

목 차

1. 웹 개요 -----	2
2. 인터넷 프로그래밍 -----	3
3. JSP 개요 -----	5
4. JSP 운영환경 및 개발환경 -----	7
5. 서블릿(Servlet) -----	15
6. JSP 소스의 서블릿 변환 -----	45
7. JSP 문법의 기본 -----	49
8. 지시자 -----	55
9. 내장 객체 -----	61
10. 액션 태그 개요 -----	116
11. 액션 태그를 활용한 템플릿 페이지 작성 -----	134
12. 웹의 비연결 특성 -----	137
13. 자바 빈즈 -----	163
14. 파일 업로드 -----	180
15. 커넥션 프로파일 만들기 -----	197
16. JDBC 개요 -----	203
17. 커넥션 풀 -----	223
18. JDBC를 위한 자바 빈즈 -----	231
19. 표현 언어 개요 -----	266
20. JSTL 개요 -----	291
21. 커스텀 태그 개요 -----	333
22. MVC 모델 구현 -----	358
23. MVC 모델2 게시판 -----	396
24. Tiles를 이용한 레이아웃 템플릿 처리 -----	447
25. 리스너와 필터 -----	456

Java Server Page

1. 웹 개요

1) WWW(World Wide Web)

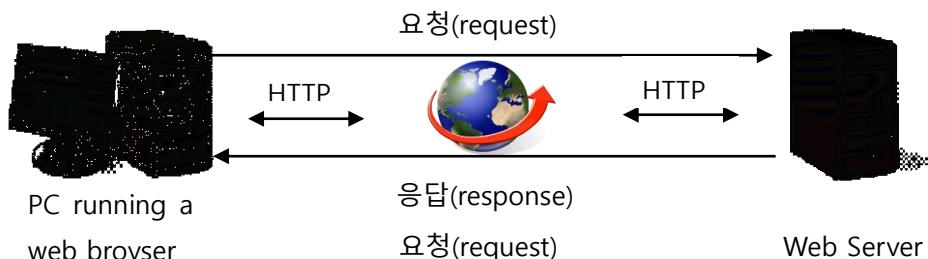
WWW인 월드와이드웹(World Wide Web)은 유럽입자물리연구소의 연구원인 팀 버너스리가 1989년에 제안하여 개발된 정보공유방안이다. WWW는 전세계에 연결된 인터넷 기반에서 하이パーテ스트(hypertext)기반의 정보를 구축하여 누구나가 쉽게 공유할 수 있도록 있는 정보 구축 방법으로 하이パーテ스트 자료들은 HTML이라는 언어를 통해 표현되며 이러한 문서들은 HTTP라는 통신 프로토콜을 사용하여 전송된다.

하이パーテ스트는 정보를 서로 연결하는 하이퍼링크에 의하여 구성된 정보를 말한다. 이 하이パーテ스트는 문자, 그림, 동영상, 음악, 파일 등의 멀티미디어 정보로 구성될 수 있으며, 멀티미디어 정보를 강조한 용어가 하이퍼미디어이다. 이러한 하이パーテ스트의 무한한 정보의 연결 방안과 인터넷이라는 지역성 파괴의 결합인 WWW는 웹브라우저의 개발과 함께 전 세계의 사람을 정보의 바다로 항해하게 만들었다.

WWW는 그 말이 표현하듯이 전 세계를 연결한 거미줄과 같은 인터넷 망에서의 정보 공유를 뜻한다. WWW는 편리하고 사용이 쉬운 장점 때문에 소수 전문가들의 전유물로 알려졌던 인터넷을 누구라도 접근하기 쉬운 것으로 변화시키면서 현재와 같이 일상 생활처럼 인터넷을 사용하게 되었다.

2) 클라이언트 서버 구조

웹은 클라이언트/서버 구조로서 웹 브라우저가 있는 클라이언트가 자료를 요청(request)하면, 웹 서버가 있는 서버는 요청에 응답(response)하여 클라이언트의 웹브라우저에 정보를 검색되는 구조를 갖는다. 요청이란 HTML 페이지 또는 동영상, 이미지, 소리 등의 자원 요청을 말하고 응답이란 이러한 요청 자원을 보내주는 것을 말한다.



3) 서버의 역할

웹 서버는 HTTP 통신 프로토콜을 사용하여 클라이언트의 요청에 응답을 하는 프로그램이다. 이 웹서버는 서버의 역할을 수행하기 위해 항상 실행되어 있어야 하며 클라이언트가 요청한 페이지 또는 프로그램을 실행하여 파일이나 그 결과를 사용자들에게 제공한다. 웹서버도 그 종류가 매우 많은데 일반적인 웹서버들로는 윈도우와 유닉스 기반의 운영체계에서 모두 쓸 수 있는 아파치와 톰캣 그리고 윈도우 서버에서 주로 이용하는 IIS로 예로 들 수 있다.

웹서버는 인터넷 서버 프로그래밍 방식과 밀접한 관련을 가지며, 만일 웹서버에 데이터베이스 관리시스템이 설치되어 있는 경우, 웹 서버가 데이터베이스 서버의 역할도 함께 수행할 수 있다. 물론 필요에 따라 웹 서버와 데이터베이스 서버를 분리 운영할 수도 있다.

2. 인터넷 프로그래밍

2.1 인터넷 클라이언트 프로그래밍

VBScript와 JavaScript는 모두 컴파일 없이 웹 브라우저 상에서 직접 수행이 가능한 스크립트 언어로 HTML 문서에서 태그로 표현할 수 없는 로직 처리를 담당하기 위해 개발된 언어이다. VBScript와 JavaScript 모두 서버가 아닌 클라이언트 웹 브라우저에서 실행되는 프로그래밍 언어이다. JavaScript는 선마이크로시스템즈 사와 넷스케이프 커뮤니케이션스 사가 공동 개발한 스크립트 언어로 1996년 2월에 발매한 웹 브라우저인 넷스케이프 내비게이터 2.0에서부터 사용할 수 있었다. 반면에 VBScript는 JavaScript에 대항하여 마이크로소프트 사가 비주얼베이직 언어를 기초로 하여 만든 스크립트 언어이다.

2.2 인터넷 서버 프로그래밍

1) CGI

CGI는 Common Gateway Interface의 약자로 동적인 웹 서버 구축을 위하여 처음으로 개발된 서버 프로그래밍 방식이다. CGI에 이용되는 언어는 달리 정해져 있지 않으며 이전부터 사용하던 C, C++, Perl 등의 일반적인 언어로 CGI 규약에 맞게 로직 처리와 데이터베이스 접속, 참조 처리를 담당한다. 1990년 중반에 동적인 서버의 필요성으로 CGI 방식이 많이 사용되었으나 CGI 방식보다 발전된 ASP와 PHP방식이 개발되면서 사용이 줄었으며 현재는 거의 사용되고 있지 않다.

2) ASP

ASP는 마이크로소프트 사가 1995년도 IIS3.0을 발표하면서 함께 발표한 기술로서 비주얼베이직을 기본으로 개발된 VBScript를 HTML문서에 직접 코딩하여 동적인 웹 페이지를 구현하는 기술이다. ASP는 발표되면서부터 기존의 웹버서 프로그래밍인 CGI를 사용하던 많은 개발자에게 빠른 시간 내에 인기를 얻게 되었다. 이러한 인기의 이유는 ASP가 윈도우 NT혹은 윈도우 2000에서 기본적으로 동작할 수 있으며 스크립트 언어로 채택한 비주얼베이직이 전 세계에 가장 많은 개발자를 보유하고 있었기 때문이다. ASP는 스크립트 언어와 함께 태그를 이용하며 보다 복잡한 비즈니스 로직은 ActiveX라는 컴포넌트를 이용해서 해결한다. ASP는 HTML 페이지에 VBScript의 소스를 내장한 프로그램이며, ASP파일은 일반 텍스트 파일로 확장자는 asp이다. ASP는 서버에서 클라이언트 사용자에게 보내지기 전에 일단 웹서버에서 asp.dll이라는 파일의 처리과정을 거쳐 모두 html 형식의 웹문서로 바꾸어 클라이언트에게 서비스된다. 그래서 윈도우 운영체제에 기반한 웹 애플리케이션 기술로 비교적 쉽고 빠르게 웹 애플리케이션 구현이 가능하다. 최근 윈도우 개발 환경이 닷넷(.Net) 플랫폼으로 변화 되면서 ASP.Net이라는 이름으로 변경되어 보다 강력해 졌다.

3) PHP

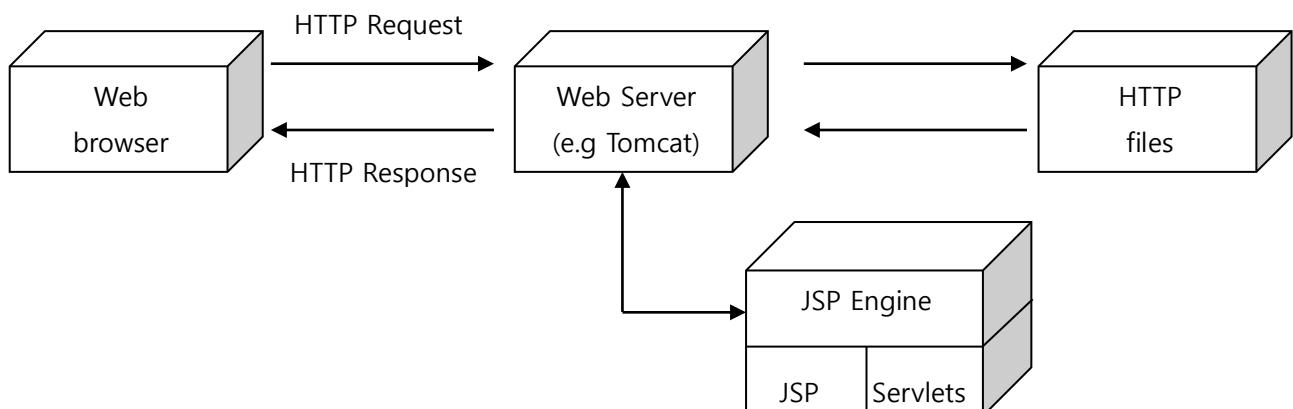
PHP(Personal Home Page 또는 Professional Hypertext Preprocessor)는 C언어와 유사한 언어를 사용하며, 적은 명령어로 서버 프로그래밍이 가능한 서버 프로그래밍 방식이다. 오픈소스 프로젝트로 다양한 운영체제와 웹서버를 지원한다. 빠른 처리속도와 메일, 데이터베이스 연동 기능 등을 통해 초기 서버 스크립트 기술의 대표로 주목 받았으나 완전한 프로그래밍 언어가 아닌 관계로 기능확장에 한계가 있다. 최근까지도 거급된 발전을 통해 처음 보다 많이 향상 된 기능을 제공하나 예전에 비해 전체적인 사용빈도는 떨어진다.

4) JSP

JSP(Java Server Page)는 썬마이크로시스템즈 사가 개발한 인터넷 서버 프로그래밍 기술이다. 선마이크로시스템즈 사는 자바 언어를 기반으로 하는 인터넷 서버 프로그래밍 방식인 서블릿(Servlets)을 먼저 개발하여 과거의 CGI 개발 방식을 대체하였다. 그러나 자바를 이용한 서블릿 개발 방식이 그리 쉽지 않고, PHP, ASP 등과 같이 HTML 코드내에 직접 비즈니스 로직을 삽입할 수 있는 개발 방식이 필요하게 되어 개발한 기술이 JSP이다. 그러나 JSP는 서블릿과 동떨어진 기술이 아니며 JSP가 실제로 웹 애플리케이션 서버에서 사용자에게 서비스가 될 때는 서블릿으로 변경되어 서비스된다. JSP는 자바 기반의 문법을 이용하여 어려운 자바 소스 대신에 태그를 사용해 자바 객체를 사용한다. 또한 JSP는 자바빈즈(JavaBeans)라는 컴포넌트를 이용하여 비즈니스 로직과 프리젠테이션 로직을 완전히 분리해 응용 시스템을 구현할 수 있다. JSP의 경우는 개발자가 태그 라이브러리 기능을 이용해 자신만의 태그를 정의해서 사용할 수 있다. 이렇게 함으로써 좀 더 많은 기능을 확장하여 사용할 수 있으며 일괄성 있는 프로그래밍 작업을 할 수 있다.

JSP는 플랫폼에 독립적인 기술 방식이다. 시스템 플랫폼이 윈도우 NT든 유닉스 시스템이든 어느 한 플랫폼에서 개발한 시스템을 다른 시스템에서 운영하는 것이 가능하다. 또한 JSP는 웹 서버에 독립적이다. 넷스케이프 엔터프라이즈 서버, 아파치 웹서버, 마이크로소프트의 IIS(Internet Information Server) 등 어떠한 웹서버 환경에서 작성되어 있던지 한번 작성된 JSP는 그 모든 웹서버에서 아무런 문제 없이 잘 작동한다.

웹서버에서 JSP를 실행시키려면 자바 모듈을 이해하는 엔진인 자바엔진이 있어야 한다. 이러한 자바 엔진의 대표가 톰캣(Tomcat)이다. 톰캣은 자바 엔진이면서 자바 서버로서 아파치 파운데이션(Apache Foundation)에서 개발되는 무료 웹서버이다.



다음은 현재의 시간을 출력하는 간단한 JSP 프로그램과 그 결과이다. <%...%> 사이에 자바 언어로 구성된 프로그램인 스크립트릿(scriptlet)이 삽입되는 것을 볼 수 있다.(Hello.jsp)

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

```

</head>
<body>
    지금 시각은 <%= new java.util.Date()%> 입니다.
</body>
</html>

```

3. JSP 개요

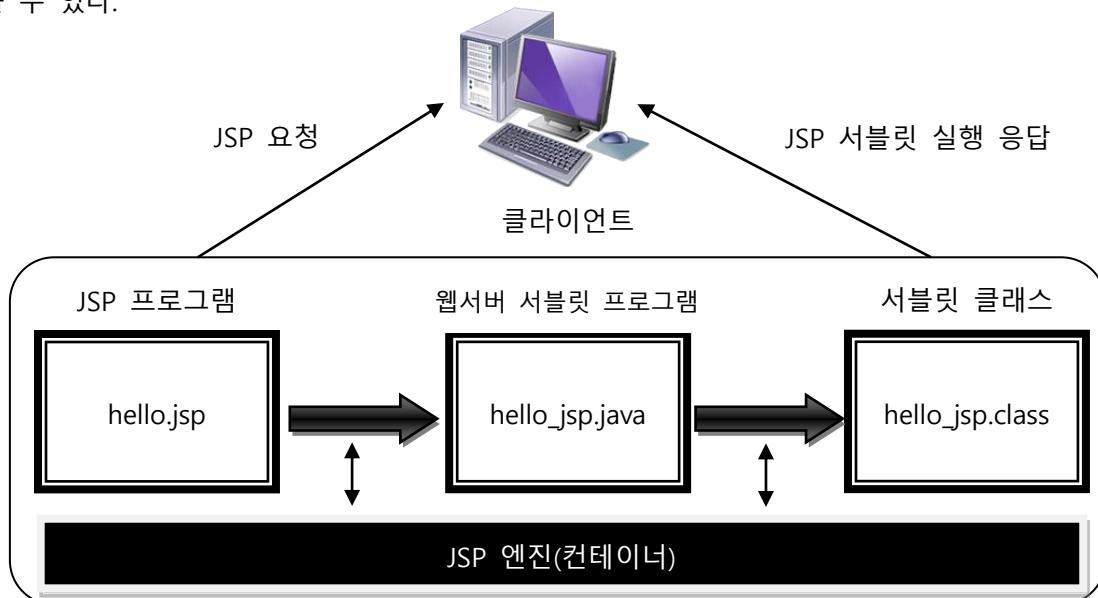
3.1 서블릿과 JSP 엔진

1) 서블릿

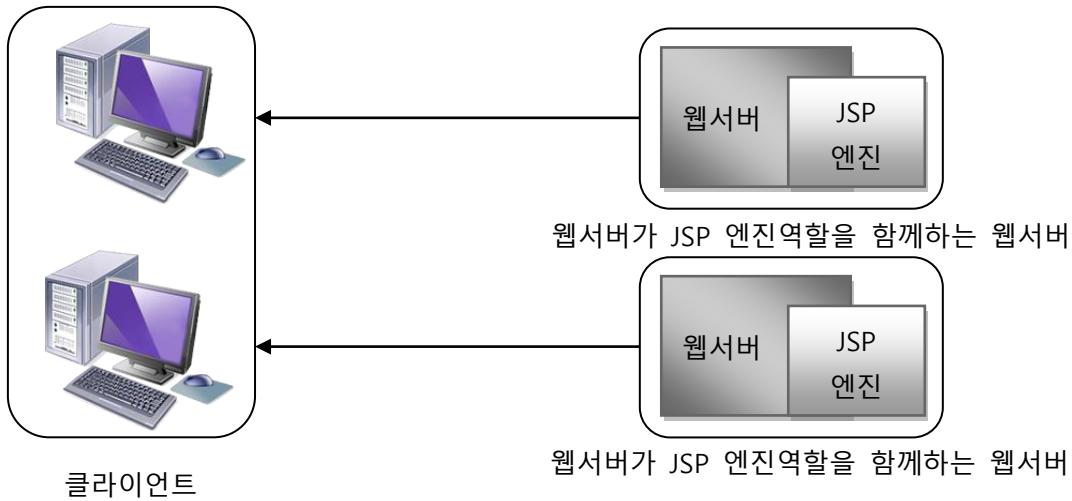
서블릿(servlet)은 자바를 이용한 확장된 CGI 방식의 서버 프로그래밍 방식으로 JSP 기술보다 먼저 발표된 기술이다. 서블릿 소스는 다음과 같이 완전한 자바 프로그래밍 방식으로 HTML의 출력을 자바 프로그램에서 구현하는 방식이다. 즉 서블릿 프로그래밍 방식은 자바 프로그램에 표현 부분인 HTML 코드를 모두 포함해야 하므로 로직 처리와 디자인 처리를 분리하기 어려운 단점이 있어 서블릿 방식이 나중에 발표된 JSP보다 어렵다고 평가되었다고 한다. 서블릿 프로그램만으로 웹 프로그래밍도 가능하지만 현재는 보다 간단한 JSP 프로그램을 함께 이용하는 방식으로서 서버 프로그래밍을 구현한다. JSP 프로그램은 내부적으로 서블릿 프로그램으로 변환되어 실행된다.

2) JSP 엔진(컨테이너)

JSP 프로그램은 하나의 서블릿 프로그램으로 변환되어 실행된다. 즉 JSP 프로그램인 hello.jsp 소스에서 hello_jsp.java의 서블릿 프로그램이 생성된 후, 이 서블릿 소스가 컴파일되어 hello_jsp.class 클래스가 생성된다. 그러므로 클라이언트 hello.jsp를 요청하면 서버는 대응하는 JSP 서블릿 클래스인 hello_jsp.class를 실행하여 클라이언트에게 응답한다. 여기서 JSP 파일에서 생성되는 서블릿 파일의 이름은 시스템마다 다를 수 있다.



이러한 JSP 소스에서 서블릿 소스 및 서블릿 클래스 생성을 처리하는 서버모듈을 JSP 엔진 또는 JSP 컨테이너라 부른다. 이러한 JSP 엔진을 웹서버와 분리되어 독립적으로 설치할 수도 있으며 JSP 엔진이 포함된 웹서버를 이용할 수도 있다. JSP 엔진으로는 톰캣(Tomcat), 레진(resin), JRun 등이 있다.

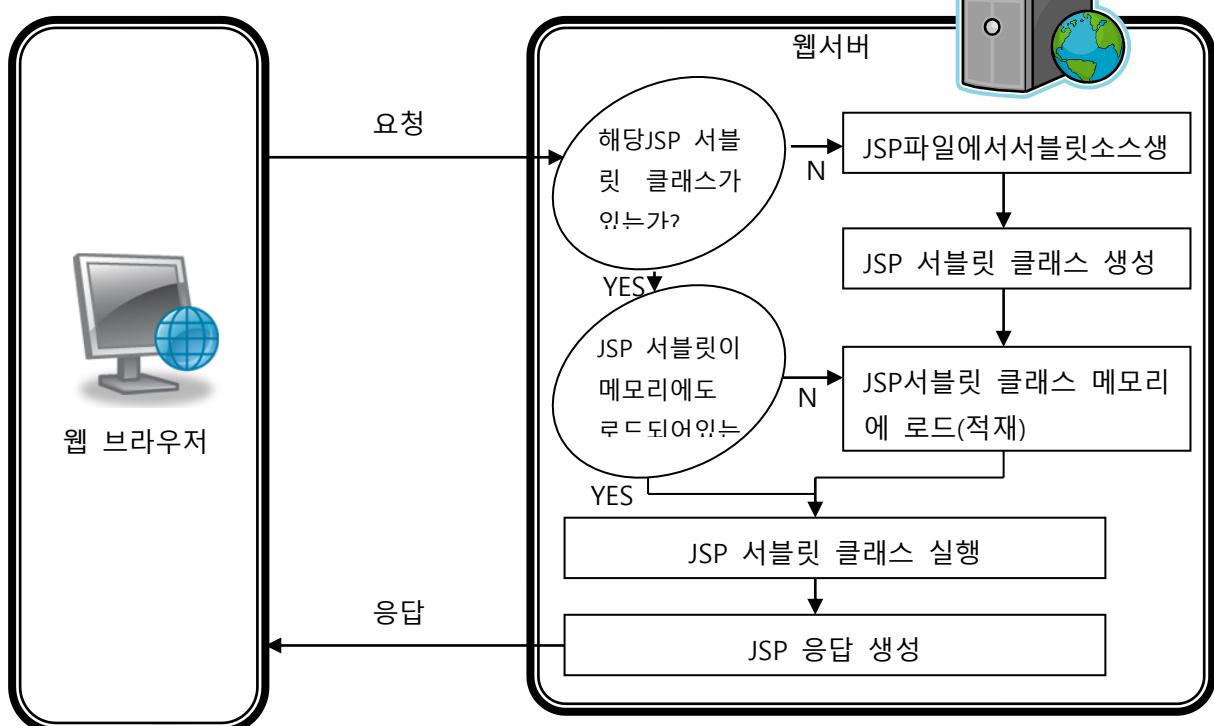


3.2 JSP 실행과 라이프 사이클

1) 서블릿 실행

JSP 엔진이 JSP 프로그램에서 JSP 서블릿 클래스를 생성하여 실행하는 과정을 살펴보자

- ① 클라이언트가 JSP 프로그램을 요청하면 JSP 소스를 해당 JSP 서블릿으로 변환하면서 시작한다.
물론 이미 서블릿 소스가 있다면 기존의 서블릿 소스를 이용한다.
- ② 이미 클래스가 있다면 메모리에 로드되어 있는지 검사한다. 이미 메모리에 로드되어 있다면 ⑤번을 실행한다.
- ③ JSP 서블릿 소스를 컴파일하여 서블릿 클래스를 생성한다.
- ④ JSP 서블릿 클래스를 메모리에 로드(적재)한다.
- ⑤ 메모리에 로드된 JSP 서블릿을 실행한다.
- ⑥ JSP 서블릿의 응답을 생성하여 클라이언트에 응답한다.

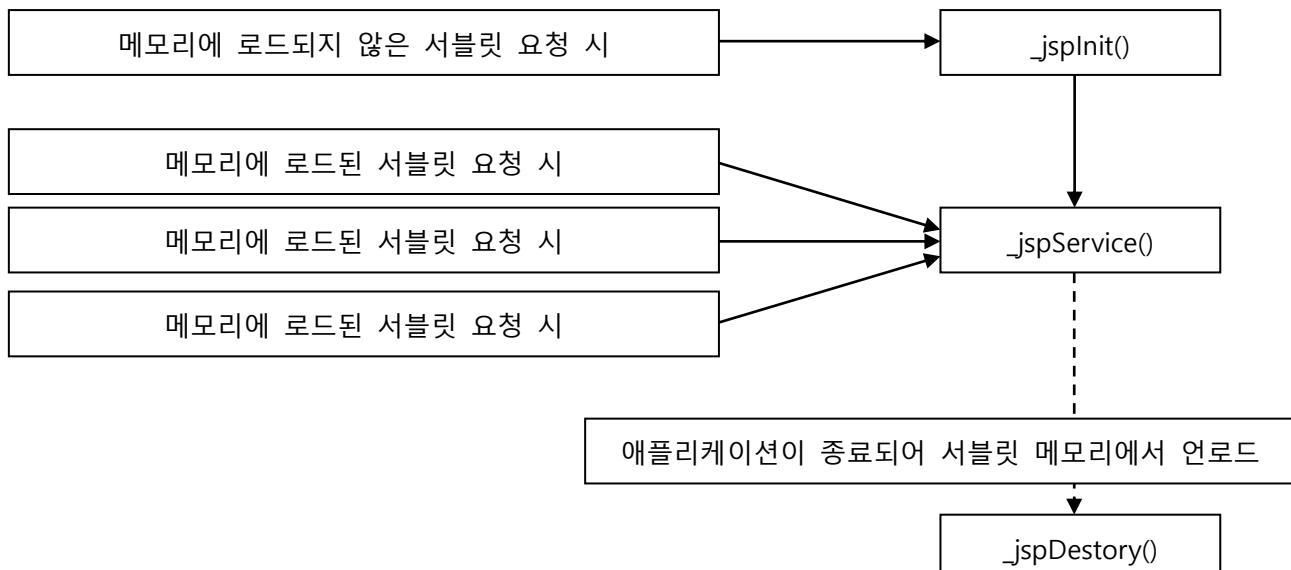


2) JSP 라이프 사이클

JSP 서블릿 클래스에는 주요 메소드 `_jspInit()`, `_jspService()`, `_jspDestory()`가 존재한다. JSP 서블릿 클래스 메소드의 기능을 살펴보면 다음과 같다.

메소드	기능
<code>_jspInit()</code>	요구되는 자원의 연결 등의 초기화 작업을 수행
<code>_jspService()</code>	실제 클라이언트의 요청에 대한 작업 처리 수행으로 클라이언트 요청 때마다 반복 수행.
<code>_jspDestory()</code>	웹서버 또는 애플리케이션이 종료되는 경우에 서블릿을 메모리에서 언로드하는 경우, JSP 서블릿 종료를 위한 작업 수행.

메소드 `_jspInit()`는 서블릿이 처음 메모리에 로드될 때 실행되는 메소드로, 요구되는 자원의 연결 등의 초기화 작업을 수행한다. 메소드 `_jspService()`는 실제 여러 클라이언트의 요청에 대한 작업 처리를 수행하는 메소드로 로드된 이후 클라이언트의 요청이 있을 때마다 반복적으로 수행되는 메소드이다. 메소드 `_jspDestory()`는 웹 서버 또는 애플리케이션이 종료될 때 서블릿을 메모리에서 언로드(unload)하는 경우, JSP 서블릿 종료를 위한 작업을 수행하는 메소드이다.



4. JSP 운영환경 및 개발환경

4.1 운영환경

JSP를 운영하기 위해서는 웹서버와 JSP 엔진이 필요하다. 웹서버는 아파치, 톰캣 등을 사용할 수 있으며, JSP 엔진으로는 JBoss, 톰캣, Resin 등을 사용할 수 있다. 아파치와 같은 웹서버를 사용한다면 JSP 엔진을 따로 설치해야 하지만, 톰캣 같은 경우는 톰캣 자체에 웹 서버와 JSP 엔진이 함께 있으므로 한 번의 설치로 JSP를 운영할 수 있다.

4.2 JSP 개발환경

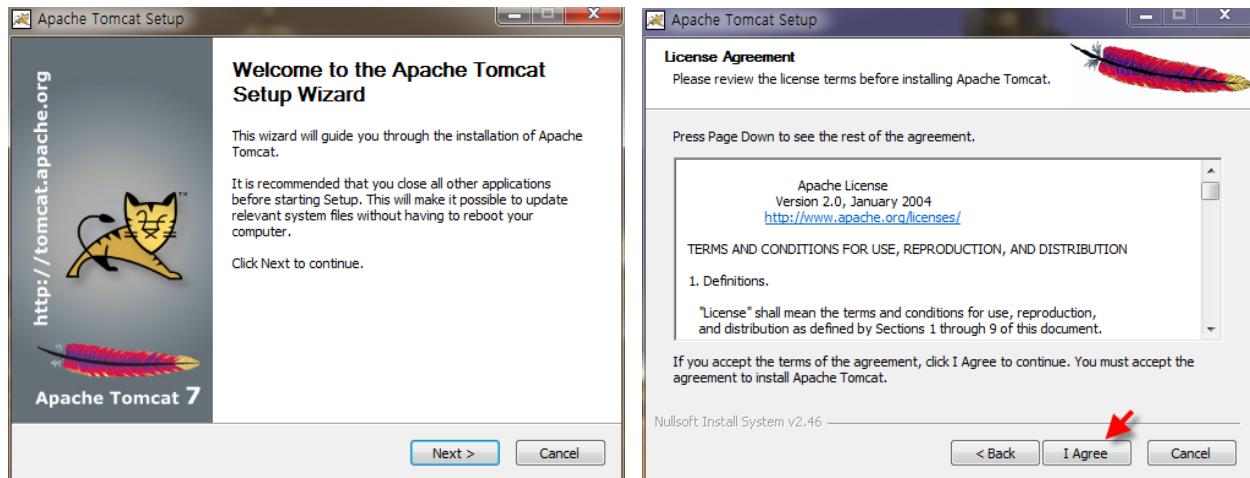
JSP 기술을 이용하여 웹 시스템을 개발하기 위해서는 다음과 같은 개발환경이 필요하다. JSP는 기본적으로 자바 기반 환경이므로 자바 운영 환경인 JDK가 필요하며, JSP 엔진과 웹서버 역할을 수행하는 시스템으로 Apache Tomcat Server가 필요하다. 또한 개발의 편의를 위하여 JSP 통합개발환경(IDE:Integrated Development Environment)이 필요한데, 여기서는 이클립스를 사용한다.

4.3 톰캣 설치 프로그램 내려받기

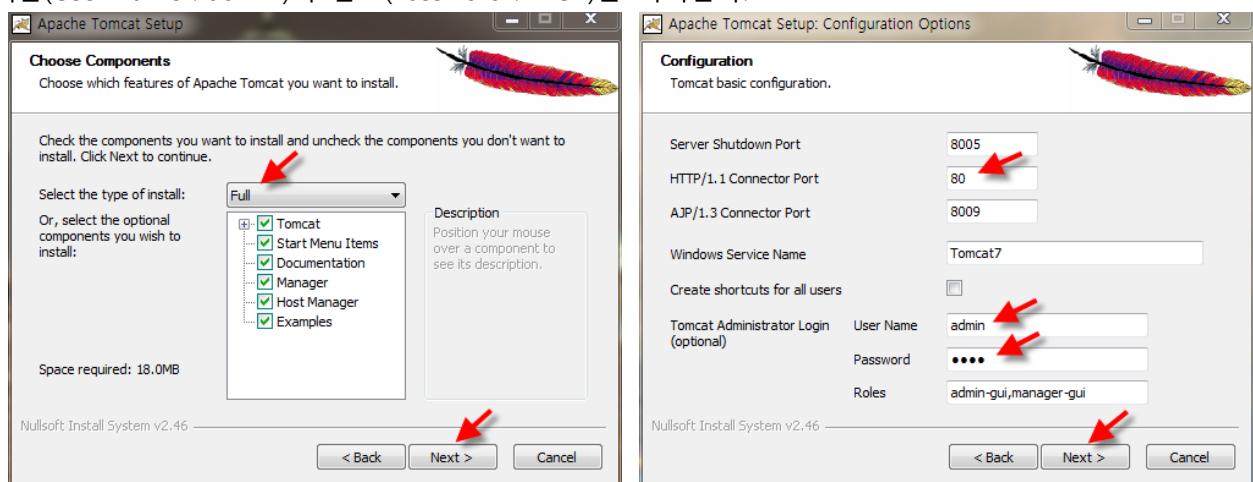
톰캣 설치 프로그램을 내려받기 위해 먼저 톰캣 홈페이지(tomcat.apache.org)에 연결(크롬)한 후, 왼쪽 부분의 [Download] 링크에서 [Tomcat 7.0]로 연결한다. 연결된 톰캣 홈페이지에서 [Binary Distributions]와 [32-bit/64-bit Windows Service Installer]를 클릭하면 실행 파일 [apache-tomcat-7.0.47.exe]을 내려 받을 수 있다. 관리자 권한으로 실행한다.

apache-tomcat-7.0.54-windows-x64.zip 을 C:\Jsp_Id\Apache-tomcat-7.0.54 에 압축 해제

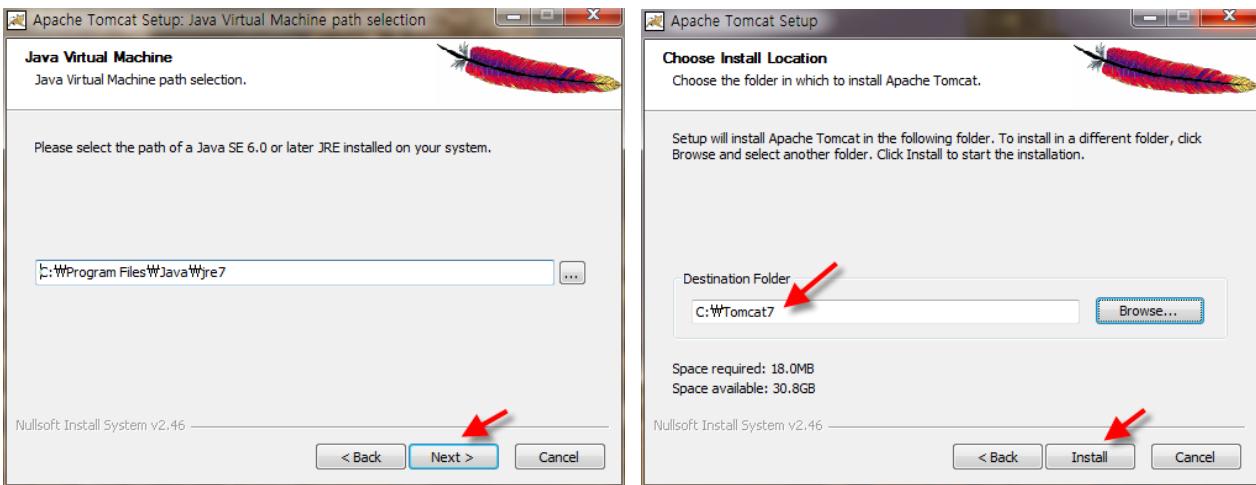
① 톰캣 설치 시작 대화상자와 라이선스 등의 대화상을 확인한다.



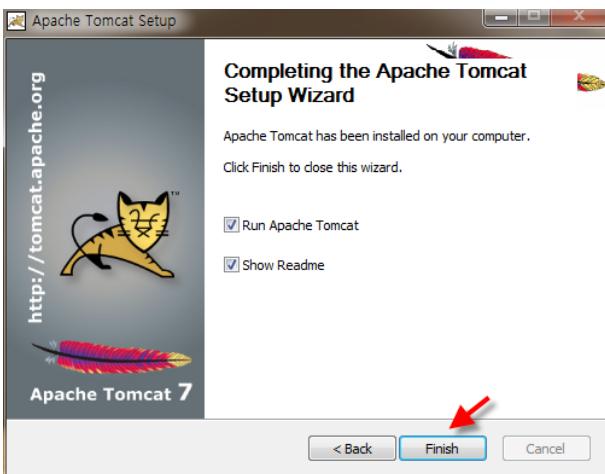
② 설치 구성요소(Choose Component) 대화상자에서 [Examples]를 선택하여 설치 후에 예제를 살펴보도록 하며, 설정 대화상자에서 연결 포트의 번호를 80으로 변경하고 관리 기능을 위한 관리자의 사용자 이름(User name : admin)과 암호>Password : 1234)를 기억한다.



③ 자바 가상 머신 대화상자에서 지정된 JRE 설치 폴더가 JDK를 설치할 때 지정된 JRE의 설치 폴더로 연결되어 있는지 확인한다. 설치 위치(Choose Installation Location) 대화상자에서 편의를 위하여 [C:\Tomcat7]으로 수정하여 설치한다.



- ④ 설치 종료 대화상자에서 톰캣 서버를 실행하는 [Run Apache Tomcat]을 체크박스의 선택을 확인하고 Finish 한다.

