodingFilter</filter-name>
 <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

# Filter Chain

- 여러 필터를 순차적으로 연결

```
<filter-mapping>
 <filter-name>CharacterEncodingFilter</filter-name>
 <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

```
<filter-mapping>
 <filter-name>LogonCheckFilter</filter-name>
 <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```



# Java Bean

# 자바 빈 소개

- 자바 빈(Java Bean)?
  - 재 사용 가능한 컴포넌트
  - 자바 빈 종류
    - Visual Bean
      - JButton, JTextField → windows program
    - NonVisual Bean
      - 데이터 저장/DB작업/작업흐름 제어
      - JSP Bean, EJB(Enterprise Java Bean)
- 자바 빈즈(Java Beans)?
  - 자바 빈을 설계하기 위한 스펙

# 자바 빈 프로퍼티 설계

```
public class BeanName {
 private datatype propertyName;
 //getter-----
 public datatype getPropertyName() {
 return this.propertyName;
 }
 //datatype이 boolean일 경우
 public datatype isPropertyName() {
 return this.propertyName;
 }
 //setter-----
 public void setPropertyName(datatype propertyName) {
 this.propertyName = propertyName
 }
}
```

# 자바 빈과 관련된 JSP 표준 액션

- <jsp:useBean  
    id=“빈변수명”  
    class=“빈개체형”  
    [type=“빈변수형(형변환:부모클래스 또는 인터페이스)”]  
        : 클래스의 형을 지정하는 생략 가능한 특성, 객체가 이 특성  
        으로 지정된 형이 아니면 ClassCastException 이 발생된다.  
    [scope=“{page|request|session|application}”]>  
  
    [<jsp:setProperty  
        name=“빈변수명”  
        property=“프로퍼티명|\*”  
        [value=“value”]  
        [param=“파라메터명”]/>  
  
</jsp:useBean>

- <jsp:setProperty  
    name=“빈 변수명”  
    property=“프로퍼티명|\*”  
    [value=“value”]  
    [param=“파라메터명”]/>
  - 프로퍼티의 데이터형이 String, 기본형일 경우에 사용
  
- <jsp:getProperty  
    name=“빈 변수명”  
    property=“프로퍼티명”/>
  - 프로퍼티의 데이터형이 String, 기본형일 경우에 사용
  - 프로퍼티가 개체형이면 EL을 사용

# <jsp:useBean> 사용 예

- <jsp:useBean id="myBean" class="package.Class"/>
  - package.Class 타입의 클래스를 myBean이라는 이름으로 초기화 한다는 의미이며 다음과 같은 것이다. Package.Class myBean = new package.Class()
- <jsp:useBean id="myBean" class="package.Class" type="MyType"/>
  - package.Class 타입의 클래스를 myBean이라는 이름으로 초기화 한다는 의미인데 그 Class의 타입이 MyType이라는 의미이다.  
MyType myBean = new package.Class() 그러므로 형 변환을 시도 하며 오류가 발생하면 ClassCastException 이 발생된다

# <jsp:setProperty> 사용 예

- <jsp:setProperty name="myBean" property="name" value="홍길동" />

<jsp:useBean에서 정의된 이름이 myBean인 JavaBean에서 자바빈의 변수(프로퍼티)중 이름이 name인 변수의 값을 "홍길동"로 set

- <jsp:setProperty name="myBean" property="name" param="pass" />

<jsp:useBean에서 정의된 이름이 myBean인 JavaBean에서 자바빈의 변수(프로퍼티)중 이름이 name인 변수의 값을 request.getParameter("pass")의 값으로 set

# Java Bean Example

- `/** 이 빈은 문자열을 뒤집거나 철자검사등을 한다.(SpellCheck.java) */`
- `public class SpellCheck {`
- `private String word;`
- `public SpellCheck() {}`
- `// 문자열을 뒤집는 메소드`
- `public String reverse() {`
- `return (new StringBuffer(word).reverse()).toString();`
- `}`
- `// 철자검사 : 그냥 true를 넘기자`
- `public boolean check() { return true; }`
- `// word 속성에 대한 접근 메소드`
- `public String getWord() {`
- `return word;`
- `}`
- `// word 속성의 값을 설정`
- `public void setWord(String myWord) {`
- `word = myWord;`
- `}`
- `}`

# Java Bean Example

- <!-- SpellCheck.html →
- <html>
- <head>
- <title> SpellCheck </title>
- </head>
- <body>
- <form action=“/jsp/SpellCheck.jsp” method=“post”>
- Enter word : <input type=“text” name=“word”>
- <select name=“mode”>
- <option value=“1” selected>Reverse</option>
- <option value=“2”>SpellCheck</option>
- </select>
- <input type=“submit” value=“submit”>
- </body>
- </html>

# Java Bean Example

- <!-- SpellCheck.jsp →
- <jsp:useBean id="help" scope="request" class="SpellCheck" />
- <jsp:setProperty name="help" property="\*" />
- <html>
- <body>
- you enter the input , <b>
- <jsp:getProperty name="help" property="word" /> </b><br>
- The processed output is : <br>
- <%= Integer.parseInt(request.getParameter("mode")) == 1 ? help.reverse() :  
  ""
- +help.check() %>
- </body>
- </html>

# Tag Extension & Library

# 태그 확장(Tag Extension)

- JSP Code의 문제
  - HTML과 자바코드의 혼재  
(서블릿 유지보수보다 JSP유지 보수가 더 어려울 수 있다.)
  - 표현과 로직의 정확한 분리어려움  
(웹 개발자와 디자이너와의 명확한 역할분담 어려움)
- 이러한 문제의 해결위해 최대한 HTML/XML등과 가까운 템플릿 메커니즘이 필요
- 결국 이상의 이유로 JSP1.1에서 Tag Extension이 도입

# 태그 확장(Tag Extension)

- JavaBean을 사용하기 위한 태그의 종류는 한계가 있으며, 그러한 태그들로는 JavaBean의 Property(속성)에만 접근이 가능하며 세밀한 초기화나 조작 등이 필요한 경우에는 스크립틀릿 안에서 메소드를 호출해야 한다.
- 커스텀 태그 라이브러리는 JSP페이지에서 자바코드를 직접 이용하지 않고 사용 할 수 있도록 하는 기술
- 자바 빈즈를 이용하여 자바코드를 JSP에서 분리 할 수도 있겠지만 자바 빈즈는 독립적으로 실행 될 수 있는 Component이기에 이를 그대로 JSP페이지에서 사용 하기 위해서는 특별한 장치가 필요한데 이것이 곧 태그라이브러리이다.

# 커스텀 태그(Custom Tag)란?

- 사용자가 정의해서 사용하는 태그
- 특징
  - 반복적인 일을 단순하게 만든다.
  - 표현영역과 연산영역을 분리 시킨다.
  - 일관성 있는 웹 페이지의 구성이 가능
  - 재사용
- 일반적인 빈에 비해 태그들은 좀더 풍부한 실행시점 프로토콜을 가진다.
  - 초기화 과정에서 태그의 실행에 꼭 필요한 속성을 설정해야 하는 경우가 있다. 이는 빈의 생성자에 대비

# 커스텀 태그(Custom Tag)란?

- JSP안에서 빈들이 가지는 문맥은 Web Application의 요구에 그리 적합하지 않다. 빈 내부에서는 pageContext 등의 기본 개념이 없다.
- JSP엔진이 하나의 태그를 읽으면 그에 해당하는 Action을 시작하고 (태그 인스턴스 생성) pageContext 객체를 태그 객체에 넘겨준다.

# 커스텀 태그(Custom Tag) 특징

- 자바코드 없는 JSP 페이지를 만들 수 있다.
- 기존태그로 처리 할 때의 복잡함을 단순화 시킬 수 있다.
  - 기존에 존재하는 HTML 태그들을 합쳐서 하나의 새로운 태그로 만들 수 있다.
  - 다음의 경우를 보자.

```
<table>
 <tr><td>학생</td><td>국어</td><td>수학</td></tr>
 <tr><td>이승엽</td><td>100</td>98</td></tr>
</table>
```

위의 태그를 태그라이브러리를 이용하면 다음과 같이 사용 가능 하다.

```
<swlee:table row="2" column="3">
 학생;국어;수학:
 이승엽;100;99
</swlee:table>
```

# 커스텀 태그(Custom Tag) 특징

- 재활용이 가능 하다.
  - 커스텀 태그 라이브러리는 .jar 형태로 압축되어 사용 될 수 있는데 일종의 package로 묶어 활용 할 수 있다는 의미이다. 즉 배포가 가능하다.
- 유지보수가 용이하다.
- 자바 API를 그대로 활용 할 수 있다.
  - 메일 기능을 이용 하기 위해서는 JavaMail을 이용하면 되고 , DB연동을 위해서는 JDBC를 이용하면 된다.

# 커스텀 태그라이브러리의 동작

- 커스텀태그의 수행은 실제 자바클래스가 담당한다. 즉 커스텀태그를 만든다는 것은 원하는 기능의 자바클래스를 설계하는 것이다. 이러한 자바클래스를 **Custom Tag Handler**라고 한다.
- 기본동작
  - JSP에서 태그라이브러리를 사용 할려면 이를 JSP Container에게 알려야 하는데 taglib 지시자를 이용한다.
  - <%@ taglib uri="/WEB-INF/sample.tld" prefix="swlee" %>
    - uri : 해당 커스텀태그 라이브러리에 대한 위치를 알린다.
    - prefix : 사용할 태그라이브러리의 이름을 가리킨다. 접두사
  - JSP콘테이너는 JSP페이지를 읽고 서블릿으로 변환 할 때 taglib지시자를 만나며 이때 .tld 파일이 로드 되었는지 확인하며 만약 로드 되지 않았다면 로드하고, 이미 로드 되었다면 넘어간다.

# 커스텀 태그 라이브러리의 구성

- 커스텀 태그 라이브러리는 TLD(Tag Library Descriptor)와 Custom Tag Handler로 구분 할 수 있다.
- TLD 파일은 JSP 컨테이너가 taglib 지시자를 만났을 때 제일 먼저 찾는 파일이다. 이 파일에는 커스텀 태그에 관한 간략한 정보들이 들어있다. 즉 TLD 파일은 JSP를 서블릿으로 컴파일 할 때 커스텀 태그가 올바른 것인지, 사용 문법에 맞는지 등을 검사 할 때 이용되는 파일이다.
- 태그의 요청이 있다면 핸들러의 작업을 담당하는 자바클래스가 JVM에 로드 되어 수행된다. 물론 로드 된 클래스는 공유가 가능하다. 매 요청마다 일일이 인스턴스를 만드는 것은 아니며 공유하여 사용 한다.
- 보통 커스텀 태그 라이브러리는 JAR 압축파일로 이루어져 있으며 이 파일은 태그 핸들러를 구성하는 자바클래스와 수행되는 태그들의 간단한 정보와 목록이 있는 TLD 파일로 구성되어 있다.

# 커스텀 태그라이브러리의 구성 (TLD)

- TLD 파일은 XML 문서의 형식이다.
- 확장자는 TLD
- 크게 4가지로 구성되어 있다.
  - TLD 문서의 전체 설정
  - 태그라이브러리의 전체 설정
  - 커스텀 태그의 전체 설정
  - 커스텀 태그의 인자 설정

# 커스텀 태그라이브러리의 구성 (TLD)

<!-- XML 헤더 정보를 나타낸다. XML 버전 및 인코딩 정보, DTD 파일 등 지정→  
<?xml version="1.0" encoding="euc-kr"?>  
<!DOCTYPE taglib  
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"  
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary\_1\_1.dtd">  
<!-- 태그라이브러리에 관한 전체적인 설정을 한다.→  
<taglib>  
<tlibversion>1.0</tlibversion> <!--Tag Library의 자체버전. 필수→  
<jspversion>1.1</jspversion> <!--JSP 스펙의 버전, 생략시 1.1, 필수→  
<shortname>swlee</shortname> <!--TagLib의 축약된 이름을 지정, 필수→  
<info>This is TagLib Sample</info> <!--정보나 설명을 적는 부분, 생략 가능→  
<uri>http://localhost/examples/sample.tld</uri> <!--생략 가능, 여러 개의 tld파일로 구분  
하여 사용되는 경우에 이용, 생략 가능→

# 커스텀 태그라이브러리의 구성 (TLD)

<!-- <tag>로 쓰인 부분은 커스텀태그에 관한 설정이다.

<tag><name>hello</name> <!-- 태그의 이름, 이 이름으로 태그를 식별 즉 다음과 같이 사용 할수 있다. <swlee:hello>안녕</swlee:hello> →

<tagclass>beans>Hello</tagclass> <!-- 커스텀 태그를 실제 수행하는 핸들러 , 필수→

<teiclass>beans>HelloTEI</teiclass> <!-- 도우미 클래스 설정, 필수아님 →

<bodycontent>JSP</bodycontent> <!-- JSP에서 커스텀 태그를 사용 할때 시작태그와 끝 태그 사이의 컨텐츠의 태입을 기술, **empty**, **tagdependent**, **JSP**등 사용

<swlee:hello>선생님</swlee:hello>, <swlee:SQL>select \* from emp</swlee:SQL>

<swlee:JSP><%=request.getParameter("name")%></swlee:JSP>의 경우를 예를들면 JSP로 설정되어 있지 않은데 위의 JSP Code가 오면 그대로 나타날 것이다.

만약 SQL문장처럼 질의를 수행하는 경우엔 tagdependent으로 사용하면 된다.

물론 맨위의 <swlee:hello>선생님</swlee:hello>인 경우에도 tagdependent로 사용하면 된다. tagdependent의 의미는 태그사에에 오는 컨텐츠는 그 해당 태그자체에 의해 처리된다는 의미이다.(태그는 일단 내포된 스트림<BodyContent>으로 보내지며 doAfterBody(), doEndTag() 메소드들이 읽어서 처리한다.

<info>인사말을 출력하는 커스텀 태그</info> //추가적인 설명을 기술

# 커스텀 태그라이브러리의 구성 (TLD)

## <attribute>

<name>name</name> <!-- 속성의 이름, 문자열 JSP Page에서는 다음과 같이 사용된다. <swlee:hello name=“홍길동”>, 필수항목 →

<required>false</required> <!-- 필수인지의 여부 →

<rtextrvalue>false</rtextrvalue> <!-- default는 false이며 이것의 의미는 정적인 값을 입력 받겠다는 의미이다. 만약 true로 설정되어 있다면 동적으로 처리가 가능하다. <swlee:hello name='<%=request.getParameter("name") %>'> 즉 커스텀 태그 실행 시에 JSP를 사용할 수 있는지를 나타낸다.

## </attribute>

## </tag>

## </taglib>

# 태그핸들러(Tag Handler)

- Tag Handler를 지원하는 자바클래스와 인터페이스는 javax.servlet.jsp.tagext Package이다.
- JSP페이지에서 커스텀 태그가 실행되는 과정을 요약하면 다음과 같이 정리 할 수 있다.
  - 커스텀태그를 사용하는 JSP페이지 → JSP container → TLD파일 →Tag Handler
- JSP컨테이너는 커스텀 태그를 처리하기 위해 태그 핸들러의 인스턴스를 Resource Pool에서 가져와서 정의된 대로 초기화, 작업을 수행하고 작업 종료 시 Resource Pool로 반환하여 다음에 사용 될 수 있는 상태로 돌아간다.

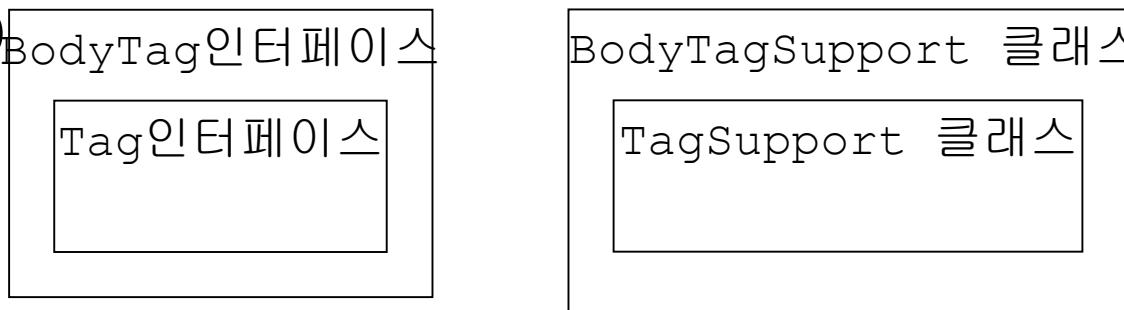
# 태그핸들러(Tag Handler)

- Tag Handler의 작업을 수행하기 위해 그 메소드는 이미 `javax.servlet.jsp.tagext.Tag` 와 `javax.servlet.jsp.tagext.BodyTag` 인터페이스에 이미 정의 되어 있다.(Body를 지원하느냐 아니냐의 차이)
- 태그핸들러를 작성할때는 반드시 `javax.servlet.jsp.tagext package`에 있는 `TagSupport` 클래스와 `BodyTagSupport` 클래스 중 하나만을 선택해서 사용해야 한다. (`TagSupport`의 경우 본체(Body)가 없거나 있어도 그대로 화면에 출력하는 기능을 하기에 본체를 처리하는 `BodyTagSupport` 태그 보다는 간단하다.)

# 태그핸들러(Tag Handler)

- javax.servlet.jsp.tagext.Tag 인터페이스를 구현한 것이 javax.servlet.jsp.tagext.TagSupport 클래스이며 javax.servlet.jsp.tagext.BodyTag 인터페이스를 구현한것이 javax.servlet.jsp.tagext.BodyTagSupport 클래스이다.

- Tag인터페이스를 상속 한 것이 BodyTag 인터페이스이며 TagSupport 클래스를 상속 한 것이 BodyTagSupport 클래스



# 태그핸들러(Tag Handler)

## - Tag 인터페이스 동작순서

### ● 태그핸들러 클래스의 인스턴스 생성

- JSP페이지 내에 커스텀 태그가 사용되면 JSP 컨테이너는 태그 핸들러 클래스의 인스턴스를 생성한다.

### ● Property 설정

- 태그 핸들러의 인스턴스 속성을 설정, 인스턴스 객체를 초기화, 인자초기화
- JSP 컨테이너에 의해 `setPageContext()`가 호출되어 `pageContext` 객체를 가지고 오며, `setParent()` 메소드가 호출된다. 이들 메소드의 역할은 태그 핸들러 클래스에서 만들어진 태그 핸들러 인스턴스를 사용이 가능하도록 한다.
- 커스텀 태그도 일반 태그와 같이 상위태그가 있다면 상위태그를 반드시 설정해 주어야 하는데 이 부분을 JSP 컨테이너가 알아서 처리해 준다.

# 태그핸들러(Tag Handler)

## - Tag 인터페이스 동작순서

### ● 태그에 정의된 속성값 설정

- 자바 빈 태그처럼 미리 태그 핸들러 클래스에 setXXX(), getXXX() 메소드를 설정하여 속성값을 정의 하는 것이다. 만일 태그에서 address라는 속성이 필요하다면 setAddress()와 getAddress()라는 메소드가 만들어져 있다면 JSP 컨테이너가 자동으로 호출하여 작업을 수행한다.

### ● doStartTag() 메소드의 실행

- 실질적인 처리를 담당
- 이 메소드는 실행 된 후 다음 두개의 값 중 하나를 Return 하는데 그 차이는 다음과 같다. **SKIP\_BODY**가 Return 되는 경우에는 시작태그 뒤에 나타나는 본문내용을 수행하지 않고 바로 doEndTag() 메소드를 JSP 컨테이너가 호출하여 실행한다. 또한 **EVAL\_BODY\_INCLUDE**가 Return되면 본문(시작태그와 종료태그 사이의 Content)의 내용이 있을 경우 본문을 수행하고 doEndTag()를 수행하게 된다.

# 태그핸들러(Tag Handler)

## - Tag 인터페이스 동작순서

- 다음의 예를 참조하자.(JSP페이지가 다음과 같이 되어 있다)  
swlee:hello 시작태그가 하는일은 “안녕하세요”를 출력한다고 할때  

```
<%@ page contentType="text/html; charset=euc-kr" %>
.....
<swlee:hello>swlee</swlee:hello>
```

이경우 태그핸들러의 doStartTag()에서 Return되는 값이 SKIP\_BODY라면 다음과 같은 결과가 나타난다.

..... 안녕하세요 .....

doStartTag ()에서 Return되는 값이 EVAL\_BODY\_INCLUDE라면  
..... 안녕하세요 swlee.....

# 태그핸들러(Tag Handler)

## - Tag 인터페이스 동작순서

### ● doEndTag() 메소드의 실행

- doStartTag()의 실행이 끝나면 JSP 컨테이너는 doEndTag() 메소드를 호출한다. 이 작업은 커스텀 태그가 끝날 때 수행되어야 하는 작업이다.
- 수행이 종료 된 후 SKIP\_PAGE와 EVAL\_PAGE의 두개의 값들이 Return 될수 있는데 SKIP\_PAGE인 경우 현재이후의 JSP 코드를 인식하지 않고 처리하지 않는다. 만약 EVAL\_PAGE가 Return되는 경우에는 JSP페이지의 나머지 부분을 수행하게 된다.