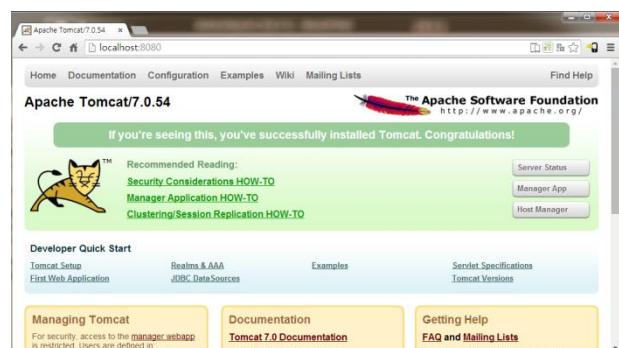


4.4 톰캣 설치 확인

정상적으로 톰캣이 설치되었다면 [시작] 메뉴의 [Apache Tomcat 7.0] 메뉴를 확인할 수 있다. [Apache Tomcat 7.0] 메뉴는 [Welcome] 등 6가지의 메뉴로 구성되며, 현재 톰캣 서버가 실행되고 있다면 메뉴 [Welcome]을 선택하여 웹브라우저로 톰캣 서버에 접속할 수 있다. 메뉴 [Welcome]은 웹브라우저로 주소 [http://localhost]에 접속하는 기능을 수행한다. 주소 URL에서 localhost는 현재 컴퓨터를 말하며 톰캣 서버의 연결 포트 번호는 생략한다. URL에서 localhost는 IP 주소로 127.0.0.1으로도 대체 가능하다.

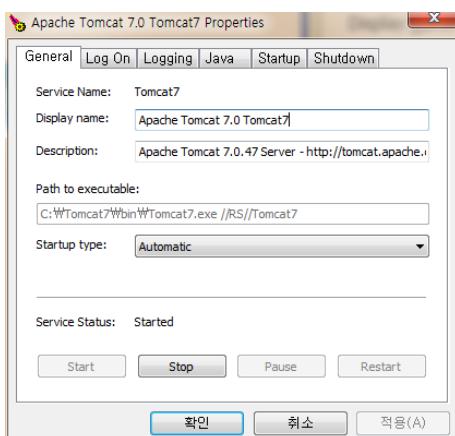
C:\Jsp_IdetApache-tomcat-7.0.54\bin\startup.bat

```
Tomcat
정보: Deployment of web application directory C:\Jsp_IdetApache-tomcat-7.0.54\webapps\host-manager has finished in 91 ms
6월 23, 2014 4:01:40 오후 org.apache.catalina.startup.HostConfig deployDirectory
정보: Deploying web application directory C:\Jsp_IdetApache-tomcat-7.0.54\webapps\manager
6월 23, 2014 4:01:40 오후 org.apache.catalina.startup.HostConfig deployDirectory
정보: Deployment of web application directory C:\Jsp_IdetApache-tomcat-7.0.54\webapps\ROOT has finished in 92 ms
6월 23, 2014 4:01:40 오후 org.apache.catalina.startup.HostConfig deployDirectory
정보: Deploying web application directory C:\Jsp_IdetApache-tomcat-7.0.54\webapps\ROOT
6월 23, 2014 4:01:40 오후 org.apache.catalina.startup.HostConfig deployDirectory
정보: Deployment of web application directory C:\Jsp_IdetApache-tomcat-7.0.54\webapps\ROOT has finished in 69 ms
6월 23, 2014 4:01:40 오후 org.apache.coyote.AbstractProtocol start
정보: Starting ProtocolHandler ["http-apr-8080"]
6월 23, 2014 4:01:40 오후 org.apache.coyote.AbstractProtocol start
정보: Starting ProtocolHandler ["ajp-apr-8009"]
6월 23, 2014 4:01:40 오후 org.apache.catalina.startup.Catalina start
정보: Server startup in 1529 ms
```



4.5 톰캣의 종료 및 실행

톰캣의 현재 실행 상태를 점검하려면 톰캣 설치 메뉴 [Monitor Tomcat]을 이용한다. [시작] 메뉴의 [Monitor Tomcat] 메뉴를 윈도우 하단 좌측 트레이에 아파치 톰캣 모니터 아이콘을 볼 수 있다. 이 아이콘을 클릭하거나 오른쪽 클릭 메뉴에서 [Configure...]를 누르면 톰캣 속성(Apache Tomcat Properties) 대화상자를 볼 수 있다. 이 톰캣 속성 대화상자는 톰캣의 설치 메뉴인 [Configure Tomcat]으로 바로 실행 할 수 있다. 이 톰캣 속성 대화상자는 현재의 톰캣 서버의 서비스 상태(Service Status)를 알 수 있으며 [Start], [Stop] 등의 버튼을 이용하여 톰캣 서버를 시작하거나 중지할 수 있다.



4.6 이클립스 설치 프로그램 내려 받기

자바 웹 개발용 이클립스 설치 프로그램을 내려 받으려면, 이클립스 다운로드 홈페이지 (www.eclipse.org/downloads)에서 [Eclipse IDE for Java EE Developers]로 연결한다. 연결된 페이지에서 [Korea, Republic Of]를 클릭하면 압축 파일[eclipse-jee-kepler-SR1-win32-x86_64.zip]을 내려받을 수 있다.

4.7 기본 환경 설정

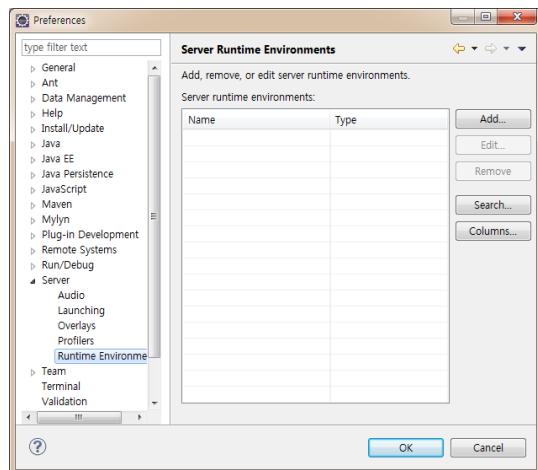
이클립스에서 주요 환경 설정은 메뉴 [Window]하부 [Preference]에서 수행한다. 메뉴[Preferences]를 실행하여 표시된 대화상자에서 왼쪽의 [java]/[Installed JREs]를 선택하면 현재 이클립스에서 이용되는 JRE를 확인할 수 있다. 초기 설치시 JRE는 다음과 같이 [Program Files]의 하부 폴더의 JRE이다.

편의를 위하여 기본 JRE를 설치된 JDK의 하부 JRE로 수정해보자. 이 수정은 편의를 위한 수정이므로 환경 설정에서 필수 사항은 아니다. 여기서 편의란 주요 자바 클래스의 소스를 확인한다든가 다른 버전의 JRE를 이용하는 등의 편리성을 말한다. [Preferences] 대화상자에서 오른쪽 [Add...] 버튼을 이용하여 [Add JRE] 대화상을 실행한 후, [Standard VM]을 선택하여 [Next]버튼을 눌러 다음 화면으로 이동한다. [JRE Home]을 지정하기 위해 버튼 [Directory]를 이용하여 설치한 JDK의 폴더 하부의 jre를 지정한다. [JRE system libraries]가 표시된 것을 확인하고 [Finish] 버튼을 누르면 현재 지정한 JRE가 추가된다.

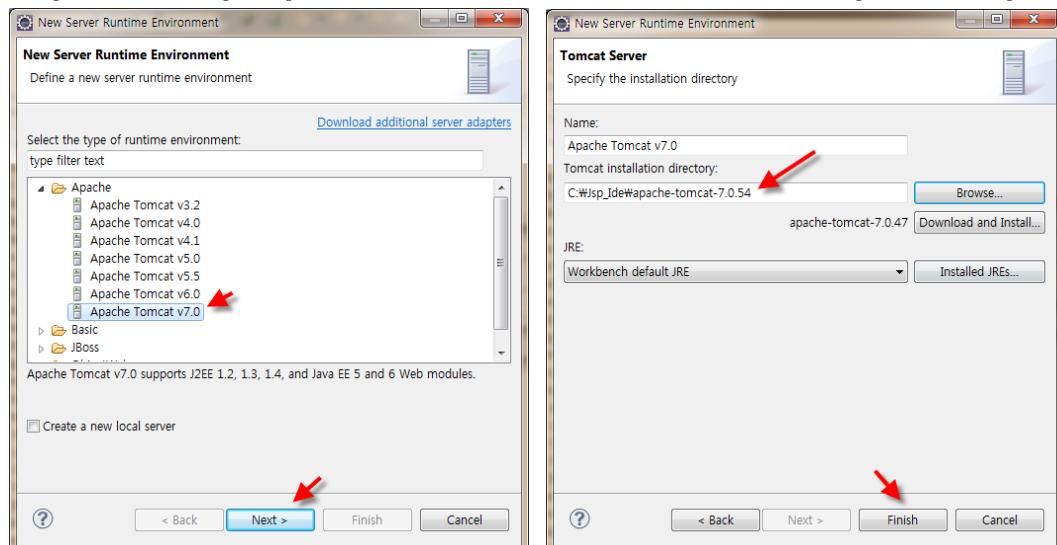
4.8 톰캣 서버 실행 환경 설정

JSP 서버로 톰캣을 이용하는 JSP프로그래밍을 수행하려면 이클립스에서 톰캣 서버를 실행 환경으로 설정해야 한다. 이를 위하여 [Preferences] 대화상자의 왼쪽 항목 [Server]/[Runtime Environment]를 선택한다. 서버 실행 환경이 하나도 지정되지 않은 초기에는 다음과 같이 오른쪽 서버 실행 환경 목록에 자료

가 전혀 없다. 이제 [Add] 버튼을 이용하여 이전에 설치된 톰캣을 지정하고 서버 실행 환경을 추가해 보자.

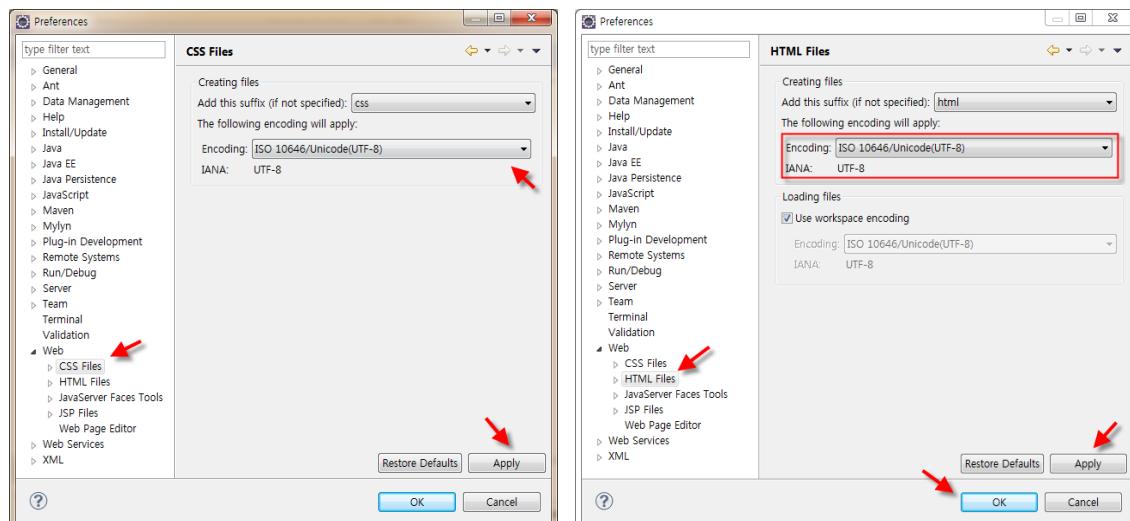


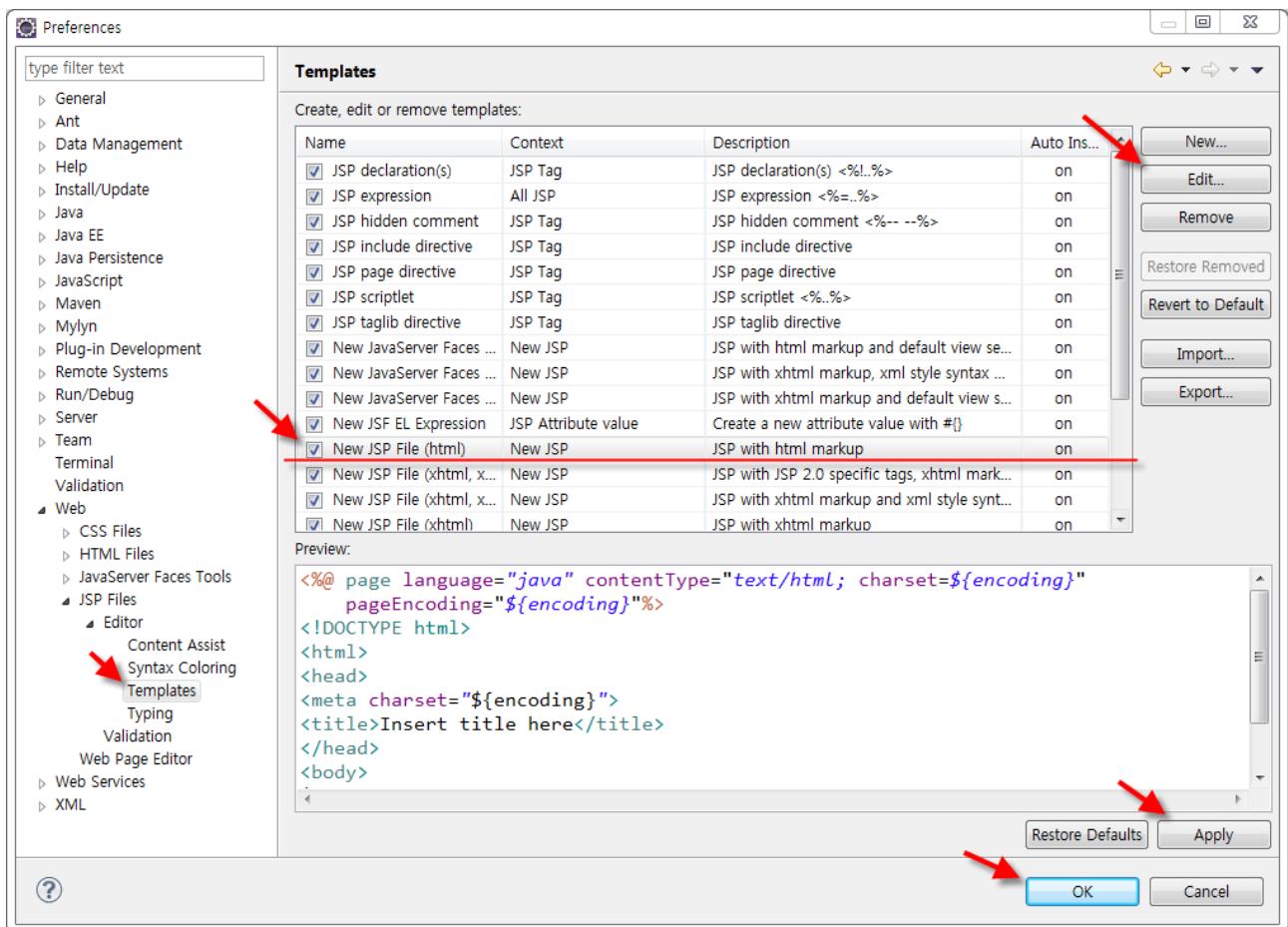
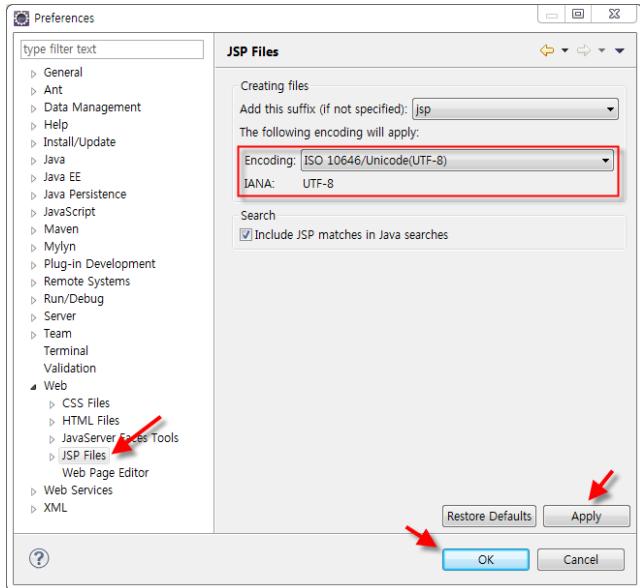
[Add]버튼에 위한 [New Server Runtime Environment] 대화상자에서 추가할 JSP서버인 [Apache Tomcat v7.0]을 선택하고 [Next]버튼을 눌러 이미 설치된 톰캣의설치 폴더인 [C:\Tomcat7]을 지정한다.



위와같이 지정한 후 [New Server Runtime Environment] 대화상자에서 [Finish] 버튼을누르면 다음과같이 새로운 서버 실행 환경으로 Apache Tomcat v7.0이 추가된 [Preferences] 대화상자를 확인할 수 있다.

Window -> Preferences 클릭한다. UTF-8로 변경한다.





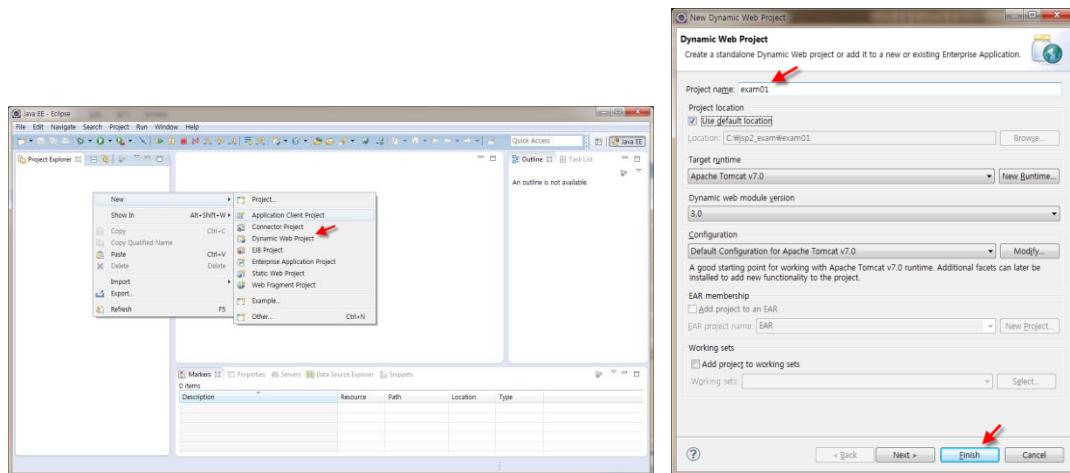
위와 같이 변경한다.

4.9 웹 프로젝트 생성

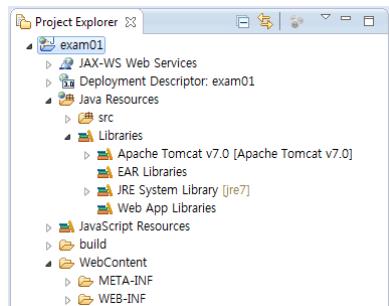
JSP 프로그래밍을 위해 제일 먼저 해야 할 일은 프로젝트를 생성하는 것이다.

메뉴 [File]/[New]/[Dynamic Web Project]를 선택하여 [New Dynamic Web Project] 대화상자에서 프로젝트 이름을 입력한다. 프로젝트 이름은 [exam01]으로 하고 표시되는 기본값을 살펴보기로 하자.

[Dynamic Web Project] 대화상자의 [Projects Contents]는 프로젝트 관련 파일이 저장되는 폴더로 작업공간 하부에 프로젝트 이름의 폴더인 [C:\Jsp_Id\workspace\exam01]으로 지정되는 것을 알 수 있다. [Target Runtime]은 JSP 프로그램이 실행되는 웹 서버로 현재 [Apache Tomcat 7.0]임을 확인 할 수 있다.

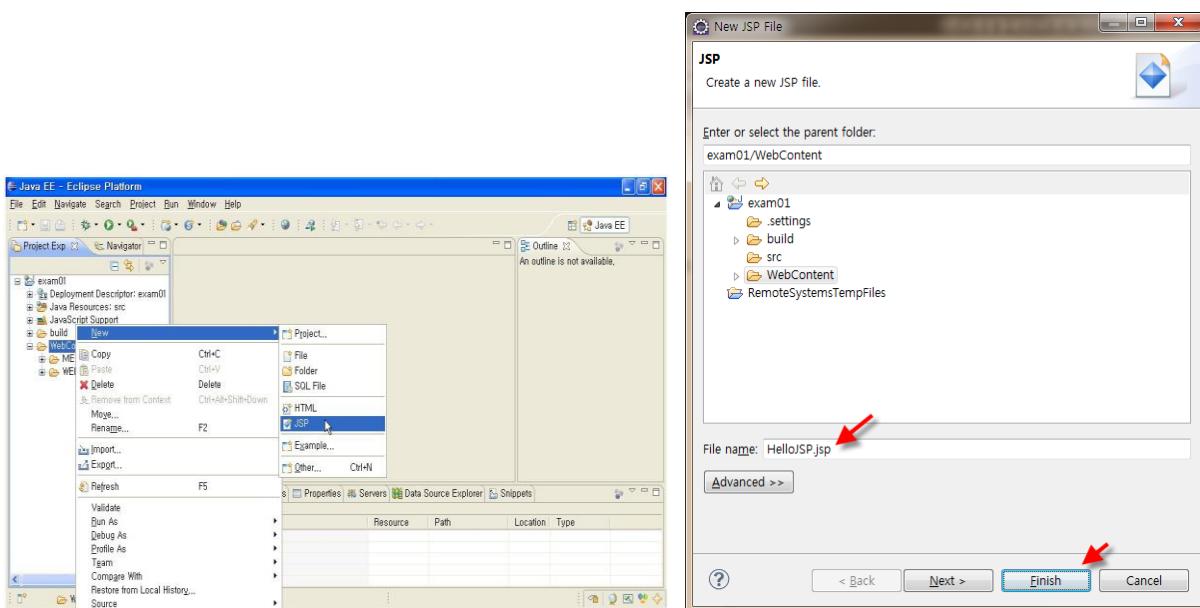


[Finish]를 클릭하면 [Project Explorer]뷰에 프로젝트 아이콘 exam01을 볼 수 있다. 프로젝트 하부에는 JSP 프로그램을 실행하기 위한 관련 라이브러리, 자바 프로그램 등이 모여있는 [Java Resources: src]가 있으며, 실제 JSP, html 파일이 저장되는 [WebContent] 등이 있다.

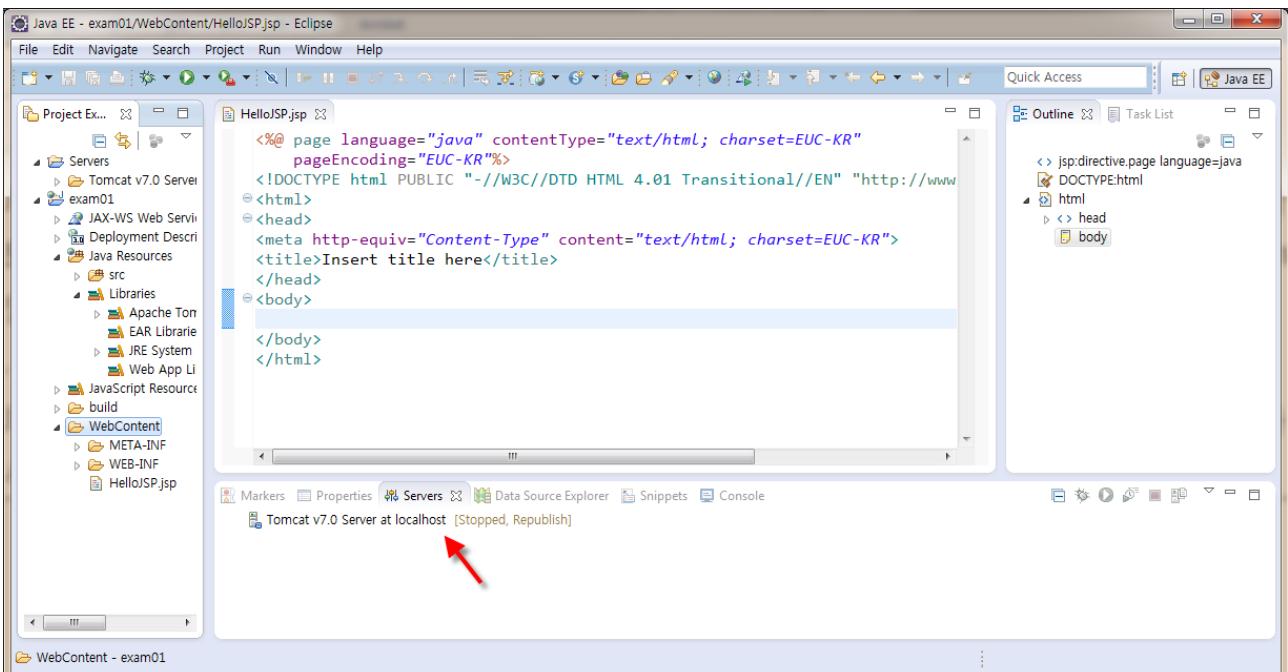


4.10 JSP 소스 생성

프로젝트를 생성했으니 이제 첫 JSP 프로그램을 작성해 보자. JSP 소스를 작성하려면 프로젝트를 오른쪽 클릭하고 메뉴 [New]/[JSP]를 선택한다. 표시된 [New JavaServer Page] 대화상자에서 소스가 저장되는 상위 폴더인 [exam01/WebContent]를 확인한 후 파일 이름 HelloJSP를 입력한다. 파일 이름은 대소문자를 구분하고 빈 문자도 가능하며, 확장자는 jsp가 없으면 자동으로 붙여준다.



다음은 프로젝트 [exam01]에 JSP 소스 HelloJSP.jsp가 작성된 이후의 이클립스 [Java EE Perspective]화면으로, 뷰[Project Explorer]에서 [WebContent] 하부에 JSP 파일이 생성된 것을 확인 할 수 있다. 만일 이 JSP 파일이 다른 장소에 있으면 문제가 생기므로 JSP 파일의 장소가 [WebContent]하부라는 사실을 반드시 확인하는 습관을 기르도록 한다. 이제 이클립스 중앙에는 JSP 소스 편집기가 자동으로 생성된 JSP 소스를 보여주고 있으며 원하는 편집이 가능하다.



JSP 프로그램을 위한 준비가 끝났으니 탐색기를 이용하여 작업공간이 있는 폴더를 확인해보자. 프로젝트 [exam01]이 작업공간 [workspace] 하부에 생성되었으며 폴더 [exam01]/[WebContent] 하부에 JSP 파일 HelloJSP.jsp가 생성된 것을 확인할 수 있다.

JSP 소스를 보면 첫 문자 <%@ page ...%>인 JSP 지시자로 현재 JSP 페이지 형식등을 지정한다. 이 지시자를 제외한 대부분이 HTML 소스인 것을 알 수 있다.

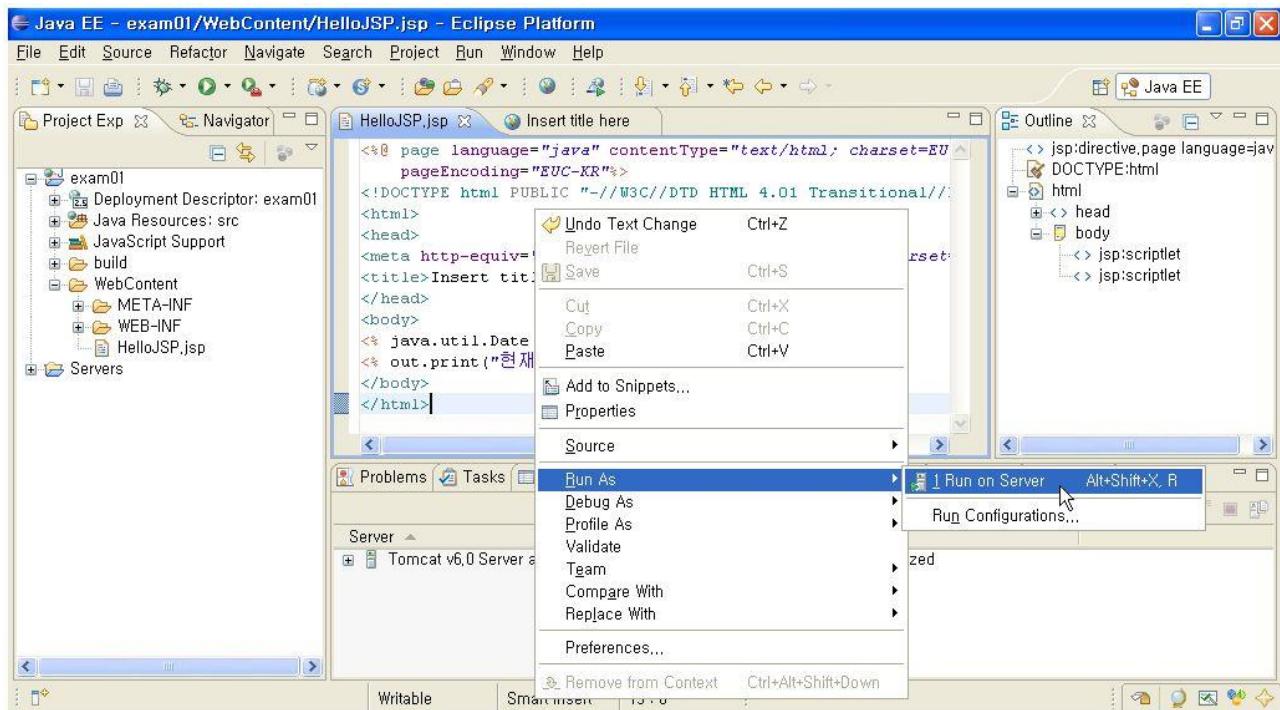
```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"><title>JSP 첫 문서 작성하기</title>
</head>
<body>
    <% java.util.Date today = new java.util.Date(); %>
    <% out.print("현재 시각은 [" + today + "] 입니다"); %>
</body>
</html>

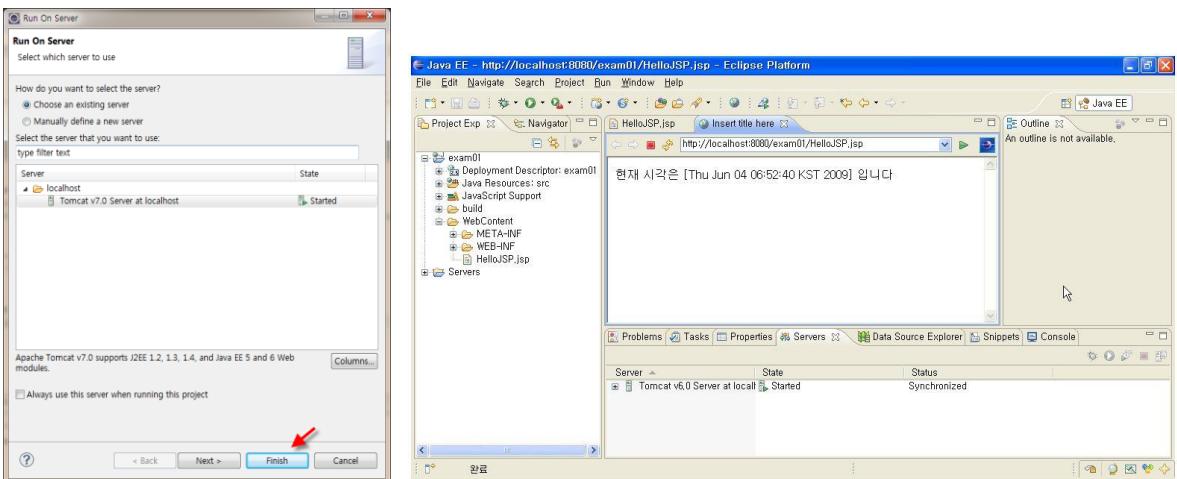
```

JSP에서 자바 문장을 기술하기 위해서는 태그 <%>로 시작하여 %>로 종료한다. 클래스 Date로 현재의 시간 정보를 저장한 객체 today를 생성한다. JSP에서 자바 문장을 이용하여 원하는 문장을 출력하는 프로그램을 작성하기 위해서 out 객체를 이용한다 out의 print() 메소드를 이용하여 브라우저에 원하는 문자열이나 객체를 출력한다.

이제 작성한 JSP 프로그램을 실행하자. 실행하려는 소스에서 메뉴 [Run As]/[Run on Server]를 선택한다.

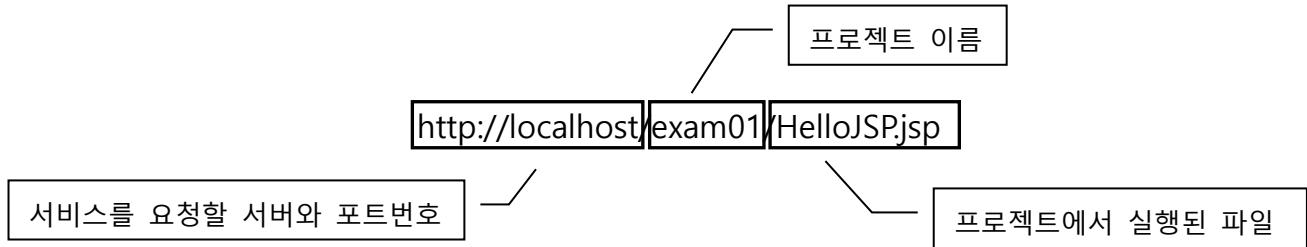


표시된 [Run on Server] 대화상자에서 서버 유형이 [Tomcat v7.0 Server]임을 확인하고, [Finish] 버튼을 누른다. 이클립스 내부에서 처음 톰캣 서버를 실행한다면 [Manually define a new server] 라디오 버튼만을 선택할 수 있다.



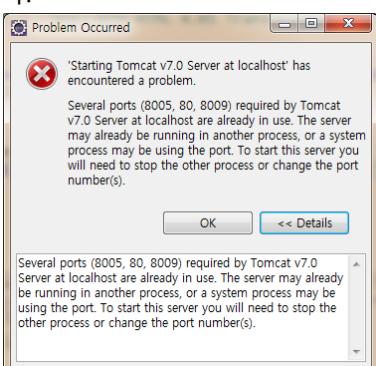
이클립스 화면 하단에 위치한 [Console] 뷰에 톰캣 서버의 실행을 알리는 메시지가 여러 줄 표시되고 [Servers] 뷰에 서버 이름과 함께 현재 상태가 표시되며 편집기가 있던 중앙 부분에 웹 브라우저가 나타나 JSP 프로그램이 실행된다.

[Servers] 뷰의 [Tomcat v7.0 Server at localhost]를 확장하면 생성된 프로젝트 [exam01]을 확인할 수 있으며 이를 클릭하면 중앙에 소스 HelloJSP.jsp가 실행된 웹 브라우저가 표시된다. 웹 브라우저의 URL을 보면 [http://localhost/exam01>HelloJSP.jsp]가 표시된 것을 알 수 있다.



JSP 프로그램에서 다음과 같은 대화상자의 오류가 발생하면 외부나 내부에 이미 톰캣 서버 또는 80 포트 번호로 다른 서버가 실행되어 있으므로 그 서버를 종료한 후 다시 실행하도록 한다. 이러한 문제가 발생하더라도 쉽게 포트를 이용하는 다른 서버를 찾을 수 없는 경우가 있으니 이런 경우, 이를 클릭해 다시 실행하거나 아예 윈도우 시스템을 종료한 후 다시 시작하기 바란다.

그리고 대화상자 [Preferences]에서 [General]/[Web Brower]를 선택한 후, 오른쪽에서 [Use external Web Brower]를 선택하면 JSP 프로그램을 실행하는 웹 브라우저를 외부에서 이용할 수 있다. 위와 같이 지정한 후, 실행하면 다음과 같이 외부 웹 브라우저에서 JSP가 실행되는 것을 볼 수 있다. 물론 외부 브라우저를 띄운 후 URL에 직접 http://localhost/exam01>HelloJSP.jsp를 기술해도 프로그램 HelloJSP.jsp가 실행된다.



5. 서블릿(Servlet)

서블릿은 웹 컨테이너에 의해서 관리되며 다양한 클라이언트로 요청에 의해서 동적인 콘텐츠로 응답 가능한 자바 기반의 웹 컴포넌트이다.

서블릿 웹 컴포넌트의 특징

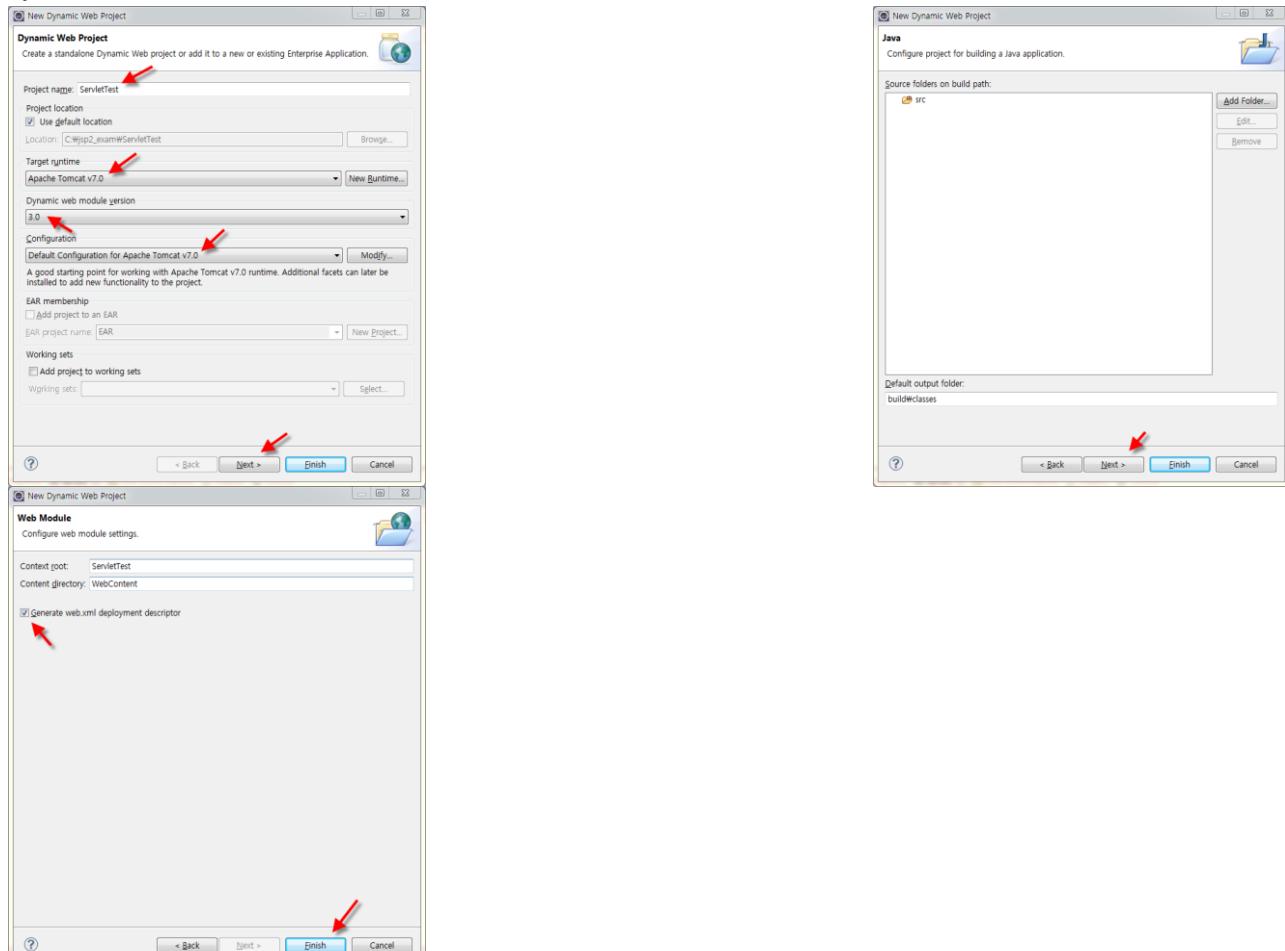
- 자바기반의 웹 컴포넌트로서 java 확장자를 갖는다.
 - 클라이언트의 요청에 의해서 동적으로 실행된다.(다양한 클라이언트 요구 사항을 처리 가능)
 - 클라이언트는 브라우저를 이용한 URL 지정을 통해 서블릿에 요청 가능하다.
 - 서블릿의 응답 결과는 일반적으로 HTML 형식으로 서비스된다. (자바 코드를 이용해서 클라이언트에 HTML 코드로 전송하는 추가 작업이 필요)
 - MVC 패턴을 적용하여 웹 어플리케이션을 개발한다면, 서블릿이 아닌 JSP에서 HTML 코드를 작성
- 서블릿은 반드시 웹 컨테이너에 의해서 관리되며, 자바 스레드로 동작되기 때문에 효율적으로 사용이 가능하다.
- MVC 패턴의 Controller 역할로서 서블릿이 사용된다.

5.1 HelloServlet 코드 작성

1) 이클립스 메뉴에서 [File]-[New]-[Dynamic Web Project]를 선택

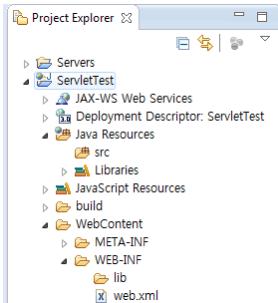
Project name에는 ServletTest문자열을 지정, Target runtime에는 Apache Tomcat 7.0으로 지정

Dynamic web module version에는 3.0으로 지정



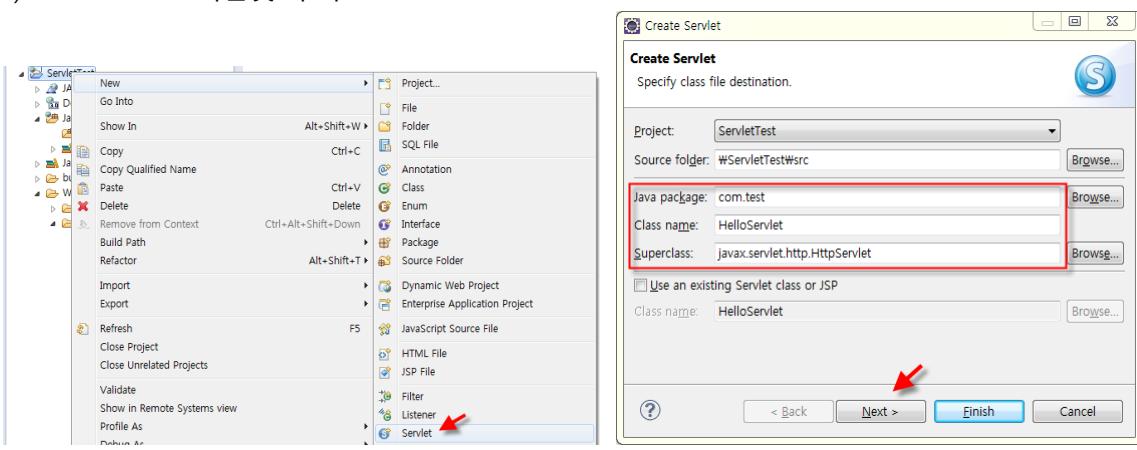
웹 컴포넌트(HTML, JSP, Servlet 및 이미지 파일)를 저장하는 디렉토리와 웹 컨테이너가 논리적으로 관리

하는 이름인 Context를 지정한다. 기본적으로 Context명은 웹 프로젝트명과 일치하고 저장 디렉토리명은 WebContent로 자동 지정된다. 임의의 변경이 가능하다. Context인 웹 애플리케이션에 대한 배치 지시자(Deployment Descriptor)인 'web.xml'을 생성시키기 위해서 체크박스에 체크하고 Finish한다.

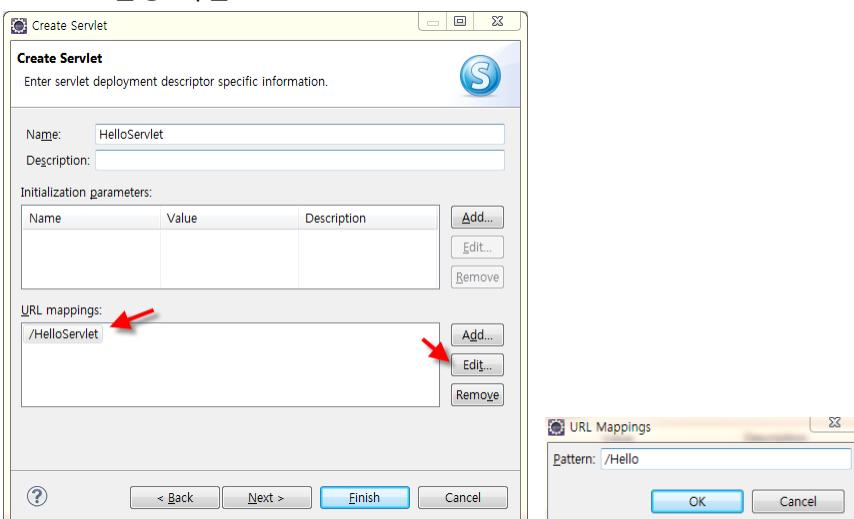


C:\WTomcat7\lib\ servlet-api.jar 와 jsp-api.jar 를 C:\Java7\jre\lib\ext 에 복사한다.

2) HelloServlet 서블릿 추가

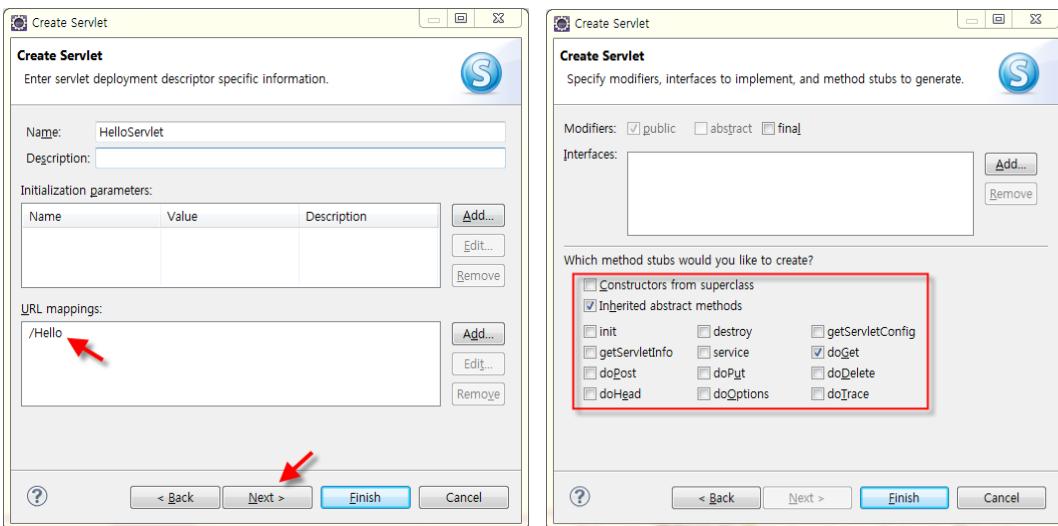


web.xml 설정 화면



URL mappings의 맵핑 이름을 변경한다.(반드시 '/' 뒤에 지정해야 한다.)

서블릿 요청 URL 형식은 'http://localhost:포트번호/URL 맵핑명'



서블릿 구현 메소드는 일반적으로 doGet() 또는 doPost() 메소드를 사용한다. 여기서는 doGet()만 선택한다.

```

package com.test;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.IOException;
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 HelloServlet
 */
@WebServlet("/Hello")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>Servlet Example</title>");
        out.println("</head><body><h1>Hello 서블릿 ~~!</h1>");
        out.println("</body></html>");
        System.out.println("서블릿 테스트");
    }
}

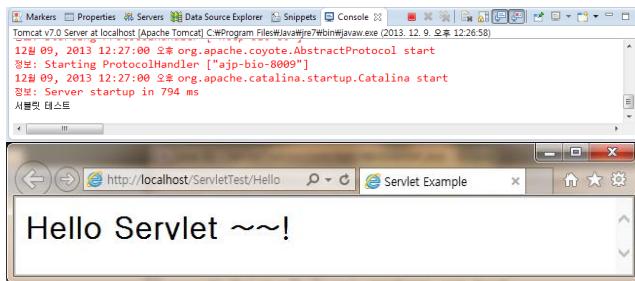
```

서블릿이 되기 위해서는 HttpServlet을 반드시 상속해야 한다. 서블릿 클래스는 public 이어야 한다.

```
}
```

Servlet 2.5에서는 web.xml에 URL 맵핑 정보를 지정했으나 Servlet 3.0에서는 @WebServlet("/Hello") 애노테이션을 이용해서 URL 맵핑 정보를 지정할 수 있다.

서블릿을 수행하는 invoker는 프로그래머가 작성한 서블릿의 doGet()이나 doPost() 메소드를 호출해서 서블릿을 실행하게 된다. 따라서 서블릿 작성시 doGet()이나 doPost() 메소드를 재정의함으로써 서블릿의 기능을 구현해야 한다.



- 서블릿 클래스 import

서블릿 프로그래밍에 필요한 클래스들을 임포트 한다.

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

javax.servlet 패키지는 서블릿을 프로그래밍하는데 필요한 클래스들이 정의되어 있으며, javax.servlet.http 는 HTTP 프로토콜로 제공되는 서블릿을 구현하기 위한 클래스들을 제공한다.

- HttpServlet 클래스

HttpServlet 클래스는 HTTP 프로토콜로 제공되는 서블릿을 구현하기 위한 클래스로써, 서블릿을 만들기 위해서는 반드시 HttpServlet 클래스를 상속받아야 한다.

```
public class HelloServlet extends HttpServlet { ... }
```

HttpServlet 클래스를 상속받은 HelloServlet 클래스를 작성한다.

- doGet()

doGet() 메소드는 클라이언트로부터 HTTP 프로토콜을 통해서 전달된 GET 메시지를 처리하기 위한 메소드이다. HttpServlet 클래스를 상속받고, doGet() 메소드를 재정의함으로써 서블릿을 만들 수 있다.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
    ...
}
```

5.2 서블릿 맵핑

1) web.xml에 설정하는 방법

Servlet 2.5까지 사용하던 방법이며 Servlet 3.0에서도 사용 가능하다.

WEB-INF 폴더안의 web.xml 파일에 <servlet> 태그와 <servlet-mapping> 태그를 사용하여 설정한다. 여러 개의 서블릿 등록이 가능하며 주의할 점은 url-pattern 값에는 임의의 값으로 지정 가능하지만, 반드

시 '/'를 사용해야 되고 <servlet> 태그의 <servlet-name> 값과 <servlet-mapping> 태그의 <servlet-name> 값도 임의의 값으로 지정 가능하지만, 반드시 일치해야 한다.

```
<servlet>
    <servlet-name>서블릿 별칭</servlet-name>
    <servlet-class>패키지를 포함한 서블릿명</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>서블릿 별칭</servlet-name>
    <url-pattern>/맵핑명</url-pattern>
</servlet-mapping>
```

예)

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.test.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

@WebServlet 애노테이션 또는 web.xml 설정 중 하나만 선택해서 사용한다.

2) @WebServlet 애노테이션(annotation)

애노테이션은 JDK 5.x부터 추가된 기능이다. 환경 설정 파일에 설정하지 않고 자바 코드에 직접 기술하는 방법으로 '@'로 시작된다.

만약 애노테이션을 사용하지 않으려면 web.xml 파일에 다음과 같은 속성을 값을 지정한다.

```
<web-app metadata-complete = "true" />
```

Servlet 3.0에 추가된 애노테이션

```
@WebServlet  @WebFilter  @WebInitParam  @DeclareRoles  @EJB  @EJBs
@PersistenceContext  @PersistenceContexts  @PersistenceUnit  @PreDestroy  @RunAs
```

@WebServlet 애노테이션은 서블릿 맵핑에 사용되면 2가지 방법이 있다.

① 서블릿 맵핑명만 지정하는 방식

```
@WebServlet("/맵핑명")
public class HelloServlet extends HttpServlet { ... }
```

② 추가 속성을 사용하는 방식

서블릿별칭과 urlPatterns 속성을 사용하여 여러 개의 맵핑명을 지정할 수 있다.

```
@WebServlet(name = "서블릿별칭", urlPatterns = {"/맵핑명", "/맵핑명2"})
public class HelloServlet extends HttpServlet { ... }
```

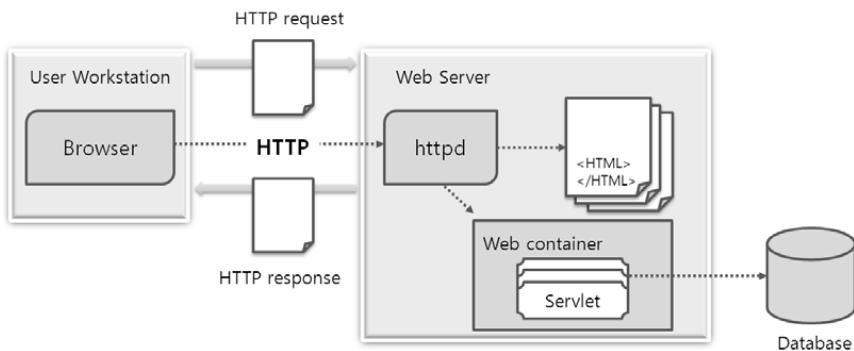
여러 개의 맵핑명을 지정할 수 있는 다른 방식은 value 속성을 사용한다.

```
@WebServlet(name = "서블릿별칭", value = {"/맵핑명", "/맵핑명2"})
```

```
public class HelloServlet extends HttpServlet { ... }
```

5.3 서블릿 아키텍처 및 핵심 API

서블릿을 사용하는 경우의 웹 아키텍처이다. 클라이언트에서 웹 브라우저를 이용하여 적절한 URL 형식으로 서블릿에 요청하면 웹 컨테이너에서 서블릿을 실행하고 결과 값을 html로 구성하여 클라이언트로 응답 처리한다.



javax.servlet.http.HttpServletRequest : HTTP Request인 요청과 관련된 핵심 API

javax.servlet.http.HttpServletResponse : HTTP Response인 응답과 관련된 핵심 API

1) HttpServletRequest API

HTTP Request 관련 작업을 처리하는 핵심 API로 클래스가 아닌 인터페이스로 제공한다.

- | | |
|-----------------------------------|--|
| · getHeader(String name):String | · setCharacterEncoding(String encoding) |
| · getHeaderNames():Enumeration | · setAttribute(String name, Object obj) |
| · getCookies():Cookie[] | · getAttribute(String name):Object |
| · getRequestURI():String | · removeAttribute(String name) |
| · getServletPath():String | · getParameter(String name) |
| · getSession(boolean):HttpSession | · getParameterNames():Enumeration |
| · getSession():HttpSession | · getParameterValues(String name):String[] |

2) HttpServletResponse API

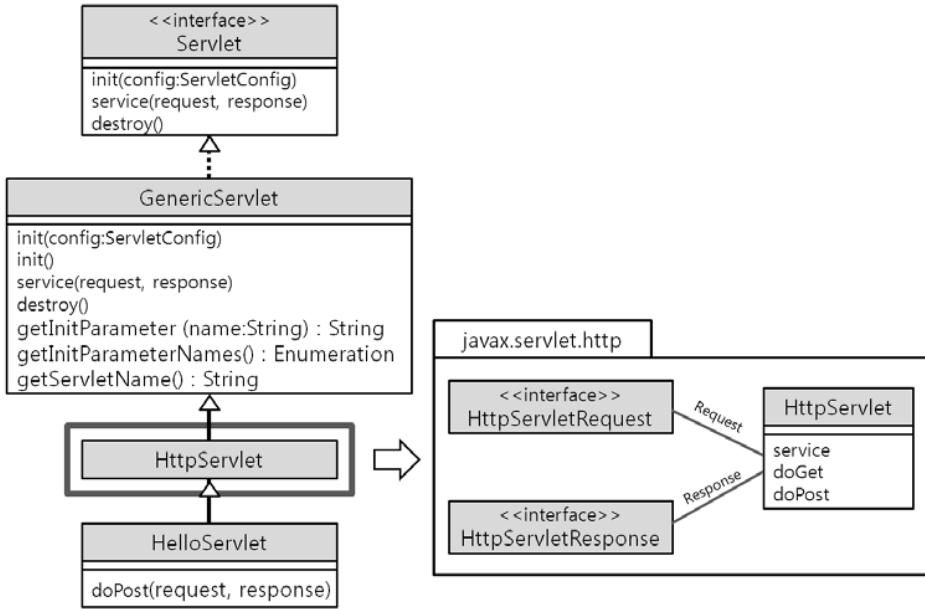
HTTP Response 관련 작업을 처리하는 핵심 API로 클래스가 아닌 인터페이스로 제공한다.

- | | |
|--|---|
| · addCookie(Cookie c) | · sendRedirect(String loc) |
| · addHeader(String name, String value) | · getWriter():PrintWriter |
| · encodeURL(String url) | · getOutputStream():ServletOutputStream |
| · getStatus() | · setContent-Type(String type) |

3) HttpServlet API

서블릿을 구현하기 위한 핵심 API로 일반 클래스가 아닌 추상 클래스로 제공한다.

HttpServlet의 계층 구조



5.4 서블릿 컨테이너의 역할

- 요청 및 응답 객체 생성

웹 서버로부터 서블릿에 대한 요청이 들어오면 서블릿 컨테이너는 요청 정보를 이용해서 HttpServletRequest 객체와 HttpServletResponse 객체를 생성하게 된다. HttpServletRequest 객체는 클라이언트의 요청 정보를 담고 있으며, HttpServletResponse 객체는 클라이언트로 전달될 응답 객체를 나타낸다.

- 서블릿 객체 생성

요청 및 응답 객체를 생성한 후 서블릿 컨테이너는 요청된 서블릿이 서블릿 컨테이너에 존재하는지 검사하게 된다. 만약 서블릿 객체가 서블릿 컨테이너에 로딩되어 있다면 해당 서블릿 객체를 생성하고, 로딩되어 있지 않다면 해당 서블릿의 클래스 파일을 로딩한 후 객체를 생성한다.

- 서블릿의 실행

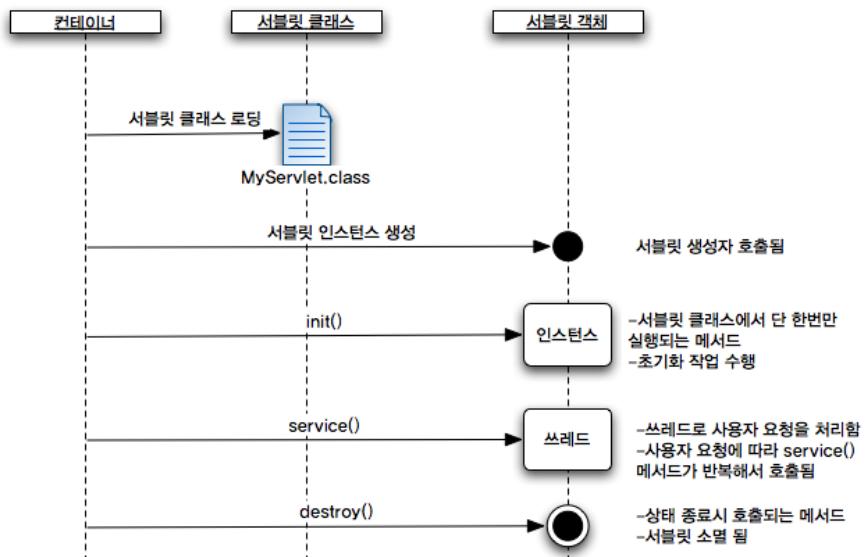
서블릿을 작서할 때 HttpServlet 클래스를 상속받아 서블릿에서 정의된 doGet()이나 doPost() 메소드를 재정의함으로써 서블릿을 구현하게 된다. 이렇게 작성된 doGet()이나 doPost() 메소드는 서블릿 객체가 생성된 후 서블릿 컨테이너에 의해 호출되어, 서블릿을 실행되는 것이다.

그 때 doGet() 메소드의 코드에서 첫번째 응답 객체인 HttpServletResponse의 setContentType() 메소드를 이용해서 응답 페이지의 MIME 타입과 인코딩 스타일을 다음과 같이 지정해 준다.

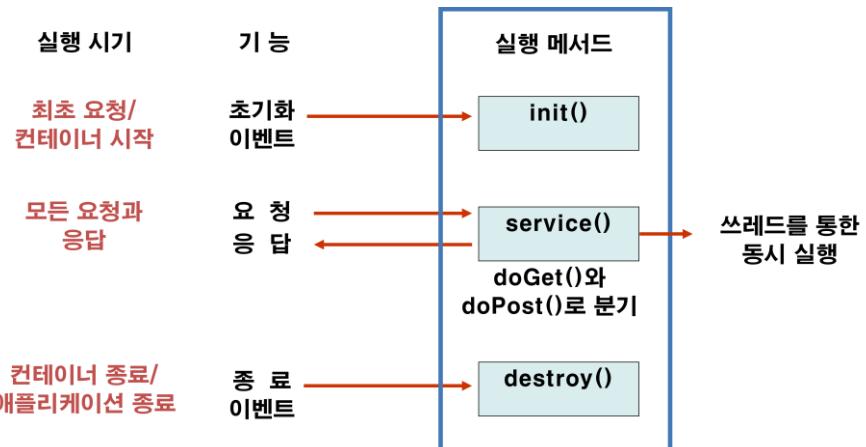
```
response.setContentType("text/html;charset=UTF-8");
```

응답 페이지의 MIME 타입 즉 웹 브라우저에게 응답을 보낼 때 응답 결과를 **text/html**이며, 문자 인코딩 방법은 한글을 처리하기 위한 **UTF-8**로 지정해 주었다. 다음으로는 웹 브라우저에 응답을 보낼 내용을 화면에 표출 할 때 PrintWriter 타입의 객체를 사용해서 이 객체의 인스턴스를 통해서 화면에 출력해야 한다. 이 때 getWriter() 메소드를 이용해서 PrintWriter 객체를 얻어냈고, 인스턴스로 out을 사용한다.

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



서블릿 생명 주기



① init() 메서드

웹 컨테이너에 의해서 서블릿 인스턴스가 처음 생성될 때, 단 한번 호출된다. 따라서 서블릿에서 필요한 초기화 작업 시 주로 사용한다.

② service() 메서드

클라이언트가 요청할 때마다 호출된다. 클라이언트가 원하는 동적인 처리 작업 시 필요하다. 일반적으로 service 메서드보다는 doGet 또는 doPost 메서드를 사용한다.

③ destroy() 메서드

서블릿 인스턴스가 웹 컨테이너에서 제거될 때 호출된다. init 메서드에서 구현했던 초기화 작업을 반납 처리하는 작업 시 주로 사용된다.

5.5 HttpServlet의 동작과 구현

- 서블릿의 상속 구조

HttpServlet(HTTP 프로토콜을 위한 HTTP 전용 서블릿) 추상 클래스는 GenericServlet(일반적인 네트워크 프로토콜을 위한 추상 클래스) 추상 클래스를 상속 받았으며, GenericServlet은 ServletConfig(서블릿 환경을 위한 인터페이스), Servlet(서블릿 기능 구현을 위한 인터페이스), Serializable(서블릿 직렬화를 위한

인터페이스) 인터페이스를 구현한 클래스이다.

- **ServletConfig**

ServletConfig 인터페이스는 서블릿과 관련된 초기화 정보를 읽고 처리하는데 필요한 메소드를 정의하고 있는 인터페이스이다.

- **Servlet**

Servlet 인터페이스는 서블릿에서 가장 기본이 되는 인터페이스로써, 서블릿의 생명 주기(init() : 서블릿이 처음 로딩될 때 한번 호출되는 메소드, service() : 클라이언트로부터 서블릿에 대한 요청이 있을 때 호출되는 메소드, destroy() : 서블릿이 메모리로부터 제거될 때 호출되는 메소드)와 관련된 메소드를 담고 있으며 서블릿을 만들기 위해서는 반드시 Servlet 인터페이스 직접적으로나 간접적으로 구현해야만 한다.

처음으로 서블릿의 요청이 들어왔을 때 서블릿은 서블릿 컨테이너에 의해 자동으로 메모리에 로딩된다. 메모리로 서블릿 클래스가 로딩되는 즉시 객체를 생성하게 되며 객체 생성과 동시에 init() 메소드를 호출하게 된다. init() 메소드는 서블릿을 로딩할 때 단 한번 호출되며 문제가 발생했을 때 ServletException을 발생시킨다.

만약 성공적으로 init() 메소드가 호출되었다면 service() 메소드를 수행해서 클라이언트의 요청을 실행한다. 두 번째 클라이언트의 요청이 있을 때부터는 service() 메소드를 호출하여 서블릿을 실행하게 된다. service() 메소드는 클라이언트의 요청 방식에 따라 GET 방식이면 doGet() 메소드를, POST 방식이면 doPost() 메소드를 호출한다.

서블릿 객체가 더 이상 서비스를 하지 않고 있는 경우, 메모리에서 제거되며 이때 호출되는 메소드가 바로 destroy() 메소드이다. destroy() 메소드가 호출되면, 가비지 콜렉터는 메모리로부터 서블릿 객체를 제거하게 된다.

결국 **init()과 destroy() 메소드는 단 한번 호출되며, service() 메소드는 클라이언트의 요청이 있을 때마다 호출되는 것이다.**

- **Serializable**

Serializable 인터페이스는 서블릿 객체의 직렬화를 위한 인터페이스이다.

객체 직렬화란 메모리에 생성된 클래스 객체의 현재 상태를 그대로 보존해서 파일에 저장하거나 네트워크를 통해 전달할 수 있는 기능을 말한다. 자바에서는 객체의 직렬화를 위한 Serializable 인터페이스를 구현하면 된다. 서블릿 Serializable 인터페이스를 구현한 이유는 네트워크를 통해 응답 스트림을 전달하기 때문이다.

- **GenericServlet**

GenericServlet 클래스는 ServletConfig, Servlet, Serializable 세 개의 인터페이스를 구현한 추상 클래스로써, 일반적인 네트워크 프로토콜에서 사용되는 서블릿을 구현하기 위한 클래스이다. GenericServlet은 추상 클래스이기 때문에 그 자체로는 사용할 수 없으며, 반드시 상속해야만 사용 할 수 있다.

GenericServlet이 구현하고 있는 Servlet, ServletConfig 인터페이스는 서블릿 통신을 할 때 필요한 기본적인 메소드를 가지고 있으며, 이를 GenericServlet에서 모두 구현하고 있다. 또한 GenericServlet 클래스는 서블릿의 직렬화를 위해 Serializable 인터페이스를 구현하고 있다.

서블릿을 만들 때 상속받아 쓰는 HttpServlet 클래스도 GenericServlet 클래스를 상속받은 클래스이다. 사실 웹 애플리케이션은 거의 모두가 HTTP 프로토콜을 기반으로 하기 때문에, HTTP 프로토콜에 알맞게 만들어진 HttpServlet 클래스를 상속받아 서블릿을 만드는 것이 일반적이다. 흔한 경우는 아니지만 HTTP 프로토콜 이외의 FTP 프로토콜 등을 사용하는 서블릿을 작성하려면 GenericServlet 클래스를 상속 받아 서블릿을 만들어야 한다.

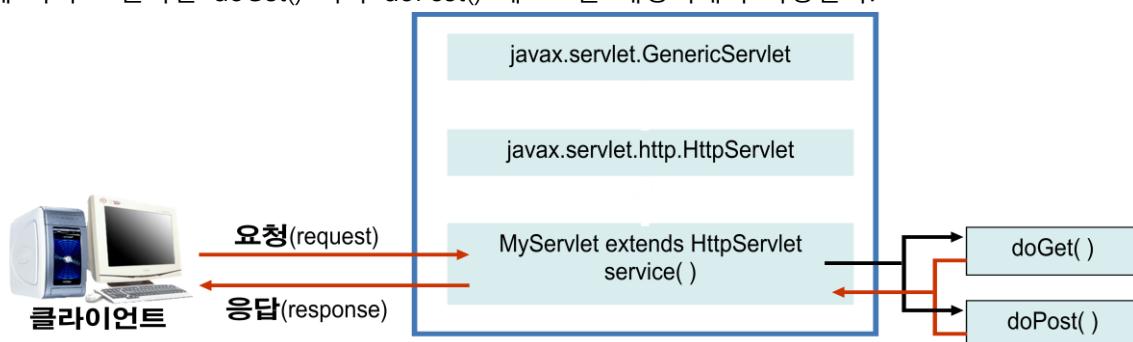
-HttpServlet

HttpServlet 클래스는 GenericServlet 클래스를 상속받아 HTTP 프로토콜에 맞게 구현된 클래스이다. HttpServlet 클래스를 상속받아 서블릿을 구현한 이유는 바로 월드 와이드 웹이 HTTP 프로토콜을 사용하기 때문이다.

5.6 HttpServlet 클래스의 동작 과정

HttpServlet 클래스는 일반적인 네트워크 프로토콜을 위한 GenericServlet 클래스를 상속받아 HTTP 프로토콜에 맞게 구현된 클래스로써, 웹 클라이언트로부터 들어온 요청을 처리하고 웹 클라이언트에게 결과를 되돌려 주는 작업을 수행한다. HttpServlet 클래스는 서블릿을 구현할 때 가장 많이 사용되는 클래스로써 서블릿을 만들 때 일반적으로 doGet() 혹은 doPost() 메소드를 재정의해서 사용한다. HttpServlet 클래스는 GenericServlet 클래스를 상속받았기 때문에 클라이언트의 요청이 전달될 경우 service() 메소드가 가장 먼저 호출된다. service() 메소드의 내부에서는 클라이언트로부터 전달된 요청의 방식, 즉 GET 방식인지 POST 방식인지에 따라 doGet() 혹은 doPost() 메소드를 호출하게 된다. 그렇기 때문에 HttpServlet 클래스를 상속받아 서블릿을 구현 할 때 클라이언트의 요청 방식에 따라 doGet() 혹은 doPost() 메소드를 재정의해서 서블릿을 구현해야 한다.

HttpServlet 클래스를 상속받아 작성한 서블릿의 생명 주기 또한 앞서 살펴본 것과 동일하게 init(), service(), destroy() 메소드가 순서대로 호출된다. 하지만 HttpServlet 클래스는 service() 메소드를 재정의해서 사용하지 않는다. service() 메소드는 자동으로 호출되고, service() 메소드가 클라이언트의 요청 방식에 따라 호출하는 doGet()이나 doPost() 메소드를 재정의해서 사용한다.



5.7 GET 방식과 POST 방식

- GET 방식

URL에 전달하고자 하는 정보를 포함해서 정보를 전달하는 방법이다. 이 때 단점으로 정보가 노출되므로 보안에 취약하며, 보내는 정보의 크기는 제한적이다. ? 이후의 값들을 서버에서 QUERY_STRING을 통해 전달된다. "속성=값" 형태로 사용해야 하며 "&"는 여러 속성 값을 전달할 때 연결해 주는 문자열이다.

GET 방식의 URL : <http://xxx.xxx.co.kr/servlet/login?id=hj&name=hong>

GET 방식의 URL: http://localhost:8080/method/viewParameter.jsp?name=java&address=seoul&pet=cat

GET 방식을 이용한 파라미터 전송 시 요청 데이터

01	GET /method/viewParameter.jsp?name=java&address=seoul&pet=cat HTTP/1.1	← 요청라인
02	Host: localhost:8080	
03	User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; ko; rv:1.9.0.3) ...	
04	Accept: text/html, application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8	
05	Accept-Language:ko-kr,ko;q=0.8,en-us;q=0.5,en;q=0.3	
06	Accept-Encoding:gzip,deflate	
07	Accept-Charset:UTF-8,utf-8;q=0.7,*;q=0.7	
08	Proxy-Connection:keep-alive	
09	Cookie:JSESSIONID=D1DEA89425A9F051B9A306511B15B328	

웹 브라우저는 HTTP 프로토콜에 맞춰 위 코드와 같은 데이터를 전송한다. HTTP 프로토콜에 따르면 첫 번째 줄은 요청방식과 HTTP 프로토콜 버전을 명시하도록 되어 있다. GET 방식으로 요청 파라미터를 전송하는 경우 파라미터가 URI와 함께 전송되는 것을 확인할 수 있다.

HTTP 프로토콜의 구성

HTTP 프로토콜은 크게 요청라인/응답 상태 라인, 헤더(header) 영역, 데이터(content) 영역의 세 부분으로 구성된다. 웹 브라우저가 웹 서버에 전송할 때에는 요청 라인이 사용되며, 웹 서버가 웹 브라우저에 요청 결과를 전송할 때에는 응답 상태 라인이 사용된다.

요청라인에는 요청 메소드와 요청 자원의 URI, 그리고 HTTP 프로토콜 버전이 명시된다.

헤더 영역에는 실제 주고 받을 데이터를 제외한 나머지 정보를 전달하는 헤더가 포함되며, 헤더는 '이름:값'으로 구성된다. GET 방식은 추가로 전송할 데이터를 갖지 않기 때문에 헤더 영역만 가지며 데이터 영역은 갖지 않는다.

- POST 방식

전달하고자 하는 정보를 첨부파일 형태로 포함해서 전달하는 방식 즉 데이터 영역을 이용해서 데이터를 전송하기 때문에 URL에 보내는 정보에 대한 노출이 없기 때문에 보안성이 좋으며 웹 브라우저나 웹 서버 등에 상관없이 전송할 수 있는 파라미터의 길이에 제한이 없다.

POST 방식을 이용한 파라미터 전송 시 요청 데이터

01	POST /method/viewParameter.jsp HTTP/1.1	← 요청라인
02	Host: localhost:8080	
03	User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; ko; rv:1.9.0.3) ...	
04	Accept: text/html, application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8	
05	...	
06	Referer: http://localhost:8080/method/testForm.jsp	
07	Cookie:JSESSIONID=D1DEA89425A9F051B9A306511B15B328	
08	Content-Type:application/x-www-form-urlencoded	
09	Content-length:22	
10		
11	name=java&address=seoul&pet=cat	← 데이터영역

- POST 방식

전달하고자 하는 정보를 첨부파일 형태로 포함해서 전달하는 방식이다. 이 때 장점으로 URL에 보내는 정보에 대한 노출이 없기 때문에 보안성이 좋으며 보내는 정보의 크기에 제한이 없다.

5.8 요청 처리를 위한 코드 작성

서블릿으로 전달된 파라미터 정보를 얻기 위해서는 HttpServletRequest 객체의 getParameter() 메소드를 이용한다. getParameter() 메소드는 서블릿으로 전달된 파라미터의 이름을 인자로 갖는다.