### ● release() 메소드의 호출

- JSP 컨테이너는 마지막 단계로 release() 메소드를 호출한다.
- 태그 핸들러가 Resource Pool로 돌아가기 전에 핸들러를 Reset, 사용된 Resource를 해제하는 등의 역할을 수행한다.

# 태그핸들러(Tag Handler)

## - BodyTag 인터페이스 동작순서

- Tag 인터페이스를 확장한 인터페이스로 태그의 내용을 여러 번 반복해서 처리할 수 있는 수단을 제공한다.
- 전체적으로 다음과 같은 순서로 동작한다.
  - 태그 핸들러의 인스턴스 생성
  - Property 생성
  - 태그에 정의된 속성값 설정
  - doStartTag() 메소드 호출
  - 본문의 내용을 초기화 하고 수행함
  - doEndTag() 메소드 호출
  - release() 메소드 호출

# 태그핸들러(Tag Handler)

## - BodyTag 인터페이스 동작순서

### ● doStartTag()

- Tag인터페이스의 동작과 동일하지만 Return 되는 값이 틀리다.
- SKIP\_BODY : 본문을 처리하지 않고 바로 넘어간다.
- EVAL\_BODY\_TAG : 본문의 내용을 처리하여 그 처리결과를 다음처리에서 이용할 수 있도록 저장한다. 즉 처리결과를 BodyContent 클래스를 이용하여 저장하게 된다.

(JSP1.2이상에서는 default로 **EVAL\_BODY\_BUFFERED**를 사용 한다. 새로운 버퍼 즉 이 태그의 본체를 평가하는 BodyContent 의 작성을 요구하며 doStartTag 가 BodyTag 를 구현하고 있는 경우는 doStartTag 로부터 버퍼가 돌려지며 이 클래스가 BodyTag 를 구현하고 있지 않는 경우는 doStartTag 로 반환값 으로 버퍼 가 돌려지지 않는다 )

# 태그핸들러(Tag Handler)

## - BodyTag 인터페이스 동작순서

- 본문의 내용을 초기화하고 수행한다.
  - doStartTag()에서 return value가 EVAL\_BODY\_TAG(JSP1.20이상에서 는 EVAL\_BODY\_BUFFERED)인 경우에 수행된다.
  - EVAL\_BODY\_TAG가 리턴 되면 BodyContent 클래스를 생성하고 초기화한다. BodyContent 클래스는 JspWriter 클래스의 서브 클래스이며 이 클래스를 이용하여 본문의 내용을 읽어오고 수행결과를 페이지에 표시한다.
  - BodyContent 클래스의 인스턴스를 생성 한 후 setBodyContent 메소드를 통해 BodyContent 클래스를 핸들러 클래스에 결합시키며 doInitBody() 메소드는 setBodyContent()를 이용해 초기화 할 때 누락된 부분을 다시 초기화를 진행한다. 즉 처리할 본체를 위한 특성들을 설정한다. doInitBody()는 Return 값이 없다.

# 태그핸들러(Tag Handler)

## - BodyTag 인터페이스 동작순서

- 이상과 같이 초기화를 진행 한 후에는 본문의 내용을 수행하는데 수행결과는 BodyContent 클래스의 인스턴스에 저장된다. 본문의 수행이 종료되면 JSP 컨테이너는 doAfterBody()를 호출한다.
- doAfterBody ()는 BodyContent의 인스턴스를 사용하여 작업을 수행하며 이 메소드가 종료되면 EVAL\_BODY\_TAG (JSP1.2이상에서는 EVAL\_BODY AGAIN) 와 SKIP\_BODY등의 값이 Return 되는데 EVAL\_BODY\_TAG가 Return되면 한번더 doAfterBody () 메소드가 호출되게 되는 것이다. 즉 아직 처리 해야 할 본문의 내용이 더 있다 는 뜻이다. 결국 이 과정을 반복하는 것이다.
- 모든 작업이 완료되면 SKIP\_BODY가 Return 된다.

# 태그핸들러(Tag Handler)

## -BodyTagSupport 클래스 메소드

public class **BodyTagSupport** extends **TagSupport** implements **BodyTag**

- doStartTag : TagSupport의 doStartMethod를 재정의, Default로 EVAL\_BODY\_BUFFERED를 Return)
- doEndTag : TagSupport의 doEndTag를 재정의 하며 , Default로 EVAL\_PAGE를 Return
- doAfterBody : doStartTag의 메소드에서 EVAL\_BODY\_INCLUDE 또는 EVAL\_BODY\_BUFFERED를 Return하는 경우에 호출, 몸체의 처리가 끝났으면 SKIP\_BODY 아니면 EVAL\_BODY\_AGAIN을 Return
- getBodyContent : 몸체의 내용을 읽음, BodyContent 클래스를 Return, BodyContent 클래스의 getString()을 통해 문자를 받아올수 있다.
- getPreviousOut : JspWriter클래스를 Return받아 print메소드를 통해 출력시킨다. 참고로 BodyContent 클래스의 getEnclosingWriter메소드를 이용해도 JspWriter 클래스를 Return 받을 수 있다.

# 인터페이스의 상수들

**Tag** 인터페이스에는 **SKIP\_BODY**, **EVAL\_BODY\_INCLUDE**, **SKIP\_PAGE**, **EVAL\_PAGE** 4가지 상수가 있다. 또한 **BodyTag** 인터페이스에는 **EVAL\_BODY\_TAG**, **EVAL\_BODY\_BUFFERED**가 있으며 **IterationTag** 인터페이스에는 **EVAL\_BODY\_AGAIN**이 있다.

| 상수                       | 메소드                           | 설명                                                         |
|--------------------------|-------------------------------|------------------------------------------------------------|
| <b>SKIP_BODY</b>         | <code>doStartT</code>         | 몸체 처리를 하지 않는다.                                             |
|                          | <code>doAfterBo<br/>dy</code> | 몸체의 반복처리를 중단                                               |
| <b>EVAL_BODY_INCLUDE</b> | <code>doStartT<br/>ag</code>  | 몸체처리를 한다. 즉 몸체의 내용이 출력되고 <code>doAfterBody</code> 메소드가 호출됨 |
| <b>SKIP_PAGE</b>         | <code>doEndTag</code>         | 이후 JSP페이지의 처리를 종료한다.                                       |
| <b>EVAL_PAGE</b>         | <code>doEndTag</code>         | 이후 JSP페이지의 처리를 계속한다.                                       |
| <b>EVAL_BODY_AGAIN</b>   | <code>doAfterBo<br/>dy</code> | 몸체를 계속해서 반복 처리한다. 즉 <code>doAfterBody</code> 메소드가 또 호출된다.  |

# 인터페이스의 상수들

| 상수                 | 메소드         | 설명                                                                              |
|--------------------|-------------|---------------------------------------------------------------------------------|
| EVAL_BODY_TAG      | doAfterBody | EVAL_BODY_AGAIN과 동일하다.<br>JSP1.2에서는 EVAL_BODY_AGAIN을 사용한다.                      |
| EVAL_BODY_BUFFERED | doStartTag  | BodyTag 인터페이스를 구현하지 않은곳의 doStartTag에서는 예러가 발생한다. 이값이 넘어오면 BodyContent 객체가 생성된다. |

# 도우미 클래스(Helper Class)

- Javax.servlet.jsp.TagExtraInfo 클래스를 상속 받아야 한다.
- TLD파일에서 <teiclass>에 기술한 클래스
- 커스텀 태그 라이브러리를 이루는 필수 요소는 아니며, 커스텀 태그에 새롭게 Input되는 변수에 관한 정보를 제공하고 TLD를 이용하여 JSP의 문법을 검사하게끔 한다.
- 태그에 추가된 정보클래스
- JSP 컨테이너는 도우미 클래스의 인스턴스를 생성 한 후 ,도우미 클래스의 getVariableInfo 메소드와 isValid 메소드를 호출
- getVariableInfo : 변수의 정보 제공, VariableInfo 배열을 Return
- isValid : 번역 시간 중에 태그 속성의 유효성을 체크

# 커스텀태그 예제(Hello oraclejava!! 출력)

## custom.tld

(사용자 정의 태그의 tag를 정의 및 이를 수행할 자바 클래스 파일 등을 정의, /WEB-INF/에 저장)

## HelloTag.java

(실제 사용자 정의 태그가 호출되었을 때의 작업을 수행하는 Tag Handler)

## HelloTag.jsp

(사용자 정의 tag를 이용한 jsp file)

## custom.tld

```
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>2.0</jsp-version>
 <short-name>Example TLD</short-name>
 <tag>
 <name>Hello</name>
 <tag-class>beans.HelloTag</tag-class>
 <body-content>empty</body-content>
 </tag>
</taglib>
```

# HelloTag.java

```
package beans;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class HelloTag extends SimpleTagSupport {

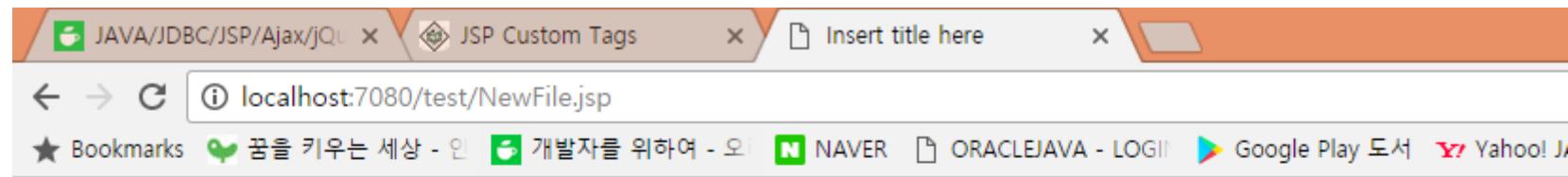
 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out = getJspContext().getOut();
 out.println("Hello OracleJava!");
 }
}
```

# HelloTagLib.jsp

```
<%@ page contentType="text/html;charset=euc-kr" %>
<html>
<title>Simple Tag Extension</title>
<body>
<!-- taglib 지시자는 태그의 동작을 지시하는 XML 문서를 참조한다. -->
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%> <h2><ex:Hello/></h2>
</body>
</html></pre>
```

사용자 정의  
Tag

# 실행결과



Hello OracleJava!

사각형 캡처(R)

## 커스텀태그 예제(마지막 방문일 표시)

### LastVisitTagLib.tld

(사용자 정의 태그의 tag를 정의 및 이를 수행할  
자바 클래스 파일 등을 정의, /WEB-INF/tlds 아래  
에 저장)

### LastVisitTag.java

(실제 사용자 정의 태그가 호출되었을 때의 작업  
을 수행하는 Tag Handler)

### LastVisitTagLib.jsp

(사용자 정의 tag를 이용한 jsp file)

# SimpleTagLib.tld(1)

```
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>2.0</jsp-version>
 <short-name>Example TLD</short-name>
<tag>
 <name>pageVisit</name>
 <tag-class>beans.LastVisitTag</tag-class>
 <body-content>empty</body-content>
</tag>
</taglib>
```

# LastVisitTag.java(1)

```
package beans;
import java.io.IOException;
..
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class LastVisitTag extends SimpleTagSupport {

 @Override
 public void doTag() throws JspException, IOException {
 Cookie info = null;
 String msg = "당신은 이곳을 처음 방문...";
 boolean found = false;
 //pageContext는 범용적인 ServletRequest, ServletResponse를 돌려주
 //며 쿠키는 지원안함
 PageContext pageContext = (PageContext) getJspContext();
 Cookie[] cookies =
 ((HttpServletRequest) pageContext.getRequest()).getCookies();
```

## LastVisitTag.java(2)

```
for (int i=0; i<cookies.length; i++) {
 info = cookies[i];
 if (info.getName().equals("MyCookie")) {
 found = true;
 break;
 }
}
String newValue = "" + System.currentTimeMillis();
if (!found) {
 //쿠키를 새로 생성후 만료일을 표시
 info = new Cookie("MyCookie",newValue);
 info.setMaxAge(60*1);
 info.setPath("/");
 ((HttpServletResponse)pageContext.getResponse()).addCookie(info);
}else {
 long conv = new Long(info.getValue()).longValue();
 msg = "당신의 마지막 방문은 " + new Date(conv);
}
```

## LastVisitTag.java(3)

```
//쿠키에 새값을 설정하고 응답에 추가
info.setValue(newValue);
 info.setMaxAge(10*24*60*60); //10일
 info.setPath("/");
 ((HttpServletResponse)pageContext.getResponse()).addCookie(info);

}

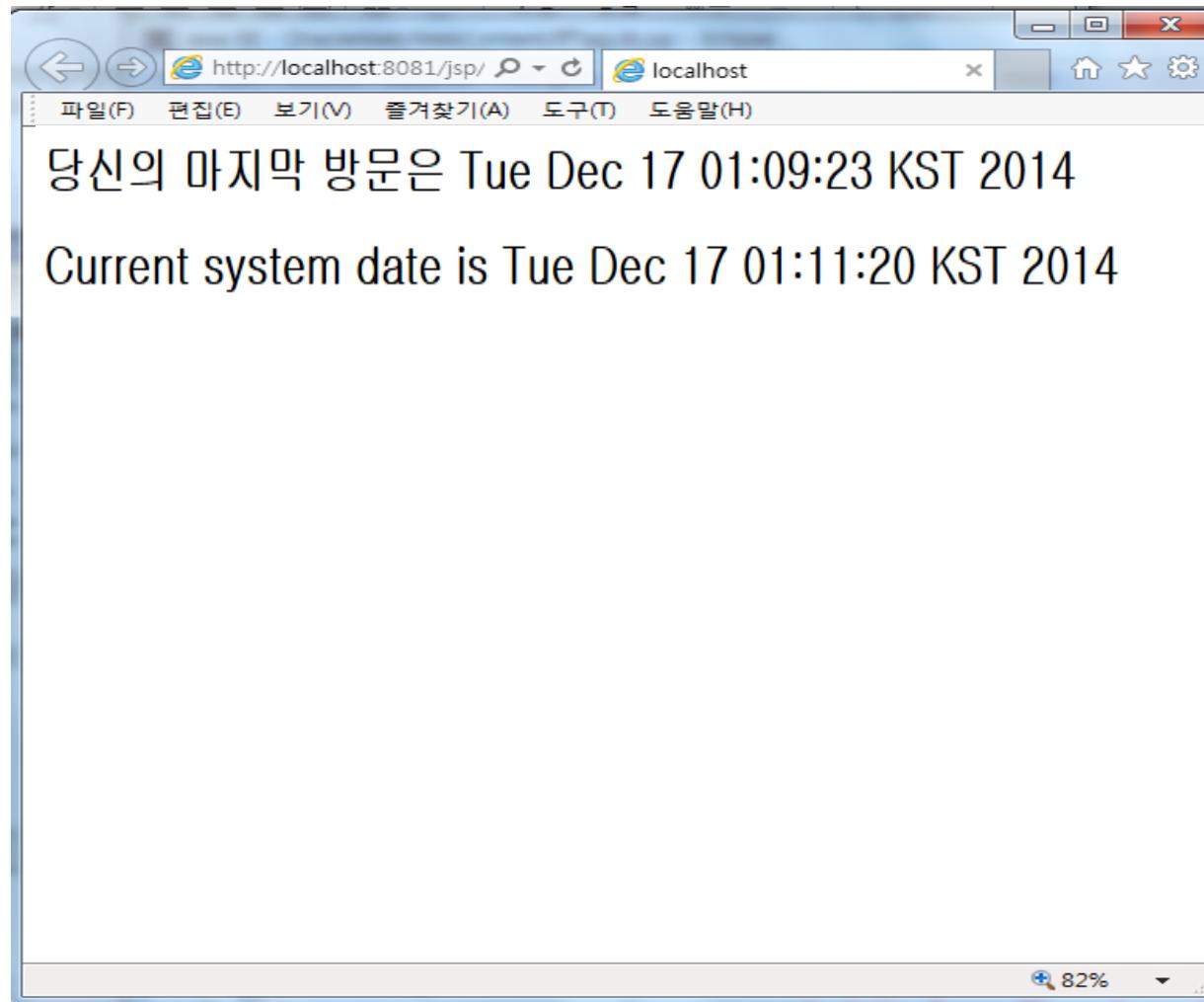
JspWriter out = getJspContext().getOut();
out.println(msg);
}

}
```

# SimpleTagLib.jsp

```
<%@ page language="java" import="java.util.Date" %>
<%@ page contentType="text/html;charset=euc-kr" %>
<html>
<title>Simple Tag Extension</title>
<body>
<!-- taglib 지시자는 태그의 동작을 지시하는 XML 문서를 참조한다. -->
<%@ taglib uri="/WEB-INF/tlds/LastVisitTagLib.tld" prefix="test" %>
<h2><test:pageVisit/></h2>
<h2>Current system date is <%= new Date().toString() %></h2>
</body>
</html>
```

# 실행결과



# Sendmail 프로그램 – TagLib 이용

[\*\*SendMailTagLib.html\*\*](#) : HTML 파일(맨처음 실행됨)

[\*\*SendMailTagLib.jsp\*\*](#) : JSP 파일

[\*\*SendMailTagLib.tld\*\*](#) : Tag Library 정의 파일  
(WEB-INF\tlds 아래 위치)

[\*\*SendMailTagLib.java\*\*](#) : Tag Handler Class

# SendMailTagLib.html (1/1)

```
<form action="SendMailTagLib.jsp" method="POST">
<table><tr> <td>Mail Sending Example</td>
 <td>Examples</td> </tr>
<tr><td>From</td><td>
 <input type="text" size= "20" name="from"></td></tr>
<tr><td>To</td><td>
 <input type="text" size= "30" name="to"></td></tr>
<tr><td>Subject</td>
 <td><input type="text" size= "40" name="subject"></td></tr>
<tr><td>Msg Body</td><td><textarea name= "contents" rows="20"
 cols="38"></textarea></td></tr>
<tr><td></td><td>
 <input type="submit" name= "B1" value="Submit"></td></tr>
</table></form>
```

# SendMailTagLib.jsp (1/1)

```
<%@ page contentType="text/html; charset=euc-kr" %>
<%@ taglib uri="/WEB-INF/tlds/SendMailTagLib.tld" prefix="mail" %>
<HTML><HEAD><TITLE>EmailForm Example</TITLE></HEAD>
<BODY>
<!-- 이페이지는 메일을 보내기 위한 TagLib을 이용한 JSP 페이지 입니다.
-->
<mail:sendmail from='<%=request.getParameter("from")%>'>
 to='<%=request.getParameter("to")%>'>
 subject='<%=request.getParameter("subject")%>'>
 contents='<%=request.getParameter("contents")%>'>
/>
</body>
</html>
```

# SendMailTagLib.tld (1/2)

```
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>2.0</jsp-version>
 <short-name>Example TLD</short-name>

<tag>
 <name>sendmail</name>
 <tag-class>beans.SendMailTag</tag-class>
 <body-content>empty</body-content>
 <attribute>
 <name>from</name>
 <required>true</required>
 <rteprvalue>true</rteprvalue >
 </attribute>
```

## SendMailTagLib.tld (2/2)

```
<attribute>
 <name>to</name>
 <required>true</required>
 <rtextrvalue>true</rtextrvalue >
</attribute>
<attribute>
 <name>subject</name>
 <required>true</required>
 <rtextrvalue>true</rtextrvalue >
</attribute>

<attribute>
 <name>contents</name>
 <required>true</required>
 <rtextrvalue>true</rtextrvalue >
</attribute> </tag>
</taglib>
```

## SendMailTagLib.java (1/5)

```
package beans;

import java.io.IOException;
import java.util.Properties;

...
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SendMailTag extends SimpleTagSupport {

 private String from="";
 private String to="";
 private String subject="";

 private String contents="";
 private String result="";
```

## SendMailTagLib.java (2/5)

```
public void setFrom(String from) {
 this.from = from;
}

public void setTo(String to) {
 this.to = to;
}

public void setSubject(String subject) {
 this.subject = subject;
}

public void setContents(String contents) {
 this.contents = contents;
}
```

## SendMailTagLib.java (3/5)

```
@Override
```

```
 public void doTag() throws JspException, IOException {
 sendMail();
 JspWriter out = getJspContext().getOut();
 out.println(result);
 }
```

## SendMailTagLib.java (4/5)

```
//실제 메일을 보내는 함수
public void sendMail() {
 try {
 Properties property = new Properties();

 property.put("mail.smtp.host", "smtp.gmail.com");

 //Gmail
 property.put("mail.smtp.auth", "true");
 property.put("mail.smtp.starttls.enable", "true");
 property.put("mail.smtp.host", "smtp.gmail.com");
 property.put("mail.smtp.port", "587");
 property.put("mail.smtp.debug", "true");
```

## SendMailTagLib.java (5/5)

```
Session session = Session.getInstance(property, new javax.mail.Authenticator(){
 protected PasswordAuthentication getPasswordAuthentication(){
 return new PasswordAuthentication("지메일id", "지메일비번");
 }
});

MimeMessage mimeMessage = new MimeMessage(session);

InternetAddress toAddress =
 new InternetAddress(to, to);

mimeMessage.setRecipient(Message.RecipientType.TO, toAddress);

InternetAddress fromAddress =
 new InternetAddress(from,from);
```

## SendMailTagLib.java (5/5)

```
mimeMessage.setFrom(fromAddress);

 mimeMessage.setSubject(subject);

 mimeMessage.setText(contents);

 Transport.send(mimeMessage);

 //메일 전송이 성공했다는 출력 생성
 result = "메일이 성공적으로 전송되었습니다...";

 }

 catch (Exception e) {
 System.out.println(e.toString());
 result ="편지가 배달되지 못했습니다. ";
 }
}
```

# 실행결과

The screenshot shows a web browser window with the URL `localhost:7080/test/sendmail.html`. The page displays a form for sending an email. The form fields are as follows:

- From:** `damansa72@gmail.com`
- To:** `damansa1@naver.com`
- Subject:** `나에게 보내는 편지`
- Msg Body:** `보냅니다.  
잘 받으세요.  
수고하세요`

At the bottom of the form is a **Submit** button.

# 몸체를 가진 커스텀 태그

FontTagLib.tld : 태그정의파일(WEB-INF\tlds\에 위치)

FontTag.java: 태그 핸들러 파일

FontTagLib.jsp: TagLib을 이용한 JSP 파일

## FontTagLib.tld (1/2)

```
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>2.0</jsp-version>
 <short-name>Example TLD with Body</short-name>
 <tag>
 <name>Hello</name>
 <tag-class>com.oraclejava.HelloTag</tag-class>
 <body-content>scriptless</body-content>
 </tag>

 <tag>
 <name>font</name>
 <tag-class>com.oraclejava.FontTag</tag-class>
 <body-content>scriptless</body-content>
```

## FontTagLib.tld (2/2)

```
<attribute>
 <name>size</name>
 <required>false</required>
 <rtexprvalue>false</rtexprvalue>
</attribute>
<attribute>
 <name>face</name>
 <required>false</required>
 <rtexprvalue>false</rtexprvalue>
</attribute>

</tag>

</taglib>
```

## FontTag.java (1/2)

```
package com.oraclejava;
import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class FontTag extends SimpleTagSupport {

 private int size = 0; //픽셀
 private String face = ""; //글꼴
 public void setSize(int size) {
 this.size = size;
 }
 public void setFace(String face) {
 this.face = face;
 }
}
```

## FontTagLib.java (2/2)

```
StringWriter sw = new StringWriter();

@Override
public void doTag() throws JspException, IOException {

 if (size == 0) size = 12;
 if ("".equals(face)) face = "돋움";

 getJspBody().invoke(sw);
 getJspContext().getOut().println("<span style=\"font-size:"
 + size + "px;font-family:" + face
 + "\">" +
 + sw.toString() + "");

}

}
```

# FontTagLib.jsp (1/1)

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

<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
 <head>
 <title>A sample custom tag</title>
 </head>
 <body>
 <ex:font face="궁서" size="50">갤럭시S8 화면 크기 5.7인치... 노트 뺑치네
 </ex:font>
 </body>
</html>
```

# 실행화면



갤럭시S8 화면 크기 5.7인치... 노트 뺄치네

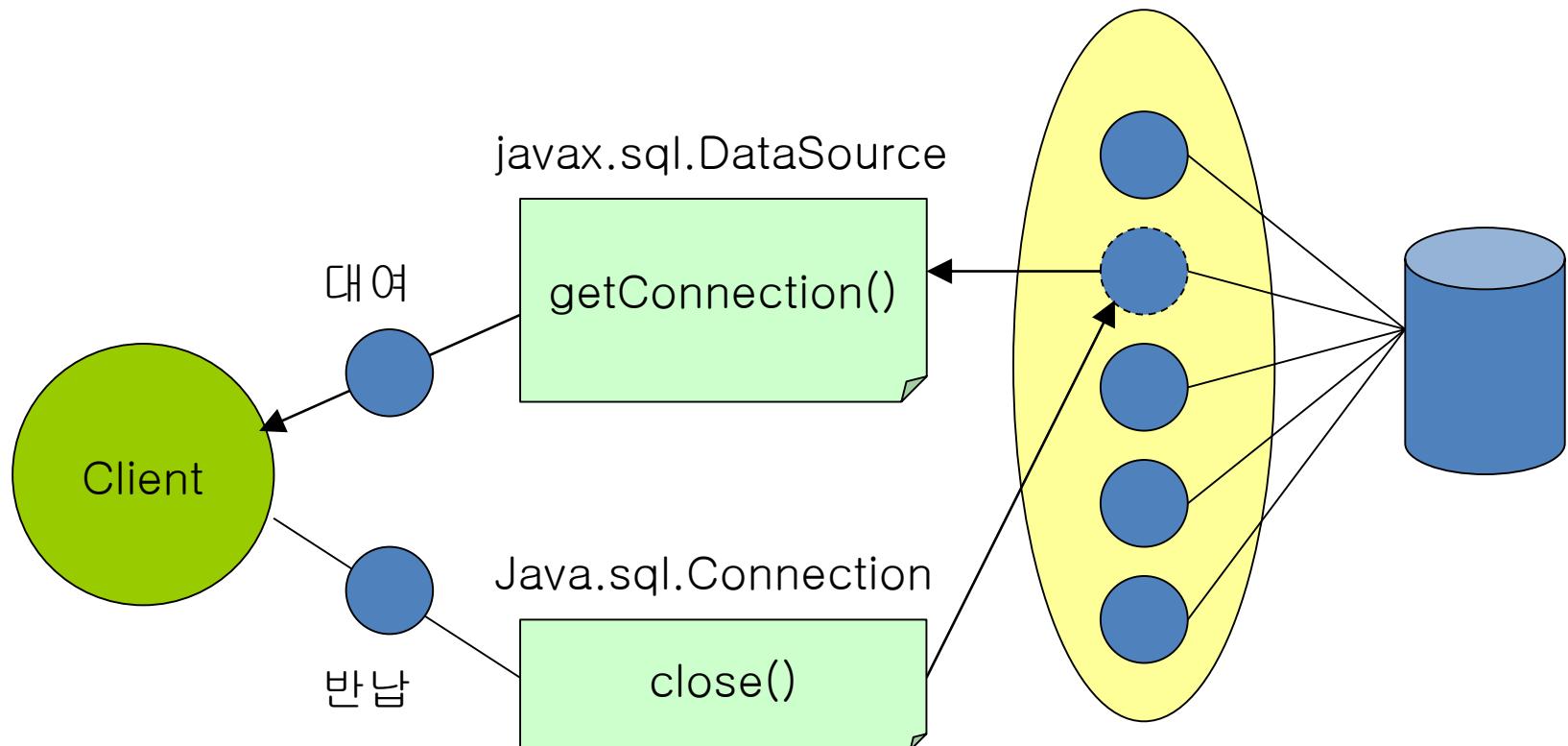
# 고급 JDBC 프로그래밍

# Connection Pooling

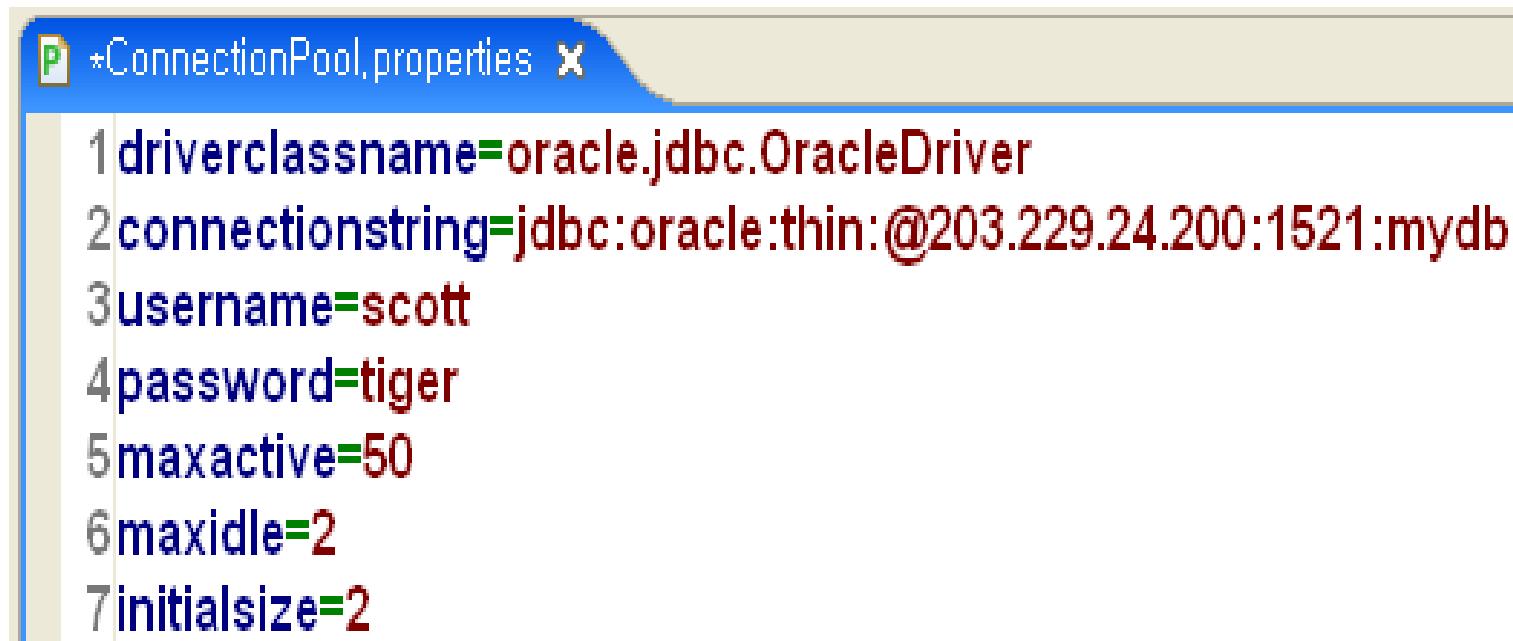
- Connection Pooling?
  - Connection을 미리 생성하여 풀로 관리
  - DB 작업이 필요한 클라이언트에게 Connection 대여
  - 클라이언트가 DB 작업을 끝내면 Connection 반납
  - 적정수의 Connection 관리
  - 비디오 테이프 대여와 유사
- Connection Pooling일 필요한 이유
  - JDBC에서 Connection 생성시 많은 리소스 소비
  - 비즈니스 로직 처리 성능 저하
  - Connection 수 관리가 필요(라이센스와 관련)

## • Connection 대여와 반납

Connection Pool  
(javax.sql.DataSource 구현체)



- Apache DBCP 사용 방법
  - javax.sql.DataSource 구현체
  - /WEB-INF/lib/commons-dbc-p-x.x.x.jar
  - 설정 properties 파일



```
*ConnectionPool.properties
1 driverclassname=oracle.jdbc.OracleDriver
2 connectionstring=jdbc:oracle:thin:@203.229.24.200:1521:mydb
3 username=scott
4 password=tiger
5 maxactive=50
6 maxidle=2
7 initialsize=2
```

# • ConnectionManager 클래스

```
public class ConnectionManager {

 private static BasicDataSource connPool;

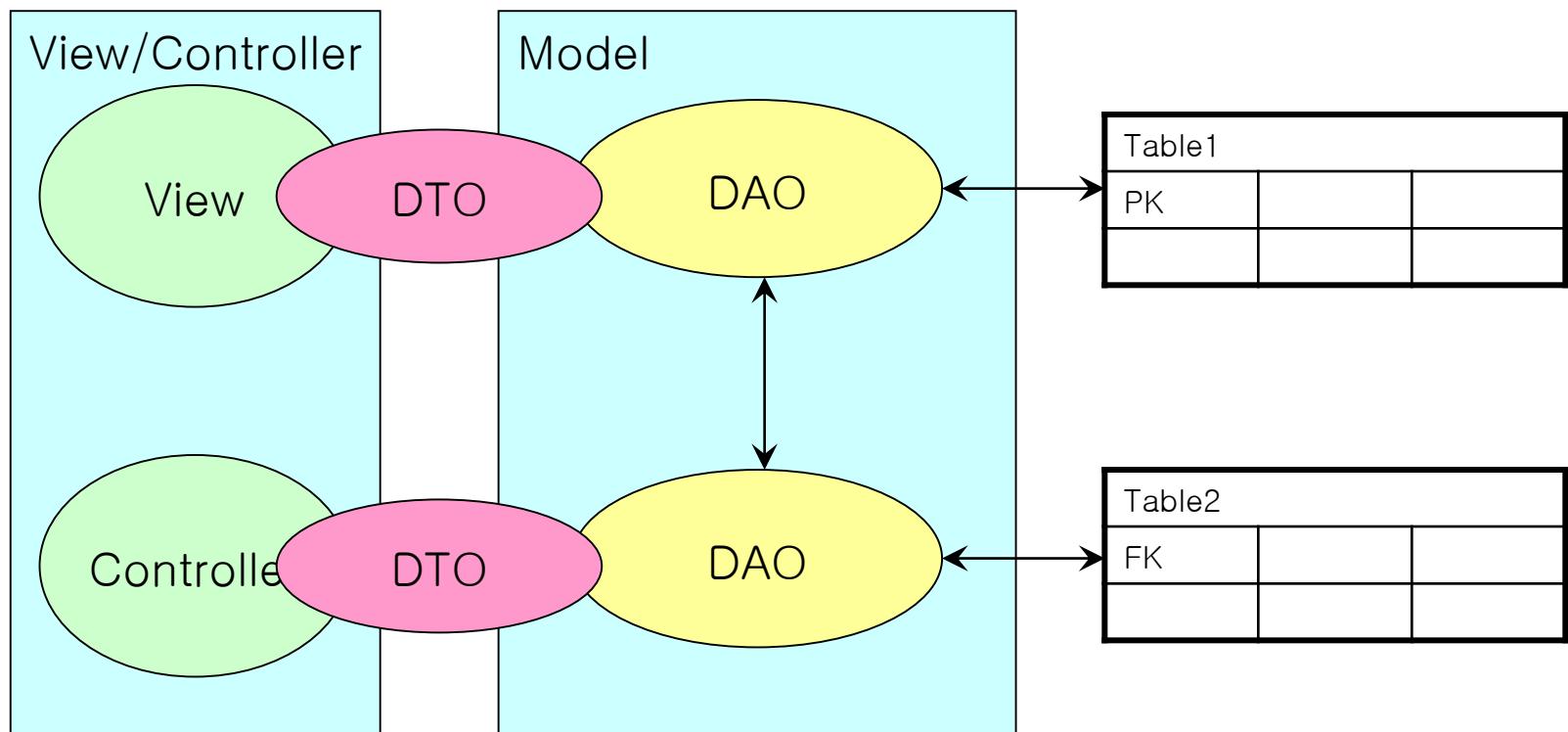
 public static Connection getConnection() {
 try {
 DataSource ds = ConnectionManager.getDataSource();
 Connection conn = ds.getConnection();
 return conn;
 } catch(Exception e) {
 e.printStackTrace();
 return null;
 }
 }

 public static BasicDataSource getDataSource() {
 if(connPool==null) {
 ResourceBundle rb = ResourceBundle.getBundle("model/ConnectionPool");
 connPool = new BasicDataSource();
 connPool.setDriverClassName(rb.getString("driverclassname"));
 connPool.setUrl(rb.getString("connectionstring"));
 connPool.setUsername(rb.getString("username"));
 connPool.setPassword(rb.getString("password"));
 connPool.setMaxActive(Integer.parseInt(rb.getString("maxactive")));
 connPool.setInitialSize(Integer.parseInt(rb.getString("maxidle")));
 connPool.setMaxIdle(Integer.parseInt(rb.getString("initialsize")));
 }
 return connPool;
 }
}
```

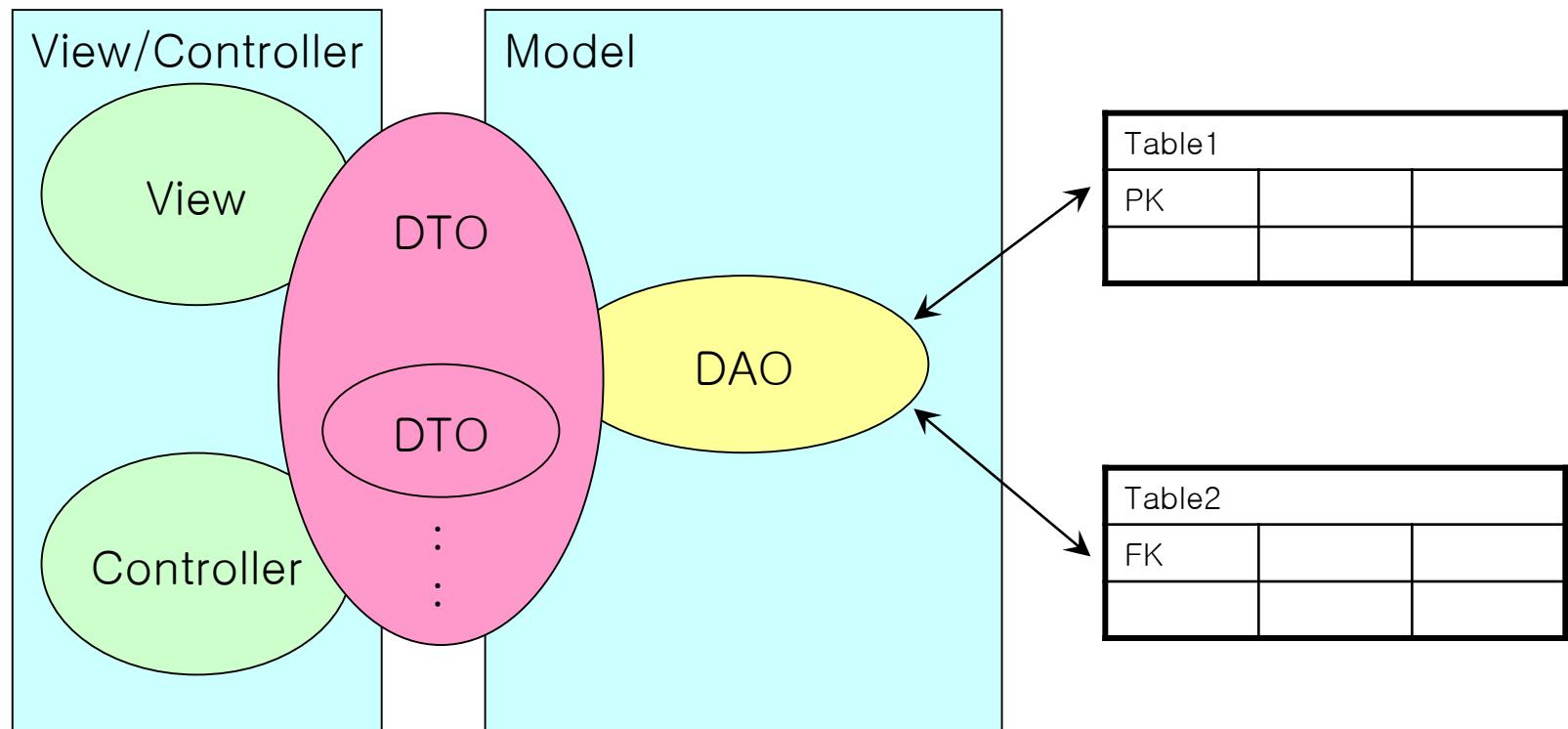
# DAO/DTO

- DAO (Data Access Object)
  - DB에 데이터를 검색/삽입/수정/삭제를 담당하는 객체
  - MVC 모델에서 Model 영역에서 사용
  - 일반적으로 DAO와 DB 테이블은 1:1, 1:n 매팅
- DTO (Data Transfer Object)
  - DAO에서 메서드의 매개변수 또는 반환형으로 사용
  - MVC간의 데이터 전달 목적으로 사용되는 객체
  - 일반적으로 DTO와 DB 테이블은 1:1 매팅

- DAO/DTO 테이블 매팅A
  - Table1과 Table2가 이종의 데이터일 경우
    - 예: 고객과 상품 주문
  - PK로서 대량의 FK를 검색할 경우



- DAO/DTO 테이블 매팅B
  - Table2가 Table1의 확장 데이터일 경우
    - 예: 고객과 고객 취미
  - PK로서 소량의 FK를 검색할 경우



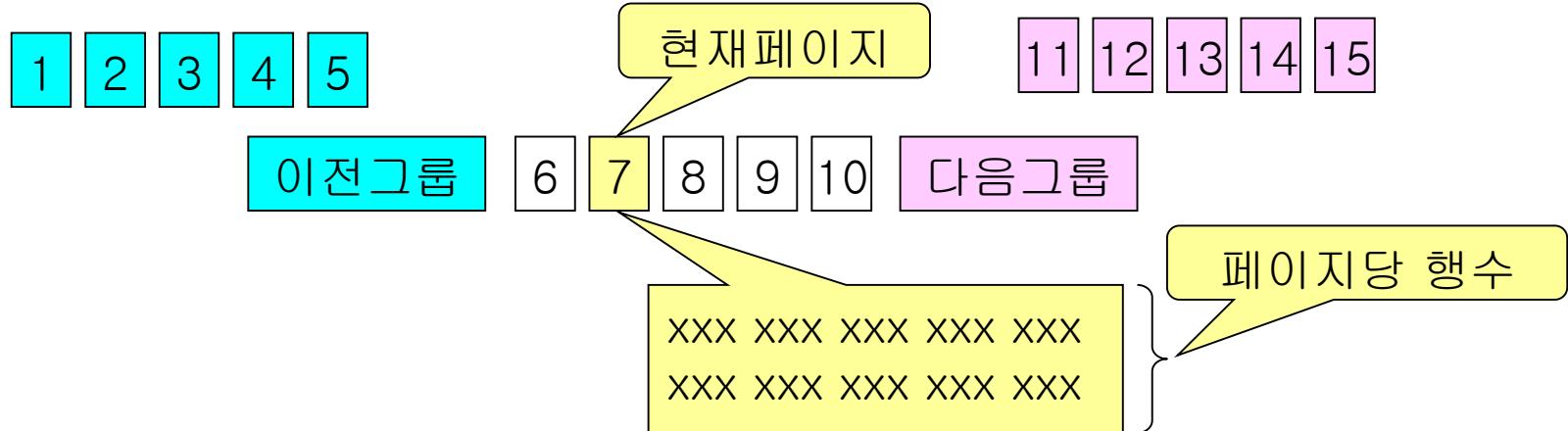
- DTO 클래스 작성
  - public class DB테이블명
  - 각 컬럼
    - 프로퍼티로 정의
      - private 필드
      - public getter,setter 메서드
  - 참조 테이블(매핑B타입)
    - ArrayList형의 프로퍼티로 정의

- DAO 클래스 작성
  - public class DbTableNameDAO

- 기본 메서드
  - int insert(param, ...) / int insert(objDTO)
  - int delete(param, ...) / int delete(objDTO)
  - int updateXXX(param, ...) / int updateXXX(objDTO)
  - ArrayList getList([param, ...])
  - long getCount([param, ...])
- 기타 메서드

# 페이지 처리

- 페이지(Paging)
  - 전체 행을 몇 개의 행으로 구성된 페이지로 나누는 것
- 페이지에 관련된 정보
  - 전체 행 수
  - 전체 페이지 수
  - 전체 그룹 수
  - 페이지당 행 수(rowsPerPage)
  - 현재 페이지(currentPage)
  - 그룹당 페이지 수



- 현재 페이지의 리스트 검색 SQL

- Oracle

- select \*
    - from
    - (
    - select **rownum rnum**, \*
    - from
    - (
    - » select \*
    - » from board
    - » order by bno desc
    - )
    - where **rownum <= (rowsPerPage\*currentPage)**
    - )
    - where **rnum > (rowsPerPage\*(currentPage-1))**

- MySql

- select \*
    - from freeboard
    - Where 조건
    - order by 정렬
    - **limit rowsPerPage\*(currentPage-1), rowsPerPage;**

## • DAO의 페이지 리스트 검색 메서드