```
String helloName = request.getParameter();
```

그리고 **HttpServletRequest** 클래스는 클라이언트 요청을 처리하는 기능을 제공하는 클래스로, 주요 메소드는 다음과 같다.

메소드	기 능
getParameter(name)	문자열 name과 같은 이름을 가진 파라미터의 값을 가져온다
getParameterValues(name)	문자열 name과 같은 이름을 가진 파라미터의 값을 배열 형태로 가져옴. checkbox, multiple list 등에 주로 사용된다.
getMethod()	현재 요청이 GET, POST 인지 가져 온다.
getRemoteAddr()	클라이언트의 IP 주소를 가져온다.
setCharacterEncoding	현재 JSP로 전달되는 내용을 지정하는 캐릭터셋을 변환한다. HTML FORM 에서 한글 입력시 정상적으로 처리하려면 반드시 필요하다.

HttpServletResponse 클래스는 서버에서 클라이언트에 응답하기 위한 기능을 제공하는 클래스로, 주요 메소드는 다음과 같다.

메소드	기 능
setContent-Type(type)	문자열 형태의 type에 지정된 MIME Type으로 ContentType을 설정한다.
setHeader(name, value)	문자열 name의 이름으로 문자열 value 값을 헤더로 설정한다.
sendError(status, msg)	오류 코드를 설정하고 메시지를 보낸다.
sendRedirect(url)	클라이언트 요청을 다른 페이지로 보낸다.

HttpServletRequest와 HttpServletResponse 클래스는 JSP에서 각각 request와 response라는 내장 객체로 사용되기 때문에 앞으로 서블릿을 사용하지 않더라도 JSP만으로 프로그래밍을 한다고 해도 꼭 알아 두어야 한다.

5.9 서블릿 응답 처리

클라이언트에서 서블릿으로 요청을 하면 서블릿은 처리한 결과를 html 형식으로 응답 처리한다. 실제로 MVC 패턴을 적용한 웹 어플리케이션 개발에서는 jsp에서 응답 처리를 담당한다. 서블릿에서 응답 처리와 관련된 API는 'HttpServletResponse' 이다.

• **response.setContentType("text/html;charset=UTF-8")**

클라이언트인 웹 브라우저에게 처리할 데이터의 MIME 타입을 알려주는 메서드이다. 기본은 일반 텍스트를 의미하는 'text/plain'이고 실습에서는 html 형식의 전송이므로 'text/html'로 지정한다.

한글 처리를 위해서 'charset=UTF-8'을 추가 지정한다.

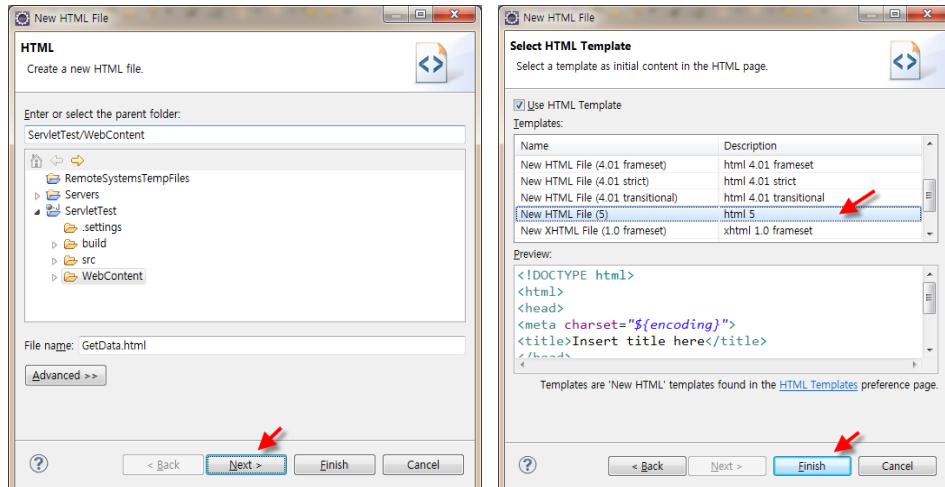
• response.getWriter()

서블릿 및 JSP를 이용한 응답 처리는 기본적으로 자바 I/O 기술을 이용한다.

자바 출력을 위한 OutputStream 또는 Writer 클래스를 사용한다. 서블릿에서는 getWriter() 메서드를 이용한 PrintWriter와 getOutputStream() 메서드를 이용한 ServletOutputStream 클래스를 사용한다.

문자 데이터를 처리하기 위해서는 PrintWriter를 이용하고 바이너리(binary) 데이터를 위해서는 ServletOutputStream 클래스를 사용한다.

HTML5 파일 생성



GetData.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Get Data</title>
</head>
<body>
<form method="get" action="getData">
    이 름 : <input type="text" size="20" maxlength="15" name="name"><br>
    주 소 : <input type="text" size="20" maxlength="15" name="address"><br>
    <input type="submit" value="전송">&nbsp;&nbsp;<input type="reset" value="취소">
</form>
</body>
</html>
```

GetData.java

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

```

@WebServlet("/getData")
/* @WebServlet( urlPatterns={"/getData"}, initParams={@WebInitParam(name="",value="")..} */
public class GetData extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException{
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String helloName = new String(request.getParameter("name"));
        String helloAddress = new String(request.getParameter("address"));
        out.println("<html>");
        out.println("<head> <title>GET DATA</title> </head>");
        out.println("<body>");
        out.println("<center><h2>GET 방식 테스트입니다.</h2></center>");
        if(!(helloName.equals("")) && helloAddress.equals(""))){
            out.println("<li> 입력한 이름 : "+helloName);
            out.println("<li> 입력한 주소 : "+helloAddress);
        }
        out.println("</body>");
        out.println("</html>");
    }
}

```



HangulConversion.java 파일

```

public class HangulConversion {
    public static String toEng (String ko) {
        if (ko == null) {
            return null;
        }
        try {
            return new String(ko.getBytes("UTF-8"),"8859_1");
        } catch(Exception e) {
            return ko;
        }
    }
}

```

```

    }

    public static String toKor (String en) {
        if (en == null) {
            return null;
        }
        try {
            return new String (en.getBytes("8859_1"), "UTF-8");
        } catch(Exception e) {
            return en;
        }
    }
}

```

- lunch.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> select & post </title></head>
<body>
    <h3>오늘점심은 무엇을 먹을까? (2개이상 선택)</h3>
    <form method="post" action="todayMenu">
        <select name="lunch" multiple size="4">
            <option>떡볶기</option>
            <option>버섯덮밥</option>
            <option>칼국수</option>
            <option>회덮밥</option>
            <option>치즈김밥</option>
            <option>피자</option>
        </select>
        <input type="submit" value="전송">
    </form>
</body>
</html>

```

TodayMenu.java

```

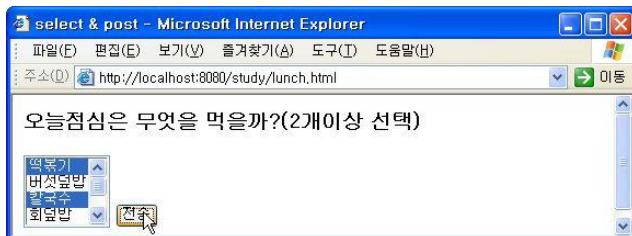
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
@WebServlet("/todayMenu")

```

```

public class TodayMenu extends HttpServlet{
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException{
        res.setContentType("text/html;charset=UTF-8");
        PrintWriter out = res.getWriter();
        out.println("<html> <title>SELECT & POST </title> </head>");
        out.println("<body>");
        out.println("<center> <h3>오늘점심은</h3> </center> ");
        String values[] = req.getParameterValues("lunch");
        for(int i=0;i<values.length;i++){
            out.print("<br>");
            out.print(HangulConversion.toKor(values[i]));
        }
        out.println("이나 먹어야 겠다");
        out.println("</body> </html>");
    }
}

```



calc.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> 계산기 </title></head>
<body>
<center>
<h3>계산기</h3>
<hr>
<!-- 서블릿 버전 계산기 실행을 위해 계산 버튼 클릭시 서블릿 호출 -->
<form name="form1" action="calcServlet" method="post">
<input type="text" name="num1" width=200 size="5">
<select name="operator">
    <option selected>+</option>

```

```

<option>-</option>
<option>*</option>
<option>/</option>
</select>
<input type="text" name="num2" width="200" size="5">
<input type="submit" value="계산" name="b1"> <input type="reset" value="다시 입력" name="b2">
</form>
</center>
</body>
</html>

```

CalcServlet.java

```

// 패키지
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
@WebServlet("/calcServlet")
// 클래스 선언 HttpServlet 을 상속 받는다.
public class CalcServlet extends HttpServlet {
    // GET 요청을 처리하기 위한 메서드
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        // doPost() 로 포워딩 하고 있다.
        doPost(request, response);
    }
    // POST 요청을 처리하기 위한 메서드
    // doGet() 에서도 호출하고 있기 때문에 모든 요청은 doPost() 에서 처리되는 구조이다.
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        // 변수선언
        int num1,num2;
        int result;
        String op;
        // 클라이언트 응답시 전달될 컨텐츠에 대한 mime type 과 캐릭터셋 지정
        response.setContentType("text/html; charset=UTF-8");
        // 클라이언트 응답을 위한 출력 스트림 확보
        PrintWriter out = response.getWriter();
        // HTML form 을 통해 전달된 num1, num2 패러미터 값을 변수에 할당.
        // 이때 getParameter() 메서드는 문자열을 리턴하므로 숫자형 데이터의 경우
        // Integer.parseInt() 를 통해 int 로 변환함.
    }
}

```

```

        num1 = Integer.parseInt(request.getParameter("num1"));
        num2 = Integer.parseInt(request.getParameter("num2"));
        op = request.getParameter("operator");
        // calc() 메서드 호출로 결과 받아옴.
        result = calc(num1,num2,op);
        // 출력 스트림을 통해 화면구성
        out.println("<html>");
        out.println("<head> <title>계산기 </title> </head> ");
        out.println("<body> <center>");
        out.println("<h2>계산결과</h2>");
        out.println("<hr>");
        out.println(num1+" "+op+" "+num2+" = "+result);
        out.println("</body></html>");
    }

    // 실제 계산 기능을 수행하는 메서드
    public int calc(int num1, int num2, String op) {
        int result = 0;
        if(op.equals("+")) {
            result = num1 + num2;
        }
        else if(op.equals("-")) {
            result = num1 - num2;
        }
        else if(op.equals("*")) {
            result = num1 * num2;
        }
        else if(op.equals("/")) {
            result = num1 / num2;
        }
        return result;
    }
}

```

calc.html에서 `<form name="form1" action="calcServlet2" method="post">` 수정한다.

CalcServlet2.java

```

// 패키지 import
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

```

```

@WebServlet("/calcServlet2")
// 클래스 선언 HttpServlet 을 상속 받는다.
public class CalcServlet2 extends HttpServlet {
    // GET 요청을 처리하기 위한 메서드
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        // doPost() 로 포워딩 하고 있다.
        doPost(request, response);
    }
    // POST 요청을 처리하기 위한 메서드
    // doGet() 에서도 호출하고 있기때문에 모든 요청은 doPost() 에서 처리되는 구조이다.
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        // 변수선언
        int num1,num2;
        int result;
        String op;
        // 클라이언트 응답시 전달될 컨텐츠에 대한 mime type 과 캐릭터셋 지정
        response.setContentType("text/html; charset=UTF-8");
        // 클라이언트 응답을 위한 출력 스트림 확보
        PrintWriter out = response.getWriter();
        // HTML form 을 통해 전달된 num1, num2 패러미터 값을 변수에 할당.
        // 이때 getParameter() 메서드는 문자열을 리턴하므로 숫자형 데이터의 경우
        // Integer.parseInt() 를 통해 int 로 변환함.
        num1 = Integer.parseInt(request.getParameter("num1"));
        num2 = Integer.parseInt(request.getParameter("num2"));
        op = request.getParameter("operator");
        // calc 클래스의 인스턴스를 생성한 후 getResult() 메서드를 통해 결과를 받아온다.
        Calc calc = new Calc(num1,num2,op);
        result = calc.getResult();
        // 출력 스트림을 통해 화면구성
        out.println("<html>");
        out.println("<head> <title>계산기 </title> </head>");
        out.println("<body><center>");
        out.println("<h2>계산결과</h2>");
        out.println("<hr>");
        out.println(num1+ " "+op+" "+num2+" = "+result);
        out.println("</body> </html>");
    }
}

```

Calc.java

```
public class Calc{  
    // 계산 결과 보관을 위한 변수로 초기값을 0으로 설정 함.  
    int result =0;  
    //생성자 : 클래스 인스턴스 생성시 자동으로 호출되는 메서드로 변수값들을 인자로 받아옴.  
    public Calc(int num1, int num2, String op) {  
        // 연산자에 따른 계산 수행  
        if(op.equals("+")) {  
            result = num1 + num2;  
        }  
        else if(op.equals("-")) {  
            result = num1 - num2;  
        }  
        else if(op.equals("*")) {  
            result = num1 * num2;  
        }  
        else if(op.equals("/")) {  
            result = num1 / num2;  
        }  
    }  
    // 결과 변수를 리턴하는 메서드  
    public int getResult() {  
        return result;  
    }  
}
```

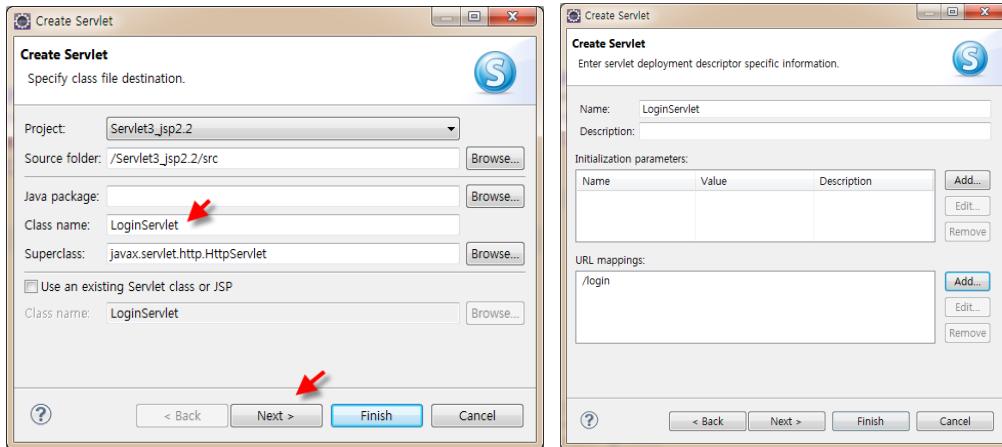
서블릿을 생성할 때 맵핑 설정(get 방식 전송)

login.html

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>로그인</title>  
</head>  
<body>  
<h1>로그인</h1>  
<form action="login" method="get">  
아이디 : <input type="text" name="id"/><br>  
비밀번호 : <input type="password" name="passwd"><br>  
<input type="submit" value="로그인"/>
```

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

서블릿 생성



LoginServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public LoginServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        String id = request.getParameter("id");
        String passwd = request.getParameter("passwd");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("아이디=" + id + "<br>");
        out.println("비밀번호=" + passwd + "<br>");
    }
}
```

```

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // TODO Auto-generated method stub
}
}

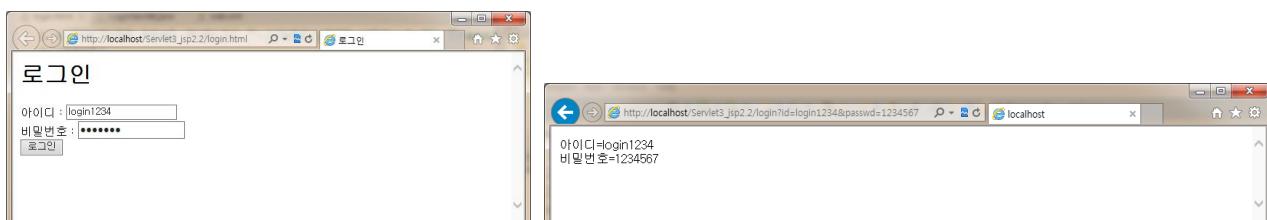
```

web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"           xmlns:web="http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>Servlet3_jsp2.2</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <!-- <servlet>
        <description></description>
        <display-name>LoginServlet</display-name>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>LoginServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping> -->
</web-app>

```



서블릿을 생성할 때 맵핑 설정(post 방식 전송)

memReg.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원가입</title>
</head>
<body>
<h1>회원가입</h1>
<form action="memberJoin" method="post">
    회원명 : <input type="text" name="name"><br>
    주소 : <input type="text" name="addr"><br>
    전화번호 : <input type="text" name="tel"><br>
    취미 : <input type="text" name="hobby"><br>
    <input type="submit" value="회원가입"/>
</form>
</body>
</html>
```

MemJoinServlet.java(매핑명 : /memberJoin)

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/memberJoin")
public class MemJoinServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public MemJoinServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}
```

```

        request.setCharacterEncoding("UTF-8"); //한글 처리
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");
        String addr = request.getParameter("addr");
        String tel = request.getParameter("tel");
        String hobby = request.getParameter("hobby");
        out.println("이름 = " + name + "<br>");
        out.println("이름 = " + addr + "<br>");
        out.println("이름 = " + tel + "<br>");
        out.println("이름 = " + hobby + "<br>");
    }
}

```

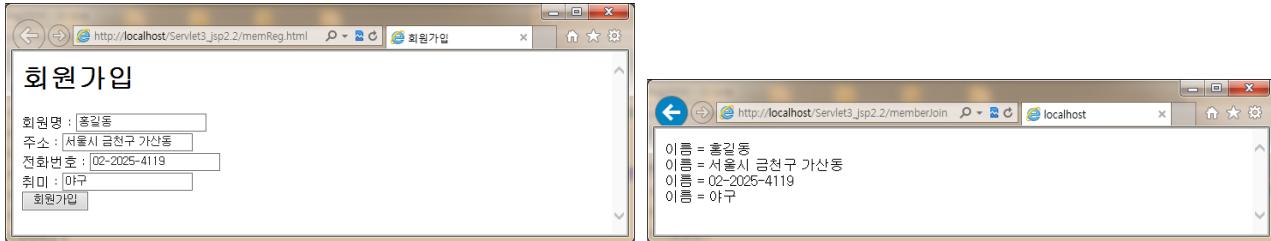
web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>Servlet3_jsp2.2</display-name>
    <!-- <servlet>
        <description></description>
        <display-name>LoginServlet</display-name>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>LoginServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping> -->
    <!-- <servlet>
        <description></description>
        <display-name>MemJoinServlet</display-name>
        <servlet-name>MemJoinServlet</servlet-name>
        <servlet-class>MemJoinServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MemJoinServlet</servlet-name>
        <url-pattern>/memberJoin</url-pattern>
    </servlet-mapping>

```

```
</servlet-mapping> -->
</web-app>
```



서블릿으로 post 방식 전송

FormServlet.java(매핑명 : /form)

```
package servletApp01;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/form")
public class FormServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String TITLE = "주문 양식";
    public FormServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        PrintWriter writer = response.getWriter();
        writer.println("<html>");
        writer.println("<head>");
        writer.println("<title>" + TITLE + "</title></head>");
        writer.println("<body><h1>" + TITLE + "</h1>");
        writer.println("<form method='post'>");
        writer.println("<table>");
        writer.println("<tr>");
```

```

writer.println("<td>이름:</td>");
writer.println("<td><input name='name' /></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>주소:</td>");
writer.println("<td><textarea name='address' " + "cols='40' rows='5'></textarea></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>국가:</td>");
writer.println("<td><select name='country'>");
writer.println("<option>한국</option>");
writer.println("<option>미국</option>");
writer.println("</select></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>배송 방법:</td>");
writer.println("<td><input type='radio' " + "name='deliveryMethod'" + " value='택' /> 택");
writer.println("<input type='radio' " + "name='deliveryMethod'" + "value='택배' /> 택배 </td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>배송 유의사항:</td>");
writer.println("<td><textarea name='instruction' " + "cols='40' rows='5'></textarea></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>&nbsp;</td>");
writer.println("<td><textarea name='instruction' " + "cols='40' rows='5'></textarea></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>최신 제품 카탈로그를 함께 보냅니다</td>");
writer.println("<td><input type='checkbox' " + "name='catalogRequest' /></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>&nbsp;</td>");
writer.println("<td><input type='reset' />" + "<input type='submit' /></td>");
writer.println("</tr>");
writer.println("</table>");
writer.println("</form>");
writer.println("</body>");
writer.println("</html>");
}

```

```

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // TODO Auto-generated method stub
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html");
    response.setCharacterEncoding("utf-8");
    PrintWriter writer = response.getWriter();
    writer.println("<html>");
    writer.println("<head>");
    writer.println("<title>" + TITLE + "</title></head>");
    writer.println("</head>");
    writer.println("<body><h1>" + TITLE + "</h1>\"");
    writer.println("<table>");
    writer.println("<tr>");
    writer.println("<td>이름:</td>");
    writer.println("<td>" + request.getParameter("name") + "</td>");
    writer.println("</tr>");
    writer.println("<tr>");
    writer.println("<td>주소:</td>");
    writer.println("<td>" + request.getParameter("address") + "</td>");
    writer.println("</tr>");
    writer.println("<tr>");
    writer.println("<td>국가:</td>");
    writer.println("<td>" + request.getParameter("country") + "</td>");
    writer.println("</tr>");
    writer.println("<tr>");
    writer.println("<td>배송 유의사항:</td>");
    writer.println("<td>");

    String[] instructions = request.getParameterValues("instruction");
    if (instructions != null) {
        for (String instruction : instructions) {
            writer.println(instruction + "<br/>");
        }
    }

    writer.println("</td>");
    writer.println("</tr>");
    writer.println("<tr>");
    writer.println("<td>배송 방법:</td>");
    writer.println("<td>" + request.getParameter("deliveryMethod") + "</td>");
    writer.println("</tr>");
}

```

```

writer.println("<tr>");
writer.println("<td>제품 카달로그 요청 여부:</td>");
writer.println("<td>");
if (request.getParameter("catalogRequest") == null) {
    writer.println("아니요");
} else {
    writer.println("예");
}
writer.println("</td>");
writer.println("</tr>");
writer.println("</table>");
writer.println("<div style='border:1px solid #ddd;' + \"margin-top:40px;font-size:90%'\>\"");
writer.println("Debug Info<br/>");
Enumeration<String> parameterNames = request.getParameterNames();
while (parameterNames.hasMoreElements()) {
    String paramName = parameterNames.nextElement();
    writer.println(paramName + ": ");
    String[] paramValues = request.getParameterValues(paramName);
    for (String paramValue : paramValues) {
        writer.println(paramValue + "<br/>");
    }
}
writer.println("</div>");
writer.println("</body>");
writer.println("</html>");
}
}

```

The left screenshot shows a form titled "주문 양식" (Order Form). It contains fields for Name (홍길동), Address (서울시 금천구 가산동), Country (한국), Delivery Method (택배), Delivery Note (경비실에 맡겨주세요), and a checkbox for "최신 제품 카달로그를 함께 보냅니다". Below the form are two buttons: "원래대로" and "취리 전송".

The right screenshot shows the same page after submission. It displays the "Debug Info" section which includes the following information:

- address: 서울시 금천구 가산동
- deliveryMethod: 택배
- catalogRequest: on
- name: 홍길동
- instruction: 경비실에 맡겨주세요
- keepAlive: 있으니 조심해서 다뤄주세요
- country: 한국

5.10 애노테이션을 이용한 서블릿의 선처리 및 후처리 작업

Servlet 3.0에서는 서블릿의 인스턴스가 init로 초기화되기 전에 필료한 작업을 할 수 있는 선처리 작업과 destroy 메소드로 제거된 후에 필요한 작업을 할 수 있는 후처리 작업이 가능하다.

1) @PostConstruct를 이용한 선처리 작업

서블릿의 LifeCycle 메서드인 init 메서드가 호출되기 전에 수행되는 선처리 작업 메서드에 지정 가능하다. 반드시 리턴 타입은 void로 지정하고, 예외 클래스를 throws 할 수 없다.

```
@PostConstruct  
public void postConstruct() {  
    ...  
}
```

2) @PreDestroy를 이용한 후처리 작업

서블릿의 LifeCycle 메서드인 destroy 메서드가 호출된 후에 수행되는 후처리 작업 메서드에 지정 가능하다. 반드시 리턴 타입은 void로 지정하고, 예외 클래스를 throws 할 수 없다

```
@PreDestroy  
public void cleanup() {  
    ...  
}
```

PostPreServlet.java

```
package com.test;  
import java.io.IOException;  
import javax.annotation.PostConstruct;  
import javax.annotation.PreDestroy;  
import javax.servlet.ServletConfig;  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
@WebServlet("/postPre")  
public class PostPreServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
    public PostPreServlet() {  
        super();  
        // TODO Auto-generated constructor stub  
    }  
    @Override  
    public void init(ServletConfig config) throws ServletException {  
        // TODO Auto-generated method stub  
    }
```

```

super.init();
System.out.println("init");
}

@Override
public void destroy() {
    // TODO Auto-generated method stub
    System.out.println("destroy");
}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // TODO Auto-generated method stub
    System.out.println("doGet");
}

@PostConstruct
public void initMethod(){
    System.out.println("initMethod");
}

@PreDestroy
public void clean(){
    System.out.println("clean");
}
}

```



6. JSP 소스의 서블릿 변환

- 이클립스에서 서블릿의 위치

톰캣 JSP 컨테이너는 JSP 소스인 *.jsp를 자바 프로그램인 서블릿 소스 *.jsp.java로 자동 생성하여 서블릿 클래스를 실행한다. 다음 예제 소스를 작성하여 실행하고 생성된 소블릿 소스를 찾아 보자.

increment.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 increment.jsp</title>
</head>
<body>

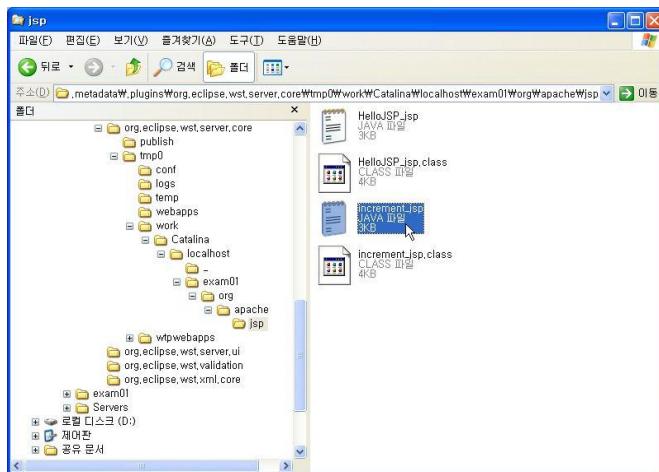
```

```

<% int i = 0; %>
i = <%= ++i %>
</body>
</html>

```

JSP 소스인 increment.jsp에서 생성되는 서블릿 소스 파일은 JSP 컨테이너마다 다를 수 있는데 톰캣 서버의 경우, 서블릿 파일은 increment_jsp.java이다. 현재 이클리스의 설정에서 JSP 파일은 실행하면 가상의 톰캣 서버를 생성하여 실행하므로 서블릿 파일의 위치는 다음과 같이 작업 공간 [C:\jsp2_exam\metadata\plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\exam01\org\apache\jsp] 폴더 하부에 위치한다. 여기서 [tmp0]는 생성하는 서버의 수에 따라 [tmp+번호]와 같으며, [exam01]는 프로젝트 이름이다.



- 서블릿 소스 확인

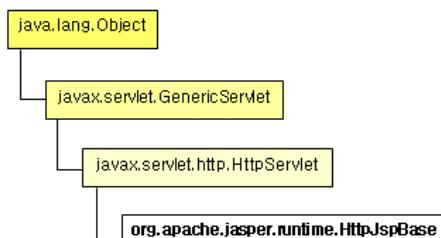
JSP 파일에서 생성된 서블릿 소스 increment_jsp.java를 이클립스에서 직접 편집기로 열어보면 다음과 같은 클래스 increment_jsp를 구현한 자바 프로그램이다. 처음에는 서블릿 increment_jsp.java 소스가 다소 복잡하고 어렵다고 느낄 수 있으나 이클립스의 오른쪽 아웃라인 뷰를 살펴보면 그 구조가 의외로 간단하다. 아웃라인 뷰에서 보듯이 클래스 increment_jsp는 _jspxFactory 등 4개의 소속 변수와 _jspxInit(), _jspDestroy(), _jspService() 등 4개의 메소드로 구성된다는 것을 알 수 있다.

```

package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class increment_jsp
    extends org.apache.jasper.runtime.HttpJspBase
        implements org.apache.jasper.runtime.JspSourceDependent {
    private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();
    private static java.util.List _jspx_dependants;
    private javax.el.ExpressionFactory _el_expressionfactory;
    private org.apache.AnnotationProcessor _jsp_annotationprocessor;
    public Object getDependants() {
        return _jspx_dependants;
    }
}

```



```

}

public void _jsplInit() {
    _el_expressionfactory =
        _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
    _jsp_annotationprocessor = (org.apache.AnnotationProcessor)
        getServletConfig().getServletContext().getAttribute(org.apache.AnnotationProcessor.class.getName());
};

}

public void _jspDestroy() {
}

public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    try {
        response.setContentType("text/html; charset=UTF-8");
        pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;
        out.write("WrWn");
        out.write("<html>WrWn");
        out.write("<head>WrWn");
        out.write("<meta http-equiv="W"Content-TypeW" content="W"text/html;
            charset=UTF-8W">WrWn");
        out.write("<title>JSP 예제 increment.jsp</title>WrWn");
        out.write("</head>WrWn");
        out.write("<body>WrWn");
        out.write("Wt");
    }
}

```

pageContext, session, application, config, out 참조 변수이다. 이들 참조 변수들은 JSP 파일이 로딩될 때 사용자가 작업한 코드 이전에 초기화되므로 JSP 코드 내에서는 별도의 선언 없이 사용할 수 있게 된다. 이러한 이유로 이를 참조 변수들은 보통 JSP 내장 객체로 불리며, 속성 관리 등 다양한 부가 기능을 제공한다.

```

int i = 0;
out.write("WrWn");
out.write("Wti = ");
out.print( ++i );
out.write("WrWn");
out.write("</body>WrWn");
out.write("</html>");
} catch (Throwable t) {
if (!(t instanceof SkipPageException)){
out = _jspx_out;
if (out != null && out.getBufferSize() != 0)
try { out.clearBuffer(); } catch (java.io.IOException e) {}
if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
}
} finally {
_jspxFactory.releasePageContext(_jspx_page_context);
}
}
}
}

```

위 서블릿 소스를 살펴보면 JSP 파일 increment.jsp에서 코딩된 HTML의 <body> 태그 부분 코딩이 메소드 _jspService() 내부에 구현된 것을 확인할 수 있다. 즉 JSP 스크립트릿 코딩인 int i=0; 부분은 모두 메소드 _jspService() 내부에 그대로 구현되었고, 표현식 부분인 ++i는 out.print(++i);로 구현된 것을 확인할 수 있다.

- 서블릿 변환 후 컴파일 오류

JSP 소스를 서블릿으로 변환한 후 컴파일 시 발생하는 오류는 문법 오류이다. 이러한 컴파일 오류는 일반 자바 프로그램의 컴파일 오류와 같은 문제에서 발생하는 오류로 이클립스 편집기에서도 소스 코딩 순간에 오류 표시를 즉시 해 준다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 error.jsp</title>
</head>
<body>
<% String []str = {"감사합니다.", "Thank you."}; %>
한국어로 [<%= str[0] %>]는 <br>
영어로 [<%= str[2] %>]이다.
</body>
</html>

```



7. JSP 문법의 기본

7.1 태그의 이용

JSP는 태그를 이용하여 고유한 문법을 기술하는 서버 프로그래밍 방식이다. JSP의 태그 방식은 스크립트 태그(Script Tag), 액션 태그(Action Tag), 커스텀 태그(Custom Tag)로 나뉜다. 다음은 스크립트 태그 방식의 5가지 종류로 모두 <% ... %>를 사용한다. 태그의 시작인 <와 %사이에 빈 공간 문자가 없어야 하며, 마찬가지로 종료 태그인 %> 사이에도 빈 문자를 허용하지 않는다.

종류	형식	사용 용도
지시어(directives)	<%@ %>	JSP 페이지의 속성을 지정
선언(declaration)	<%! %>	소속변수 선언과 메소드 정의
표현식(expression)	<%= %>	변수, 계산식, 함수 호출 결과를 문자열 형태로 출력
스크립트릿(scriptlet)	<% %>	자바 코드를 기술
주석(comments)	<%-- --%>	JSP 페이지의 설명 추가

액션 태그는 XML 스타일의 태그로 기술한 동작 수행 기능을 수행하는 방식이며, 커스텀 태그는 새로운

태그를 정의하여 이용하는 방법이다.

종류	태그형식	사용 용도
액션 태그 (Action Tag)	<jsp:include page="test.jsp" />	현재 JSP 페이지에서 다른 페이지를 포함.
	<jsp:forward page="test.jsp" />	현재 JSP 페이지의 제어를 다른 페이지에 전달
	<jsp:plugin type="applet" code="test" />	자바 애플릿을 플러그인
	<jsp:useBean id="login" class="LoginBean" />	자바빈을 사용
	<jsp:setProperty name="login" property="pass" />	자바빈의 속성을 지정하는 메소드를 호출
	<jsp:getProperty name="login" property="pass" />	자바빈의 속성을 반환하는 메소드를 호출
커스텀 태그 (Custom Tag)	<tag:printData dbname="mydb" table="memb" />	사용자가 직접 정의한 태그를 이용

7.2 JSP 스크립트 요소

자바 프로그래밍 코드가 삽입되는 선언, 표현식, 스크립트릿 그리고 문법과 관계 없이 설명을 기술하는 주석을 JSP 스크립트 요소(JSP Script Elements)라 한다.