```

public class BoardDAO {
 public Vector getPageList(long currentPage, long rowsPerPage){
 Vector vector = new Vector();
 Connection conn = ConnectionManager.getConnection();
 try {
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery(
 "select * " +
 " from " +
 " (" +
 " select rownum rnum, bno, title, writeday, readcount" +
 " from " +
 " (" +
 " select bno, title, to_char(writeday, 'yyyy-mm-dd') as writeday, readcount " +
 " from board order by bno desc" +
 ") " +
 " where rownum <= " + rowsPerPage + " * " + currentPage +
 ") " +
 " where rnum >" + rowsPerPage + " * (" + currentPage + "-1)" +
);
 while(rs.next()) {
 Board board = new Board();
 board.setBno(rs.getLong(2));
 board.setTitle(rs.getString(3));
 board.setWriteday(rs.getString(4));
 board.setReadcount(rs.getInt(5));
 vector.add(board);
 }
 rs.close();
 stmt.close();
 } catch(Exception e) {
 e.printStackTrace();
 } finally {
 if(conn!=null) try { conn.close(); } catch(Exception e) {}
 }
 return vector;
 }
}

```

- 페이지와 관련된 클래스 및 JSP

- util.Pager

- 페이지와 관련된 정보 초기화
    - Eclipse에서 소스 확인

- common/include/pager.jsp

- 페이지모양을 표현
    - Eclipse에서 소스 확인

## • Pager 클래스

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

public class Pager {
 //전체 행수
 private long totalRows;
 //페이지당 보여줄 행 수
 private long rowsPerPage;
 //전체 페이지수
 private long totalPages;
 //그룹당 페이지 수
 private long pagesPerGroup;
 //전체 그룹수
 private long totalPageGroups;
 //현재 페이지 번호
 private long currentPage;
 //현재 그룹 번호
 private long currentPageGroup;

 public void init(HttpServletRequest request,
 long totalRows, long rowsPerPage, long pagesPerGroup) throws Exception {
 //필드에 저장
 this.totalRows = totalRows;
 this.rowsPerPage = rowsPerPage;
 this.pagesPerGroup = pagesPerGroup;
 //전체 페이지수
 totalPages = totalRows/rowsPerPage;
 if(totalRows%rowsPerPage != 0) totalPages++;
 //전체 그룹수
 totalPageGroups = totalPages/pagesPerGroup;
 if(totalPages%pagesPerGroup != 0) totalPageGroups++;
 }
}
```

```
String strFirstPageGroup = request.getParameter("firstPageGroup");
String strPreviousPageGroup = request.getParameter("previousPageGroup");
String strNextPageGroup = request.getParameter("nextPageGroup");
String strCurrentPage = request.getParameter("currentPage");
String strLastPageGroup = request.getParameter("lastPageGroup");
if(strFirstPageGroup!=null) {
 currentPageGroup = 1;
 currentPage = 1;
} else if(strPreviousPageGroup!=null) {
 currentPageGroup = Integer.parseInt(strPreviousPageGroup);
 currentPage = (currentPageGroup-1)*pagesPerGroup + pagesPerGroup;
} else if(strNextPageGroup!=null) {
 currentPageGroup = Integer.parseInt(strNextPageGroup);
 currentPage = (currentPageGroup-1)*pagesPerGroup + 1;
} else if(strCurrentPage!=null) {
 currentPage = Integer.parseInt(strCurrentPage);
 currentPageGroup = ((currentPage-1)/pagesPerGroup) + 1;
} else if(strLastPageGroup!=null) {
 if(totalPageGroups!=0) {
 currentPageGroup = totalPageGroups;
 currentPage = totalPages;
 } else {
 currentPageGroup = 1;
 currentPage = 1;
 }
} else {
 currentPageGroup = 1;
 currentPage = 1;
}

HttpSession session = request.getSession();
session.setAttribute("pager", this);
}
```

```
HttpSession session = request.getSession();
session.setAttribute("pager", this);
}

public long getCurrentPage() {
 return currentPage;
}

public long getCurrentPageGroup() {
 return currentPageGroup;
}

public long getTotalPageGroups() {
 return totalPageGroups;
}

public long getRowsPerPage() {
 return rowsPerPage;
}

public long getPagesPerGroup() {
 return pagesPerGroup;
}

public long getTotalPages() {
 return totalPages;
}

public long getStartPageOfCurrentPageGroup() {
 long result = (currentPageGroup-1)*pagesPerGroup + 1;
 return result;
}
```

- pager.jsp

```
<%@page import="bean.*"%>
<%@page contentType="text/html; charset=euc-kr"%>

<%request.setCharacterEncoding("euc-kr");
String url = request.getParameter("url");
if(url.indexOf("?") == -1) {
 url = url + "?";
} else {
 url = url + "&";
}%>

<%Pager pager = (Pager) session.getAttribute("pager");%>

<table width="100%" border="0" cellspacing="0">
<tr>
 <td align="center">
 <%if(pager.getTotalPages() != 0) {%
 <a href="<%url%>firstPageGroup=true">[First]
 %}>

 <%if(pager.getCurrentPageGroup() > 1) {%
 <a href="<%url%>previousPageGroup=<%pager.getCurrentPageGroup()-1%>">[Previous]
 %}>

 <%long startPage = pager.getStartPageOfCurrentPageGroup();
 long currentPage = pager.getCurrentPage();
 long pagesPerGroup = pager.getPagesPerGroup();
 long totalPages = pager.getTotalPages();%>
```

```
<%for(long i=startPage; (i<=totalPages && pagesPerGroup!=0); i++) {%
 <a href="<%url%>currentPage=<%i%>"%
 <%if(i==currentPage) {%>[<%i%>]<%}
 else {%>[<%i%>]<%}>
 <%pagesPerGroup--;%>
}>%>

<%if(pager.getCurrentPageGroup()<pager.getTotalPageGroups()) {%
 <a href="<%url%>nextPageGroup=<%pager.getCurrentPageGroup()+1%>">[Next]
}>%>

<%if(pager.getTotalPages()!=0) {%
 <a href="<%url%>lastPageGroup=true">[Last]
}>%>
</td>
</tr>
</table>
```

## ■ JSP에서 리스트 표현

```

<%-- DAO 객체 생성 --%>
<jsp:useBean id="boardDAO" class="model.BoardDAO"></jsp:useBean>

<%-- 페이징 정보 초기화 --%>
<%long totalRows = boardDAO.getTotalRows(); %>
<jsp:useBean id="pager" class="util.Pager"></jsp:useBean>
<%pager.init(request, totalRows, 5, 5);%>

<%--현재 페이지 리스트 검색 --%>
<%Vector vector = boardDAO.getagedList(pager.getCurrentPage(), pager.getRowsPerPage()); %>

<%-- 현재 페이지를 테이블로 표시 --%>

| | | | |
|---|---|--------------------------|---------------------------|
| 번호 | 제목 | 날짜 | 방문 |
| <%=board.getBno()%> | <a href=" <u>detailview.jsp?bno=%=board.getBno()%>"><%=board.getTitle()%></u> | <%=board.getWriteday()%> | <%=board.getReadcount()%> |
| <%-- 페이저 표시 --%> <jsp:include page="../common/include/pager.jsp"> <jsp:param name="url" value="list.jsp"/> </jsp:include> | | | |


```

# 저장 프로시저와 함수 호출

- 저장 프로시저(Stored Procedure)와 함수(Function)
  - 호출방식으로 실행가능한 SQL 코드 블록
  - DBMS에서 컴파일하여 저장한 후, 호출 실행하는 방식
- 저장 프로시저와 함수의 차이점은 함수는 호출자에게 단일값(single value)를 반환함.
- 각 DBMS마다 작성 방법이 조금씩 다름
  - Oracle : PL/SQL
  - SQL Server: TSQL
- 장점
  - 네트워크 통신량 절감 및 성능 향상
    - 클라이언트에서 여러가지의 SQL문을 보내는 것 보다, 이들 SQL 문들이 포함된 저장 프로시저나 함수를 한번만 호출함으로서 통신량 절감
      - 컴파일이 미리 되어 있으므로, 재 컴파일 과정이 필요없음
    - 데이터의 무결성 강화
      - 테이블의 데이터를 저장 프로시저 또는 함수로만 제어하므로 무결성이 강화되고, 보안에 유리
- 단점
  - 저장 프로시저와 함수 작성 방법이 다소 어려움

- 테이블

```
create table bankaccount (
 ano varchar(11) primary key,
 aname varchar(8) not null,
 amoney number(20) not null
);
```

```
insert into bankaccount values('111-111-111','홍길동', 10000);
insert into bankaccount values('222-222-222','신민철', 10000);
commit;
```

# • 오라클 저장 프로시저

--procedure

```
create or replace procedure withdraw1 (
```

```
 v_ano in bankaccount.ano%type,
```

```
 v_amoney in bankaccount.amoney%type,
```

```
 v_result out int)
```

**is**

```
 v_old_amoney bankaccount.amoney%type;
```

```
 v_new_amoney bankaccount.amoney%type;
```

**begin**

```
 select amoney into v_old_amoney from bankaccount where ano=v_ano;
```

```
 v_new_amoney := v_old_amoney - v_amoney;
```

```
 if v_new_amoney < 0 then
```

```
 v_result := -1;
```

**else**

```
 update bankaccount set amoney = v_new_amoney where ano=v_ano;
```

```
 commit;
```

```
 v_result := v_new_amoney;
```

**end if;**

**end;**

/

-- testing

```
variable v_result number;
```

```
execute withdraw1('111-111-111', 100, :v_result);
```

```
print v_result;
```

## • 저장 프로시저 호출

//Procedure 호출

```
CallableStatement cstmt1 = conn.prepareCall("{call withdraw1(?, ?, ?)}");
cstmt1.setString(1, "333-333-333");
cstmt1.setInt(2, 1000);
cstmt1.registerOutParameter(3, Types.INTEGER);
cstmt1.execute();
int value1 = cstmt1.getInt(3);
if(value1 == -1)
 System.out.println("잔액이 부족합니다.");
else
 System.out.println("출금후 잔액: " + value1);
cstmt1.close();
```

## • 오라클 함수

```
--function
create or replace function withdraw2 (
 v_ano IN bankaccount.ano%type,
 v_amoney IN bankaccount.amoney%type)
return int
is
 v_old_amoney bankaccount.amoney%type;
 v_new_amoney bankaccount.amoney%type;
begin
 select amoney into v_old_amoney from bankaccount where ano=v_ano;
 v_new_amoney := v_old_amoney - v_amoney;
 if v_new_amoney < 0 then
 return(-1);
 else
 update bankaccount set amoney = v_new_amoney where ano=v_ano;
 commit;
 return(v_new_amoney);
 end if;
end;
/
-- testing
variable v_result number;
execute :v_result := withdraw2('333-333-333', 5000);
print v_result;
```

## • 함수 호출

//Function 호출

```
CallableStatement cstmt2 = conn.prepareCall("{?=call withdraw2(?,?)}");
```

//주어야될 인자값 설정

```
cstmt2.registerOutParameter(1, Types.INTEGER);
```

```
cstmt2.setString(2, "333-333-333");
```

```
cstmt2.setInt(3, 1000);
```

//실행을 해라

```
boolean result = cstmt2.execute();
```

```
System.out.println(result);
```

//결과값 얻기

```
int value2 = cstmt2.getInt(1);
```

```
if(value2 == -1)
```

```
 System.out.println("잔액이 부족합니다.");
```

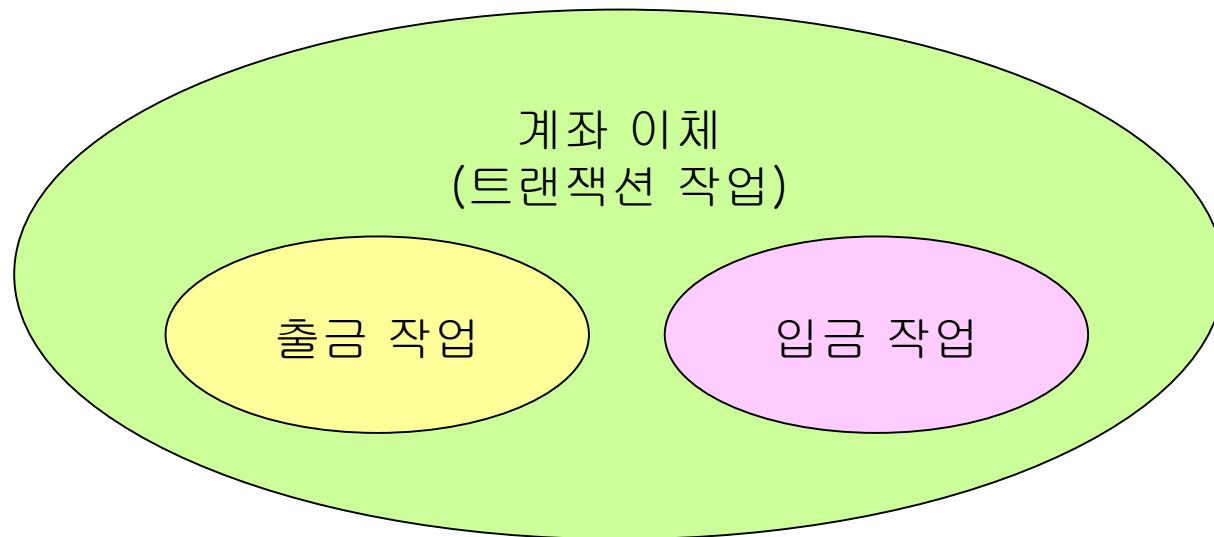
```
else
```

```
 System.out.println("출금후 잔액: " + value2);
```

```
cstmt2.close();
```

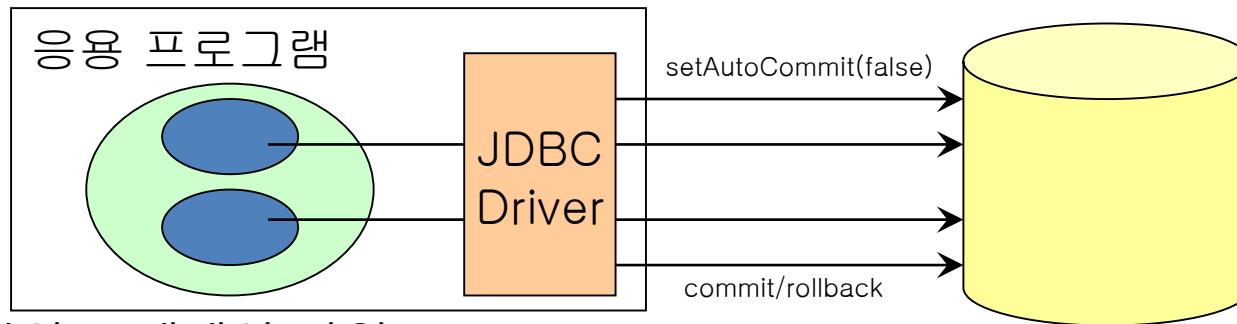
# 트랜잭션 작업 처리

- 트랜잭션(Transaction) 작업
  - 소작업들로 이루어진 하나의 논리적 작업
  - All or Nothing



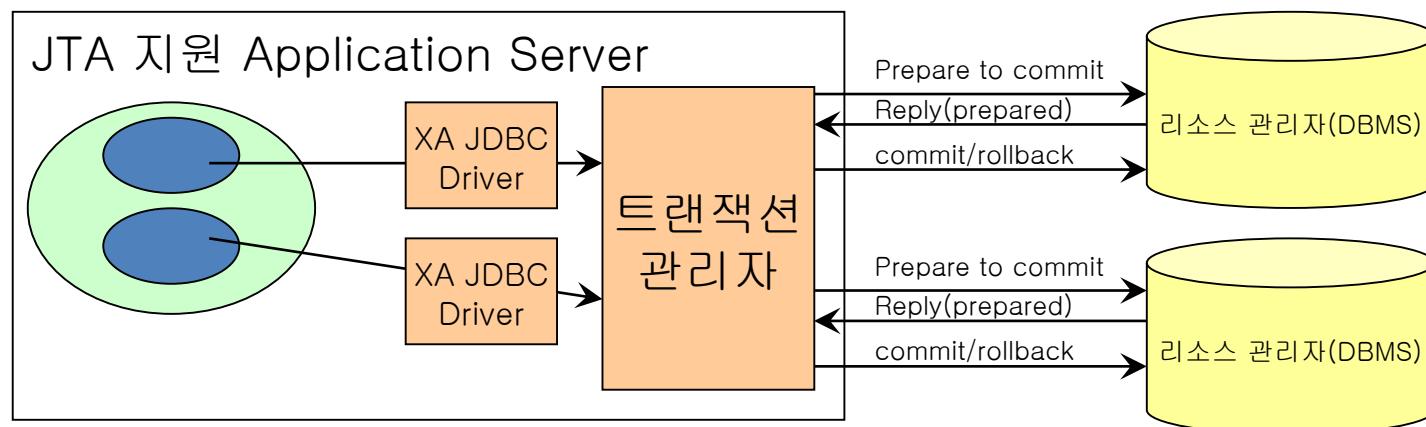
## • 트랜잭션 작업의 종류

- 로컬 트랜잭션 작업
  - 하나의 데이터베이스에서 모든 작업이 진행되는 경우



- 분산 트랜잭션 작업

- 여러 데이터베이스에서 작업이 진행되는 경우
- 2 Phase Commit(2단계 커밋) 이용



- JDBC 드라이버를 이용한 로컬 트랜잭션 구현
  - conn.setAutoCommit(false);
    - 기본적으로 JDBC 드라이버는 단일 SQL문과 함께 commit문을 같이 전송하여 영구적 변경도록 지시
    - setAutoCommit(false)는 commit문을 전송하지 않도록 하여 데이터베이스 메모리에서만 작업도록 지시
  - conn.commit()/conn.rollback();
    - 소작업들이 모두 성공적으로 처리되면 commit()을 실행시켜 메모리의 모든 작업을 영구적으로 변경도록 지시
    - 만약 소작업들 중 한 개라도 실패되면 rollback()을 실행시켜 메모리의 모든 작업을 영구적으로 변경하지 않도록 지시

```
public void transfer(String from, String to, int money) {
 try {
 conn.setAutoCommit(false);
 //Transaction 작업 -----
 withdraw(from, money);
 deposit(to, money);
 //-----
 conn.commit();
 System.out.println("계좌이체 성공");
 } catch(Exception e1) {
 try { conn.rollback(); } catch(Exception e2) {}
 System.out.println("계좌이체 실패");
 } finally {
 try {
 conn.setAutoCommit(true);
 } catch(Exception e) {}
 }
}
```

```
public void withdraw(String ano, int money)
 throws Exception {
 Statement stmt = conn.createStatement();
 int result = stmt.executeUpdate(
 "UPDATE bankaccount SET amoney=amoney-" + money +
 " WHERE ano='' + ano + ''"
);
 if(result==0) throw new Exception("계좌없음");
 stmt.close();
}
public void deposit(String ano, int money)
 throws Exception {
 Statement stmt = conn.createStatement();
 int result = stmt.executeUpdate(
 "UPDATE bankaccount SET amoney=amoney+" + money +
 " WHERE ano='' + ano + ''"
);
 if(result==0) throw new Exception("계좌없음");
 stmt.close();
}
```

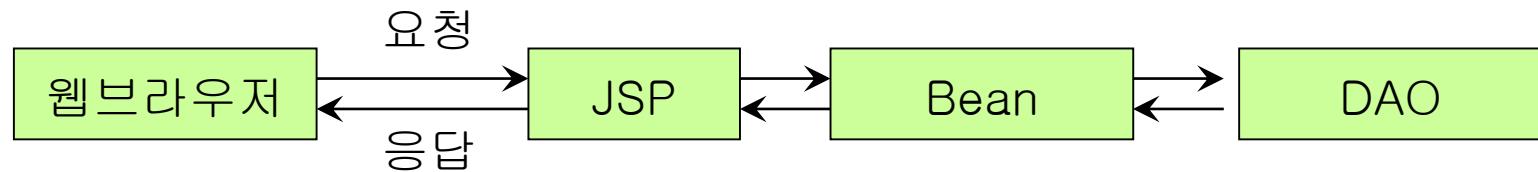
# MVC 개발 패턴

# MVC 소개

- 초기의 웹 응용 프로그램
  - 비즈니스 로직과 프레젠테이션 로직이 서블릿 또는 JSP에 뒤섞여 있다.
  - 개발시 디자이너와 개발자간의 역할 분담이 어렵다.
  - 재 사용성이 떨어지므로 확장성이 좋지 못하다.
- Model
  - 데이터의 계산/삽입/저장/삭제등의 비즈니스 로직에만 집중하는 역할
  - DAO, DTO
- View
  - 클라이언트에게 보여줄 프리젠테이션(표현)에만 집중하는 역할
  - JSP
- Controller
  - 클라이언트의 요청을 처리하기 위해 두 역할간의 제어 흐름을 담당하는 역할
  - Servlet / JSP, 주로 Servlet이 담당
- 효과
  - Model과 View를 따로 개발이 가능해지므로 개발 기간의 단축을 가져올 수 있다.
  - 재 사용성이 높아 확장성에 유리하다.
  - 초기 설계가 어렵다.

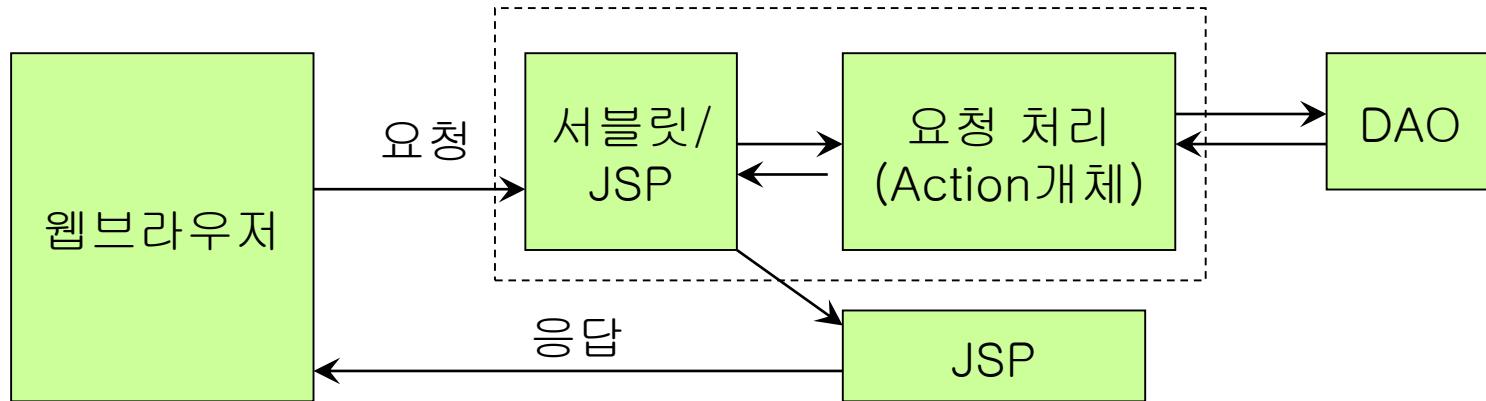
# MVC 패턴 종류

- 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



- Controller: 서블릿, Action 개체
- Model: DAO
- View: JSP
- 모든 클라이언트의 요청은 서블릿이 접수
- 요청에 따라 처리 객체(Action 개체) 선택
- 각 Action 개체는 Model을 이용해서 비즈니스 로직 실행
- 비즈니스 로직 수행 결과에 따라 JSP 선택
- JSP에서 프레젠테이션 생성
- 각 역할별 재 사용성이 높고, 확장성이 우수함
- 컨트롤러 설계시 많은 주의가 필요 → 프레임워크의 필요성이 증대
- 스트럿츠 프레임워크

# EL 소개

- Expression Language
- JSP 2.0에 추가됨
- JSP에서 자바빈즈 내의 데이터에 좀 더 쉽게 접근하기 위한 용도로 정의된 새로운 표현식
- JSP에서 자바코드를 없애기 위함
- JSP 실행시 EL 인터프리터에 의해 번역되어 실행
- JSTL 및 JSF(Java Server Faces)에서 사용

# EL 문법

- **도트(.)연산자: \${firstThing.secondThing}**

- firstThing
  - EL 맵형 내장 객체
  - 스코프 객체에 저장된 속성명(객체 저장명)
    - page, request, session, application 순으로 찾음
- secondThing
  - firstThing이 맵 내장 객체일 경우, secondThing은 키명
  - firstThing이 스코프 객체의 속성일 경우, 빈의 프로퍼티
    - getSecondThing(), setSecondThing(...)

- **[ ] 연산자1: \${firstThing["secondThing"]}**

- fistThing
  - EL 맵형 내장 객체
  - 스코프 객체에 저장된 속성명(객체 저장명)
  - List형 객체, 배열
- secondThing
  - firstThing이 맵 내장 객체일 경우, secondThing은 키명
  - firstThing이 스코프 객체의 속성일 경우, 빈의 프로퍼티
  - firstThing이 List형 객체 또는 배열일 경우 숫자(1 또는 "1")

- **[ ] 연산자2: \${firstThing[secondThing]}**

- secondThing
  - 스코프 객체에 저장된 속성명(객체 저장명)

# EL 내장 객체

- pageContext
  - JSP의 pageContext 내장 객체에 해당
  - JSP에서 얻을 수 있는 대부분의 객체들은 이 객체를 통해 접근할 수 있다.
    - servletContext(application), session, request, response
- 스코프 객체
  - pageScope
  - requestScope
  - sessionScope
  - applicationScope

Ex) <%=session.getAttribute("name")%> → \${sessionScope.name}

- 맵형 객체((key,value)의 쌍으로 저장하는 객체)
  - **param**
    - 파라미터명에 대한 단일/복수 값을 조회  
Ex) <%=request.getParameter("name")%> → \${param.name}
  - **paramValues**
    - 파라미터명에 대한 복수 값을 조회
  - **header**
    - HTTP 헤더의 단일 값을 조회
  - **headerValues**
    - HTTP 헤더의 복수 값을 조회
  - **cookie**
    - 쿠키값을 조회
  - **initParam**
    - 컨텍스트의 초기화 파라미터를 조회

```


[TestBean의 프로퍼티값 얻기]


```

```
<%
TestBean testBean = new TestBean();
testBean.setStringField1("JAVA 언어");
testBean.setStringField2("Servlet/JSP");
testBean.setStringField3("EL");
session.setAttribute("testBean", testBean);
%>
StringField1: ${testBean.stringField1}

StringField2: ${testBean['stringField2']}

StringField3: ${sessionScope.testBean.stringField3}


```

```

[요청 파라미터 값 얻기]

<form method="post" action="el.jsp">
아이디: <input type="text" name="id">

이름: <input type="text" name="name">

취미1: <input type="text" name="hobby">

취미2: <input type="text" name="hobby">

취미3: <input type="text" name="hobby">

<input type="submit" value="전송">
</form>
id: ${param.id}

pw: ${param['name']}

hobby1: ${paramValues.hobby['0']}

hobby2: ${paramValues['hobby'][1]}

hobby3: ${paramValues['hobby'][2]}

[요청 헤더값 얻기]

클라이언트 브라우저: ${header['User-Agent']}


```

```

[쿠키값 얻기]

<%
 String key = "name";
 String value = "Shin Yong Gwon";
 Cookie cookie = new Cookie(key, value);
 response.addCookie(cookie);
%>
name: ${cookie['name'].value}

[컨택스트의 초기화 파라미터값 얻기]

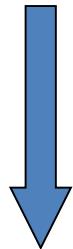
adminEmail: ${initParam.adminEmail}
```

# EL 연산자

- Arithmetic
  - +, -, \*, / and div, % and mod
- Logical
  - and, &&, or, ||, not, !
- Relational
  - ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le.
- Conditional
  - A ? B : C
- empty
  - whether a value is null or empty.
- ()
- ., []

EL Expression	Result	352
<code> \${1 &gt; (4/2) }</code>	false	
<code> \${4.0 &gt;= 3}</code>	true	
<code> \${100.0 == 100}</code>	true	
<code> \${ (10*10) ne 100}</code>	false	
<code> \${'a' &lt; 'b'}</code>	true	
<code> \${'hip' gt 'hit'}</code>	false	
<code> \${4 &gt; 3}</code>	true	
<code> \${1.2E4 + 1.4}</code>	12001.4	
<code> \${3 div 4}</code>	0.75	
<code> \${10 mod 4}</code>	2	
<code> \${empty param.Add}</code>	True if the request parameter named Add is null or an empty string	
<code> \${pageContext.request.contextPath}</code>	The context path	
<code> \${sessionScope.cart.numberOfItems}</code>	The value of the numberOfItems property of the session-scoped attribute named cart	
<code> \${param['mycom.productId']}</code>	The value of the request parameter named mycom.productId	
<code> \${header["host"]}</code>	The host	
<code> \${departments[deptName]}</code>	The value of the entry named deptName in the departments map	
<code> \${requestScope['javax.servlet.forward.servlet_path']}</code>	The value of the request-scoped attribute named javax.servlet.forward.servlet_path	

```
<%Cart cart = (Cart) session.getAttribute("cart");
if(cart != null && cart.getNumberOfItems()>0) { %>
...
<% }%>
```



\*간결하다.  
\*null 처리/형변환을 자동으로 처리한다.

```
<c:if test = "${sessionScope.cart.numberOfItems>0}">
</c>
```

JSTL

EL

# JSTL 구현체 다운로드

- 다운로드
  - <http://tomcat.apache.org/taglibs/>
  - Standard-1.0: JSTL 1.1
    - Servlet 2.5, JSP 2.1
    - Tomcat 6.0 이상

# Standard-1.1: JSTL 1.1 설치

- WEB-INF/lib
  - jstl.jar
    - JSTL API classes: 인터페이스(스펙)
  - standard.jar
    - JSTL의 인터페이스를 구현한 클래스
- 자바 실행 환경
  - J2SE 1.4 이상
    - JAXP 1.2
    - Xalan 2.5
    - JDBC Standard Extension 2.0

# JSTL을 기능별로 분류

Area	Function	URI	Prefix	Example
Core	-변수 지원 -흐름 제어 -URL 관리	http://java.sun.com/jsp/jstl/core	c	<c:tagname ...>
XML	-Core -흐름 제어 -변환	http://java.sun.com/jsp/jstl/xml	x	<x:tagname ...>
I18N	-로케일 -메시지 포맷팅 -숫자,날짜 포맷팅	http://java.sun.com/jsp/jstl/fmt	fmt	<fmt:tagname ...>
Database	-SQL	http://java.sun.com/jsp/jstl/sql	sql	<sql:tagname ...>
Functions	-컬렉션 길이 -문자열 조작	http://java.sun.com/jsp/jstl/functions	fn	fn:functionName(...)

\*<http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html>

# @ 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"%>

# Core Tag Library

Area	Function	Tags	Prefix
Core	Variable support	remove set	c
	Flow control	choose when otherwise forEach forTokens if	
	URL management	import param redirect param url param	
	Miscellaneous	catch out	

# • Variable Support

- <c:set var="varName" [scope="{page|request|session|application}"] value="value"/>
- <c:set var="varName" [scope="{page|request|session|application}"]> value</c:set>
- <c:set target="\${objectName}" property="propertyName" value="value"/>
- <c:set target="\${objectName}" property="propertyName"> value</c:set>
- <c:remove var="varName" [scope="{page|request|session|application}"]/>

모든 스코프에서 검색

```
<%-- 스코프에 데이터 저장 --%>
<c:set var="userid" scope="session" value="white">
userid: ${sessionScope.userid}

<%-- 빈개체의 프로퍼티에 저장--%>
<%-- 이미 세션에 개체가 있다면 생략 --%>
<jsp:useBean id="testBean" class="bean.TestBean" scope="session">
<%-- 모든 스코프에서 개체를 찾은 다음 프로퍼티값을 설정 --%>
<c:set target="\${testBean}" property="stringField1" value="blue">
stringField1: ${testBean.stringField1}

<%-- 맵형 개체에 저장 --%>
<jsp:useBean id="hashMap" class="java.util.HashMap" scope="request">
<c:set target="\${hashMap}" property="name" value="신용권">
name: ${hashMap['name']}

<c:remove var="userid">
<c:remove var="testBean" scope="session">
<c:remove var="hashMap">
```

- Flow Control

- <c:if test="\${condition}">...</c:if>

```
<c:if test="${empty param.id}">
<%--<c:if test="${param.id==null || param['id'] eq ''}"%>--%>
 <c:redirect url="/index.jsp"></c:redirect>
</c:if>
```

- <c:choose>
  - <c:when test="\${condition}"> ... </c:when>
  - <c:when test="\${condition}"> ... </c:when>
  - <c:otherwise> ... </c:otherwise>
- </c:choose>

```
<c:choose>
 <c:when test="${param.color=='red'}">
 ${param.color}
 </c:when>

 <c:when test="${param.color eq 'blue'}">
 ${param.color}
 </c:when>

 <c:when test="${param.color=='green'}">
 ${param.color}
 </c:when>

 <c:otherwise>
 black
 </c:otherwise>
</c:choose>
```

– <c:forEach var="varName" items="\${collection}" varStatus="varStatusName">

...

</c:forEach>

javax.servlet.jsp.jstl.core.LoopTagStatus 개체 변수 참조:

<http://java.sun.com/products/jsp/jstl/1.1/docs/api/index.html>

- Collection, Map, Iterator, Enumeration
- Object[], 기본형[]
- ","로 분리된 문자열
- javax.servlet.jsp.jstl.sql.Result

– <c:forEach var="varName" begin="0"  
– end="10" step="1">

...

</c:forEach>

– <c:forTokens var="varName" items="\${stringOfTokens}" delims="delimiters" varStatus="varStatusName">

...

</c:forTokens>

프로퍼티	설명
index	0으로 시작하는 인덱스
Count	1로 시작하는 인덱스
first	루프의 첫 라운드 여부
last	루프의 마지막 라운드 여부

```
<%@page contentType="text/html; charset=UTF-8"
%>
<%ArrayList bookList = new ArrayList();
bookList.add("JAVA");
bookList.add("Servlet/JSP");
bookList.add("EJB");
request.setAttribute("bookList", bookList);%>

<c:forEach var="book" items="${bookList}" varStatus="loop">
<c:if test="${loop.first}">
 <table border="1" cellspacing="0">
 <tr bgcolor="yellow">
 <td>순번</td>
 <td>제목</td>
 </tr>
</c:if>

<tr>
 <td>${loop.count}</td>
 <td>${book}</td>
</tr>

<c:if test="${loop.last}">
 </table>
</c:if>
</c:forEach>
```

```
<%String cityList = "서울,제주,대전";
request.setAttribute("cityList", cityList);%>

<c:forEach var="city" items="${cityList}" varStatus="loop">
<c:if test ="${loop.first}">
<select>
</c:if>

<option>${city}</option>

<c:if test ="${loop.last}">
</select>
</c:if>
</c:forEach>

<hr/>

<%String jobList = "프로그래머|디자이너|매니저|공무원|교사";
request.setAttribute("jobList", jobList);%>

<c:forTokens var="job" items="${jobList}" delims="|" varStatus="loop" >
<c:if test ="${loop.first}">
<select size="3">
</c:if>

<option>${job}</option>

<c:if test ="${loop.last}">
</select>
</c:if>
</c:forTokens>
```

# • URL management

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

- url 속성으로 지정된 리소스를 가져와 지정된 스코프에 저장
- var 속성이 지정되지 않으면 가져온 리소스를 저장하지 않고 바로 출력한다.
- url 속성에는 외부 웹 서버의 웹 응용 프로그램 Full URL을 지정할 수도 있다.
- 동일한 서블릿 컨테이너에 설치된 다른 웹 응용 프로그램에 리소스가 위치한다면 context 속성에 다른 웹 응용 프로그램 명을 지정해야 한다.

```
<c:import url="importHTML.jsp">

<c:import url="importHTML.jsp" var="importHTML" scope="page">

${importHTML}

${importHTML}

```

- <c:redirect url="url" [context="context"]>  
[<c:param name="name" value="value"/>]  
</c:redirect>

```
<c:if test="${empty param.id}">
<c:redirect url="/index.jsp"></c:redirect>
</c:if>
```

```

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

- 컨텍스트명이 포함된다.
- 요청 매개변수 이름과 값이 URL 인코딩 된다.

```

<c:url value="/" var="contextPath">
${contextPath}

el.jsp

<%-- el.jsp?id=%c8%ad%c0%cc%c6%ae --%>
<c:url value="/el/el.jsp" var="el">
 <c:param name="id" value="화이트"></c:param>
</c:url>
el.jsp

```

# • Miscellaneous

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

- <%=%>와 동일하나 EL을 사용할 수 있다.
- default 속성에는 value 속성을 통해 지정된 값이 null일 경우 출력할 값을 지정한다.
- escapeXml 속성을 true로 지정하게 되면 XML이나, HTML에서 <,>,&,'."을 인코딩하여 출력한다.