- 스트립트릿

JSP에서 자바 코드를 삽입하려면 <% ... %>의 **스트립트릿(scriptlet)**을 이용한다.

간단한 문장은 주로 한 줄에 태그와 코드를 함께 표현한다. 변수나 객체 또는 문자열의 출력을 자바 코드로 표현하려면 객체 out의 print() 또는 println() 메소드를 이용한다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 scriptlet.jsp</title>
</head>
<body>
    <% int i = 24; %>
    <%
        out.println("하루는 " + i + " 시간이며, <br>");
        out.println("하루는 " + i*60 + " 분 입니다.");
    %>
</body>
</html>
```

println()을 이용하더라도 출력에
 태그를 사용하지 않으면 웹 브라우저 화면에서 두 줄로 출력되지 않는다. 즉 출력에 이용되는 out.println()은 html 소스에서 출력되는 내용을 다음 줄로 이동시키지만 브라우저의 결과에는 다음줄로 이동이 반영되지 않는다. 차이점은 소스 보기 하였을 경우 print()로 작성한

경우 출력된 문장이 모두 한줄에 출력된 것을 확인할 수 있다.

- 표현식(변수의 출력)

표현식 태그 <%= ...%> 를 이용하여 변수, 계산식, 함수 호출 결과를 문자열 형태로 출력한다. 표현식에서 expression은 String 유형으로 변환되어 out 객체를 통해 클라이언트에 전달된다.

표현식 태그에서 <%와 = 사이에 공백이 없어야 하며, 출력하는 변수 또는 계산식 expression은 하나의 결과 값이어야 하고, expression 이후에 ;(세미콜론)이 없어야 한다. 이는 JSP 페이지가 서블릿으로 변환될 경우, expression이 변환된 자바 문장인 `out.print(expression);` 의 인자로 이용되기 때문이다. 그러므로 자바의 기본 자료형은 모두 표현식으로 출력 가능하며, 참조 자료형으로는 메소드 `toString()`의 결과를 출력된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 expression.jsp</title>
</head>
<body>
    <% int year = 365; %>
    <% out.println("1년은 약 몇 주일까요? <p>"); %>
    <%= year / 7 %>
    <%= " 주 입니다." %>
</body>
</html>
```

표현식에서 expression은 `out.print(expression);`의 인자로 문법상 문제가 있으면 서블릿 변환 시 오류가 발생하며 expression을 String유형으로 변환할 수 없는 경우, 실행시 ClassCastException 예외가 발생할 수 있다.

7.3 변수의 선언과 메소드 구현

선언 태그는 <%! %>를 이용하여 소속 변수를 선언하거나 메소드를 구현하는 태그이다.

선언태그에서 선언되는 변수는 소속 변수(membered variables)로 선언하는 것이다. 반대로 스크립트릿에서 선언되는 변수는 `_jspService()` 내부에서 선언된 지역 변수이다. 다음 예제는 원의 반지름을 인자로 원의 면적을 구하는 프로그램으로, 메소드 `getArea()`를 구현하며 변수 `radius`는 서블릿 클래스의 소속 변수로 선언된다.

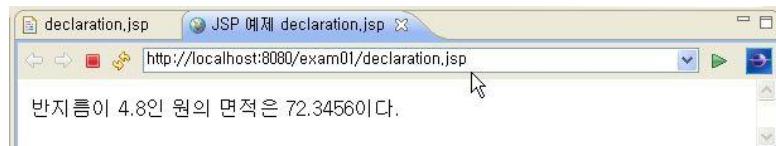
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 declaration.jsp</title>
</head>
<body>
    <%! double radius = 4.8; %>
```

```

<%!
    public double getArea(double r) {
        return r * r * 3.14;
    }
%>
반지름이 <%= radius %>인 원의 면적은 <%= getArea(radius) %>이다.
</body>
</html>

```

[결과]



다음 소스는 스크립트릿 태그로 지역변수를 선언하는 것과 선언 태그로 소속 변수를 선언하는 것의 차이를 알아보는 프로그램이다. 즉 변수 i는 지역변수이고 변수 memi는 소속 변수이다. 그러므로 변수 i는 변수를 선언한 문장 이후, 스크립트릿에서만 참조가 가능하나 변수 memi는 변수 선언 문장 이전에도 스크립트릿에서 참조가 가능하다

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 membervar.jsp</title>
</head>
<body>
    <% int i = 0; %>
    [지역변수] i = <%= ++i %>
    <p>
        [소속변수] memi = <%= ++memi %>
        <%! int memi = 0; %>
</body>
</html>

```

[결과]





지역변수 i를 참조하는 `<%= ++i %>`은 `<% int i = 0; %>` 이전에 위치할 수 없으나, 소속 변수 memi를 참조하는 `<%= ++memei %>`은 변수 선언 `<%! int memi = 0; %>`보다 앞에 위치할 수 있다. 위 예제를 처음 실행한 후, 웹 브라우저에서 계속 refresh를 하면 지역변수 i는 계속 1이 출력되나 소속 변수인 memi는 1씩 계속 증가하는 것을 알 수 있다. 즉 변수 memi는 소속 변수로서 이전 값이 계속 남아있기 때문이다.

7.4 주석

JSP에서 이용되는 주석은 HTML에서 이용하는 주석과 JSP 자체의 주석으로 나뉜다. HTML 주석은 `<!-- ... -->`이며, JSP 주석(JSP 컨테이너에서 웹서버로 전달시 버려지므로 클라이언트에 전송되지 않는다.)은 `<%-- ... --%>`이다. HTML 주석은 HTML 태그를 위한 주석으로 웹 브라우저의 [소스보기]에서 HTML 내용과 함께 그 주석 내용이 보이나, JSP 주석은 JSP 서버 프로그램을 위한 주석으로 실행된 웹 브라우저의 [소스보기]에서 표시되지 않는다. 다음과 HTML 주석 내부에서 JSP의 스크립트릿 태그나 표현식 태그를 출력으로 이용할 수 있다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import = "java.util.Calendar,java.util.GregorianCalendar ,java.text.SimpleDateFormat"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 jspcomments.jsp</title>
</head>
<body>
<h1> HTML 주석과 JSP 주석의 차이 </h1>
<!-- 이것은 HTML 주석으로 웹 브라우저의 [소스 보기]에서 보입니다. -->
<%-- 다음은 JSP 주석문 입니다.--%>
<%! String str; %>
<%
    String today;
    Calendar now = Calendar.getInstance();
    if (now.get(Calendar.HOUR_OF_DAY) >= 12)
        today = "오후";
    else
        today = "오전";
<%
    GregorianCalendar calendar = new GregorianCalendar();
```

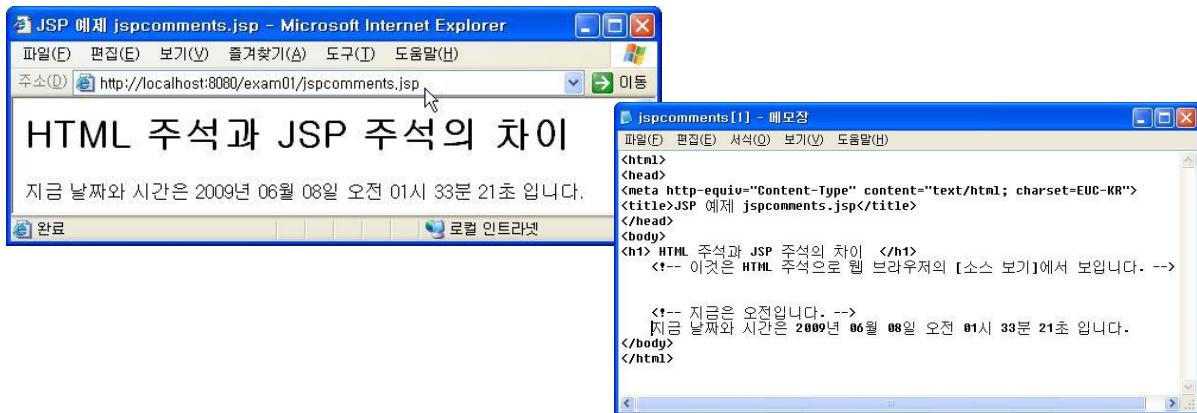
```

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy년 MM월 dd일 aa
hh시 mm분 ss초");
str = dateFormat.format(calendar.getTime());
%>
<!-- 지금은 <%= today %>입니다. -->
지금 날짜와 시간은 <%= str %> 입니다.
</body>
</html>

```

그런데 이 때 주의할 점은 format 메소드에 넘겨 주는 파라미터로 GregorianCalendar 객체를 사용 할 수 없다는 것이다. format 메소드가 받을 수 있는 파라미터로 Date 클래스 타입의 객체인데, 이 클래스는 날짜와 시간 정보를 담는데 사용되는 클래스이긴 하지만 지금은 거의 사용하지 않는다. Calendar 클래스에는 Calendar 객체가 가지고 있는 것과 똑같은 시각 정보를 갖는 Date 객체를 만들어내는 getTime이라는 메소드가 있다.

[결과]



- 자바 주석의 이용

JSP 주석은 *.jsp의 소스에서만 보이는 주석이며 실행 시 생성된 서블릿 프로그램에서는 보이지 않는다. 일반 자바 주석은 JSP 소스의 자바 코딩이 가능한 부분에서 이용될 수 있을 뿐 아니라, 실행시 생성된 서블릿 프로그램에서도 볼 수 있다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"><title>JSP 예제 comments.jsp</title>
</head>
<body>
<!-- 이것은 HTML 주석으로 웹 브라우저의 [소스 보기]에서 보입니다. -->
<%-- 이것은 JSP 주석으로 브라우저의 [소스 보기]에서 안보입니다.--%>
<%!
/*
    절대값을 반환하는 메소드 abs()
*/
public int abs(int a) { //메소드 구현
    //if 문장을 활용
}

```

```

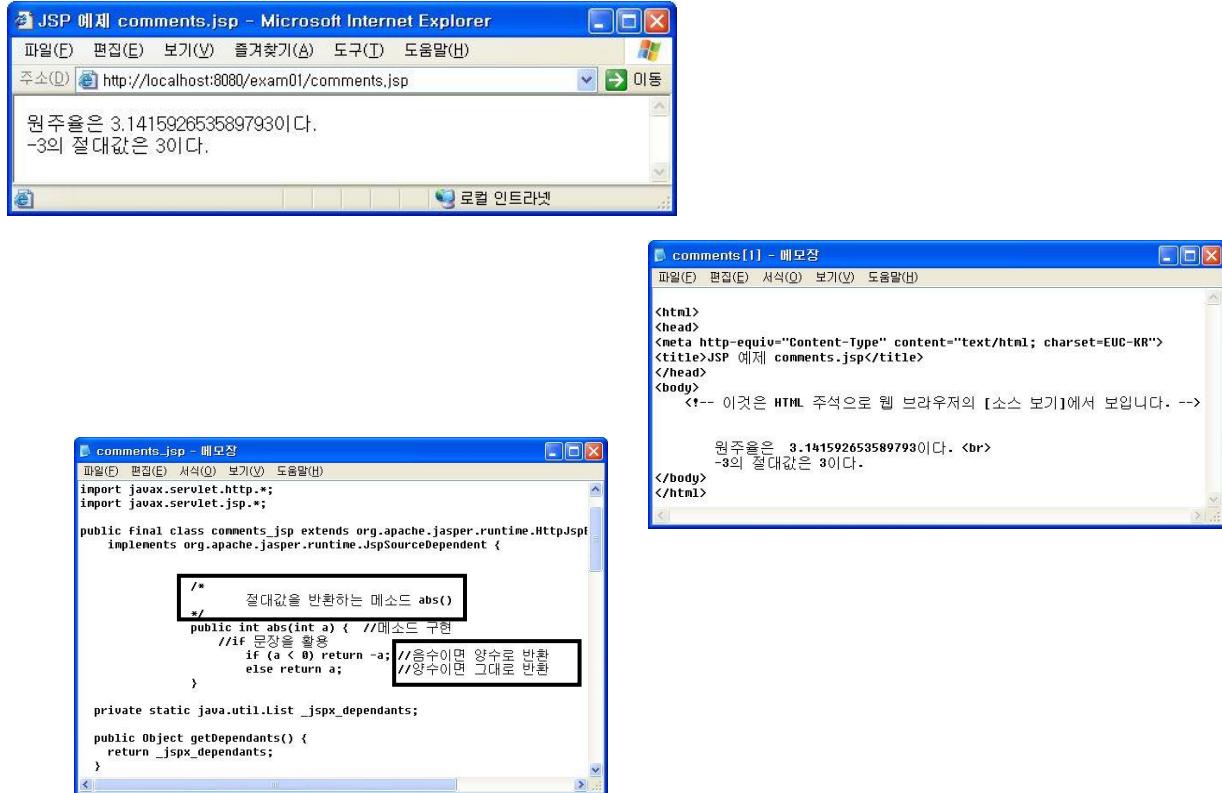
        if (a < 0) return -a; //음수이면 양수로 반환
        else return a;           //양수이면 그대로 반환
    }

%>
원주율은 <%= Math.PI %>이다. <br>
-3의 절대값은 <%= abs(-3) %>이다.

</body>
</html>

```

[결과]



8. 지시자

8.1 지사자의 개요

- 지시자 형식과 종류

지시자(directives)는 일반적인 프로그램 언어와는 달리 태그 형태를 이용하여 JSP 페이지에 대한 속성 또는 특별한 지시 사항을 지정하는 태그이다. 지시자는 다음과 같은 형식을 취하며 지시어 directives와 속성 property 모두 대소문자를 구분하고 속성 값은 반드시 "속성값"과 같이 큰 따옴표 또는 작은 따옴표를 이용하여 둘러싼다.

```
<%@ directives property = "property-value" %>
```

JSP 지시자의 종류는 page, include, taglib 3가지가 있다. taglib지사자는 새로이 정의된 커스텀 태그의 이용을 선언하는 지시자로 나중에 확인해 보도록 하겠다.

종류	형태	용도
----	----	----

page	<code><%@ page property = "property-value" %></code>	JSP 페이지에 대한 속성 지정
include	<code><%@ include file="file-name" %></code>	태그 부분에 지정된 페이지를 정적으로 삽입
taglib	<code><%@ taglib uri="uri_value" prefix="pfx-value" %></code>	새로운 태그를 정의하여 이용

8.2 page 지시자

- 속성종류

page 지시자는 JSP 컨테이너에서 JSP 페이지에 대한 여러 속성과 값을 지정하는 지시자이다. Page 지시자는 language, contnetType, pageEncoding 등의 속성을 지정하고, 한 번에 서로 다른 여러 개의 속성을 지정할 수 있다.

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

다음은 page 지시자의 모든 속성 지정 방법을 표현하고 있는데, 대괄호 [...]는 필요하면 해당하는 속성을 기술할 수 있다는 옵션을 의미한다. 설정값에서 "true | false"는 true와 false 두 개 중에서 하나를 기술할 수 있으며 기술하지 않으면 진한 글씨인 true가 기본값이라는 것을 의미한다. 또한 설정값에서 "text"와 같이 뉴어 쓴 문자열은 그에 해당하는 적당한 값을 넣으라는 것을 의미한다.

```
<%@ page
[language="java"] [extends="package.class"]
[import="{package.class | package.*}, ..."]
[session="true | false"]
[buffer="none | 8kb | sizekb"] [autoFlush="true | false"]
[isThreadSafe="true | false"] [info="text"]
[errorPage="relativeURL"] [isErrorPage="true | false"]
[contentType="{mimeType [ ; charset=characterSet ] | text/html ; charset =ISO-8859-1}"]
[pageEncoding="{characterSet | ISO-8859-1}"]
[isELIgnored="true | false"]
%>
```

- language 속성

language 속성은 JSP 페이지의 표현식, 선언, 스크립트릿에서 사용할 스크립트 언어의 종류를 지정하는 속성이다. 현재 language 속성은 기본값도 java이고, 대부분의 JSP 컨테이너가 java이외에 지정할 다른 언어를 제공하고 있지 않으나 향후 JSP의 확장을 위해 만든 속성이다.

```
<%@ page language="java" %>
```

- contentType 속성

contentType 속성은 JSP 페이지 MIME(Multipurpose Internet Mail Extension)유형(type)을 지정하는 속성으로 지정하지 않으면 다음과 같이 "text/html"이 기본값이다. MIME 유형이란 JSP 페이지 자료를 네트워크에서 주고 받을 때, 서로 주고 받는 문서의 타입을 정의함으로써 이를 보내고 받는 시스템에서 원활하게 자료를 처리하려는 목적에서 나온 속성이다.

```
<%@ page contentType="text/html" %>
```

contentType 속성은 MIME 유형과 함께 MIME에서의 문자셋을 지정할 수 있는데, MIME 속성 다음에 구분자인 ;을 연결하여 "; charset=방식" 형태로 기술한다.

```
<%@ page contentType="text/html; charset=ISO-8859-1" %>
```

charset 값을 기술하지 않으면 위와 같이 [ISO-8859-1]이 기본값이므로 한글을 지원하기 위해서는 다음과 같이 [UTF-8]을 지정한다

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

여기서 [UTF-8]은 소문자인 [UTF-8]도 가능하고 [[Extended Unix Code KOREA]를 의미하고 한글지원을 위한 문자 코드를 말한다.

- pageEncoding

pageEncoding속성은 JSP 페이지의 문자 인코딩 방식을 기술하는 속성으로 지정하지 않으면 기본값이 [ISO-8859-1]이다.

```
<%@ page pageEncoding=" ISO-8859-1" %>
```

한글을 지원하기 위해서는 다음과 같이 pageEncoding속성을 [UTF-8]로 지정한다.

```
<%@ page pageEncoding="UTF-8" %>
```

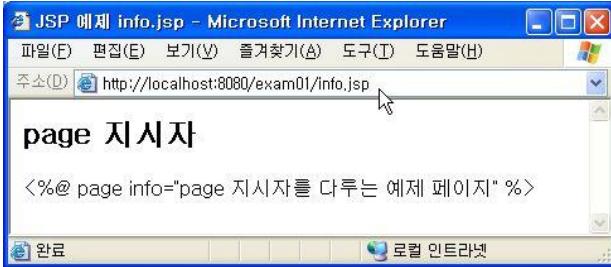
- info 속성

info속성은 JSP 페이지 전체에 대한 설명이나 버전, 작성자, 작성일자와 같은 정보를 문자열로 기술하는 부분으로 길이에는 제한이 없다. info속성을 이용하여 페이지 관리를 손쉽게 할 수 있다.

```
<%@ page info=" JSP 페이지에 대한 설명이나 정보" %>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page info="page 지시자를 다루는 예제 페이지" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 info.jsp</title>
</head>
<body>
<h2>page 지시자 </h2>
<%@ page info="page 지시자를 다루는 예제 페이지" %>
</body>
</html>
```

[결과]



- import

import 속성은 자바의 import문장을 대체하는 속성으로 이용할 클래스를 지정하는 방법이다. 자바의 import 문장과 달리, 필요하면 구분자 ,(콤마)를 이용하여 여러 개의 클래스를 지정할 수 있다. 또한 JSP 페이지에 여러 개의 import 페이지 속성도 기술할 수 있다.

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

일반 자바 프로그램과 같이, JSP 페이지는 패키지 [java.lang.*]을 import 할 필요가 없으며, 서블릿에서 기본적으로 제공되는 패키지인 [javax.servlet.*], [javax.servlet.http.*], [javax.servlet.jsp.*]도 import 할 필요가 없다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.Calendar" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 import.jsp</title>
</head>
<body>
    <h2> page 지시자의 import 속성</h2>
    <%
        Calendar now = Calendar.getInstance();
        StringBuffer sb = new StringBuffer("현재 시각 : ");
        sb.append(now.get(Calendar.YEAR));
        sb.append("년 ");
        sb.append(now.get(Calendar.MONTH)+1);
        sb.append("월 ");
        sb.append(now.get(Calendar.DAY_OF_MONTH));
        sb.append("일");
        out.println(sb);
    %>
</body>
</html>
```

[결과]

서블릿 소스 파일[import_jsp.java]



- isErrorPage 속성

isErrorPage 속성은 JSP 페이지가 오류를 처리하는 페이지인지를 true 또는 false로 지정하는 속성이다. isErrorPage 속성은 지정하지 않으면 기본값이 false이고 필요하면 true로 지정한다.

```
<%@ page isErrorPage="true" %>
```

실질적으로 에러처리(error handling)을 담당하는 JSP페이지에 위와 같이 isErrorPage를 true로 지정하면 내장 객체라 부르는 exception 변수를 사용하여 에러를 처리할 수 있다.

- errorPage 속성

errorPage 속성은 JSP 페이지에서 발생한 오류를 처리하는 JSP 페이지를 기술하는 방법이다.

```
<%@ page errorPage="exception.jsp" %>
```

이러한 errorPage를 지정해서 오류를 처리하는 전담 JSP 페이지를 지정하면, 이 시스템을 사용하는 사용자에게 일관성 있게 오류 처리를 해 줄수 있어 시스템의 신뢰성을 높일 수 있는 장점이 있다.

- isThreadSafe 속성

isThreadSafe 속성은 동시 사용자 접속 처리에 대한 지정 방법으로 true 또는 false로 지정하며, 지정하지 않으면 true가 기본 값이다.

```
<%@ page isThreadSafe="false" %>
```

JSP는 하나의 서블릿에 대하여 여러 사용자가 접속할 경우 쓰레드(thread)처리를 한다. 쓰레드 처리란 동시에 여러 명의 사용자가 접속한 경우라도 자원 참조에 문제가 발생할 수 없도록 동기화를 해주는 것이다. isThreadSafe 속성을 false로 지정하면 쓰레드를 사용하지 않겠다는 것으로 이러한 경우는 매우 드문 일이다.

- isELIgnored 속성

isELIgnored 속성은 표현 언어인 EL(Exception Language)의 사용 여부를 지정하는 방법으로, 지정하지 않으면 false가 기본값으로 표현 언어를 사용한다는 의미이다. 만일 표현 언어를 사용하지 않으려면 값을 true로 지정한다.

```
<%@ page isELIgnored="true" %>
```

- buffer 속성

buffer 속성은 JSP 페이지의 출력 버퍼링 메모리 크기를 지정하는 방법으로, 지정하지 않으면 8KB가 기본값이다. buffer 속성 값은 none 또는 16KB와 같이 다른 크기의 값으로 지정할 수 있다.

```
<%@ page buffer="16kb" %>
```

버퍼링은 일반적으로 입력이나 출력에 이용하는 방식으로 프로세스의 처리 속도보다 입출력의 속도가

느리기 때문에 어느 정도 자료를 모아서 입출력을 처리하는 방법이다. 즉 양동이에 어느 정도의 물이 차야 물을 쏟아내는 양동이와 같이 버퍼링 메모리에 지정한 크기만큼의 자료가 쌓여야 출력을 하는 방식이 버퍼링이다. buffer 속성 값이 none이면 버퍼링을 하지 않겠다는 의미로 출력 자료가 버퍼를 거치지 않고 바로 웹 브라우저에 출력된다.

```
<%@ page buffer="none" %>
```

- autoFlush 속성

autoFlush 속성은 버퍼가 모두 찼을 때 자동으로 출력하는지를 지정하는 방법으로 지정하지 않으면 true가 기본값으로, 버퍼 크기의 자료가 모두 찼을 때 자동으로 웹 브라우저에 출력한다는 의미이다. 만일 autoFlush 속성을 false로 지정하면 버퍼 크기만큼 차기 전, 중간 중간에 수동으로 직접 버퍼를 비워야 출력이 가능하며, 버퍼 크기의 자료가 모두 찼을 경우, 오버플로우(overflow)예외가 발생한다.

```
<%@ page autoFlush="false" %>
```

다음과 같이 autoFlush 값이 false이면 수동으로 버퍼링을 해야 하는데, buffer 값을 none으로 지정하면 버퍼링을 하지 않겠다는 것으로 잘못된 지정 방법이다.

```
<%-- 다음은 잘못된 page 버퍼 지정 방법이다. --%>
<%@ page buffer="none" %>
<%@ page autoFlush="true" %>
```

- session 속성

session 속성은 JSP 페이지에서 세션의 사용 여부를 지정하는 방법으로 지정하지 않으면 true가 기본값으로 세션을 이용할 수 있다.

```
<%@ page session="false" %>
```

세션(session)은 웹 브라우저의 사용자를 구분하는 단위로 사용자 별로 웹 서버에 필요한 정보를 임시로 저장하는 방법이다. 어느 사이트에 접속했을 때 한 번 로그인 한 후 어느 정도 시간이 지나면 세션이 끊어져 더 이상 사용할 수 없다거나, 장바구니에 새로운 상품을 이전 목록에 추가하는 것이 모두 세션을 이용한 구현 방법이다.

8.3 include 지시자

include 지시자는 태크를 기술한 위치에, 지정한 파일을 삽입하는 유일한 속성으로 삽입 파일을 지정하는 file을 갖는다.

```
<%@ include file="file_name" %>
```

다음 예제 include.jsp는 include 지시자를 사용하는 두 파일 header.jsp와 footer.html 파일을 각각 처음과 마지막에 삽입하는 예제와 그 결과이다.

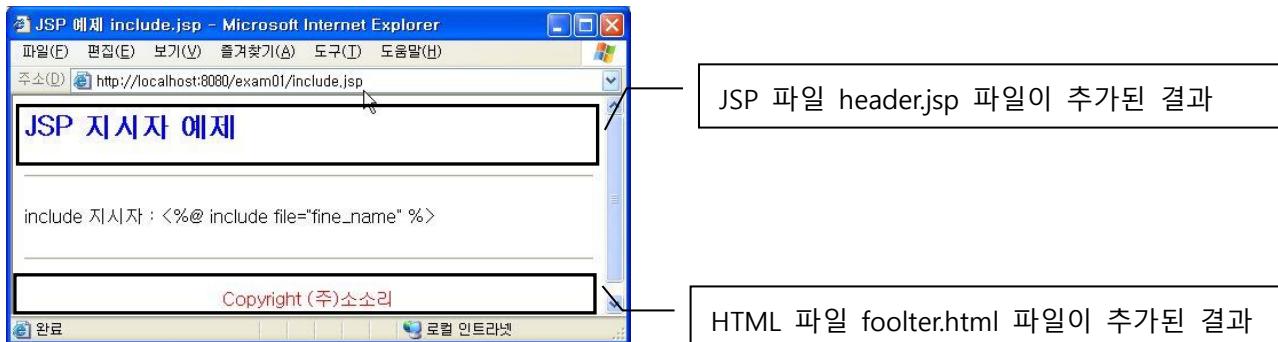
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```

<title>JSP 예제 include.jsp </title>
</head>
<body>
    <%@ include file="header.jsp" %>
    <hr> <p>
        include 지시자 : &lt;%@ include file="fine_name" %&gt; <p>
        <hr> <p>
        <%@ include file="footer.html" %>
</body>
</html>

```

[결과]



header.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제</title>
</head>
<body>
    <h2><font color="blue">JSP 지시자 예제</font></h2>
</body>
</html>

```

footer.html

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<center>
<font color="red">Copyright (주)대한민국</font>
</center>

```

삽입되는 JSP파일 header.jsp는 일반적인 JSP 파일과 동일한 구조이나 현실적으로 <html>,<head> 등의

구조적인 HTML 태그는 원래의 include.jsp파일에 있으므로 위와 같이 생략하는 것이 좋다. 즉 웹 브라우저에서 [소스 보기]로 HTML 소스를 살펴 보면 <html>, <head> 등의 구조적인 HTML 태그가 중복되는 것을 알 수 있으나 다행히 page 지시자는 중복되지 않아 오류는 발생하지 않는다. Include 지시자는 삽입되는 파일의 JSP 지시자에 해당하는 부분을 그대로 삽입하지 않으며 처리 단계를 거쳐 HTML 또는 일반 텍스트만 삽입된다. 삽입되는 파일은 JSP, HTML, 일반 텍스트 파일 등이며, 결과 화면에서 한글이 잘 처리되려면 JSP 파일이 아닌 경우에는 다음 문장을 첫 줄에 추가해야 한다.

9. 내장 객체

9.1 내장 객체 개요

- 내장 객체란?

내장 객체(Implicit Object)란 JSP의 스크립트릿과 표현에서 선언 없이 이용할 수 있는 객체 변수를 말한다. 즉 웹 브라우저의 출력에 이용하던 객체 변수 out은 JSP 서블릿의 `_jspService()` 메소드에서 자동으로 선언되므로 JSP 페이지의 스크립트릿에서 선언 없이 `out.println()`을 사용할 수 있었다. 객체 변수로는 `out`을 비롯하여 `request`와 `response`, `session`, `application`, `config`, `exception`, `page`, `pageContext` 9개가 있다.

내장객체	소속 패키지	클래스이름	사용용도
request	javax.servlet.http	<<interface>> HttpServletRequest	클라이언트의 요청에 의한 품 양식 정보 처리
response	javax.servlet.http	<<interface>> HttpServletResponse	클라이언트의 요청에 대한 응답
session	javax.servlet.http	<<interface>> HttpSession	클라이언트에 대한 세션 정보 처리
application	javax.servlet	<<interface>> ServletContext	웹 애플리케이션 정보 처리
config	javax.servlet	<<interface>> ServletConfig	현재 JSP 페이지에 대한 환경 처리
exception	java.lang	<<interface>> Throwable	예외 처리를 위한 객체
page	java.lang	<<class>> Object	현재 JSP 페이지에 대한 클래스 정보
pageContext	javax.servlet.jsp	<<class>> PageContext	현재 JSP 페이지에 대한 페이지 컨텍스트
out	javax.servlet.jsp	<<class>> JspWriter	출력 스트림

내장 객체의 자료유형을 살펴보면 `javax.servlet`과 `javax.servlet.http` 그리고 `javax.servlet.jsp`, `java.lang`에 속하는 인터페이스와 클래스로 구성된다.

분류	java.lang	javax.servlet	javax.servlet.http	javax.servlet.jsp
JSP 페이지에	page	config		

관련된 객체				
페이지 입출력에 관련된 객체			request response	out
컨텍스트에 관련된 객체		application	session	pageContext
예외에 관련된 객체	exception			

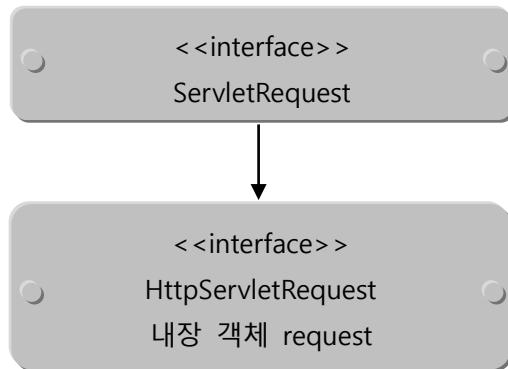
내장 객체는 JSP 서블릿 `_jspService()` 메소드의 첫 부분에 선언되거나 메소드의 매개변수 목록의 변수이다. 이전에 생성했던 JSP 서블릿 소스를 살펴보면 다음과 같이 내부 객체의 변수를 확인할 수 있다. 내장 객체 중에서 `exception`은 페이지 지시자의 속성 `isErrorPage="true"`인 경우에 선언되는 변수이다. 내부 객체는 서블릿 객체의 `_jspService()` 메소드 내부에서 이용할 수 있는 지역변수 또는 매개변수이므로 JSP의 선언에서는 이용할 수 없다. 또한 내장 객체와 같은 이름으로 JSP의 선언에 이용하더라도 지역 변수인 내부 객체와 이름이 충돌하므로 소속 변수로 이용할 수 없다.

```
<%! int application = 0; %>
<%= application /* 정수 0이 아니라 내부객체 application임 */ %>
```

9.2 내장 객체 request

- `request`의 자료 유형과 인자 전달

내장 객체 `request`는 클라이언트가 서버에서 전송하는 관련 정보를 처리하는 객체이다. 즉 HTML 폼에 입력하여 값을 전송하는 경우, 내장 객체 `request`를 사용한다. 내장 객체 `request`는 인터페이스 `HttpServletRequest`로, 상위 인터페이스는 인터페이스 `javax.servlet.ServletRequest`를 갖는다.



`request` 내장 객체가 제공하는 기능은 다음과 같다.

- 클라이언트(웹 브라우저)와 관련된 정보 읽기 기능
- 서버와 관련된 정보 읽기 기능
- 클라이언트가 전송한 요청 파라미터 읽기 기능
- 클라이언트가 전송한 요청 헤더 읽기 기능
- 클라이언트가 전송한 쿠기 읽기 기능
- 속성 처리 기능

그러므로 내장 객체인 `request`는 인터페이스 `javax.servlet.ServletRequest`의 다음과 같은 여러 메소드를

상속 받아 이용할 수 있다.

반환값	메소드	사용용도
void	setCharacterEncoding(String env)	요청 페이지에 env의 인코딩 방법을 적용
String	getParameter(String name)	name의 요청 인자 값을 반환. 없으면 null을 반환. 만일 값이 여러 개이면 첫 번째 값만 반환.
String[]	getParameterValues(String name)	지정한 name의 요청 인자값을 문자열 배열로 반환. 없으면 null을 반환.
Enumeration	getParameterNames()	모든 인자의 이름을 Enumeration으로 반환.
String	getProtocol()	사용중인 프로토콜을 반환.
String	getRemoteAddr()	클라이언트의 IP 주소를 반환
String	getRemoteHost()	클라이언트의 호스트 이름을 반환
String	getServerName()	서버 이름을 반환
int	getServerPort()	요청된 서버의 포트 번호를 반환.