```
<c:out value="${param.id}" default="none">

<hr>
<%-- 파일을 읽어 변수로 저장 --%>
<c:import url="importHTML.jsp" var="var2">
<%-- escapeXml="true" --%>
<c:out value="${var2}">
<hr>
<%-- escapeXml="false" --%>
<c:out value="${var2}" escapeXml="false">
```

- <c:catch [var="varException"]>  
    ...  
  </c:catch>

```
<c:catch var="exception">
 <c:set var="x" value="a">
 <c:out value="${10/x}">
</c:catch>

<c:if test="${exception != null}">
 <c:out value="${exception.message}">

 잘못된 변수값
</c:if>
```

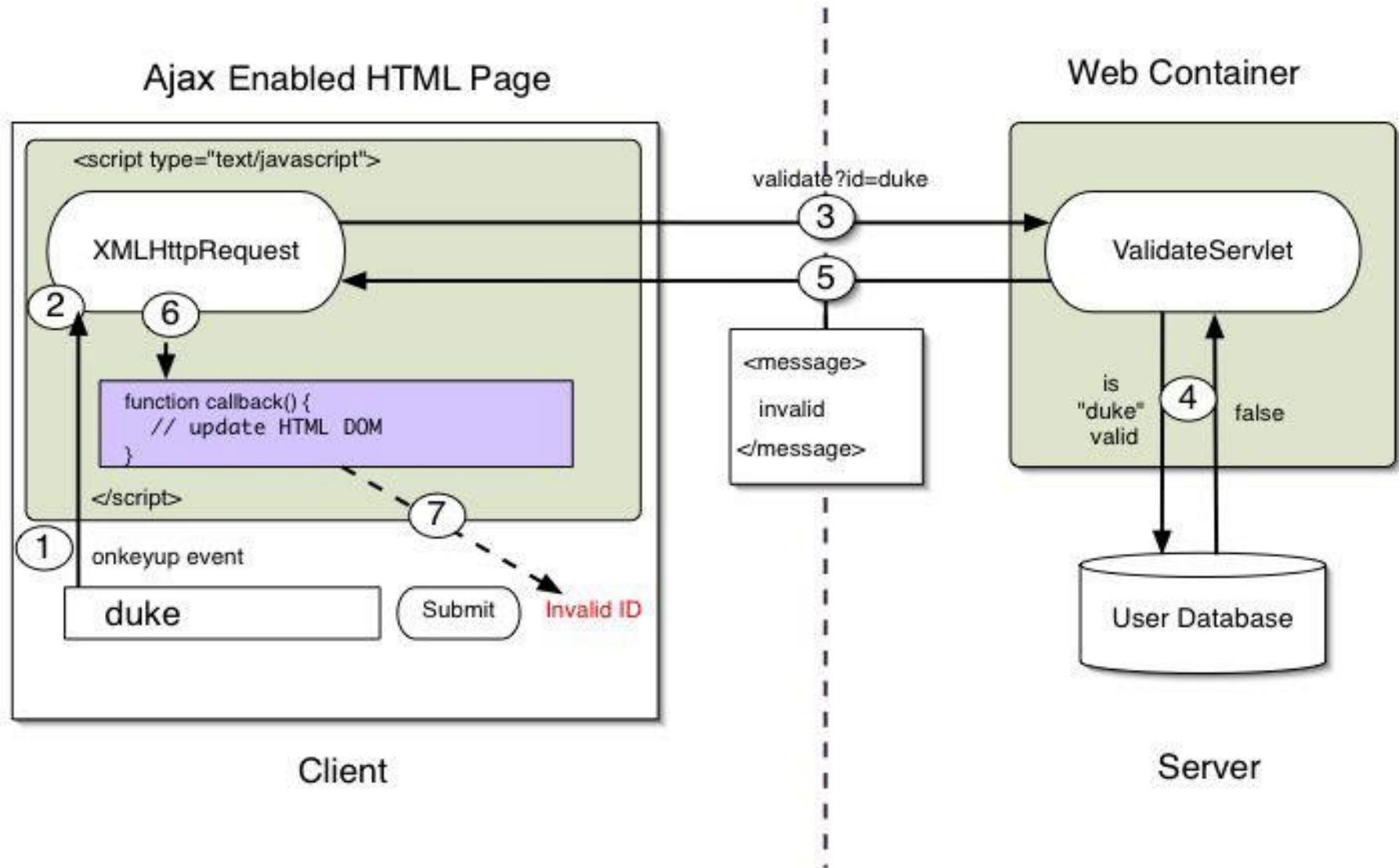
- 메모

# AJAX 프로그래밍

# AJAX 소개

- Asynchronous JavaScript and XML
  - 자바스크립트로 동적 파일 실행 요청
  - 응답이 오기전에 다른 작업 가능
  - 자바스크립트로 HTML 요소 제어
  - 브라우저의 페이지 전체 갱신이 없음
- 적용 사례
  - <http://map.google.com>
  - <http://www.naver.com>

# AJAX 동작 방식



# Ajax에 필요한 JavaScript 기술

- JavaScript 클래스 작성법
- JSON(JavaScript Object Notation) 표기법
- XMLHttpRequest 객체

# 클래스 작성법

```
//클래스 선언(생성자 선언)
클래스이름 = function([파라미터,...]) {
 [this.속성이름 = 파라미터; ...]
}

//속성 선언
클래스이름.prototype.속성이름 = [값|null];

//메서드 선언
클래스이름.prototype.메소드이름 = function([파라미터,...]) {
 ...
}
```

```
<script type="text/javascript">
 Book = function() {}
 Book.prototype.title = null;
 Book.prototype.author = null;
 Book.prototype.getTitle = function() { return this.title; }
 Book.prototype.setTitle = function(title) { this.title = title; }
 Book.prototype.getAuthor = function() { return this.author; }
 Book.prototype.setAuthor = function(author) { this.author = author; }
</script>
```

```
<script type="text/javascript">
 function fun1() {
 var book = new Book();
 book.title = "Ajax";
 book.setAuthor("신용권");
 alert(book.title);
 alert(book.getAuthor());
 }
</script>
```

# JSON 표기법

- JavaScript Object Notation
  - JavaScript로 개체를 표현하는 또 다른 방법
- 개체 생성법

```
var 변수 = {
 [속성이름:값, ...]
 [메소드이름: function([파라미터]) { ... }, ...]
}
```
- 배열 생성법

```
var 변수 = [값0, 값1, 값2, ...]
```

```
<script type="text/javascript">
function fun2() {
 var book = {
 title: null,
 author: null,
 getTitle: function() { return this.title; },
 setTitle: function(title) { this.title = title; },
 getAuthor: function() { return this.author; },
 setAuthor: function(author) { this.author = author; }
 };

 book.title = "Ajax";
 book.setAuthor(["신용권", "신민규"]);
 alert(book.title);
 alert(book.author.length + "명:" + book.author[0] + ";" + book.author[1]);
}
</script>
```

# JSON을 이용한 클래스 정의

```
//클래스 선언(생성자 선언)
클래스이름 = function([파라미터,...]) {
 [this.속성이름 = 파라미터; ...]
}
```

```
//속성 및 메서드 선언
클래스이름.prototype = {
 [속성이름: 값, ...]
 [메소드이름: function([파라미터,...]) { ... }, ...]
}
```

```
<script type="text/javascript">
JsonBook = function() {}
JsonBook.prototype = {
 title: null,
 author: null,
 getTitle: function() { return this.title; },
 setTitle: function(title) { this.title = title; },
 getAuthor: function() { return this.author; },
 setAuthor: function(author) { this.author = author; }
}
</script>

<script type="text/javascript">
function fun3() {
 var book = new JsonBook();
 book.title = "Ajax";
 book.setAuthor("신용권");
 alert(book.title);
 alert(book.getAuthor());
}
</script>
```

# XMLHttpRequest 객체

- 웹 서버와 데이터를 주고 받을 때 사용
- 표준, 그러나 대부분의 브라우저에서 지원
  - 인터넷 익스플로러 4.0 이후
  - 파이어폭스(Firefox) 1.0 이후
  - 오페라(Opera) 7.6 이후
  - 사파리(Safari) 1.2 이후
  - 네스케이프(Netscape) 7 이후
  - 컨쿼러(Kongqueror) 3 이후

```
createRequest:function() {
 try {
 //IE7, Firefox, Opera...
 return new XMLHttpRequest();
 } catch(microsoft) {
 //IE6 이전
 try {
 return new ActiveXObject("Msxml2.XMLHTTP");
 } catch(othermicrosoft) {
 try {
 return new ActiveXObject("Microsoft.XMLHTTP");
 } catch(failed) {
 alert("XMLHttpRequest 객체를 생성하지 못했습니다.");
 return null;
 }
 }
 }
},
```

# 요청시 사용되는 XMLHttpRequest 주요 멤버

- **open(method, url [, async])** 함수

- 요청 방식 구성
- method : GET/POST
- url : 접속할 URL
- async : 비동기(true) / 동기(false)
- Ex) open("POST", "member/login.jsp", true);

- **setRequestHeader(name, value)** 함수

- 요청 HTTP의 헤더 구성
- Ex) setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); //문자 폼 데이터 전송
- Ex) setRequestHeader("Content-Type", "multipart/form-data"); //바이너리를 포함하는 폼 데이터 전송

- **readyState 속성**

- 요청 진행 상태에 대한 값
- **0** (Uninitialized): 객체만 생성되고 아직 초기화되지 않은 상태(open() 메서드가 호출되지 않은 상태)
- **1** (Open): open() 메서드가 호출되고, 아직 send() 메서드가 호출되지 않은 상태
- **2** (Sent): send() 메서드가 호출되었지만, 아직 응답이 도착하지 않은 상태
- **3** (Receiving): 응답의 일부를 받은 상태
- **4** (Loaded): 응답 전부가 도착, 응답 데이터 이용 가능 상태

- **onreadystatechange 속성**
  - readyState 속성값이 변경될 때마다 호출되는 콜백 함수 지정
  - Ex) onreadystatechange = callbackFun;
- **send([body]) 함수**
  - 요청 전송
  - varBody : POST일 경우, 바디 내용
  - Ex) send("mid=white&mpassword=123");

```
sendRequest:function(url, param, callback) {
 //XMLHttpRequest 생성
 var request = this.createRequest();
 if(request==null) return;
 //요청 구성
 request.open("POST", url, true);
 //요청 헤더 구성
 request.setRequestHeader(
 "Content-Type", "application/x-www-form-urlencoded");
 //콜백 함수 지정
 request.onreadystatechange = function() {
 callback(request);
 };
 //요청 파라미터 인코딩
 param = ((param==null) || (param=="")) ? null : param;
 if(param!=null) param = encodeURI(param);
 //요청 보내기
 request.send(param);
}
```

# 파라미터의 한글 처리 방법

- XMLHttpRequest 객체가 웹 서버에 전송하는 요청 파라미터는 반드시 UTF-8로 인코딩
- JavaScript 인코딩 관련 함수
  - encodeURI("xxx.jsp?name=신용권&job=developer")
    - 인코딩 제외 문자들: ~!@#\$&\*()=:/,,?+'
    - 경로 전체를 인코딩할 때 사용
  - encodeURIComponent("신용권")
    - 인코딩 제외 문자들: ~!\*()'
    - 파라미터값만 인코딩할 때 사용
- JSP에서 파라미터값을 얻을 때
  - <%request.setCharacterEncoding("utf-8");%>
  - <%String name = request.getParameter("name");%>

# 응답 데이터 얻기 위해 사용되는 XMLHttpRequest 주요 멤버

- **status 속성**
  - 응답 HTTP의 상태 코드
  - 200: 요청 성공
  - 403: 접근 거부
  - 404: 페이지 없음
  - 500: 서버 오류 발생
- **responseText 속성**
  - 응답의 바디 내용을 문자열로 가지고 있음
  - String 개체 타입
- **responseXML 속성**
  - 응답의 바디 내용을 XML DOM 형태로 가지고 있음
  - XmlDocument 개체 타입

```
function callbackFun() {
 if(xmlHttpRequest.readyState==4 &&
 xmlHttpRequest.status==200) {

 //응답 내용 얻기
 var text = xmlHttpRequest.responseText;
 var xmlDoc = xmlHttpRequest.responseXML;

 //브라우저의 화면을 동적으로 변경하는 코드

 }
}
```

# HTML 응답 처리

- 응답이 HTML 태그를 포함하고 있을 경우
- 컨테이너 태그의 innerHTML 속성 =  
`xmlHttpRequest.responseText`
- 컨테이너 태그
  - 자식 태그를 가질 수 있는 태그
  - `<p>`, `<div>`, `<span>` ...
  - The property is read/write for all objects except the following, for which it is read-only: **COL**, **COLGROUP**, **FRAMESET**, **HTML**, **STYLE**, **TABLE**, **TBODY**, **TFOOT**, **THEAD**, **TITLE**, **TR**. The property has no default value

```
<input id="name" type="text" name="name">
<input type="button" value="전송" onclick="getGreet()">
<div id="divGreet"> </div>

function getGreet() {
 var name = document.getElementById("name").value;
 var param = "name=" + name;
 new Ajax().sendRequest("greet.jsp", param, callbackGetGreet);
}

function callbackGetGreet(request) {
 if(request.readyState==4 && request.status==200) {
 document.getElementById("divGreet").innerHTML = request.responseText;
 }
}
```

☞ alert(document.documentElement.innerHTML);로 변경된 소스 확인

# XML 응답 처리

- 응답이 XML 태그를 포함하고 있을 경우
- DOM API를 이용하여 XML 정보 얻기
  - var xmlDocument = XMLHttpRequest.responseXML
  - var nodeList = xmlDocument.getElementsByTagName("태그명");
  - var value = nodeList.item(0).firstChild.nodeValue;
- DOM API를 이용해서 HTML 화면 갱신
  - 태그 및 CSS 개체에 접근해서 속성값 변경

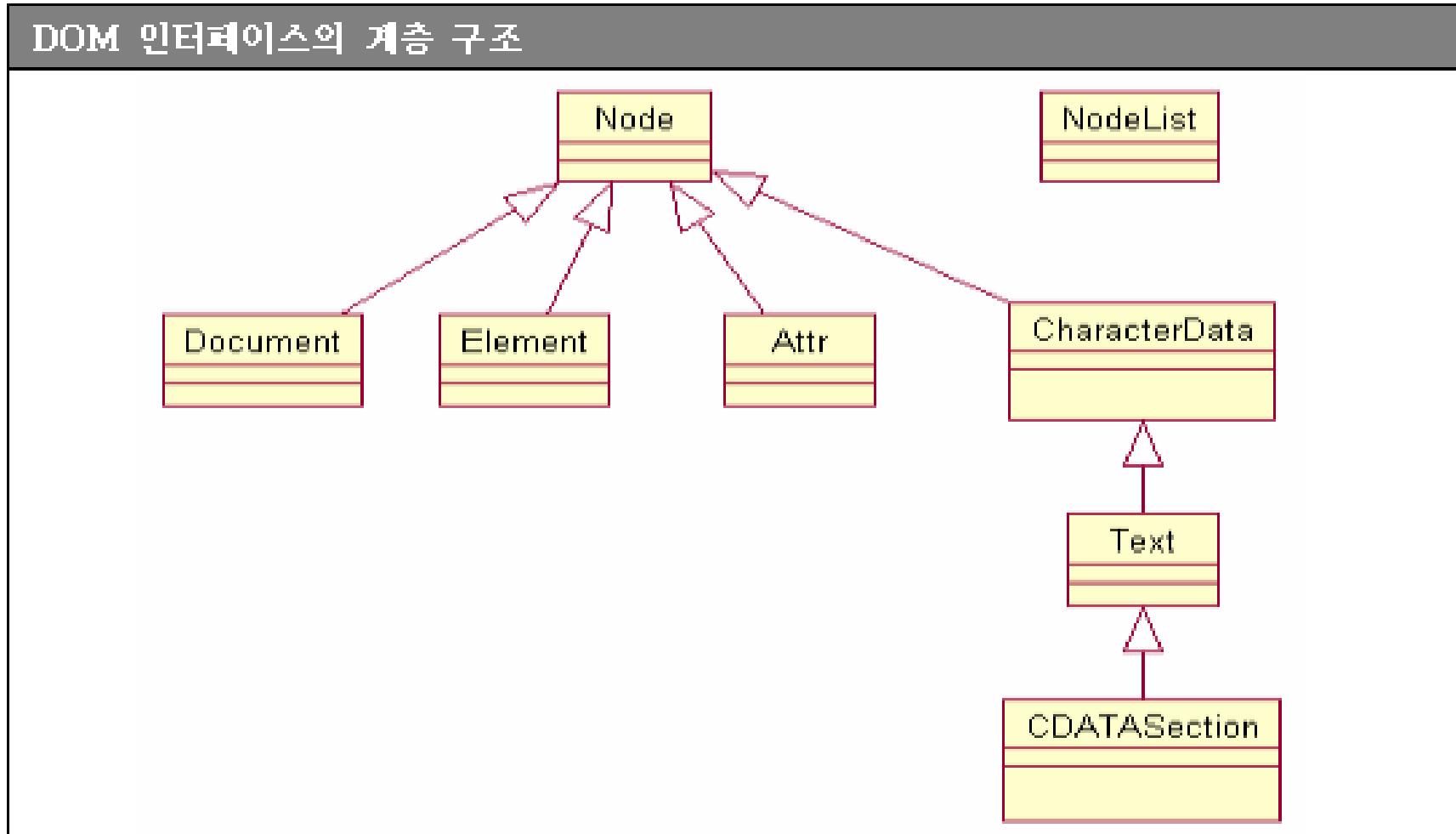
- DOM(Document Object Model)
  - W3c ([www.w3c.org](http://www.w3c.org)) 스펙화
  - XML과 HTML로 된 문서의 요소를 개체화할 때 개체의 종류와 개체가 필수적으로 가져야 할 멤버(속성, 메서드)를 정의한 것(언어중립 언어:IDE)
  - 프로그래머 입장에서 본다면, HTML 또는 XML 문서의 내용을 얻기 위해 각종 메소드와 속성들을 모아 놓은 API(Application Programming Interface)들의 집합

## • DOM Core Module 인터페이스

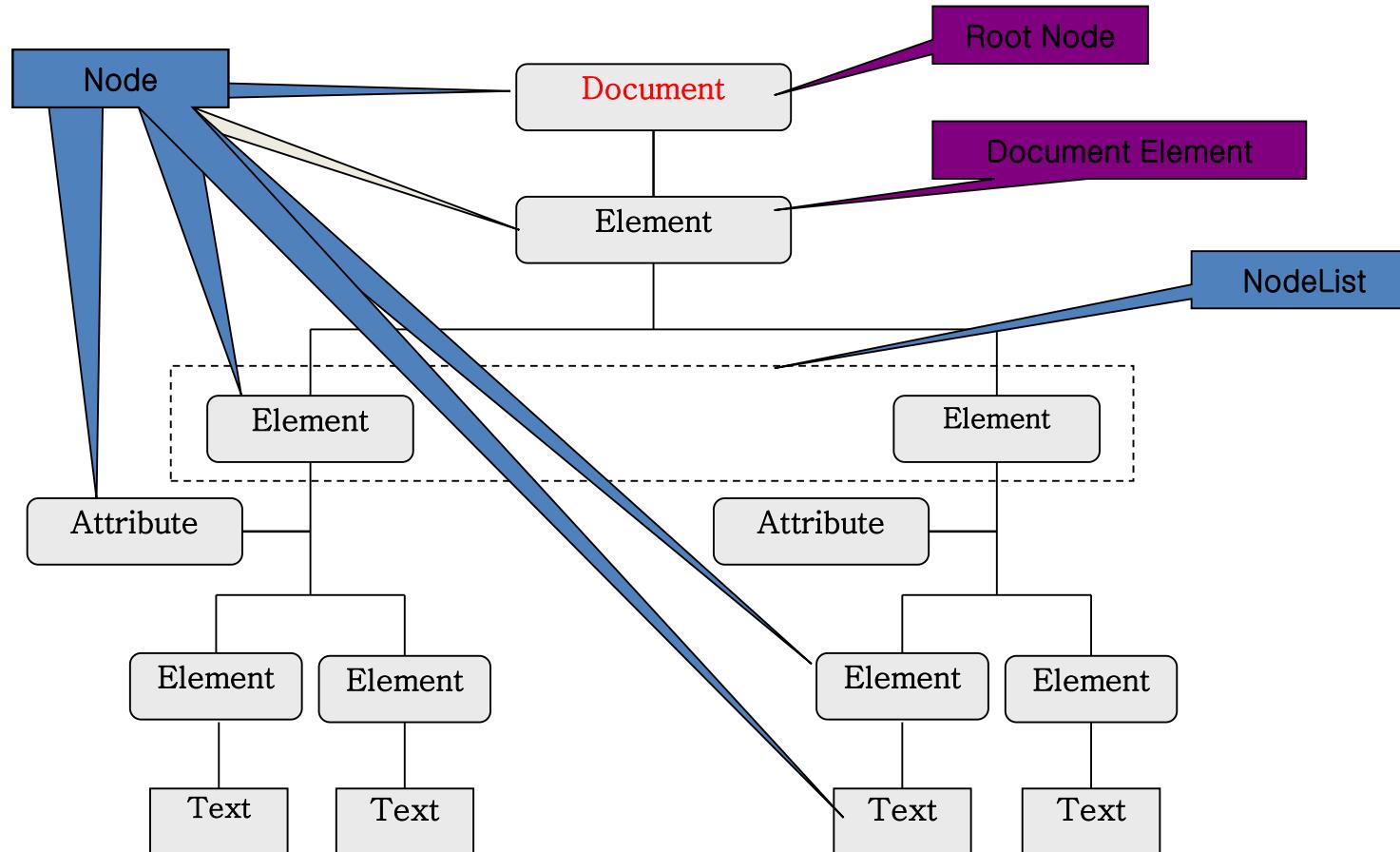
언어중립  
(IDL언어)

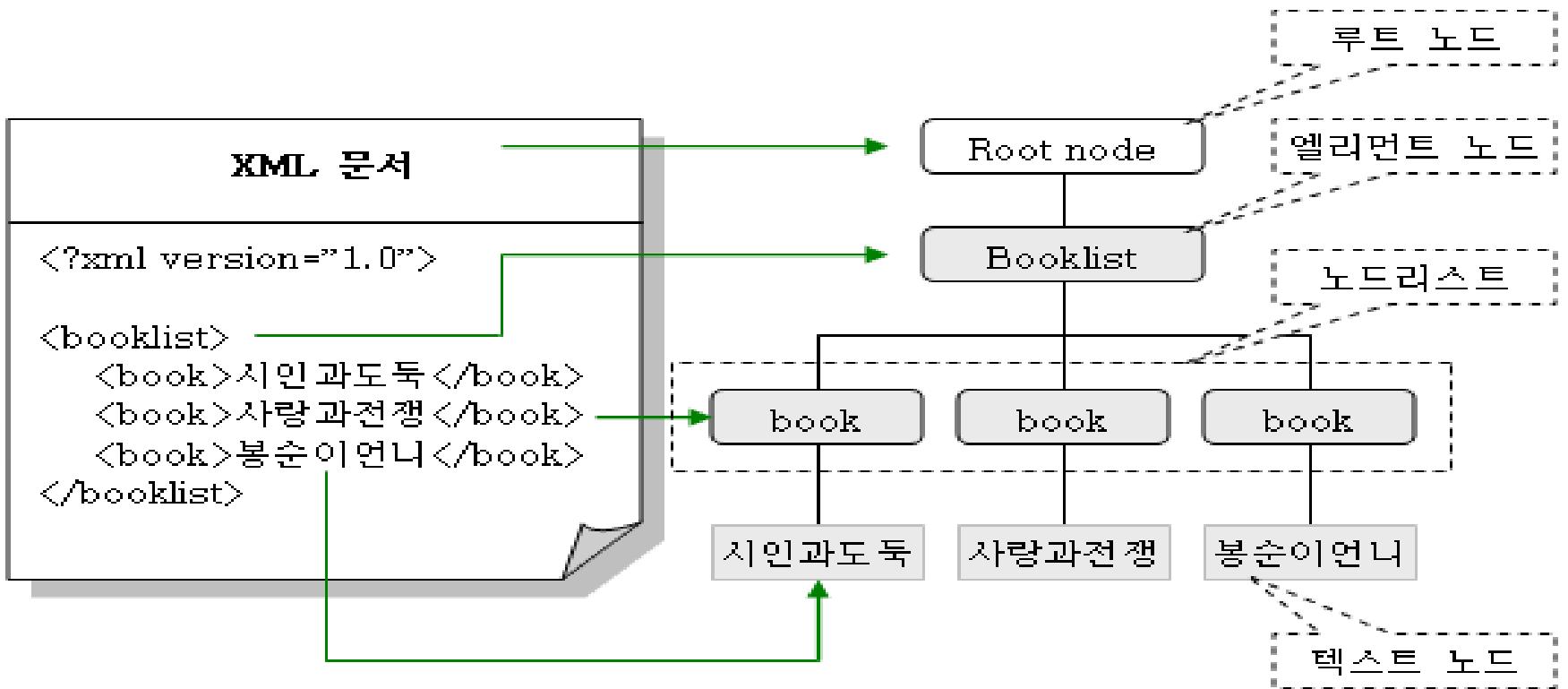
기초 인터페이스	확장 인터페이스
1. DOMException 2. DOMImplementation 3. DocumentFragment 4. Document 5. Node 6. NodeList 7. NamedNodeMap 8. CharacterData 9. Attr 10. Element 11. Text 12. Comment	1. CDATASection 2. DocumentType 3. Notation 4. Entity 5. EntityReference 6. ProcessingInstruction

## • DOM Core Module API의 계층 구조



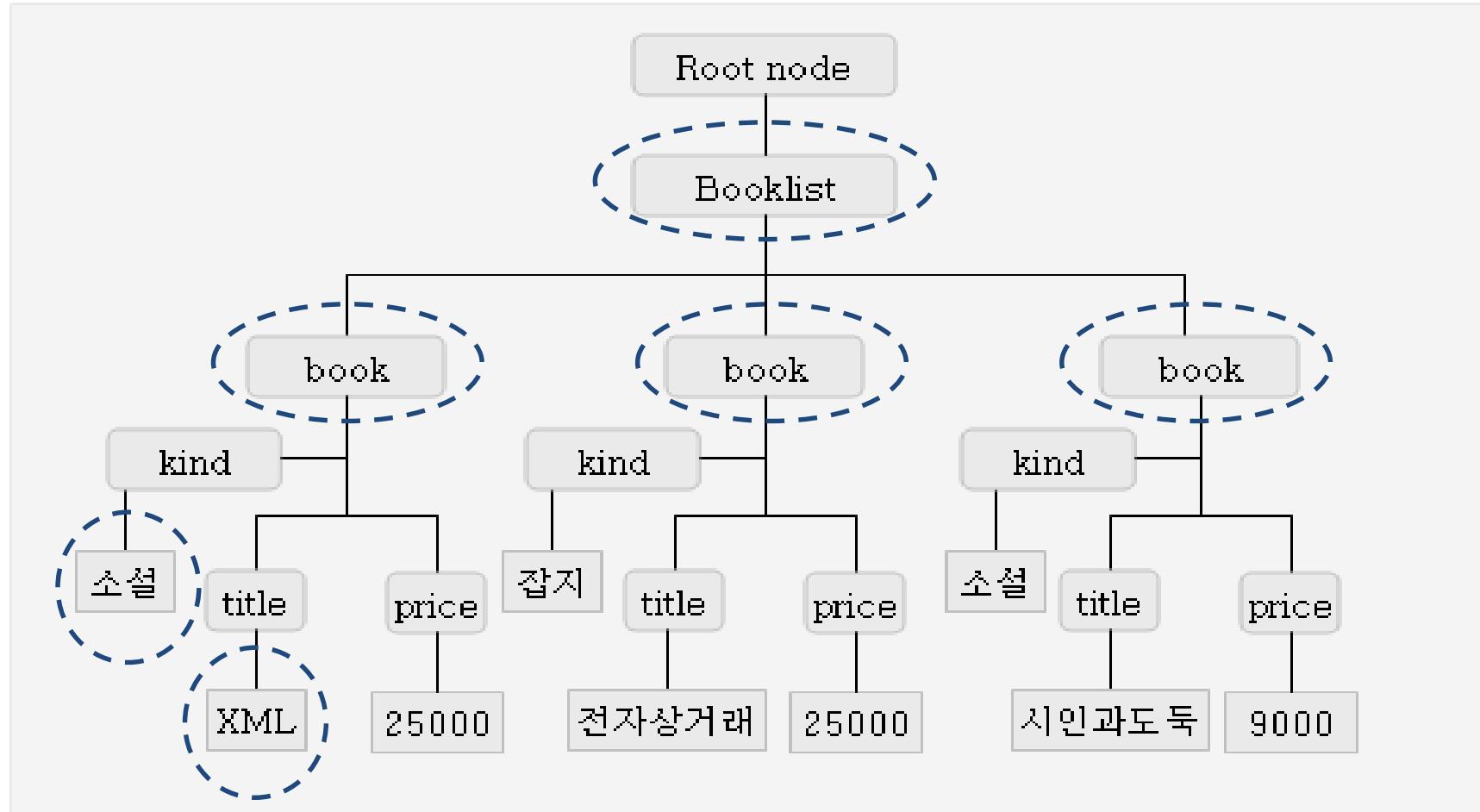
## • HTML 및 XML 문서를 객체화





기호 구분	설명
루트 노드	XML 문서 자체를 뜻하는 노드
엘리먼트 노드	루트 엘리먼트를 포함해서 자식 엘리먼트를 뜻하는 노드
텍스트 노드	데이터 부분을 뜻하는 노드
노드리스트	특정 조건에 맞는 노드들 집합

- 위치에 의한 XML DOM 객체 검색