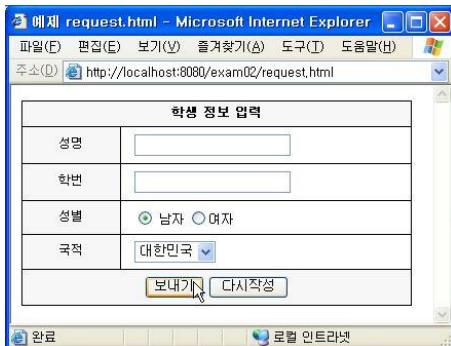
request.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>예제 request.html</title>
<style type="text/css">
  table{font:9pt 굴림}
</style>
</head>
<body>
<form method="post" action="request.jsp">
  <table border="0" cellspacing="1" cellpadding="5" width="380" bgcolor="#000000"
    height="200">
    <tr><th colspan="2" align="center" bgcolor="#F7F7F7">학생 정보 입력</th></tr>
    <tr>
      <td width="200" align="center" bgcolor="#F7F7F7" style="letter-spacing:20">성명</td>
      <td bgcolor="#FFFFFF" width="450">&ampnbsp<input type="text" name="name"></td>
    </tr>
    <tr>
      <td width="200" align="center" bgcolor="#F7F7F7" style="letter-spacing:20">학번</td>
      <td bgcolor="#FFFFFF" width="450">&ampnbsp<input type="text" name="studentNum"></td>
    </tr>
    <tr>
      <td width="200" align="center" bgcolor="#F7F7F7" style="letter-spacing:20">성별</td>
      <td bgcolor="#FFFFFF" width="450">&ampnbsp
```

```

        <input type="radio" name="sex" value="man" checked> 남자
        <input type="radio" name="sex" value="woman"> 여자</td>
    </tr>
    <tr>
        <td width="200" align="center" bgcolor="#F7F7F7" style="letter-spacing:20">국적 </td>
        <td bgcolor="#FFFFFF" width="450">&nbsp;
            <select name="country">
                <option SELECTED value="대한민국">대한민국</option>
                <option value="일본">일본</option>
                <option value="중국">중국</option>
                <option value="터키">터키</option>
                <option value="태국">태국</option>
            </select></td>
        </tr>
        <tr><td colspan="2" align="center" bgcolor="#F7F7F7">
            <input type="submit" value="보내기">
            <input type="reset" value="다시작성"></td></tr>
    </table>
</form>
</body>
</html>

```



예제 request.jsp는 request.html에서 학생 정보를 전송받아 웹 브라우저에 출력하는 JSP 페이지로서 내장 객체 request의 메소드 setCharacterEncoding()과 getParameter()를 이용한다.

특히 전송 방식 post 방식에서 한글 처리를 위하여 메소드 setCharacterEncoding()을 인자 "UTF-8"로 호출한다. 폼 태그 내부의 여러 형태에 입력된 자료는 메소드 getParameter("인자이름")에 의하여 각각 JSP 페이지에 전송된다.

request.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 request.jsp</title>
<style type="text/css">
    fieldset{border-width:1 3; border-color:#000000; border-style:solid;
        width:380 ; height:180 ; padding:10; font-size:9pt ; color:#039AFD}

```

```

table{font:9pt 굴림}
</style>
</head>
<body>
<%
    request.setCharacterEncoding("UTF-8");
    String name = request.getParameter("name");
    String studentNum = request.getParameter("studentNum");
    String sex = request.getParameter("sex");
    String country = request.getParameter("country");
    if (sex.equalsIgnoreCase("man")){
        sex = "남자";
    } else {
        sex = "여자";
    }
%>
<fieldset class="01">
    <legend><b>학생 정보 입력 결과</b></legend>!-- 글상자제목을 표시 해주는 태그이다 --
    <table border="0" cellspacing="1" cellpadding="5" width="360" bgcolor="#000000" height="160">
        <tr>
            <td width="200" align="center" bgcolor="#F7F7F7">0 &nbsp;&nbsp;&nbsp; 틈</td>
            <td bgcolor="#FFFFFF" width="450">&nbsp;&nbsp;<%= name%></td>
        </tr>
        <tr>
            <td width="200" align="center" bgcolor="#F7F7F7">학 &nbsp;&nbsp;&nbsp; 번</td>
            <td bgcolor="#FFFFFF" width="450">&nbsp;&nbsp;<%= studentNum%></td>
        </tr>
        <tr>
            <td width="200" align="center" bgcolor="#F7F7F7">성 &nbsp;&nbsp;&nbsp; 별</td>
            <td bgcolor="#FFFFFF" width="450">&nbsp;&nbsp;<%= sex%></td>
        </tr>
        <tr>
            <td width="200" align="center" bgcolor="#F7F7F7">국 &nbsp;&nbsp;&nbsp; 적</td>
            <td bgcolor="#FFFFFF" width="450">&nbsp;&nbsp;<%= country%></td>
        </tr>
    </table>
</fieldset>
</body>
</html>

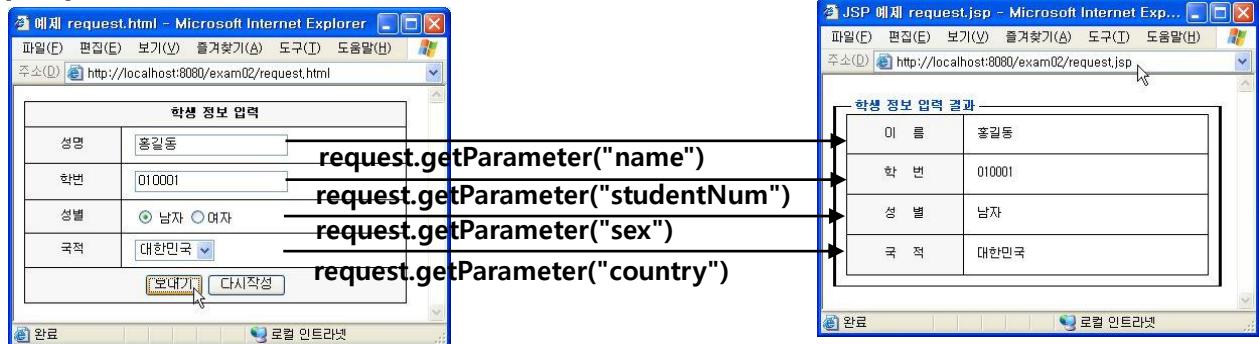
```

메소드 getParameter("namevalue")의 인자는 입력 태그에 지정한 속성 name 값인 [name="namevalue"]

이며, 반환값은 입력된 문자열 값이다. 그러므로 입력된 값을 문자열 변수에 저장하기 위해 다음과 같은 문장을 이용한다.

```
String name = request.getParameter("name");
String studentNum = request.getParameter("studentNum");
String sex = request.getParameter("sex");
String country = request.getParameter("country");
```

[결과]



- 서블릿에서 `request.getParameter(String name)` 처리

loginForm.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>getParameter 실습 </title>
</head>
<body>
<h1>getParameter 실습</h1>
<form action="login" method="get">
    <fieldset>
        <legend>로그인 폼</legend>
        <ul>
            <li>
                <label for="userid">아이디</label>
                <input type="text" name="userid" >
            </li>
            <li>
                <label for="passwd">비밀번호</label>
                <input type="password" name="passwd" >
            </li>
            <li><input type="submit" value="전송"></li>
        </ul>
    </fieldset>
</form>
```

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

LoginServlet.java

```
package com.test;
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;
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String userid = request.getParameter("userid");
        String passwd = request.getParameter("passwd");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("아이디값: " + userid + "<br>");
        out.print("비밀번호값: " + passwd + "<br>");
        out.print("</body></html>");
    }
}
```

- 서블릿에서 getParameterValues(String name) 처리

sport.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>getParameterValues 실습</title>
</head>
<body>
<h1>getParameterValues 실습</h1>
<form action="sport" method="post">
    <fieldset>
        <legend>좋아하는 운동 및 성별</legend>
```

```

<ul>
    <li>
        <label for="baseball">야구</label>
        <input type="checkbox" name="sports"" value="야구" >
        <label for="football">축구</label>
        <input type="checkbox" name="sports"" value="축구">
        <label for="basketball">농구</label>
        <input type="checkbox" name="sports"" value="농구">
    </li>
    <li>
        <label for="sex">남</label>
        <input type="radio" name="sex" value="남자" checked>
        <label for="sex">여</label>
        <input type="radio" name="sex" value="여자">
    </li>
    <li><input type="submit" value="전송"></li>
</ul>
</fieldset>
</form>
</body>
</html>

```

SportServlet.java

```

package com.test;
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;
@WebServlet("/sport")
public class SportServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String [] sports = request.getParameterValues("sports");
        String sex = request.getParameter("sex");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
    }
}

```

```

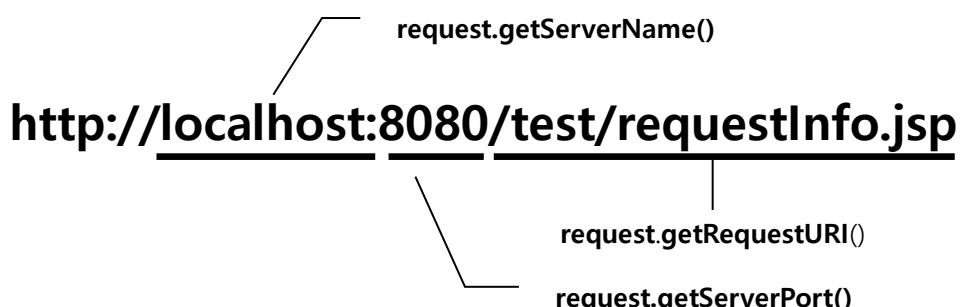
        out.print("<html><body>");
        for (String sport : sports) {
            out.print("좋아하는 운동2: " + sport + "<br>");
        }
        out.print("성별 : " + sex + "<br>");
        out.print("</body></html>");
    }
}

```

- request의 메소드

내장 객체 request의 자료유형인 인터페이스 HttpServletRequest는 다음과 같은 주요 메소드를 제공한다.

반환값	메소드	사용용도
Cookie[]	getCookies()	클라이언트에 보내진 쿠키 배열을 반환
String	getQueryString()	URL에 추가된 Query 문자열을 반환
String	getRequestURI()	클라이언트가 요청된 URI 반환. URI는 프로토콜, 서버이름, 포트번호를 제외한 서버의 컨텍스트와 파일의 문자열 URI 가 더 큰 개념이며 URI는 Uniform Resource Identifier 의 약자로 URL(Uniform Resource Locator)와 URN(Uniform Resource Name) 을 포함한다.
String	getRequestURL()	클라이언트가 요청된 URL 반환, URL은 프로토콜과 함께 주소 부분에 기술된 모든 문자열
HttpSession	getSession()	현재의 세션을 반환, 세션이 없으면 새로 만들어 반환
String	getMethod()	요청 방식인 get, post 중의 하나를 반환



request2.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>예제 request2</title>
</head>
<body>
<h2> 학생 정보 입력</h2>

```

```

<form method="post" action="request2.jsp">
    학번 : <input type="text" name="studentNum"><p>
    전공 : <select multiple name="major">
        <option SELECTED value="전산과">전산과 </option>
        <option value="국문과">국문과 </option>
        <option value="기계공학과">기계공학과 </option>
        <option value="회계학과">회계학과 </option>
        <option value="전자공학과">전자전학과 </option>
        <option value="경영학과">경영학과 </option>
        <option value="법학과">법학과 </option>
    </select> <p>
    <input type="submit" value="보내기">
</form>
</body>
</html>

```

request2.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 request2.jsp</title>
</head>
<body>
<%
    request.setCharacterEncoding("UTF-8");
    String studentNum = request.getParameter("studentNum");
    String[] majors = request.getParameterValues("major");
%>
<h2> 학생 정보 입력 결과</h2>
학번 : <%= studentNum%> <p>
전공 : <%
    if (majors == null) {
        out.println("전공 없음.");
    } else {
        //for (int i=0; i < majors.length; i++)
        //out.println(majors[i] + " ");
        //JDK 1.5 이후부터 다음 코딩 가능
        for ( String eachmajor : majors )

```

```

        out.println(eachmajor + " ");
    }

%>

<h2> 요청 정보</h2>
요청 방식 : <%= request.getMethod()%><p>
요청 URL : <%= request.getRequestURL()%><p>
요청 URI : <%= request.getRequestURI()%><p>
클라이언트 주소 : <%= request.getRemoteAddr()%><p>
클라이언트 호스트 : <%= request.getRemoteHost()%><p>
프로토콜 방식 : <%= request.getProtocol()%><p>
서버 이름 : <%= request.getServerName()%><p>
서버 포트 번호 : <%= request.getServerPort()%><p>
</body>
</html>

```

예제 request2.jsp에서 메소드 request.getParameterValues("major")는 반환 값이 문자열 배열이므로 다음과 같이 변수 majors에 저장한다. 만일 선택된 전공이 없다면 메소드 request.getParameterValues ("major") 는 null 값을 반환하므로 변수 majors에는 null값이 저장된다. 선택된 여러 전공이 저장된 문자열 배열 변수 majors를 브라우저에 출력하려면 for문을 이용하면 된다.

위 for 문장은 JDK 1.5(5.0) 이후부터 다음과 같은 for each 문장으로도 가능한데, for 문장이 반복되면서 String 변수 eachmajor에 배열 majors의 원소 값이 각각 저장되어 처리되는 것을 알 수 있다.

```

for ( String eachmajor : majors )
    out.println(eachmajor + " ");

```

[결과]

The figure consists of two screenshots of Microsoft Internet Explorer. The left screenshot shows a form titled '학생 정보 입력' (Student Information Input). It has a text input field containing '학번 : 90010001' and a dropdown menu for '전공 :'. The dropdown menu has three options: '전산과', '국문과', and '기계공학과', with '국문과' currently selected. Below the dropdown is a button labeled '보내기' (Send). An arrow points from this button to the right screenshot. The right screenshot shows the browser window title 'JSP 예제 request2.jsp - Microsoft Internet Explorer'. The page content displays various request parameters:

```

요청 방식 : POST
요청 URL : http://localhost:8080/exam02/request2.jsp
요청 URI : /exam02/request2.jsp
클라이언트 주소 : 127.0.0.1
클라이언트 호스트 : 127.0.0.1
프로토콜 방식 : HTTP/1.1
서버 이름 : localhost
서버 포트 번호 : 8080

```



- 인자의 이름 전달 메소드 `getParameterNames()`

내장 객체 `request`의 메소드 `getParameterNames()`는 반환값이 `Enumeration` 유형으로 요청 페이지의 모든 인자 이름이 저장된 목록을 반환한다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>예제 request3.html</title>
</head>
<body>
<h2> 취미와 가보고 싶은 국가</h2>
<form method="post" action="request3.jsp">
    1. 좋아하는 취미를 선택하시오. <p>
        영화 <input type="checkbox" name="hobby" value="영화"><br>
        독서 <input type="checkbox" name="hobby" value="독서"><br>
        스키 <input type="checkbox" name="hobby" value="스키"><br>
        자전거 <input type="checkbox" name="hobby" value="자전거"> <p> <hr>
    2. 여행하고 싶은 국가를 하나 선택하시오. <p>
        영국 <input type="radio" name="country" value="영국" checked><br>
        미국 <input type="radio" name="country" value="미국"><br>
        브라질 <input type="radio" name="country" value="브라질"><br>
        터키 <input type="radio" name="country" value="터키"> <p>
        <input type="submit" value="보내기">
</form>
</body>
</html>
```

실행 페이지 `request3.jsp`에서 인자 이름 목록을 알기 위해 메소드 `getParameterNames()`를 이용하며, 이 메소드의 반환 유형 `java.util.Enumeration`을 사용하기 때문에 페이지 지시자 `import` 속성을 이용하면 편리하다. 다음 소스에서 변수 `e`의 자료 유형 `Enumeration<String>`은 `Enumeration` 목록의 각 원소 자료 유형이 `String`임을 표시하는 일반화 유형을 나타낸다.

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

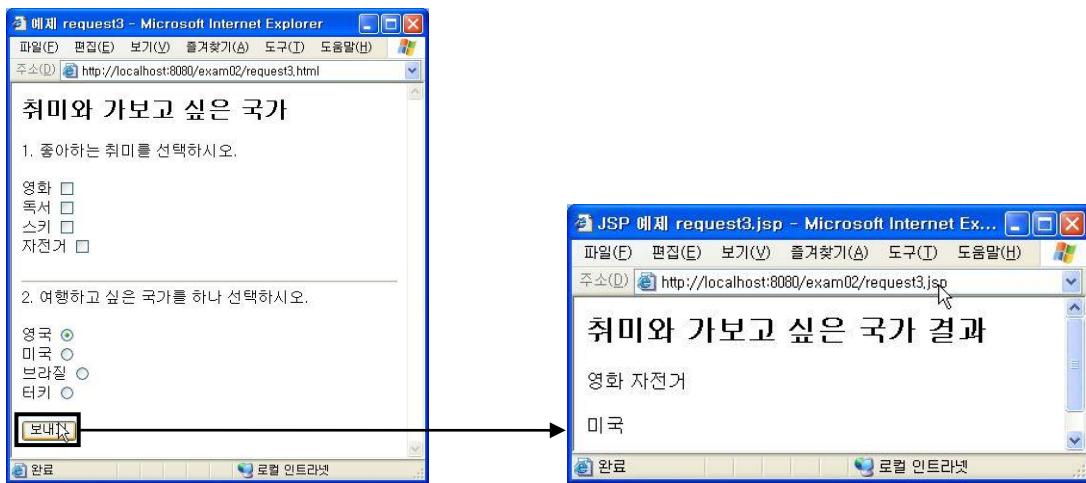
```
<% Enumeration<String> e = request.getParameterNames(); %>
```

위에서 일반화 유형 `Enumeration<String>`을 이용하면 `e.nextElement()`에서 `String`으로 자료 유형 변환이 필요없이 반환값을 `String` 유형 변수 `name`에 저장할 수 있는 장점이 있다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.Enumeration" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 request3.jsp</title>
</head>
<body>
<% request.setCharacterEncoding("UTF-8"); %>
<h2> 취미와 가보고 싶은 국가 결과</h2>
<%
//Enumeration e = request.getParameterNames();
Enumeration<String> e = request.getParameterNames();
while ( e.hasMoreElements() ) {
    //String name = (String) e.nextElement();
    String name = e.nextElement();
    String [] data = request.getParameterValues(name);
    if ( data != null ) {
        for ( String eachdata : data )
            out.println(eachdata + " ");
    }
    out.println("<p>");
}
%>
</body>
</html>
```

다음과 같이 요청된 인자의 이름을 직접 알지 못하더라도 `e.nextElement()`로 인자의 이름을 알 수 있으며, 다시 인자의 이름을 이용한 `request.getParameterValues(name)`으로 선택한 자료 값 배열 `data`를 얻을 수 있다.

[결과]



9.3 한글 처리

- post 방식에서 request.setCharacterEncoding("UTF-8")

HTML의 <form> 태그에서 정보 전송 방법 중 post 방식은 전송 자료 크기의 제한 없이 사용자가 입력한 내용을 공개하지 않고 전송하는 방법이다. 폼 양식의 post 방식인 경우, 한글 처리를 위하여 정보를 전송받은 JSP 파일에서 내장 객체 request를 사용하기 이전에 메소드 request.setCharacterEncoding("euc-kr")을 호출한다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>예제 postrequest.html</title>
</head>
<body>
<h2> 메소드 post 방식에서 한글 처리</h2>
<form method="post" action="postrequest.jsp">
    한글 성명 : <input type="text" name="korname"><p>
    영문 성명 : <input type="text" name="engname"><p>
    <input type="submit" value="보내기">
</form>
</body>
</html>
```

postrequest.jsp

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

```

<meta charset="UTF-8">
<title>JSP 예제 postrequest.jsp</title>
</head>
<body>
<% request.setCharacterEncoding("UTF-8"); %>
<h2> 메소드 post 방식에서 한글 처리</h2>
<hr>
한글 성명 : <%= request.getParameter("korname")%><p>
영문 성명 : <%= request.getParameter("engname")%><p>
</body>
</html>

```

[결과]



- get 방식에서 설정 파일 server.xml에서 URIEncoding="UTF-8"

폼 양식에서 get 전송 방식은 post와는 달리 전송 자료 크기의 제한이 있으며 사용자가 입력한 내용을 공개하여 전송하는 방식이다.

GET 방식을 이용해서 파라미터를 전송하는 방법은 세가지가 존재하며 각 방법에 따라서 파라미터 값을 인코딩할 때 사용하는 캐릭터 셋이 달라질 수 있다.

GET 방식 이용 시 파라미터 전송 방법	인코딩 결정
<a> 태그의 링크 태그에 쿼리 문자열 추가	웹 페이지 인코딩 사용
HTML 폼의 method 속성값을 "get"으로 지정해서 폼을 전송	웹 페이지 인코딩 사용
웹 브라우저에 주소를 직접 쿼리 문자열 포함한 URL 입력	웹 브라우저마다 다름

먼저 get 방식으로 파라미터를 전송하는 첫번째 방법은 <a>태그의 링크에 쿼리 문자열을 이용하는 것이다. 이때 웹 브라우저는 사용자가 링크를 클릭하면 post 방식의 경우와 마찬가지로 웹 페이지의 캐릭터 셋을 이용해서 파라미터를 인코딩한다. 즉 웹 페이지의 인코딩이 UTF-8이면 UTF-8 캐릭터 셋을 이용해서 파라미터 값을 인코딩한다.

웹 브라우저의 주소에 직접 쿼리문자열을 입력하는 경우에는 웹 브라우저에 따라서 선택하는 캐릭터 셋이 달라질 수 있다. 인터넷 익스플로러는 현재 웹 페이지에 선택되어 있는 캐릭터 셋을 이용해서 파라

미터값을 인코딩한다. 이때 인터넷 익스플로러나 파이어폭스는 UTF-8 캐릭터 셋을 이용해서 파라미터 값을 인코딩 한 URL을 웹 서버에 요청하게 된다. 반면 애플 사파리나 구글 크롬의 경우는 파라미터 값을 utf-8 캐릭터 셋을 이용해서 인코딩 한 URL을 웹 서버에 요청하게 된다. 사실 HTTP, URL, URI 등의 표준에는 GET 방식으로 전달되는 파라미터 값을 인코딩 할 때 어떤 캐릭터셋을 사용해야 하는지에 대한 규칙이 정해져 있지 않다. 예를 들어 톰캣 6.0은 GET 방식으로 전달되는 파라미터 값을 읽어올 때 기본적으로 ISO-8859-1 캐릭터 셋을 사용하기 때문에 웹 브라우저에서 UTF-8이나 UTF-8을 이용해서 인코딩 한 파라미터를 올바르게 읽어 올 수 없다. 또한 GET 방식으로 전송된 파라미터에 대해서는 request.setCharacterEncoding() 메소드로 지정한 캐릭터 셋이 적용되지 않는다.

그래서 GET 방식으로 전달된 파라미터를 읽을 때 사용할 캐릭터 셋을 다음과 같은 방법으로 지정할 수 있다.

- server.xml파일에서 <Connector>의 useBodyEncodingForURI 속성의 값을 true로 지정하는 방법
- server.xml파일에서 <Connector>의 URLEncoding 속성의 값으로 원하는 캐릭터 셋을 지정하는 방법

JSP컨테이너의 서버 설정 파일 server.xml 파일은 [톰캣 설치 폴더]의 하부[conf]폴더에 저장되어 있다. 폼 양식에서 get 방식인 경우, 한글 처리를 위해서 JSP 엔진의 서버 설정 파일 server.xml를 수정한다.

<connector port="8080" ... />에서 속성 [URLEncoding="UTF-8"]을 추가

사용하는 포트번호의 <Connector> 속성에 다음과 같이 코딩하면 된다.

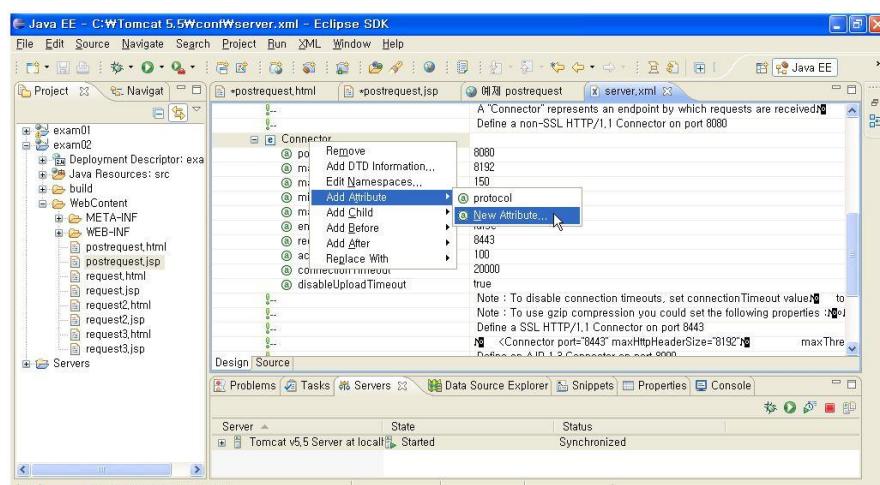
<Connector port="80" protocol="HTTP/1.1" connectionTimeout="20000"
 redirectPort="8443"
 useBodyEncodingForURI="true" />

useBodyEncodingForURI 속성값을 "true"로 지정하면 GET 방식으로 전달된 파라미터 값을 읽어올 때 request.setCharacterEncoding()메소드로 지정한 캐릭터 셋이 적용된다.

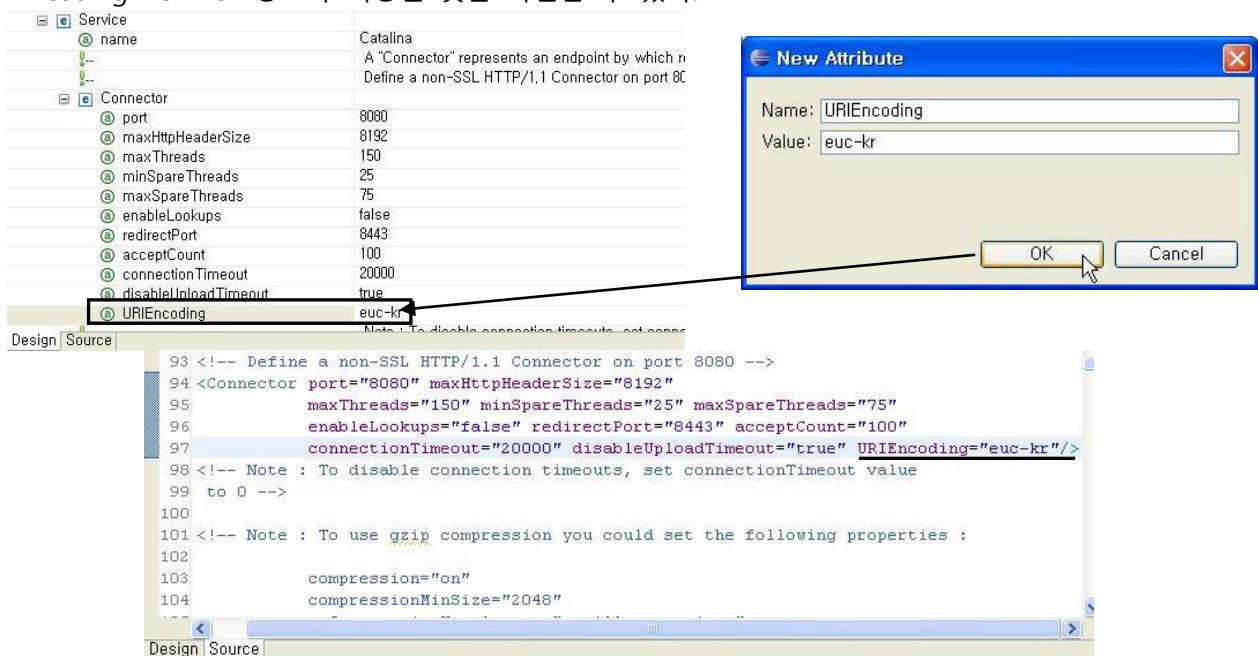
<Connector port="80" protocol="HTTP/1.1" connectionTimeout="20000"
 redirectPort="8443"
 URLEncoding="UTF-8" />

톰캣에서 GET 방식 파라미터의 캐릭터 셋을 올바르게 처리하는 두 번째 방법은 server.xml 파일에서 <Connector>의 URLEncoding 속성값으로 원하는 캐릭터 셋을 지정하는 것이다.

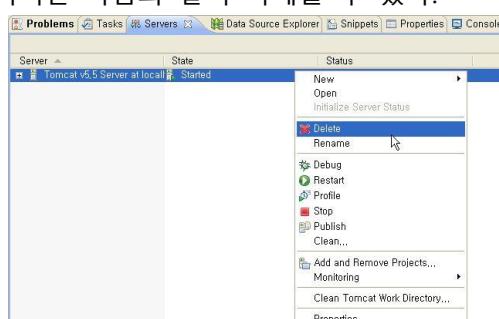
server.xml 파일을 간편하게 수정하기 위해, 이클립스 메뉴 [File/Open File...]에서 [톰캣 설치 폴더/conf] 폴더 server.xml을 열어 다음과 같이 항목 [Server]하부, [Service]하부에서 [port=80]인 [Connector]를 찾는다.



찾은 [Connector]를 클릭한 후, 오른쪽 메뉴에서 [Add Attribute/New Attribute...]를 선택하여 대화상자 [New Attribute]를 띄운다. 대화상자 [New Attribute]에서 다음과 같이 각각 URIEncoding, UTF-8을 입력하여 server.xml 파일을 저장한다. 입력시 XML 파일은 대소문자 구분하니 주의하도록 한다. 수정된 server.xml 파일에서 편집기 왼쪽 하단부의 [Design]과 [Source]를 각각 누르면 표와 소스 형태로 다음과 URIEncoding="UTF-8" 정보가 저장된 것을 확인할 수 있다.



수정된 server.xml 파일을 톰캣 서버에 반영하기 위해서는 이클립스에서 새로운 톰캣 서버를 다시 생성하여 실행해야 한다. 한글 처리 설정을 반영하지 않은 server.xml로 생성하여 사용하던 이전 서버는 한글 처리가 되지 않으므로, 이러한 서버는 다음과 같이 삭제할 수 있다.



getrequest.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>예제 getrequest.html</title>
```

```

</head>
<body>
<h2> 메소드 get 방식에서 한글 처리</h2>
<form method="get" action="getrequest.jsp">
    한글 성명 : <input type="text" name="korname"> <p>
    영문 성명 : <input type="text" name="engname"> <p>
    <input type="submit" value="보내기">
</form>
</body>
</html>

```

getrequest.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 getrequest.jsp</title>
</head>
<body>
<h2> 메소드 get 방식에서 한글 처리</h2>
<hr>
한글 성명 : <%= request.getParameter("korname")%> <p>
영문 성명 : <%= request.getParameter("engname")%> <p>
</body>
</html>

```

get 방식은 URL 부분에 전송 자료가 [name1=값1&name2=값2] 형식으로 추가되는 특징이 있다.

[결과]



URL 부분에 추가되는 부분을 질의 문자열(query string)이라 한다. 질의 문자열은 다음과 같은 구조를 갖는다.

`http://localhost:8080/exam02/getrequest.jsp?korname=%C8%AB%B1%E6%B5%BF&engname=Hong+Gil`



```

9<form method="get" action="getrequest.jsp">
10    한글 설명 : <input type="text" name="korname"><p>
11    영문 설명 : <input type="text" name="engname"><p>
12    <input type="submit" value="보내기">
13</form>

```

질의문자열

korname=%C8%AB%B1%E6%B5%BF&engname=Hong+Gil+Dong

- 서블릿에서 한글처리

memberForm.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>getParameterNames 실습 </title>
</head>
<body>
<h1>getParameterNames 실습 </h1>
<form action="Member" method="post">
<input type="hidden" name="action" value="write">
    제목<input type="text" name="title"><br />
    작성자<input type="text" name="author"><br />
    내용<textarea name="content" rows="10" ></textarea><br />
    <input type="submit" value="저장">
</form>
<a href="/myboard/board?action=page">목록</a>
</body>
</html>

```

MemberServlet.java

```

package com.test;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/Member")
public class MemberServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {

```

```

        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        Enumeration<String> enu = request.getParameterNames();
        while ( enu.hasMoreElements() ){
            String name = enu.nextElement();
            String value = request.getParameter(name);
            out.print( name + " : " + value + "<br>" );
        }
        out.print("</body></html>");
    }
}

```

9.4 내장 객체 response와 out

- response(인터페이스 HttpServletResponse)

내장 객체 response는 서버가 클라이언트에게 요청에 대한 응답을 보내기 위한 객체이다. 내장 객체 response의 자료유형인 인터페이스 HttpServletResponse는 상위 인터페이스로 ServletResponse를 가지며 다음과 같은 주요 메소드를 제공한다.

반환값	메소드	사용용도
void	addCookie(Cookie cookie)	쿠키 데이터 기록
void	addHeader(String name, String value)	response 헤더 내용 기록
void	sendRedirect(String location)	지정된 location 페이지로 이동
void	setBufferSize(int size)	버퍼 크기 지정
void	setContentType(String type)	Content Type 지정
int	getBufferSize(int size)	버퍼 크기 반환

- 메소드 sendRedirect()

내장 객체 response의 메소드 sendRedirect()를 이용하여 원하는 페이지로 이동할 수 있다.

```

<%
    String URL="http://www.naver.com";
    response.sendRedirect(URL);
%>

```

다음은 입력한 단어로 [네이버]에서 검색하도록 하는 기능을 response의 메소드 sendRedirect()를 이용하

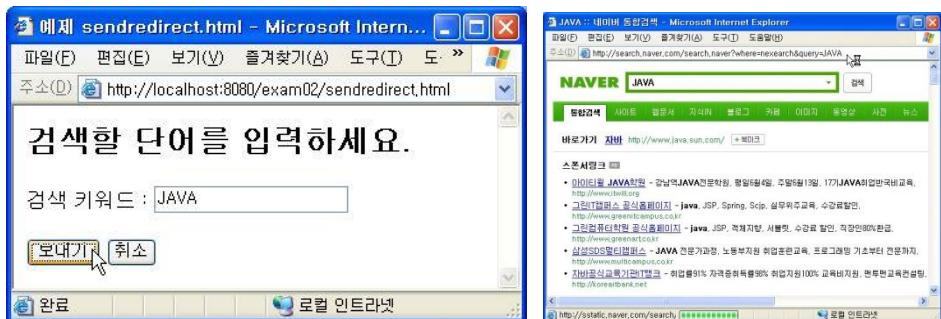
여 구현한 예제이다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>예제 sendredirect.html</title>
</head>
<body>
<h2> 검색할 단어를 입력하세요.</h2>
<form method="get" action="sendredirect.jsp">
    검색 키워드 : <input type="text" name="word"><p>
    <input type="submit" value="보내기">
    <input type="reset" value="취소">
</form>
</body>
</html>
```

sendredirect.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 sendredirect.jsp</title>
</head>
<body>
<%
    String URL = "http://search.naver.com/search.naver?where=nexearch";
    String keyword = request.getParameter("word");
    URL += "&" + "query=" + keyword;
    response.sendRedirect(URL);
%>
</body>
</html>
```

[결과]



- out(클래스JspWriter)

내장 객체 `out`은 클래스 `javax.servlet.jsp.JspWriter` 자료유형으로 JSP 페이지의 출력을 위한 객체이다. 내장 객체 `out`은 다음과 같이 출력과 버퍼링에 관련된 주요 메소드를 제공한다.

반환값	메소드	사용용도
void	print(여러 자료값)	여러 자료 유형을 출력
void	println(여러 자료값)	여러 자료 유형을 출력하고 현재 줄을 종료
void	clearBuffer()	버퍼의 현재 내용물을 제거
void	flush()	버퍼 크기 지정
void	clear()	버퍼의 내용물을 제거
void	close()	스트림을 닫음
int	getBufferSize()	버퍼의 전체 크기를 반환
int	getRemaining()	버퍼의 남아 있는 크기를 반환
boolean	isAutoFlush()	현재 autoFlush 상태를 반환

플러시(flush)

버퍼가 다 찼을 때, 버퍼에 쌓인 데이터를 실제로 전송되어야 할 곳(저장되어야 할 곳)에 전송하고 버퍼를 비우는 것을 플러시라고 한다.

내장 객체 `out`의 메소드 `clear()`를 호출하면 버퍼의 내용이 모두 제거되므로 이전에 버퍼에 출력한 내용이 모두 사라지며, 메소드 `getBufferSize()`, `getRemaining()`, `isAutoFlush()`를 이용하여 버퍼의 상태를 알 수 있다.

out.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 out.jsp</title>
</head>
<body>
<%
    out.println("이 부분은 출력되지 않습니다.");
%>
```

```

        out.clear();

%>
<h2>현재 페이지의 출력 버퍼 상태</h2><p>
초기 출력 버퍼 크기 : <%=out.getBufferSize()%> byte<p>
남은 출력 버퍼 크기 : <%=out.getRemaining()%> byte<p>
autoFlush : <%=out.isAutoFlush()%><p>

</body>
</html>

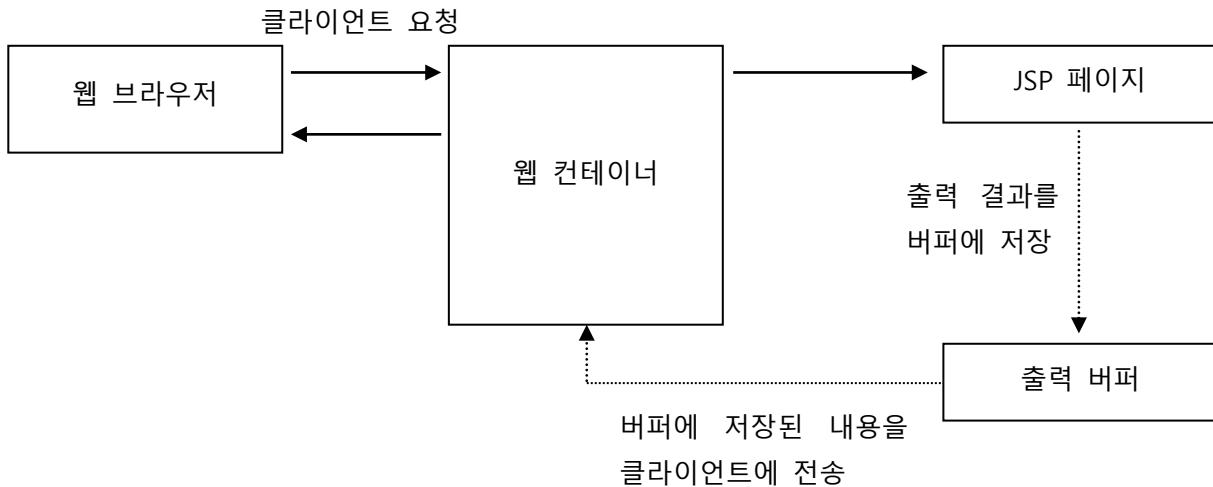
```

[결과]



- 버퍼링

JSP 페이지는 생성된 결과를 곧바로 웹 브라우저에 전송하지 않고 출력 버퍼라고 불리는 곳에 임시로 출력 결과를 저장했다가 한 번에 웹 브라우저에 전송한다.



JSP 페이지가 생성하는 출력 내용을 곧바로 웹 브라우저에 전송하지 않고 버퍼에 저장했다가 한꺼번에 전송함으로써 생기는 장점은 다음과 같다.

- 데이터 전송 성능이 향상된다.
- 곧바로 웹 브라우저로 전송되지 않기 때문에 JSP 실행 도중에 버퍼를 비우고 새로운 내용을 보여 줄 수 있다.

그래서 page 디렉티브에서 buffer 속성은 JSP 페이지가 사용할 버퍼를 설정하는데 사용되며 autoFlush 속성은 버퍼가 다 찼을 때 어떻게 처리할지를 결정할 때 사용하는 속성이다.

autoFlush가 true이면 버퍼가 다 찼을 경우 버퍼를 플러시하고 계속해서 작업을 진행하며, false이면 버퍼가 가득 차기 전에 flush()를 호출하여 출력을 수동으로 해야 한다. 그러므로 다음과 같이 만일 flush하기 전에 버퍼가 가득 차면 버퍼 오버플로(buffer overflow)오류가 발생한다.