```
<script type="text/javascript">
 function getXML() {
 new Ajax().sendRequest("booklist.jsp", null, callbackGetXML);
 }
 function callbackGetXML(request) {
 if(request.readyState==4 && request.status==200) {
 var xmlDoc = request.responseXML;
 var booklist = xmlDoc.documentElement;

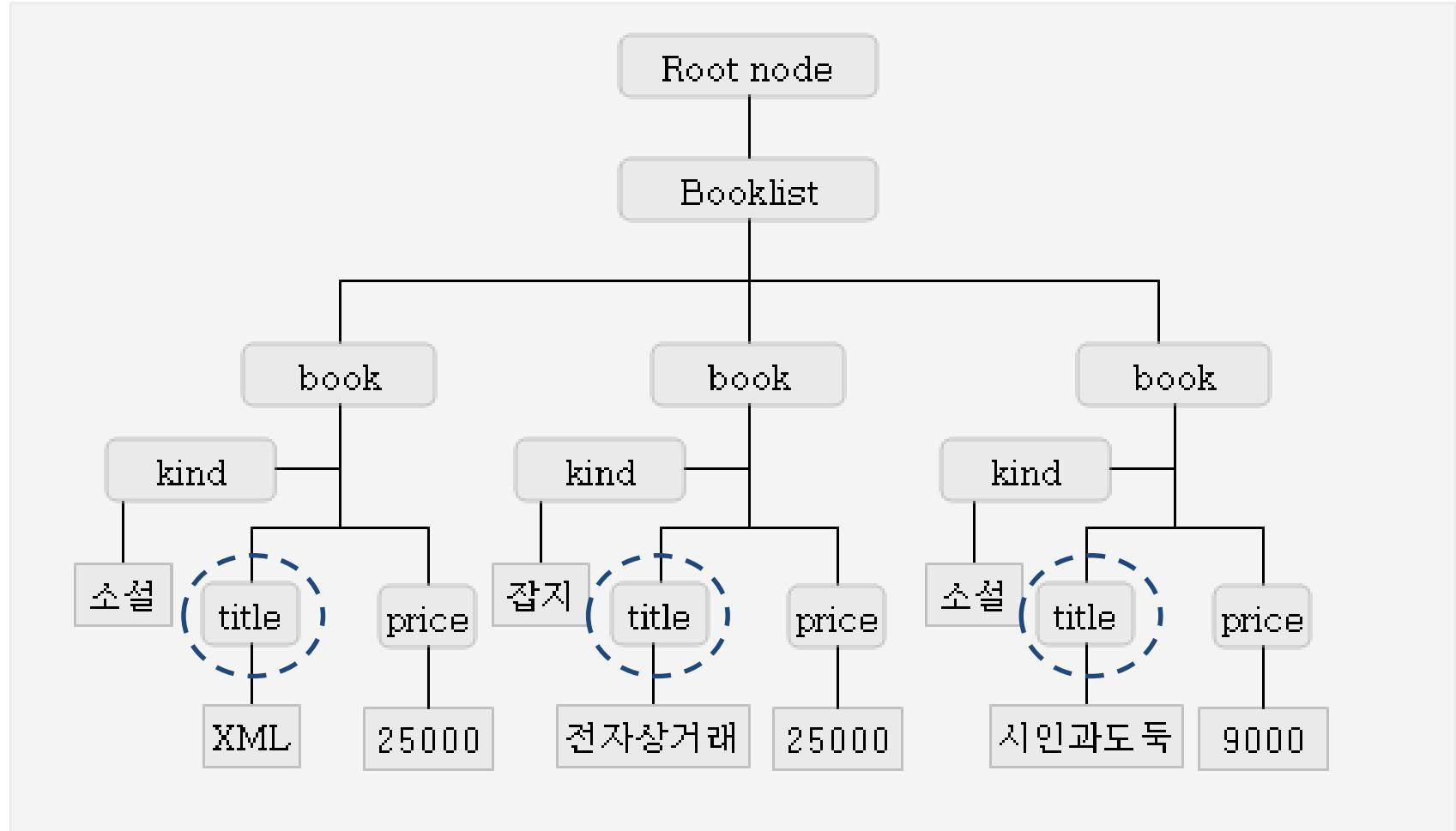
 var html = "<table border='1' cellspacing='0' width='500'>"
 html += "<tr bgcolor='#9DE54C'><td>제목</td><td>분류</td><td>가격</td></tr>";
 var book = null; var title = null; var kind = null; var price = null;

 book = booklist.firstChild;
 title = book.firstChild.firstChild.nodeValue;
 kind = book.getAttribute("kind");
 price = book.firstChild.nextSibling.firstChild.nodeValue;
 html += "<tr><td>" + title + "</td><td>" + kind + "</td><td>" + price + "</td></tr>";

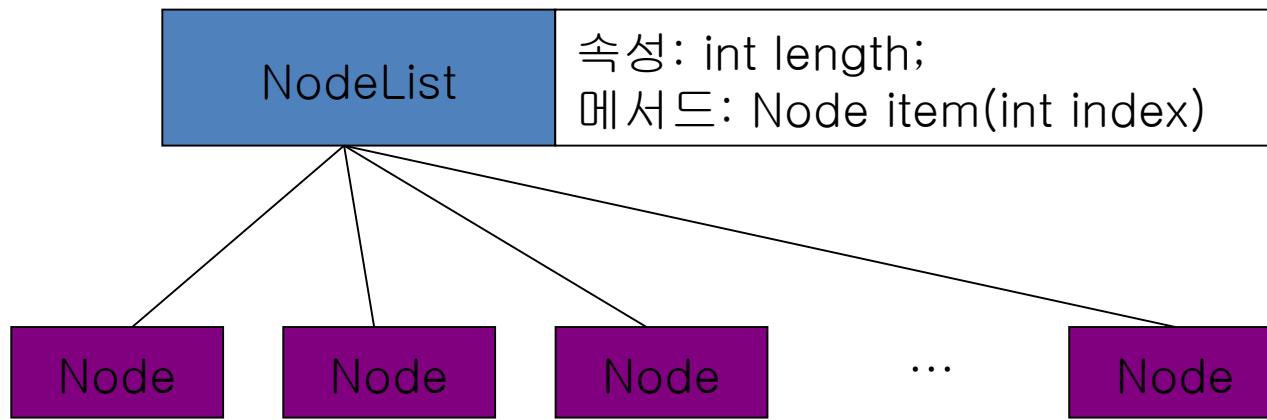
 html += "</table>";

 var tableBooklist = document.getElementById("divTable");
 tableBooklist.innerHTML = html;
 }
 }
</script>
```

- 태그 이름으로 XML DOM 객체 검색



```
NodeList getElementsByTagName(String tagname);
```



```
<script type="text/javascript">
 function getXML() {
 new Ajax().sendRequest("booklist.jsp", null, callbackGetXML);
 }
 function callbackGetXML(request) {
 if(request.readyState==4 && request.status==200) {
 var xmlDoc = request.responseXML;
 var nodeList = xmlDoc.getElementsByTagName("book");

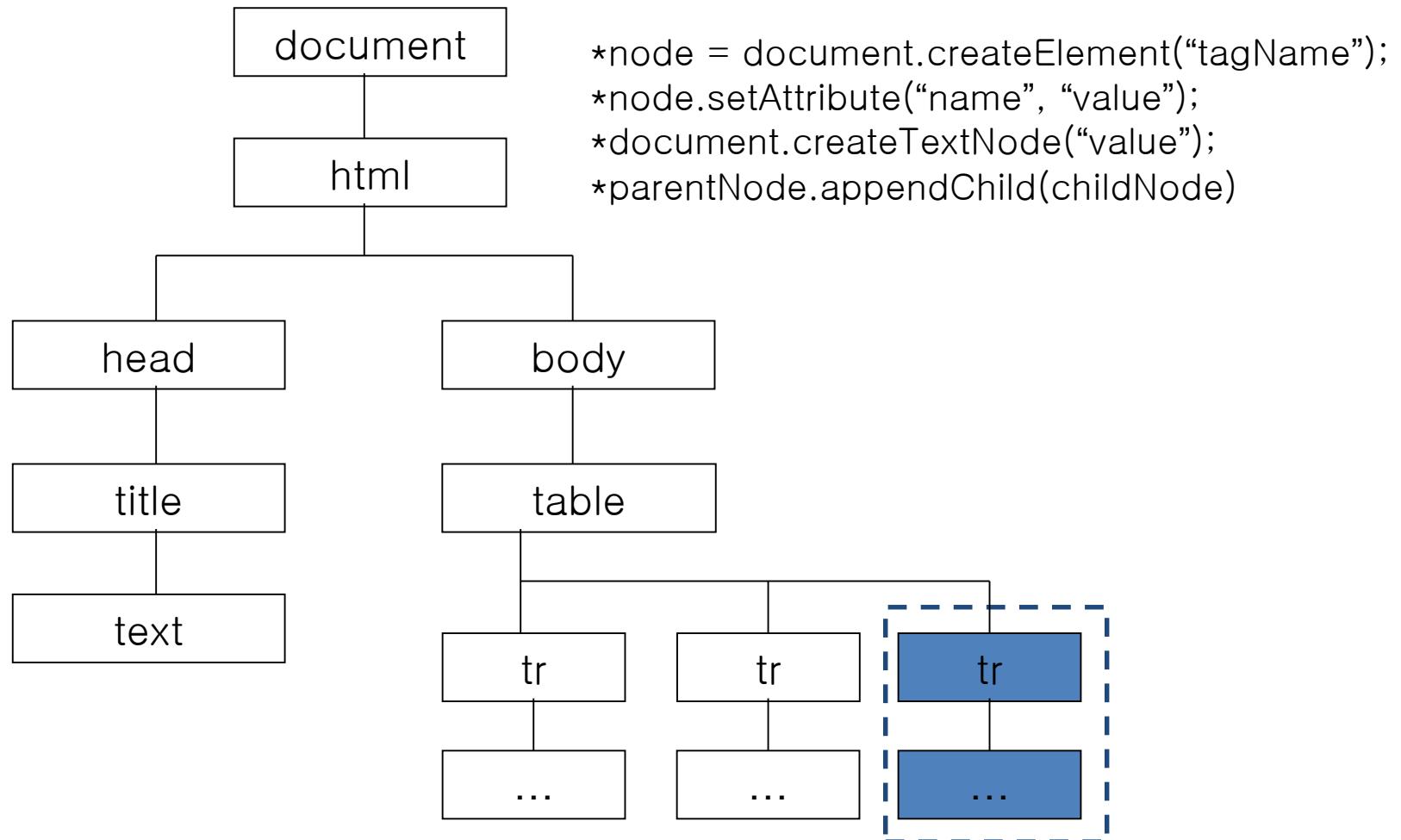
 var html = "<table border='1' cellspacing='0' width='500'>"
 html += "<tr bgcolor='#9DE54C'><td>제목</td><td>분류</td><td>가격</td></tr>";
 var book = null; var title = null; var kind = null; var price = null;

 for(var i=0; i<nodeList.length; i++) {
 book = nodeList.item(i);
 title = book.firstChild.firstChild.nodeValue;
 kind = book.getAttribute("kind");
 price = book.firstChild.nextSibling.firstChild.nodeValue;
 html += "<tr><td>" + title + "</td><td>" + kind + "</td><td>" + price + "</td></tr>";
 }

 html += "</table>";

 var tableBooklist = document.getElementById("divTable");
 tableBooklist.innerHTML = html;
 }
 }
</script>
```

## • HTML DOM에 자식 노드 및 속성 추가



```
function appendTr(title, kind, price) {
 var tbodyBooklist = document.getElementById("tbodyBooklist");

 var tr = document.createElement("tr");
 var td = null; var text = null;

 td = document.createElement("td");
 text = document.createTextNode(title);
 td.appendChild(text);
 tr.appendChild(td);

 td = document.createElement("td");
 text = document.createTextNode(kind);
 td.appendChild(text);
 tr.appendChild(td);

 td = document.createElement("td");
 text = document.createTextNode(price);
 td.appendChild(text);
 tr.appendChild(td);

 tbodyBooklist.appendChild(tr);
}
```

# JSON 응답 처리

```
<%@ page language="java" contentType="text/plain; charset=EUC-KR" %>
```

```
[
 {
 kind: "소설",
 title: "XML",
 price: "25000"
 },

 {
 kind: "잡지",
 title: "전자상거래",
 price: "25000"
 },

 {
 kind: "소설",
 title: "시인과도둑",
 price: "9000"
 }]
```

```
<script type="text/javascript">
function getXML() {
 new Ajax().sendRequest("json.jsp", null, callbackGetXML);
}
function callbackGetXML(request) {
 if(request.readyState==4 && request.status==200) {
 var booklist = eval(request.responseText);

 var html = "<table border='1' cellspacing='0' width='500'>" +
 html += "<tr bgcolor='#9DE54C'><td>제목</td><td>분류</td><td>가격</td></tr>";

 for(var i=0; i<booklist.length; i++) {
 var book = booklist[i];
 html += "<tr><td>" + book.title + "</td>" +
 "<td>" + book.kind + "</td>" +
 "<td>" + book.price + "</td></tr>";
 }

 html += "</table>";

 var tableBooklist = document.getElementById("divTable");
 tableBooklist.innerHTML = html;
 }
}
</script>
```

# XML/JSON 응답 처리

```
<?xml version="1.0" encoding="euc-kr"?>
<%@ page language="java" contentType='text/xml; charset=EUC-KR' %>

<result>
 <code>success</code>
 <data>
 <![CDATA[
 [
 {
 kind: "소설",
 title: "XML",
 price: "25000"
 },
 {
 kind: "잡지",
 title: "전자상거래",
 price: "25000"
 }
]
]]>
 </data>
</result>
```

```
<script type="text/javascript">
function getXML() {
 new Ajax().sendRequest("xmljson.jsp", null, callbackGetXML);
}
function callbackGetXML(request) {
 if(request.readyState==4 && request.status==200) {
 var xmlDoc = request.responseXML;

 var code = xmlDoc.getElementsByTagName("code").item(0).firstChild.nodeValue;

 if(code=="success") {
 var json = xmlDoc.getElementsByTagName("data").item(0).firstChild.nodeValue;
 var booklist = eval(json);

 var html = "<table border='1' cellspacing='0' width='500'>" +
 html += "<tr bgcolor='#9DE54C'><td>제목</td><td>분류</td><td>가격</td></tr>";

 for(var i=0; i<booklist.length; i++) {
 var book = booklist[i];
 html += "<tr><td>" + book.title + "</td>" +
 "<td>" + book.kind + "</td>" +
 "<td>" + book.price + "</td></tr>";
 }

 html += "</table>";

 var tableBooklist = document.getElementById("divTable");
 tableBooklist.innerHTML = html;
 }
 }
}
</script>
```

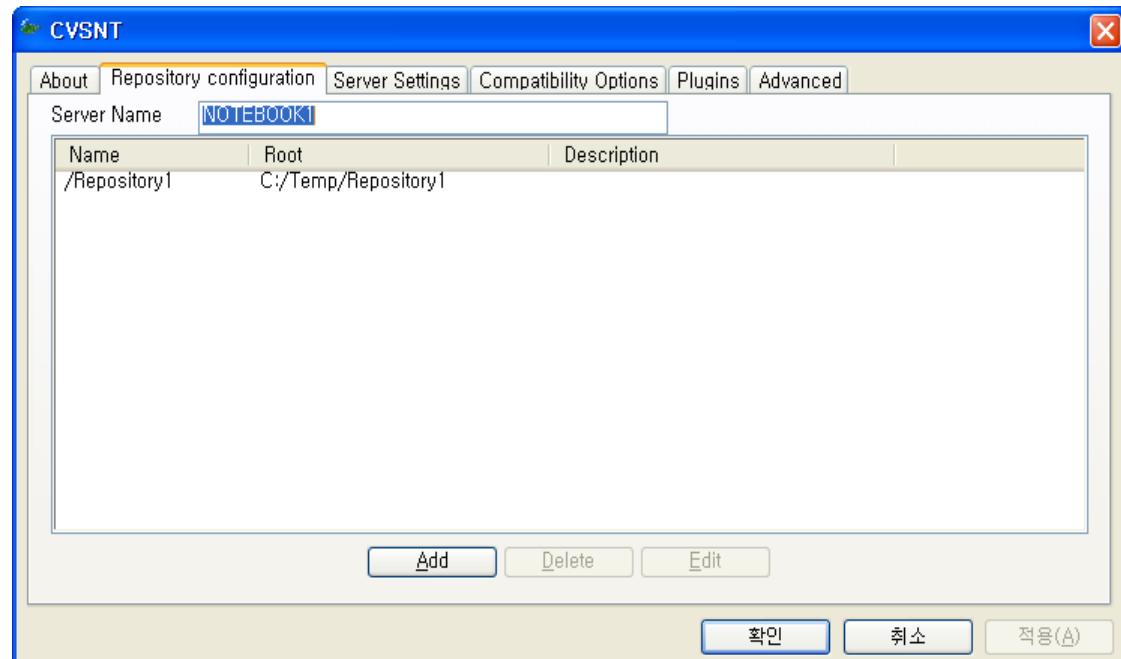
# CVS를 이용한 공동 개발

# CVS 소개

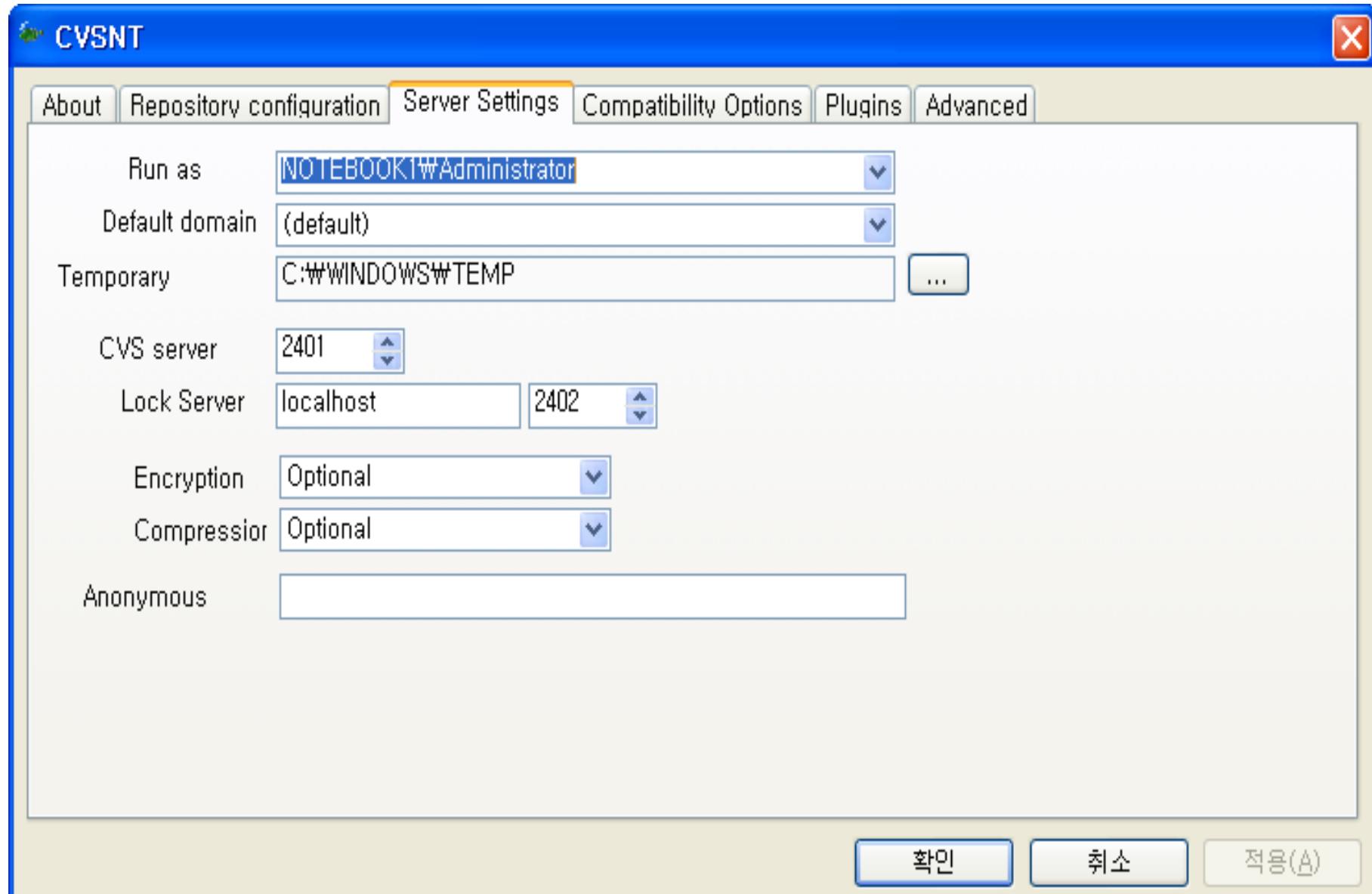
- Concurrent version system
- 소스 버전 관리 소프트웨어
- 팀 단위로 프로젝트를 개발할 때 사용
- 프로젝트 소스 취합이 용이

# CVS 설치

- CVS 서버 다운로드 및 설치
  - <http://www.cvsnt.org>
- Repository 생성

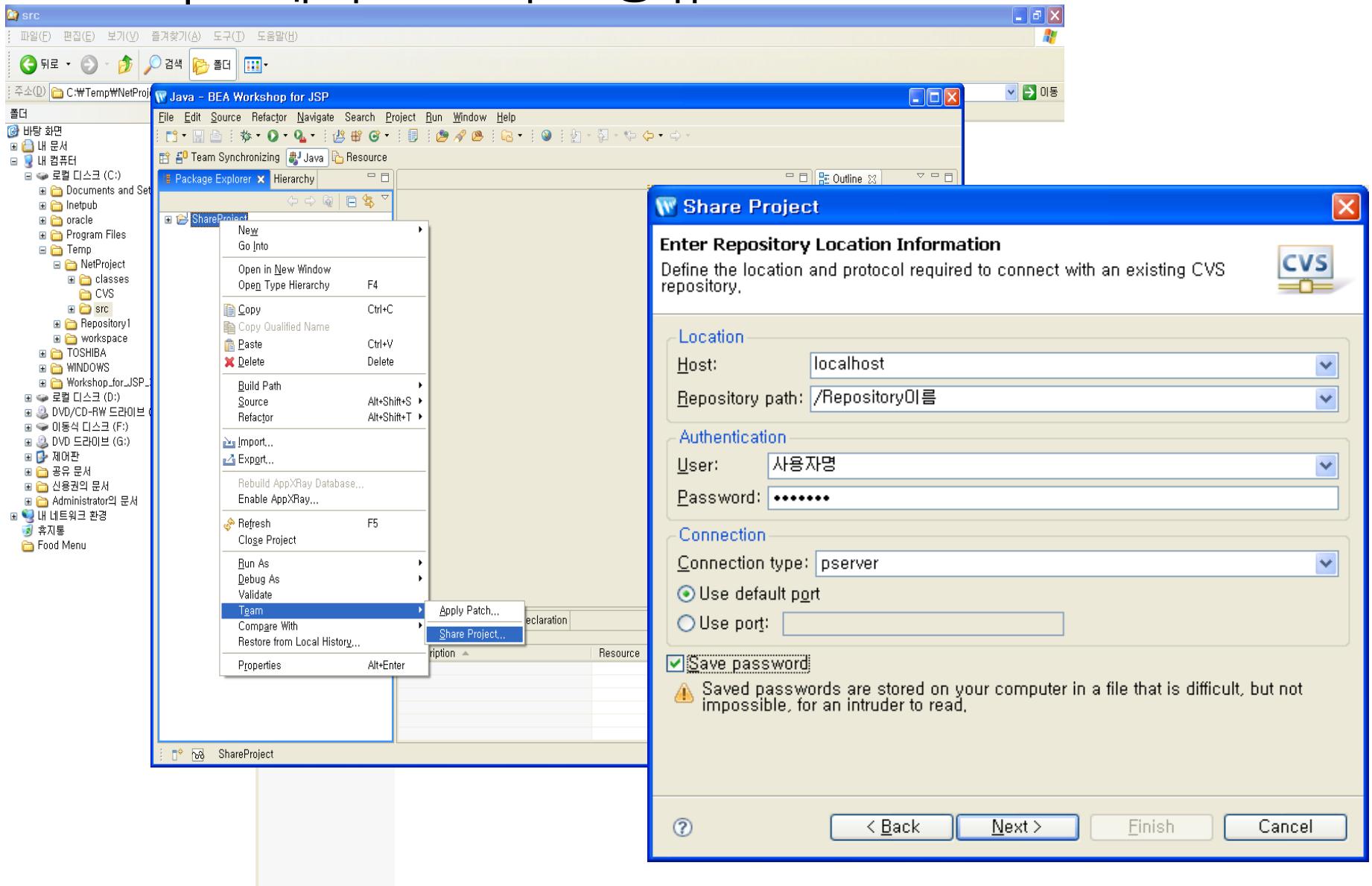


- 서버 실행 계정 설정

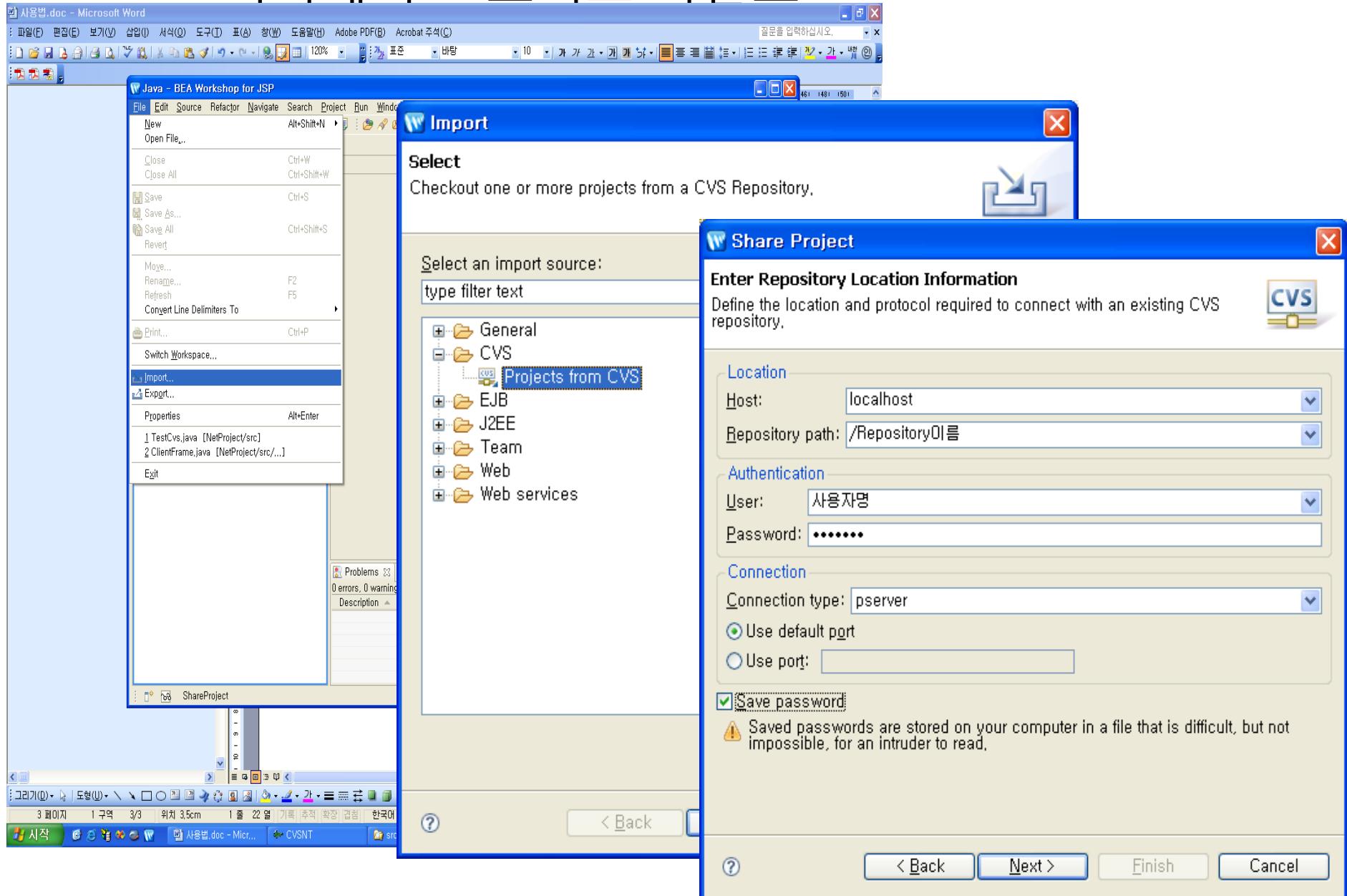


- CVS 사용자 등록
  - CVS 설치 폴더에서 DOS 창으로 실행
  - set cvsroot=:sspi:localhost:/Repository이름
  - cvs passwd –a 사용자이름

## ● Eclipse에서 프로젝트 공유

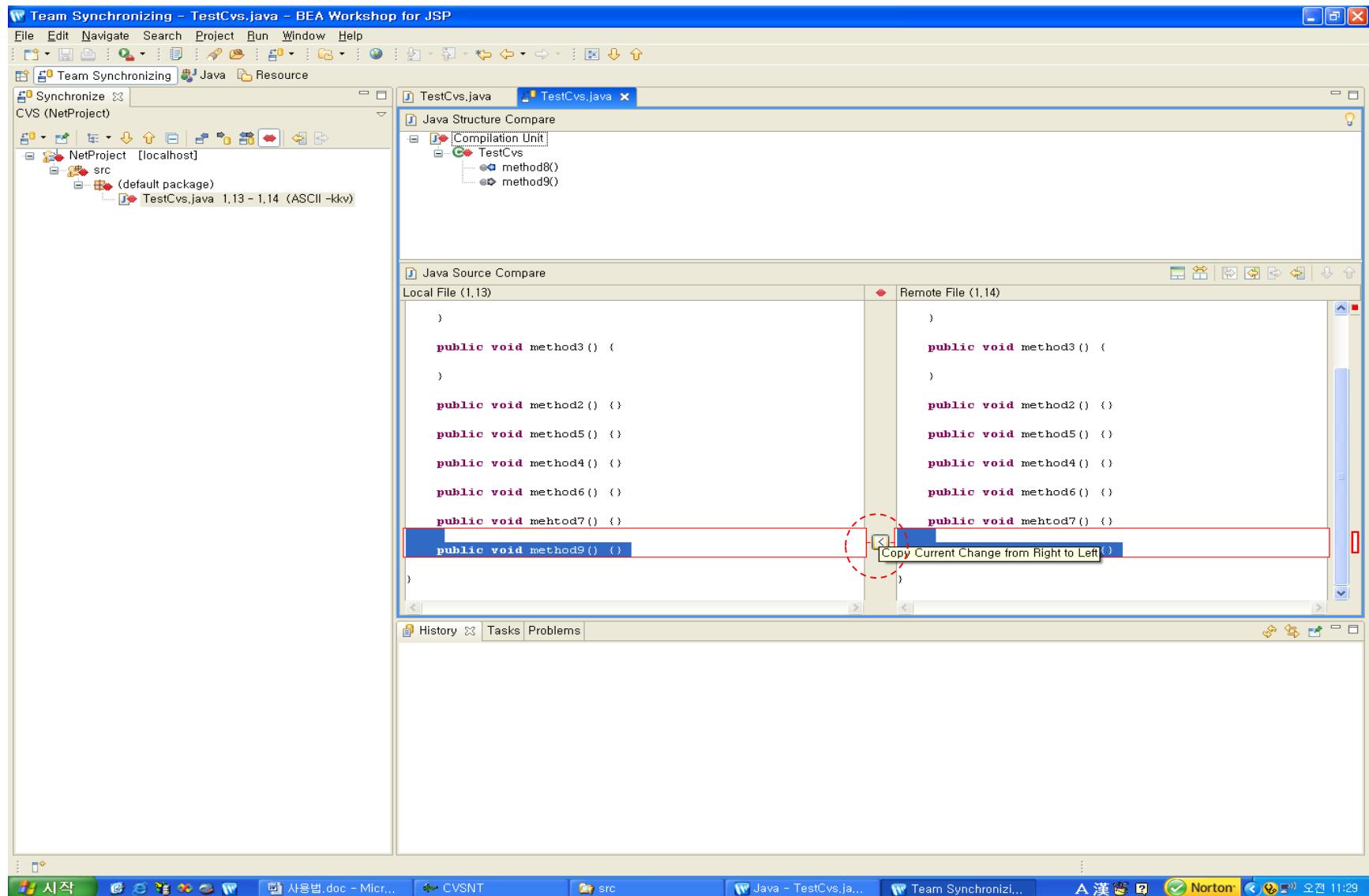


## ● CVS 서버에서 프로젝트 다운로드



- 다운로드 받은 로컬 버전을 수정한 후 서버로 올려  
새로운 버전을 할당받기
  - Commit
- 다운로드 받은 로컬 버전이 수정되지 않았을 경우,  
다시 서버에서 새로운 버전을 다운로드 받기하거나  
서버에 새로 생성된 파일을 다운로드 받기
  - update

- 다운로드 받은 로컬 버전이 서버에 있는 버전보다 낮아 commit 되지 않을 경우



# 웹 응용 프로그램 개발 실습

- 메모