```
<%@ page autoFlush="false" buffer="1kb" %>
<%
    for (int i = 1; i < 25; i++) {
        out.println("남은 출력 버퍼 크기(out.getRemaining()) : " + out.getRemaining() + "<br>");
    }
%>
```

```
<%@ page autoFlush="true" buffer="1kb" %>
<%
    for (int i = 1; i < 25; i++) {
        out.println("buffer 확인");
    }
%>
```

JSP 페이지에서 autoFlush는 기본이 true이므로 버퍼가 가득 차기 전에 flush()를 자동으로 호출한다. 그러나 위와 같이 속성 autoFlush를 false로 지정하면 메소드 out.getRemaining()을 이용하여 버퍼가 남은 양이 적으면 flush()를 호출하여 출력을 수동으로 해야 한다.

autoflush.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제 autoflush.jsp</title>
</head>
<body>
    <%@ page autoFlush="false" buffer="1kb" %>
    <h2>현재          autoFlush</h2>
    <%=out.isAutoFlush() %> </h2> <p>
    <%
        for (int i = 1; i < 25; i++) {
            out.println("남은      출력      버퍼      크기");
            (out.getRemaining() : " + out.getRemaining() + "<br>");

            //autoFlush가 false이면 알아서 버퍼를 출력해야 한다.
            if (out.getRemaining() < 50) {
                out.println("<br>");
                out.flush();
            }
        }
    </p>

```

현재 autoFlush = false

남은 출력 버퍼 크기(out.getRemaining()) : 843
남은 출력 버퍼 크기(out.getRemaining()) : 800
남은 출력 버퍼 크기(out.getRemaining()) : 757
남은 출력 버퍼 크기(out.getRemaining()) : 714
남은 출력 버퍼 크기(out.getRemaining()) : 671
남은 출력 버퍼 크기(out.getRemaining()) : 628
남은 출력 버퍼 크기(out.getRemaining()) : 585
남은 출력 버퍼 크기(out.getRemaining()) : 542
남은 출력 버퍼 크기(out.getRemaining()) : 499
남은 출력 버퍼 크기(out.getRemaining()) : 456
남은 출력 버퍼 크기(out.getRemaining()) : 413
남은 출력 버퍼 크기(out.getRemaining()) : 370
남은 출력 버퍼 크기(out.getRemaining()) : 327
남은 출력 버퍼 크기(out.getRemaining()) : 284
남은 출력 버퍼 크기(out.getRemaining()) : 241
남은 출력 버퍼 크기(out.getRemaining()) : 198
남은 출력 버퍼 크기(out.getRemaining()) : 155
남은 출력 버퍼 크기(out.getRemaining()) : 112
남은 출력 버퍼 크기(out.getRemaining()) : 69
남은 출력 버퍼 크기(out.getRemaining()) : 1024 byte
남은 출력 버퍼 크기 : 754 byte

```

        }
    }

%>
<hr>
초기 출력 버퍼 크기 : <%=out.getBufferSize()%> byte
<br>
남은 출력 버퍼 크기 : <%=out.getRemaining()%> byte
</body>
</html>

```

9.5 내장 객체 application과 exception

- application(인터페이스 ServletContext)

내장 객체 application은 java.servlet.ServletContext 인터페이스 자료 유형으로 웹 애플리케이션에서 유지 관리되는 여러 환경 정보를 관리한다. 여기서 웹 애플리케이션이란 여러 개의 서블릿과 JSP로 구성되는 웹 서비스 응용 프로그램 단위로 내장 객체 application은 서블릿과 서버 간의 자료를 교환하는 여러 메소드를 제공한다. 즉 웹 컨테이너가 구현하여 제공하는 객체로 해당 웹 어플리케이션의 실행환경을 제공하는 서버의 정보와 서버측 자원에 대한 정보를 얻어 내거나 해당 어플리케이션의 이벤트 로그를 다루는 메소들을 제공한다. application은 각 웹 어플리케이션 당 오직 하나만의 객체만이 생성하므로 해당 웹 어플리케이션 전체 영역에서 자원을 공유해야 할 때(방문자 수 등의 통계를 다룰 때) 주로 사용된다.

반환값	메소드	사용용도
String	getServerInfo()	JSP 컨테이너의 이름과 버전 반환
String	getRealPath(String path)	path에 지정된 자원 파일시스템의 실제 경로 리턴
Object	getAttribute(String name)	웹 응용에서 지정된 이름의 속성을 반환.
void	log(String msg)	지정된 msg의 로그를 저장
void	setAttribute(String name, Object object)	웹응용에서 지정된 이름으로 object를 저장
void	removeAttribute(String name)	웹응용에서 지정된 이름의 속성을 삭제

- 웹 응용 프로그램에서 조회 수 관리

내장 객체 application에서 메소드 getServerInfo()를 이용하면 웹 응용에서 이용하는 JSP 컨테이너의 이름과 버전을 알 수 있다.

서버 컨테이너 정보 : <%=application.getServerInfo() %>

application0.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">

```

```

<title>application0.jsp 내장 객체 예제 </title>
</head>
<body>
    <h4> application 내장 객체 예제 </h4>
    <%
        String serverInfo = application.getServerInfo();
        String realPath = application.getRealPath("/");
    %>
    Server: <%= serverInfo %><br>
    Path of Document: <%= realPath %><br>
</body>
</html>

```

[결과]



내장 객체 application의 속성 관련 메소드를 이용하여 접속자가 페이지 조회수를 관리하는 프로그램을 작성해 보자. 메소드 setAttribute(속성이름, 속성값)를 이용하여 속성 이름은 "count"로 하고 속성값은 조회수를 문자열로 만든 값으로 저장한다.

```
application.setAttribute("count", Integer.toString(++count));
```

이미 지정된 속성 "count"가 있다면 이 값을 가져와 1을 더한 후 출력하며, 만일 반환값이 null이라면 처음 웹 응용이 시작한 것으로 파악하여 count를 0으로 초기화한다.

application.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 application.jsp</title>
</head>
<body>
<%! int application = 0; %>
<%! int count = 0; %>
<%
    String scount = (String) application.getAttribute("count");
    if (scount != null) {
        count = Integer.parseInt(scount);
    } else {

```

```

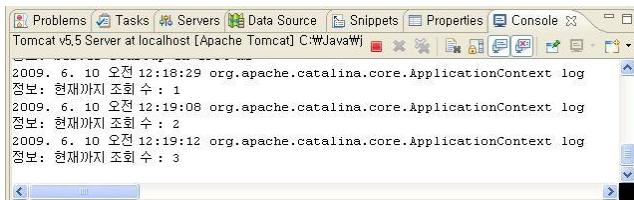
        count = 0;
    }
    application.setAttribute("count", Integer.toString(++count));
    application.log("현재까지 조회 수 : " + count);
%>
    서버 컨테이너 정보 : <%=application.getServerInfo() %> <p>
    현재까지 조회 수 : <%=count %>
</body>
</html>

```

[결과]



위 예제에서 로그 정보를 남기므로 콘솔 뷰에도 조회 수가 보이는 것을 확인할 수 있다.



- 예외 처리

JSP에서는 JSP 페이지를 처리하는 도중에 예외가 발생할 경우 예러 화면 대신 지정한 JSP 페이지를 보여줄 수 있는 기능을 제공하고 있다. 예러가 발생할 때에 보여 줄 JSP 페이지는 page 딕렉티브의 `errorPage` 속성을 사용해서 지정할 수 있다.

readParameter.jsp

```

<%@ page contentType = "text/html; charset=UTF-8" %>
<%@ page errorPage = "/error/viewErrorMessage.jsp" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>readParameter.jsp</title> </head>
<body>
    name 파라미터 값: <%= request.getParameter("name").toUpperCase() %>
</body>
</html>

```

- 예러 페이지 작성하기

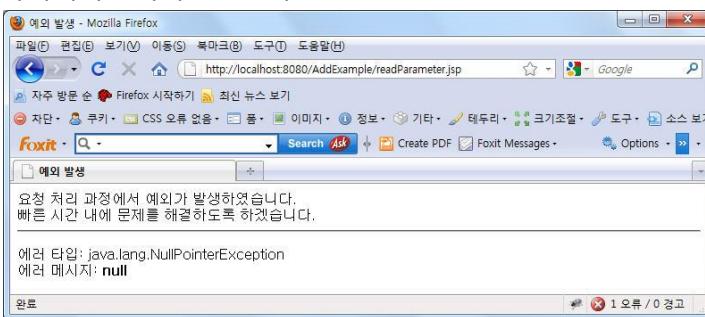
page 딕렉티브의 `errorPage` 속성을 사용해서 에러 페이지를 지정하면 에러가 발생할 때 지정된 에러 페이지를 보여주게 된다. 에러 페이지에 해당하는 JSP 페이지는 page 딕렉티브의 `isErrorPage` 속성의 값을 "true"로 지정해 주어야 한다. 그때 사용할 수 있는 내부 객체가 `exception`이다. 내부 객체 `exception`은 다음과 같은 메소드를 이용하여, 지정한 예외 처리 페이지에서 적절한 예외 처리를 구현한다.

반환값	메소드	사용용도
String	<code>getMessage()</code>	예외를 표시하는 문자열을 반환
Object	<code>toString()</code>	예외 자체 문자열을 반환
void	<code>printStackTrace()</code>	표준 출력으로 스택 추적 정보 출력

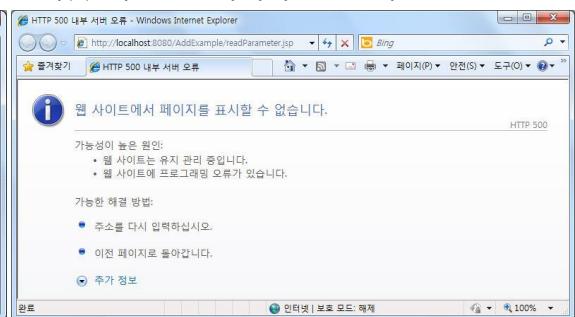
/error/viewErrorMessage.jsp

```
<%@ page contentType = "text/html; charset=UTF-8" %>
<%@ page isErrorPage = "true" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>/error/viewErrorMessage.jsp</title></head>
<body>
    요청 처리 과정에서 예외가 발생하였습니다.<br>
    빠른 시간 내에 문제를 해결하도록 하겠습니다.
    <p>
        예외 타입: <%= exception.getClass().getName() %> <br>
        예외 메시지: <b><%= exception.getMessage() %></b>
    </p>
</body>
</html>
```

파이어폭스에서 실행 화면



인터넷 익스플로러에서 실행 화면



그런데 인터넷 익스플로러에서 실행하면 에러 페이지가 출력한 내용이 아닌 인터넷 익스플로러가 자체적으로 제공하는 'HTTP 오류 메시지'가 화면에 출력된다. 인터넷 익스플로러는 다음과 같은 경우에 서버에서 전송한 응답 화면이 아닌 자체적으로 제공하는 오류 메시지 화면을 출력한다.

- 응답의 상태 코드가 404나 500과 같은 에러 코드이고
- 전체 응답 결과 데이터의 길이가 513 바이트보다 작을 때

따라서 인터넷 익스플로러에서도 에러 페이지의 내용이 올바르게 출력되길 원한다면, 에러 페이지가 생성하는 응답 화면의 데이터 크기가 513 바이트 이상이어야 한다. 그래서 다음과 같이 HTML 주석을 포

함시켜서 응답 결과의 길이가 513 바이트를 넘도록 해주면 된다.

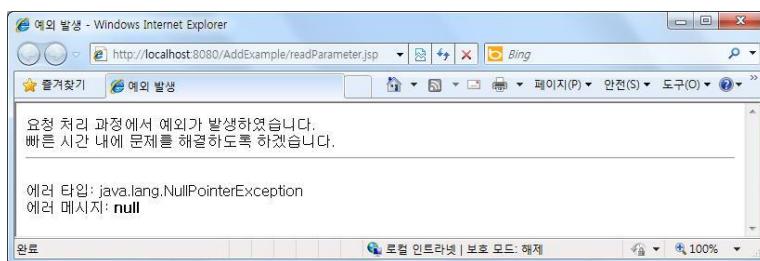
위에서 작성한 /error/viewErrorMessage.jsp 파일에 주석문 추가

```
<%@ page contentType = "text/html; charset=UTF-8" %>
<%@ page isErrorPage = "true" %>
<!DOCTYPE html>
<html>
...
</html>
<!--
```

만약 에러 페이지의 길이가 513 바이트보다 작다면, 인터넷 익스플로러는 이 페이지가 출력하는 에러 페이지를 출력하지 않고 자체적으로 제공하는 'HTTP 오류 메시지' 화면을 출력할 것이다.

만약 에러 페이지의 길이가 513 바이트보다 작은데 에러 페이지의 내용이 인터넷 익스플로러에서도 올바르게 출력되길 원한다면, 응답 결과에 이 주석과 같은 내용을 포함시켜서 에러 페이지의 길이가 513 바이트 이상이 되도록 해 주어야 한다. 참고로 이 주석은 456바이트이다.

```
-->
```



. 응답 상태 코드별로 에러 페이지 지정하기

JSP는 /WEB-INF/web.xml 파일을 통해서 각각의 응답 상태 코드별로 보여줄 페이지를 지정할 수 있도록 하고 있다. 에러 코드에 대해서 보여줄 에러 페이지는 다음과 같이 web.xml 파일에 지정할 수 있다.

```
<error-page>
    <error-code>404</error-code>
    <location>/error/error404.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/error/error500.jsp</location>
</error-page>
```

<error-page>태그는 하나의 에러 페이지를 지정하며, <error-code>와 <location>태그는 각각 에러 상태 코드와 에러 페이지의 위치를 지정한다.

주요 응답 상태 코드

HTTP 프로토콜은 응답 상태 코드를 이용해서 서버의 처리 결과를 웹 브라우저에 알려주며, 주요 응답 상태 코드로는 다음과 같다.

-
- 200 - 요청이 정상적으로 처리됨.
 - 400 - 클라이언트의 요청이 잘못된 구문으로 구성됨.
 - 401 - 접근이 허용되지 않음
 - 404 - 지정된 URL을 처리하기 위한 자원이 존재하지 않음
 - 405 - 요청된 메소드는 허용되지 않음
 - 500 - 서버 내부 에러. 예를 들어 JSP에서 예외가 발생하는 경우가 해당된다.
 - 503 - 서버가 일시적으로 서비스를 제공할 수 없음. 급격하게 부하가 몰리거나 서버가 임시 보수 중인 경우가 해당된다.
-

/error/error404.jsp

```
<%@ page contentType = "text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>/error/error404.jsp</title></head>
<body>
    <strong>요청한 페이지는 존재하지 않습니다:</strong>
    <br> <br>
    주소를 올바르게 입력했는지 확인해보시기 바랍니다.
</body>
</html>
```

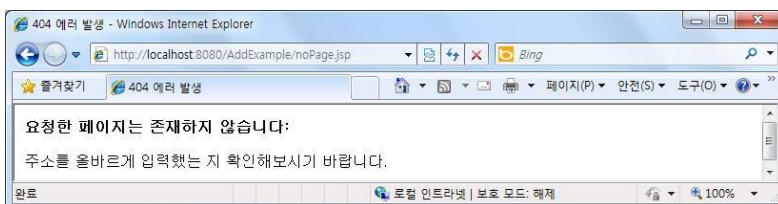
<!--

만약 에러 페이지의 길이가 513 바이트보다 작다면, 인터넷 익스플로러는 이 페이지가 출력하는 에러 페이지를 출력하지 않고 자체적으로 제공하는 'HTTP 오류 메시지' 화면을 출력할 것이다.

만약 에러 페이지의 길이가 513 바이트보다 작은데 에러 페이지의 내용이 인터넷 익스플로러에서도 올바르게 출력되길 원한다면, 응답 결과에 이 주석과 같은 내용을 포함시켜서 에러 페이지의 길이가 513 바이트 이상이 되도록 해 주어야 한다.

참고로 이 주석은 456바이트이다.

-->



· 예외 타입별로 에러 페이지 지정하기

JSP 페이지에서 발생하는 에러 종류에 따라서 에러 페이지를 지정할 수 있다. 앞에서 작성한 에러 코드 별로 에러 페이지를 지정하는 방법과 동일합니다. /WEB-INF/web.xml 파일에 다음과 같이 코딩한다.

```
<error-page>
    <exception-type>java.lang.NullPointerException</exception-type>
```

```
<location>/error/errorNullPointer.jsp</location>
```

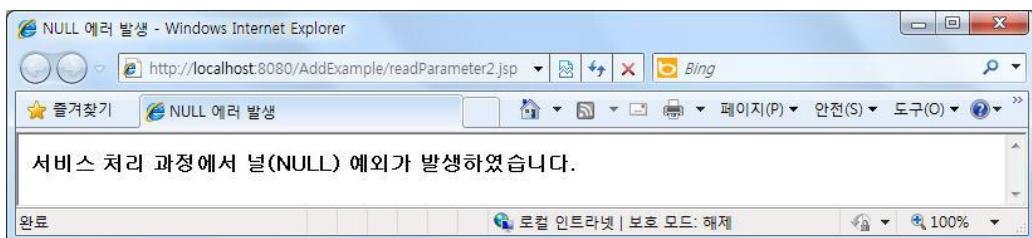
```
</error-page>
```

/error/errorNullPointer.jsp

```
<%@ page contentType = "text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>/error/errorNullPointer.jsp </title></head>
<body>
    <strong>서비스 처리 과정에서 널(NULL) 예외가 발생하였습니다.</strong>
</body>
</html>
<!--
만약 에러 페이지의 길이가 513 바이트보다 작다면,
인터넷 익스플로러는 이 페이지가 출력하는 에러 페이지를 출력하지 않고
자체적으로 제공하는 'HTTP 오류 메시지' 화면을 출력할 것이다.
만약 에러 페이지의 길이가 513 바이트보다 작은데
에러 페이지의 내용이 인터넷 익스플로러에서도 올바르게 출력되길 원한다면,
응답 결과에 이 주석과 같은 내용을 포함시켜서
에러 페이지의 길이가 513 바이트 이상이 되도록 해 주어야 한다.
참고로 이 주석은 456바이트이다.
-->
```

앞에서 작성한 readParameter.jsp 파일에서 errorCode 설정을 지우고 readParameter2.jsp 파일을 생성한다.

```
<%@ page contentType = "text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>파라미터 출력</title></head>
<body>
    name 파라미터 값: <%= request.getParameter("name").toUpperCase() %>
</body>
</html>
```



9.6 내장 객체 pageContext, page, session, config

- pageContext (클래스 pageContext)

내부 객체는 pageContext는 자료유형 클래스 javax.servlet.jsp.PageContext로 JSP 페이지에 관한 정보와 다른 페이지로 제어권을 넘겨줄 때 이용되는 메소드를 제공한다. 즉 pageContext 내장 객체는 하나의 JSP 페이지와 1:1 매핑되는 객체로서 다음과 같은 기능을 제공한다.

- 다른 내장 객체 구하기
- 속성 처리하기
- 페이지의 흐름 제어하기
- 여러 데이터 구하기

반환값	메서드	설명
void	forward(String)	다른 서블릿 혹은 JSP로 요청을 이동 (현재 페이지의 요청과 응답에 관한 제어권을 url로 지정된 주소로 영구적으로 넘긴다. forward된 페이지의 요청처리가 종료되면 응답도 종료된다.)
void	include(String)	지정된 페이지를 현재의 위치에 삽입 (현재 페이지의 요청과 응답에 관한 제어권을 url로 지정된 주소로 임시로 넘긴다. include된 페이지의 처리가 끝나면 제어권은 다시 원래의 페이지로 돌아온다. 따라서 include로 지정된 페이지의 내용을 원래 페이지에 삽입하는 효과를 가진다.)
Exception	getException()	Exception 객체를 반환
Object	getPage()	서블릿 인스턴스 객체를 리턴한다. (page 내장 객체를 리턴한다.)
JspWriter	getOut()	응답 출력 스트림을 리턴한다. (out 내장 객체를 리턴한다.)
ServletRequest	getRequest()	클라이언트의 요청정보를 담고 있는 객체를 리턴한다. (request 내장 객체를 리턴한다)
ServletResponse	getResponse()	요청에 대한 응답 객체를 리턴한다. (response 내장 객체를 리턴한다)
ServletConfig	getServletConfig()	서블릿의 초기 설정 정보를 담고 있는 객체를 리턴한다. (config 내장 객체를 리턴한다.)
ServletContext	getServletContext()	서블릿의 실행 환경 정보를 담고 있는 객체를 리턴한다. (application 내장 객체를 리턴한다.)
HttpSession	getSession()	클라이언트의 세션 정보를 담고 있는 객체를 리턴한다. (session 내장 객체를 리턴한다.)
Object	findAttribute(String)	page, request, session, application 범위 내에서 사용 가능한 속성의 값을 반환
void	removeAttribute(String)	지정한 이름의 속성 객체를 제거

Object	getAttribute(String)	page 범위 내에서 특정한 이름에 해당하는 속성 객체를 반환
void	setAttribute(String, Object)	pageContext 객체 안에 지정한 이름과 연관된 속성 객체를 저장

특히 내장 객체 pageContext는 8개의 다른 내부 객체를 얻을 수 있는 메소드를 제공한다. JSP 서블릿 소스의 _jspService() 메소드의 pageContext의 자료유형 pageContext로 선언된 것을 시작으로 session, application, config, out, page가 선언된 것을 볼 수 있다.

내장 객체 pageContext의 메소드 getServletContext(), getServletConfig(), getSession(), getOut 등을 이용하여 내장 객체 application, config, session, out 등에 각각의 객체를 저장하여 내장 객체를 설정한다. 그러므로 JSP에서 내장 객체 out을 이용하여 대신 pageContext.getOut()을 이용해도 out의 메소드를 이용할 수 있다.

```
<% pageContext.getOut().println("출력할 문자열"); %>
```

pagecontext.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 pagecontext.jsp</title>
</head>
<body>
    <h2> pageContext 예제</h2>
    <% pageContext.getOut().println("include.html을 추가"); %>
    <hr>
    <% pageContext.include("include.html"); %>
</body>
</html>
```

include.html

```
<font color=blue>
    다른 파일을 삽입하는 include(), 제어권을 넘기는 forward() 메소드 제공
</font>
```

[결과]



- page(JSP 페이지 자체를 표현)

내장 객체 page는 JSP 페이지 자체를 나타내는 객체로서 _jspService()에 다음과 같이 this가 저장되어 있다. 다시 말해 page 객체는 JSP 페이지에 의해 생성되는 서블릿 인스턴스 자체를 나타내는 객체이다. this는 자바 코드에서 자기 자신을 참조하는 레퍼런스 이므로 page 객체는 서블릿 인스턴스 자체를 참조하는 객체임을 알 수 있다.

```
Object page = this;
```

JSP 페이지는 스크립트 언어가 자바이기 때문에 page 객체를 사용하지 않고 단지 this 예약어를 사용하더라도 동일한 효과를 얻는다. 따라서 page 객체는 거의 사용할 일이 없다고 할 수 있다. 다만 컨테이너가 자바 이외의 다른 스크립트 언어를 지원하게 된다면 해당 스크립트 언어를 사용한 스크립트 코드에서 page 객체를 사용하여 서블릿 인스턴스를 참조할 수도 있게 된다.

내장 객체 page는 자바에서 자기 자신을 나타내는 키워드 this로 사용한다. 톰캣에서 this는 자료유형 org.apache.jasper.runtime.HttpJspBase의 객체로서 메소드 getServletInfo()를 제공하며, JSP 페이지 지시자의 info에 지정한 값을 반환한다.

page.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 page.jsp</title>
</head>
<body>
    <%@ page info="내장 객체 page : page 자기 자신의 객체" %>
    <%= this.getServletInfo() %> <p>
    <%= ((org.apache.jasper.runtime.HttpJspBase) (page)).getServletInfo() %>
</body>
</html>
```

[결과]



- session

인터넷 쇼핑몰에서 상품을 구매하는 경우, 장바구니를 생각해 보자. 사용자는 장바구니를 확인하면서 이미 선택한 상품도 보고 다시 다른 페이지로 이동하여 원하는 다른 상품을 더 살 수도 있으며 필요 없는 물건은 장바구니의 구매 목록에서 제거시킬 수도 있다. 이런 경우, 장바구니 페이지는 다른 페이지로 이동하더라도 현재 선택된 상품 목록과 관련 정보를 지속적으로 유지 관리하는데, 이렇게 클라이언트 사용자의 지속성 서비스를 하기 위해 session 내장 객체를 이용한다. 즉 내장 객체 session으로 클라이언트

마다 세션 정보를 저장 및 유지 관리하기 위한 객체이다

내장 객체 session은 자료유형이 인터페이스 javax.servlet.http.HttpSession으로 세션관리를 위한 다양한 메소드를 제공한다.

- config(javax.servlet.ServletConfig)

내장 객체 config는 자료유형 javax.servlet.ServletConfig 인터페이스로 서블릿이 초기화되는 동안, JSP 컨테이너가 환경 정보를 서블릿으로 전달할 때 사용하는 객체이다. (JSP 페이지가 서블릿 클래스로 변환되어 서블릿 인스턴스가 생성될 때 참조해야 할 초기 설정 정보들을 저장해 놓은 객체이다. 이러한 초기 설정 정보들은 웹 컨테이너가 구동될 때 내부에서 자체적으로 생성/관리되기 때문에 서블릿 당 1개 만의 객체가 생성되며 같은 웹 컨테이너 안의 동일 서블릿 인스턴스는 동일한 config 객체를 참조하게 된다. 일반적으로 config 객체가 JSP 페이지에서 사용되는 일은 거의 없으며, JSP 컨테이너의 구동시에 로드되는 컨테이너의 초기 설정 파일 안의 정보를 얻고자 할 때 사용될 수 있다.)

- 연습문제

1. 다음 프로그램의 실행 결과를 쓰시오.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 예제</title>
</head>
<body>
<%= "1. request" %><br>
<%= "2. response" %><br>
<%= "3. out" %><br>
<% out.clear(); %><br>
<%= "4. application" %><br>
<%= "5. exception" %><br>
</body>
</html>
```

2. 다음과 같이 [기술 정보 입력서]를 입력받아 출력하는 하나의 HTML 문서와 하나의 JSP 프로그램을 작성하시오.

기술 정보 입력	
이 름	김희진
주민등록번호	950213 - 2056443
학 력	<input type="radio"/> 고졸 <input checked="" type="radio"/> 대졸 <input type="radio"/> 대학원졸
전 공	전산과
사용가능 플랫폼	UNIX IBM OS/390 리눅스
<input type="button" value="보내기"/> <input type="button" value="다시작성"/>	



9.7 영역 객체와 속성

1) 영역(Scope)과 속성(Attribute)

JSP에서 제공하는 내장 객체들 중 session, request, application 들은 해당 객체에 정의된 유효 범위 안에서 서로 다른 페이지라 할지라도 필요한 객체(데이터)들을 저장하고 읽어들임으로써 서로 공유할 수 있는 특정한 공간을 가지고 있다. 공유되는 데이터를 속성(Attribute)이라 하고 속성을 공유할 수 있는 유효 범위를 영역(Scope)이라고 한다.

page 영역: 하나의 JSP 페이지를 처리할 때 사용되는 영역.

request 영역: 하나의 HTTP 요청을 처리할 때 사용되는 영역.

session 영역: 하나의 웹 브라우저와 관련된 영역

application 영역: 하나의 웹 어플리케이션과 관련된 영역

즉 session 내장 객체는 세션이 유지되고 있는 범위 안에서(세션 영역 안에서) 서로 다른 페이지라 할지라도 객체(데이터)들을 공유할 수 있는 속성을 가지고 있으며 이 속성에 저장된 객체(데이터)는 세션이 종료되는 순간에 반환된다(버려진다). request 객체는 클라이언트의 요청이 처리되는 동안에 속성을 사용할 수 있으며, application 객체는 해당 웹 어플리케이션이 실행되고 있는 동안에 속성을 사용할 수 있다. 이렇게 해당 영역에서 속성을 사용할 수 있는 내장 객체들을 특별히 영역 객체라고 부른다.

그럼 영역 객체가 사용하는 영역은 위의 3개가 전부가 아니라 바로 page 영역이다. page 영역은 오직 하나의 페이지 내에서만 유효성을 갖는 영역으로 주의 해야 할 점은 page 내장 객체가 아닌 pageContext 내장 객체를 통해 접근할 수 있는 영역이라는 점이다.

정리해 보면 JSP에서 정의하는 영역은 page, request, session, application으로 구성되며 이들 영역은 각각 pageContext, request, session, application 내장 객체를 통해서 속성을 설정하거나 읽어 들일 수 있다. 특별히 pageContext 내장 객체는 page 영역뿐만 아니라 모든 영역의 속성에 대한 접근이 가능하다.

영역	영역객체	속성의 유효 범위
page	pageContext	해당 페이지가 클라이언트에 서비스를 제공하는 동안에만 유효 (서블릿 인스턴스의 _jspService() 메소드가 실행되는 동안에만 유효)
request	request	클라이언트의 요청이 처리되는 동안 유효 (포워딩 또는 include를 이용하는 경우 여러 개의 페이지에서도 요청 정보가 계속 유지되므로 request 영역의 속성을 여러 페이지에 공유할 수 있다.)
session	session	세션이 유지되는 동안 유효(하나의 브라우저에 1개의 세션이 생성되므로 같은 웹 브라우저 내에서 실행되는 페이지들이 속성을 공유할 수 있다.)
application	application	웹 어플리케이션이 실행되고 있는 동안 유효 (웹 컨테이너에서 해당 어플리케이션은 오직 하나만이 실행하므로 4가지의 속성이 모두 같은 범위에서 유효)

		지 영역 중 가장 큰 영역에 해당한다.)
--	--	------------------------

2) 속성과 관련된 메소드들

영역 객체의 속성을 설정하고 읽어 들이기 위해서 JSP는 다음과 같이 대표적인 4가지 메소드들을 제공한다. 이들 4가지 메소드들은 pageContext, request, session, application 내장 객체들이 동일하게 정의하고 있는 메소드들이다.

반환형	메소드	설명
Object	getAttribute(String key)	key값으로 등록되어 있는 속성을 Object 타입으로 리턴 (key값에 해당하는 속성이 없을 경우 null을 리턴)
Enumeration	getAttributeNames()	해당 영역에 등록되어 있는 모든 속성들의 이름을 Enumeration 타입으로 리턴
void	setAttribute(String key, Object obj)	해당 영역에 key값의 이름을 obj 객체를 등록
void	removeAttribute(String key)	key값으로 등록되어 있는 속성을 제거

attributeTest1_Form.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Attribute Test Form</title></head>
<body>
<h2>영역과 속성 테스트</h2>
<form action="attributeTest1.jsp" method="post">
<table border="1" width="250">
    <tr><td colspan="2">Application 영역에 저장할 내용들</td></tr>
    <tr>
        <td>이름 </td>
        <td><input type="text" name="name"></td>
    </tr>
    <tr>
        <td>아이디 </td>
        <td><input type="text" name="id"></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit" value="전송"></td>
    </tr>
</table>
</form>
</body>
```

```
</html>
```

attributeTest1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Attribute Test</title>
</head>
<body>
<h2>영역과 속성 테스트</h2>
<%
request.setCharacterEncoding("UTF-8");
String name=request.getParameter("name");
String id=request.getParameter("id");
if(name!=null&&id!=null){
    application.setAttribute("name",name);
    application.setAttribute("id",id);
}
name=(String)application.getAttribute("name");
id=(String)application.getAttribute("id");
%>
<h3><%=name %> 님 반갑습니다. <br><%=name %>님의 아이디는 <%=id %>입니다.</h3>
<form action="attributeTest2.jsp" method="post">
<table border="1" width="250">
    <tr><td colspan="2">Session 영역에 저장할 내용들</td></tr>
    <tr>
        <td>e-mail 주소</td>
        <td><input type="text" name="email"></td>
    </tr>
    <tr>
        <td>집 주소</td>
        <td><input type="text" name="address"></td>
    </tr>
    <tr>
        <td>전화번호</td>
        <td><input type="text" name="tel"></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit" value="전송"></td>
    </tr>

```

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

attributeTest2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Attribute Test</title>
</head>
<body>
<h2>영역과 속성 테스트</h2>
<%
request.setCharacterEncoding("UTF-8");
String email=request.getParameter("email");
String address=request.getParameter("address");
String tel=request.getParameter("tel");
session.setAttribute("email",email);
session.setAttribute("address",address);
session.setAttribute("tel",tel);
String name=(String)application.getAttribute("name");
%>
<h3><%=name %> 님의 정보가 모두 저장되었습니다.</h3>
<a href="attributeTest3.jsp" style="text-decoration:none">확인하러 가기 </a>
</body>
</html>
```

attributeTest3.jsp

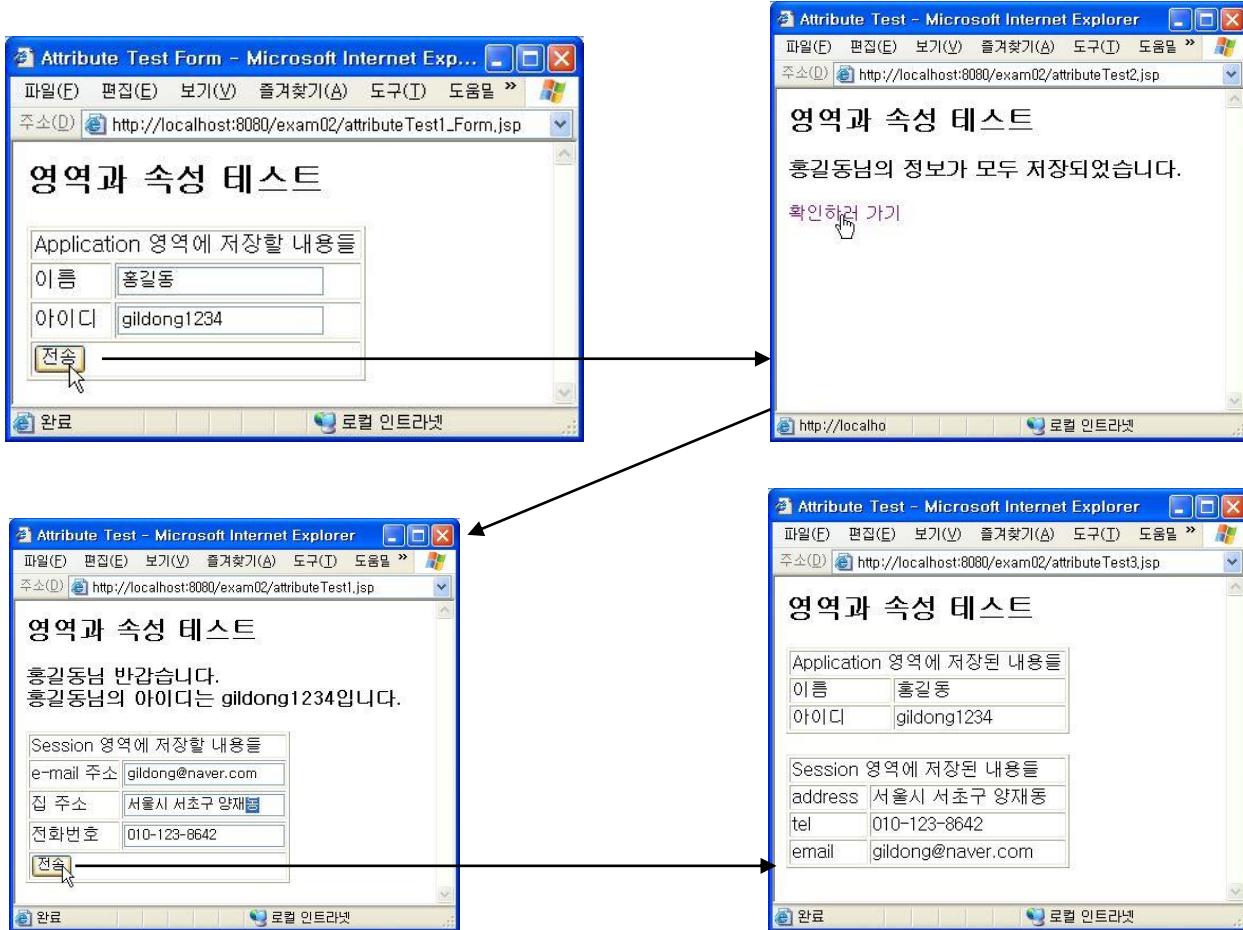
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.Enumeration"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Attribute Test</title>
</head>
<body>
<h2>영역과 속성 테스트</h2>
<table border="1" width="250">
<tr><td colspan="2">Application 영역에 저장된 내용들</td></tr>
<tr>
```

```

<td>이름</td>
<td><%=application.getAttribute("name") %></td>
</tr>
<tr>
    <td>아이디</td>
    <td><%=application.getAttribute("id") %></td>
</tr>
</table>
<br>
<table border="1" width="250">
    <tr><td colspan="2">Session 영역에 저장된 내용들</td></tr>
<%
/*
session 영역에 "id"란 이름으로 등록된 속성을 읽어 들여 표현식으로 출력한다.
AttributeName이란 변수로 받아서 각 AttributeName의 이름을 가진 속성을getAttribute()메소드를
사용해 AttributeValue의 변수에 저장한 후 표현식으로 속성이름과 그 값을 출력한다
*/
Enumeration e = session.getAttributeNames();
while(e.hasMoreElements()){
    String attributeName = (String)e.nextElement();
    String attributeValue = (String)session.getAttribute(attributeName);
%>
    <tr>
        <td><%= attributeName %></td>
        <td><%= attributeValue %></td>
    </tr>
<%
}
%>
</table>
</body>
</html>

```

[결과]



application 영역에 저장하기 때문에 브라우저를 닫은 후 다시 attributeTest1.jsp 파일을 실행하여도 다음 화면처럼 이전에 저장한 저장한 이름과 아이디는 사라지지 않는다 (단 이클립스의 톰캣 서버를 재시작하면 저장된 내용은 사라진다.)

application와 session 영역에 속성으로 저장되어 있기 때문에 application 영역에 저장된 내용들은 서버가 종료되기 전까지는 사라지지 않고 session 영역에 저장된 내용은 브라우저가 종료되기 전까지는 사라지지 않는다. 다만 session 영역에서 저장된 내용들의 경우 세션 유효 시간이 지나면 브라우저가 종료되거나 않더라도 사라질 수 있다. page 영역의 경우 오직 하나의 JSP 페이지에서만 속성이 지속되기 때문에 실질적으로 사용할 일이 거의 없다고 볼 수 있고, request 영역에서 속성을 사용하는 것은 나중에 살펴보도록 하겠다.

9.8 서블릿 핵심 클래스

1) ServletConfig API를 활용한 초기화 파라미터

서블릿이 초기화될 때, 공통적으로 적용해야 되는 작업들이 필요 경우가 있다.(예;외부 파일 및 디렉터리 경로, JDBC에서 사용하기 위한 데이터베이스 경로, 계정 및 비밀번호와 같은 정보들) 이런 정보들을 서블릿에서 설정하지 않고 web.xml에서 설정한 후 서블릿에서 접근해서 사용한다.

서블릿에서 설정하는 경우에는 정보가 변경되면 반드시 서블릿을 재컴파일 시켜야 한다. 하지만 web.xml에서 설정하면 재컴파일 없이 변경된 정보를 참조할 수 있기 때문에 유지보수가 쉬워진다.

web.xml에 설정된 설정 값을 '초기화 파라미터(Initialization Parameter)'라고 하며 ServletConfig API를 이

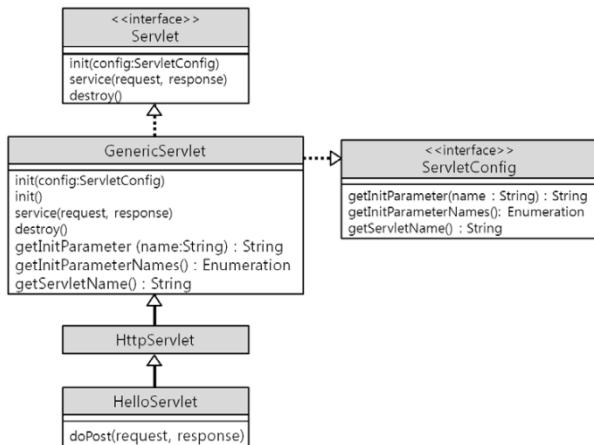
용해서 접근할 수 있다.

여러 서블릿에서 공유해서 사용하지 못하고 <init-param>으로 등록된 서블릿에서만 사용 가능하다.

서블릿 코드 내에서 @WebInitParam 어노테이션을 이용하여 초기화 파라미터를 등록할 수도 있다.

- ServletConfig API의 계층 구조

ServletConfig는 인터페이스로서 GenericServlet에서 구현했기 때문에 사용자가 작성한 서블릿에서 ServletConfig의 메서드를 제약없이 사용 가능하다.



- web.xml에 초기화 파라미터 등록

<servlet> 태그 안에서 <init-param> 태그를 사용하여 지정한다. 여러 개의 태그 등록이 가능하고 name, value 쌍으로 설정한다.

```
<servlet>
    <servlet-name>서블릿 별칭</servlet-name>
    <servlet-class>패키지를 포함한 서블릿명</servlet-class>
    <init-param>
        <param-name>초기화 파라미터 이름</param-name>
        <param-value>초기화 파라미터 값</param-value>
    </init-param>
    <init-param>
        <param-name>초기화 파라미터 이름</param-name>
        <param-value>초기화 파라미터 값</param-value>
    </init-param>
</servlet>
[예]
<servlet>
    <servlet-name>InitParam</servlet-name>
    <servlet-class>com.test.InitParamServlet</servlet-class>
    <init-param>
        <param-name>dirPath</param-name>
        <param-value>c:\wwwtest</param-value>
```

```

</init-param>
<init-param>
    <param-name>userid</param-name>
    <param-value>admin</param-value>
</init-param>
</servlet>

```

ServletConfig의 핵심 메소드

리턴 타입	메서드명	내용
String	String getInitParameter(name)	name에 해당되는 파라미터 값을 리턴한다. 만약 지정된 name의 파라미터 값이 없으면 null을 리턴 한다.
Enumeration	getInitParameterNames()	모든 초기화 파라미터 name 값을 Enumeration 타입으로 리턴한다. 얻은 name 값을 이용하여 초기화 파라미터 값을 얻는다.
String	getServletName()	요청한 서블릿의 이름을 리턴한다.

- web.xml 초기화 파라미터 설정과 서블릿에서 ServletConfig를 이용하여 참조하는 예제

InitParamServlet.java

```

package com.test;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class InitParamServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // 초기화 파라미터 얻기
        String dirPath = getInitParameter("dirPath");
        String userid = getInitParameter("userid");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("디렉터리 경로 :" + dirPath + "<br>");
        out.print("아이디 값 :" + userid + "<br>");
        out.print("</body></html>");
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws

```

```

ServletException, IOException {
    doPost(request, response);
}
}

```

web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <servlet>
        <servlet-name>InitParam</servlet-name>
        <servlet-class>com.test.InitParamServlet</servlet-class>
        <!-- 초기화 파라미터 설정 -->
        <init-param>
            <param-name>dirPath</param-name>
            <param-value>c:\WWWtest</param-value>
        </init-param>
        <init-param>
            <param-name>userid</param-name>
            <param-value>admin</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>InitParam</servlet-name>
        <url-pattern>/InitParam</url-pattern>
    </servlet-mapping>
</web-app>

```

@WebInitParam 어노테이션을 이용한 초기화 파라미터 등록

서블릿 코드내에서 @WebInitParam 어노테이션을 사용하여 초기화 파라미터를 등록할 수 있다.
ServletConfig의 getInitParameter(name) 메서드를 사용하여 초기화 파라미터 값을 얻는다.

InitParamAnnoServlet.java

```

package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet( name = "InitParamAnnoServlet" ,
    urlPatterns={"/initParamAnno"} ,
    initParams = { @WebInitParam ( name="dirPath", value = "c:WWtest") ,
                  @WebInitParam ( name="userid", value = "system") })
public class InitParamAnnoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        System.out.println(getInitParameter("dirPath"));
        System.out.println(getInitParameter("userid"));
    }
}

```

3) ServletContext API

웹 어플리케이션에는 여러 가지 자원을 포함할 수 있다. html 파일, 미디어 파일, 이미지 파일, 다수의 JSP 파일과 서블릿 등이 유기적으로 동작된다.

ServletContext는 웹 어플리케이션(Context)마다 하나씩 생성되는 객체로서, 다수의 JSP 파일과 서블릿에서 공유해서 사용할 수 있다. ServletContext 객체는 웹 어플리케이션의 LifeCycle과 일치하기 때문에, 웹 어플리케이션이 Tomcat 컨테이너에 존재한다면 계속 사용 가능하다. 이것을 'application scope'라고 한다.

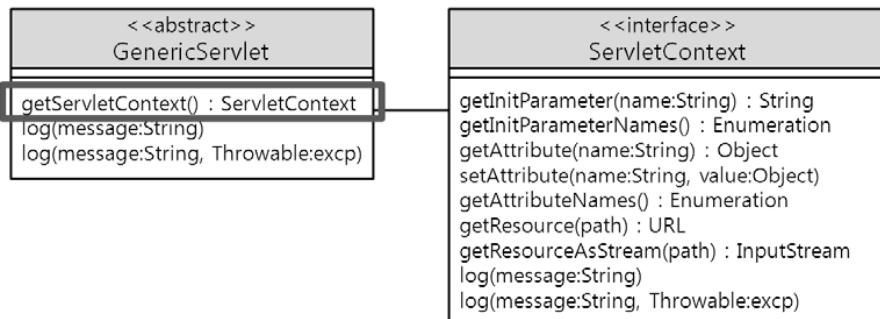
- ServletContext 객체를 이용한 핵심 기능

여러 서블릿에서 사용 가능한 초기화 파라미터 사용할 수 있고 일반적으로 '컨텍스트 파라미터(Context Parameter)'라고 한다. 서블릿에서 파일 접근 가능(읽기 모드만 가능)하다.

application scope에 해당되는 속성(Attribute)을 저장하고 조회할 수 있다.

- ServletContext API의 계층 구조

GenericServlet을 HttpServlet이 상속받았기 때문에 ServletContext 객체를 얻기 위해서 사용자가 생성한 서블릿에서는 getServletContext() 메소드를 사용한다.



- ServletContext의 핵심 메소드

리턴 타입	메소드명	내용
-------	------	----

String	getInitParameter(name)	name에 해당되는 컨텍스트 파라미터 값을 리턴한다. 만약 지정된 name의 파라미터 값이 없으면 null을 리턴한다.
InputStream	getResourceAsStream(path)	웹 어플리케이션의 path 경로에 해당되는 파일을 읽기 모드로 접근 가능하다.
void	setAttribute(name,value)	application scope 해당되는 속성 값을 저장할 때 사용 한다. 브라우저를 종료해도 속성 값을 사용 가능하다.
Object	getAttribute(name)	name에 해당되는 속성 값을 리턴한다.

Context Parameter 설정

다수의 서블릿이 공통적으로 사용되는 특정 데이터가 필요하다면, 컨텍스트 파라미터를 사용하는 것이 바람직하다. 초기화 파라미터는 특정 서블릿만이 파라미터 값을 사용 가능할 수 있지만 컨텍스트 파라미터는 application scope이기 때문에 어플리케이션의 모든 서블릿이 공유해서 사용 가능하다.

초기화 파라미터와 마찬가지로 web.xml에 등록하여 사용하고 ServletConfig 대신에 ServletContext 객체의 getInitParameter(name) 메서드를 사용해서 컨텍스트 파라미터 값을 얻는다.

- 예제

ContextParamServlet.java

```
package com.test2;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ContextParam")
public class ContextParamServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // 컨텍스트 파라미터 얻기
        String driver = getServletContext().getInitParameter("driver");
        String savePath = getServletContext().getInitParameter("savePath");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("드라이버명: " + driver + "<br>");
        out.print("저장 경로: " + savePath + "<br>");
        out.print("</body></html>");
    }
}
```

```
    }  
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xmlns="http://java.sun.com/xml/ns/javaee"  
      xmlns:web="http://java.sun.com/xml/ns/javaee/web-  
      app_2_5.xsd"  
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
      http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">  
    <!-- 컨텍스트 파라미터 설정 -->  
    <context-param>  
      <param-name>driver</param-name>  
      <param-value>oracle.jdbc.driver.OracleDriver</param-value>  
    </context-param>  
    <context-param>  
      <param-name>savePath</param-name>  
      <param-value>c:\WWsave</param-value>  
    </context-param>  
    <servlet>  
      <servlet-name>InitParam</servlet-name>  
      <servlet-class>com.test.InitParamServlet</servlet-class>  
      <!-- 초기화 파라미터 설정 -->  
      <init-param>  
        <param-name>dirPath</param-name>  
        <param-value>c:\WWtest</param-value>  
      </init-param>  
      <init-param>  
        <param-name>userid</param-name>  
        <param-value>admin</param-value>  
      </init-param>  
    </servlet>  
    <servlet-mapping>  
      <servlet-name>InitParam</servlet-name>  
      <url-pattern>/InitParam</url-pattern>  
    </servlet-mapping>  
</web-app>
```

서블릿에서 파일 접근(읽기 모드)

서블릿에서 웹 어플리케이션내의 특정 파일을 접근하기 위해서 ServletContext 객체를 사용 가능하다.

읽기 모드만 가능하고 쓰기는 불가능 하다.

- 파일 접근 예제

testFile.txt 파일을 WEB-INF 폴더에 저장한다.

ContextFileServlet.java

```
package com.test2;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
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;
@WebServlet("/ContextFile")
public class ContextFileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String readFile = "/WEB-INF/testFile.txt";
        InputStream is = getServletContext().getResourceAsStream(readFile);
        BufferedReader reader = new BufferedReader(new InputStreamReader(is));
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html> <body>");
        String str = reader.readLine();
        while(str != null){
            out.print(str + "<br>");
            str = reader.readLine();
        }
        reader.close();
        out.print("</body> </html>");
    }
}
```

4) 서블릿에서 attribute 설정 및 참조

웹 어플리케이션에서 브라우저를 종료해도 지속적으로 사용해야 되는 데이터가 필요하다면, application scope에 해당되는 속성(attribute)을 사용할 수 있다.

대표적인 예로 '웹 사이트의 방문자수 조회' 형태로서, 웹 어플리케이션을 종료할 때까지 속성 값은 유지

된다.

- 속성 설정 및 참조 예제

브라우저를 종료하고 다시 접속해도 계속 유지되는 데이터를 ServletContext 객체의 setAttribute(name, value) 메소드로 저장하고, getAttribute(name) 메소드로 참조한다.

ContextSetServlet.java 서블릿을 먼저 실행하고 브라우저를 닫는다.

```
package com.test2;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ContextSet")
public class ContextSetServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        // 속성값 설정
        String name = "홍길동";
        int age = 20;
        getServletContext().setAttribute("name", name );
        getServletContext().setAttribute("age", age );
    }
}
```

ContextGetServlet.java

```
package com.test2;
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;
@WebServlet("/ContextGet")
public class ContextGetServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        //속성값 얻기
        String name = (String)getServletContext().getAttribute("name");
    }
}
```

```

int age = (Integer)getServletContext().getAttribute("age");
response.setContentType("text/html; charset=UTF-8");
PrintWriter out = response.getWriter();
out.print("<html><body>");
out.print("이름 : " + name + "<br>");
out.print("나이 : " + age + "<br>");
out.print("</body></html>");
}
}

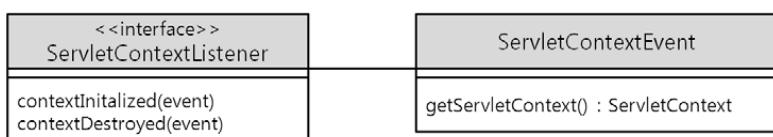
```

5) ServletContextListener API

서블릿이 LifeCycle를 가지고 있는 것처럼, 웹 어플리케이션도 LifeCycle을 갖는다. Tomcat 컨테이너가 시작될 때 웹 어플리케이션도 초기화되고, Tomcat 컨테이너가 종료될 때 웹 어플리케이션도 제거된다. 웹 어플리케이션이 초기화되고 제거되는 이벤트를 감지하는 ServletContextListener API를 사용하면, 언제 초기화되고 제거되었는지를 쉽게 알 수 있다.

이 이벤트는 JDBC의 Pooling기법에 적용 가능하다. 웹 어플리케이션이 초기화될 때 Pooling을 활성화하고 제거될 때 Pooling을 비활성화 시키면 효율적으로 Connection을 관리할 수 있다.(JDBC에서 자세히)

- ServletContextListener API 계층 구조



- 예제 실습 순서

- ① ServletContextListener 인터페이스를 구현하는 클래스를 작성한다.
- ② web.xml에 구현한 클래스를 <listener> 태그로 등록 또는 @WebListener 어노테이션을 이용한다.
- ③ Tomcat 컨테이너를 시작하고 종료하는 작업을 실행하여 이벤트 감지를 확인한다.

ContextListenerImpl.java

```

package com.test2;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
public class ContextListenerImpl implements ServletContextListener {
    @Override
    public void contextDestroyed(ServletContextEvent event) {
        System.out.println("웹 어플리케이션 제거");
    }
    @Override
    public void contextInitialized(ServletContextEvent event) {

```

```

        System.out.println("웹 어플리케이션 초기화");
    }
}

```

web.xml에 <listener> 등록

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"           xmlns:web="http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd"                                         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <!-- 컨텍스트 파라미터 설정 -->
    <context-param>
        <param-name>driver</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
    </context-param>
    <context-param>
        <param-name>savePath</param-name>
        <param-value>c:\WWsave</param-value>
    </context-param>
    <!-- Listener 설정 -->
    <listener>
        <listener-class>com.test2.ContextListenerImpl</listener-class>
    </listener>
    <servlet>
        <servlet-name>InitParam</servlet-name>
        <servlet-class>com.test.InitParamServlet</servlet-class>
        <!-- 초기화 파라미터 설정 -->
        <init-param>
            <param-name>dirPath</param-name>
            <param-value>c:\WWtest</param-value>
        </init-param>
        <init-param>
            <param-name>userid</param-name>
            <param-value>admin</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>InitParam</servlet-name>
        <url-pattern>/InitParam</url-pattern>
    </servlet-mapping>

```

```
</web-app>
```

console 창에서 이벤트 감지 확인하고 서버를 종료하고 이벤트 감지 확인한다.

@WebListener 애노테이션으로 등록하는 방법

web.xml에 등록한 <listener> 태그를 제거하거나 주석 처리하고 Context ListenerImpl 클래스에 @WebListener 어노테이션을 추가하고 테스트한다.

ContextListenerImpl.java

```
package com.test2;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
@WebListener
public class ContextListenerImpl implements ServletContextListener {
    @Override
    public void contextDestroyed(ServletContextEvent event) {
        System.out.println("웹 어플리케이션 제거");
    }
    @Override
    public void contextInitialized(ServletContextEvent event) {
        System.out.println("웹 어플리케이션 초기화");
    }
}
```

5) Filter API

클라이언트인 웹 브라우저에서 서블릿으로 요청하면, 웹 컴포넌트인 서블릿이 요청을 받아서 작업을 처리하고 결과를 HTML 형식으로 작성하여 웹 브라우저에게 응답 처리한다.

서블릿이 요청 받기 전과 결과를 웹 브라우저에게 응답하기 전에 특정 작업을 수행할 수 있도록 Filter API를 사용할 수 있다. 웹 컴포넌트가 실행되기 전의 선처리(pre-processing) 작업과 응답되기 전의 후처리(post-processing) 작업을 수행하는 API이다.

다수의 Filter를 체인(Chain)처럼 묶어서 적용시킬 수도 있다.

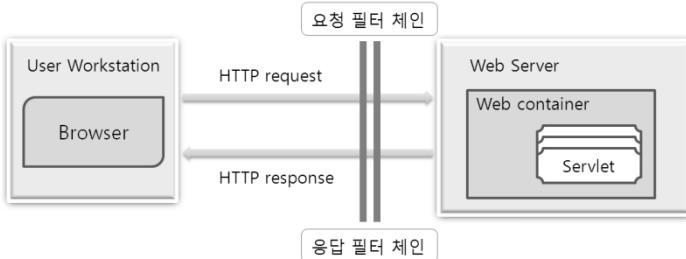
선처리 작업의 필터 → 요청 필터(Request Filter)

후처리 작업의 필터 → 응답 필터(Response Filter)

선처리 작업의 적용 예 : 한글인코딩 및 보안관련 작업 등

후처리 작업의 적용 예 : 압축 및 데이터 변환 작업 등

Filter를 적용한 아키텍처



request 요청이 요청 필터 체인을 거쳐서 서블릿으로 전송된다. 서블릿이 처리하기 전에 실행되어야 하는 선처리 작업 수행할 수 있다. response 응답이 응답 필터 체인을 거쳐서 전송된다. 요청과 마찬가지로 웹 브라우저로 응답되기 전에 실행되어야 하는 후처리 작업 수행할 수 있다.

Filter를 적용하기 위한 실습 순서

- ① Filter 인터페이스를 구현한 클래스를 작성한다.
- ② web.xml에 <filter> 태그를 사용하여 등록하거나, @WebFilter 어노테이션을 사용하여 등록한다.
- ③ 웹 서버에 요청하여 Filter가 적용되었는지를 확인한다.

- web.xml에 <filter> 태그로 등록하는 방법

웹 어플리케이션의 서블릿이 요청을 처리하기 전의 선처리(pre-processing) 작업과 후처리(post-processing) 작업을 처리하는 Filter 예제이다. 코드를 간단하게 하기 위해서 console 창에 문자열을 출력

MyFilter.java

```
package com.test2;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class MyFilter implements Filter {
    public void init(FilterConfig fConfig) throws ServletException {
        System.out.println("MyFilter 초기화");
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        System.out.println("MyFilter 요청필터 코드작업");
        request.setCharacterEncoding("UTF-8");
        chain.doFilter(request, response);
        System.out.println("MyFilter 응답필터 코드작업");
    }
    public void destroy() {
```

```

        System.out.println("MyFilter 제거");
    }
}

```

web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"           xmlns:web="http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd"                                         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <!-- 컨텍스트 파라미터 설정 -->
    <context-param>
        <param-name>driver</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
    </context-param>
    <context-param>
        <param-name>savePath</param-name>
        <param-value>c:\WWsave</param-value>
    </context-param>
    <!-- Filter 설정 -->
    <filter>
        <filter-name>myFilter</filter-name>
        <filter-class>com.test2.MyFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>myFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <servlet>
        <servlet-name>InitParam</servlet-name>
        <servlet-class>com.test.InitParamServlet</servlet-class>
        <!-- 초기화 파라미터 설정 -->
        <init-param>
            <param-name>dirPath</param-name>
            <param-value>c:\WWtest</param-value>
        </init-param>
        <init-param>
            <param-name>userid</param-name>
            <param-value>admin</param-value>
        </init-param>

```

```

</servlet>
<servlet-mapping>
    <servlet-name>InitParam</servlet-name>
    <url-pattern>/InitParam</url-pattern>
</servlet-mapping>
</web-app>

```

@WebFilter 애노테이션을 사용하면 web.xml에서 <filter>관련 태그를 주석처리한다.

MyFilter.java

```

package com.test2;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
@WebFilter(urlPatterns={ "/"})
public class MyFilter implements Filter {
    public void init(FilterConfig fConfig) throws ServletException {
        System.out.println("MyFilter 초기화");
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        System.out.println("MyFilter 요청필터 코드작업");
        request.setCharacterEncoding("UTF-8");
        chain.doFilter(request, response);
        System.out.println("MyFilter 응답필터 코드작업");
    }
    public void destroy() {
        System.out.println("MyFilter 제거");
    }
}

```

6) url-pattern의 개요

서블릿 맵핑과 필터 맵핑은 web.xml 파일 및 어노테이션을 사용하여 설정한다. 이때 공통적으로 사용하는 값이 url-pattern이다.

url-pattern은 웹 브라우저에서 클라이언트가 요청하는 URL 값의 패턴에 따라서, 서버의 어떤 웹 컴포넌트가 실행될지를 결정하는 방법이다.

① 디렉터리 패턴

디렉토리 패턴과 일치하는 형태의 url-pattern을 지정한 웹 컴포넌트가 수행

이때, url-pattern 값은 반드시 '/디렉토리 패턴 값' 형식으로 지정해야 되며, 계층 구조를 가질 수 있음

만약에 '/*' 형식으로 지정하면, 클라이언트가 요청하는 URL 값의 패턴과 무관하게 항상 수행

다음은 웹 컨테이너에 4개의 서블릿을 작성하고 각각 url-pattern 값을 지정한 후에, 웹 브라우저에서 특정 패턴을 사용하여 요청한 경우

http://localhost/ServletTest/ATest	ATestServlet 서블릿 /ATest
http://localhost/ServletTest/test/BTest	BTestServlet 서블릿 /test/BTest
http://localhost/ServletTest/test	CTestServlet 서블릿 /test
http://localhost/ServletTest/ http://localhost/ServletTest/xxx	ATestServlet 서블릿 /*

ATestServlet.java

```
package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ATest")
// @WebServlet("/test/BTest")    클래스명 BTestServlet
// @WebServlet("/test")    클래스명 CTestServlet
// @WebServlet("/*")    클래스명 DTestServlet
public class ATestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        System.out.println("디렉토리 패턴");
    }
}
```

② 확장자 패턴

일치하는 확장자 형태의 url-pattern을 지정한 웹 컴포넌트가 수행

url-pattern 값은 반드시 '*.확장자' 형식으로 지정해야 됨

주의할 점은 '/'를 사용하면 안되며 Spring, Struts, Struts2 프레임워크 같은 다양한 웹 프레임워크에서 많이 사용되는 패턴

다음은 웹 컨테이너에 3개의 서블릿을 작성하고 각각 url-pattern 값을 지정한 후에, 웹 브라우저에서 특정 패턴을 사용하여 요청한 경우이며, /test, /xyz 또는 /my 같은 경로 값 설정과 무관하게 요청

http://localhost/ServletTest/xxx.do http://localhost/ServletTest/xyz/a.do	ETestServlet 서블릿 *.do
http://localhost/ServletTest/my/ss.go http://localhost/ServletTest/kk.go	FTestServlet 서블릿 *.go
http://localhost/ServletTest/test/a.nhn http://localhost/ServletTest/xyz.nhn	GTestServlet 서블릿 *.nhn

ETestServlet.java

```

package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("*.do")
// @WebServlet("*.go")    클래스명 FTestServlet
// @WebServlet("*.nhn")   클래스명 GTestServlet
public class ETestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        System.out.println("ETestServlet");
    }
}

```

필터를 이용한 문자 인코딩 설정

web.xml의 일부분

```

<!-- Filter 문자 인코딩 설정 -->
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>com.filters.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

위 처럼 설정하면 `request.setCharacterEncoding("UTF-8");` 을 사용할 필요가 없다.

애노테이션을 이용한 필터 배치

```
@WebFilter( urlPatterns = "*",
    initParams={ @WebInitParam(name="encoding", value="UTF-8") })
```

10. 액션 태그 개요

액션 태그란 JSP 페이지에서 자바 코드 등의 스트립트 언어를 사용하지 않고도 다른 페이지의 서블릿이나 자바빈의 객체에 접근할 수 있도록 태그를 이용해 구현된 기능을 말한다. 액션 태그를 통해서 개발자는 페이지의 흐름을 제어하거나 자바빈의 속성을 읽고 쓰며 애플릿을 사용하는 등의 다양한 기능을 활용할 수 있다.

또한 이러한 기능들은 스크립트릿 등의 스크립트 요소(자바 코드)를 사용하지 않기 때문에 개발자는 JSP 페이지의 내부적인 프로그램 로직을 사용자로부터 감출 수가 있다. 이것은 즉 액션 태그를 사용하면 사용자에게 보여지는 프리젠테이션 부분과 사용자의 요청을 처리하는 비즈니스 로직 부분(프로그램 부분)을 분리하는 것이 가능하다는 것을 의미하며, 웹 프로그램에 있어서 이러한 프리젠테이션 부분과 비즈니스 로직 부분의 분리는 매우 중요한 의미를 가진다.

- JSP에서 제공하는 액션 태그의 종류

페이지 흐름 제어 액션(forward/include 액션)

자바빈 사용 액션(useBean 액션)

애플릿 사용 액션(plugin 액션)

10.1 액션 태그의 유형

- XML 스타일 태그

JSP 액션 태그는 XML 스타일의 태그로 기술하며 특정한 동작 기능을 수행한다. JSP 액션 태그는 다음과 같이 <와 접두어 `jsp:` 그리고 `forward`, `include`, `param`과 같은 고유한 태그 키워드로 구성된 `<jsp:forward>` 와 같은 시작 태그로 시작하고, 속성값을 지정하며, 마지막 종료 태그는 `>`로 종료한다.

- 단독으로 태그가 쓰여졌을 경우

```
<jsp:태그키워드 태그속성="태그값" />
<jsp:include page="sub.jsp" />
```

액션 태그에서 매개변수지정과 같은 내용이 있다며, 다음과 같이 시작 태그 `<jsp:태그키워드 ...>`와 종료 태그 `</jsp:태그키워드>` 사이에 `<jsp:param ... />`과 같은 `param` 태그를 기술한다. 여기서 주의할 점은 시작 태그 `<jsp:태그키워드 ...>`에서 마지막 종료 태그인 `>`이 아니라 `>`라는 것이다.

```
<jsp:태그키워드 태그속성="태그값">
    매개변수 지정과 같은 다른 내용
</jsp:태그키워드>

<jsp:include page="includesub.jsp" >
    <jsp:param name="week" value="52" />
</jsp:include>
```

10.2 액션 태그의 종류

- 태그 종류 `forward`, `include`, `param`, `plugin`

JSP 액션 태그는 `<jsp:forward ... />`, `<jsp: include ... />`, `<jsp: param ... />`, `<jsp: plugin ... />` 등이 있다. 자바빈즈를 활용하는 액션 태그로 `<jsp:useBean ... />`, `<jsp:setProperty ... />`, `<jsp:getProperty ... />` 등이

있으며 이와 같은 자바 빈즈 활용을 앞으로 배울 예정이다.

태그 종류	태그형식	사용 용도
include param	<jsp:include page="test.jsp" /> <jsp:include page="test.jsp"> <jsp:param name="id" value="hong" /> </jsp:include>	현재 JSP 페이지에서 다른 페이지를 포함.
forward param	<jsp:forward page="test.jsp" /> <jsp:forward page="test.jsp" /> <jsp:param name="id" value="hong" /> </jsp:forward>	현재 JSP 페이지의 제어를 다른 페이지에 전달
plugin	<jsp:plugin type="applet" code="test" />	자바 애플릿을 플러그인
useBean	<jsp:useBean id="login" class="LoginBean" />	자바빈을 사용
setProperty	<jsp:setProperty name="login" property="pass" />	자바빈의 속성을 지정하는 메소드를 호출
getProperty	<jsp:getProperty name="login" property="pass" />	자바빈의 속성을 반환하는 메소드를 호출

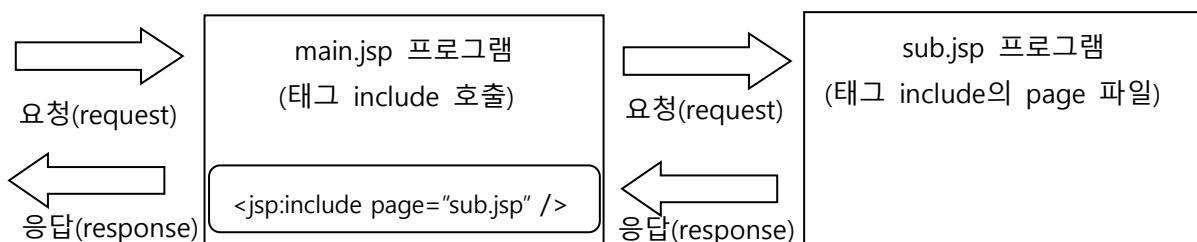
10.3 액션 태그 include

- 태그 정의 <jsp:include page="filename" />

include 액션은 임시로 제어권을 include되는 페이지 즉 page로 설정된 페이지로 넘겼다가 그 페이지의 처리가 끝나면 다시 원래의 페이지로 제어권을 반환하는 방식이다.

```
<jsp:include page="포함될 페이지" flush="false" />
```

flush 속성은 포함될 페이지로 제어가 이동될 때 현재 포함하는 페이지가 지금까지 출력 버퍼에 저장한 결과를 처리하는 방법을 결정하는 것으로, flush 속성의 값이 true로 지정하면 포함될 페이지로 이동될 때 현재 페이지가 지금까지 버퍼에 저장한 내용을 웹 브라우저에 출력하고 버퍼를 비운다. (웹 브라우저에 전송할 때 헤더 정보도 같이 전송된다. 일단 헤더 정보가 웹 브라우저에 전송이 되고 나면 헤더 정보를 추가해도 결과가 반영되지 않는다.)



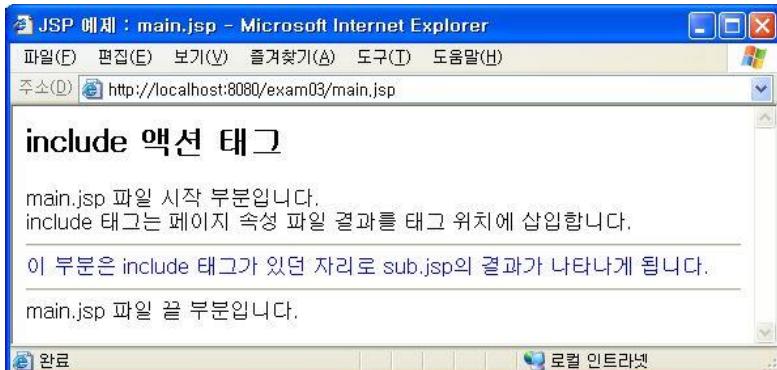
main.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP 예제 : main.jsp</title>
</head>
<body>
    <h2> include 액션 태그 </h2>
    main.jsp 파일 시작 부분입니다.<br>
    include 태그는 페이지 속성 파일 결과를 태그 위치에 삽입합니다.<br>
    <jsp:include page="sub.jsp" />
    main.jsp 파일 끝 부분입니다.
</body>
</html>

```

[결과]



sub.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : sub.jsp</title>
</head>
<body>
    <hr><font color=blue>
    이 부분은 include 태그가 있던 자리로 sub.jsp의 결과가 나타나게 됩니다.
    </font><hr>
</body>
</html>

```

- 지시자 include와 액션 태그 include

지시자 include와 액션 태그 include의 차이를 알아보자. 지시자 include의 경우 원래 페이지 안으로 include 지시어로 지정한 페이지의 소스가 그대로 복사되어 들어가는데 반해 include 액션은 소스코드가

복사되는 것이 아니라 제어권 자체가 include 액션으로 지정된 페이지로 넘어갔다가 다시 원래 페이지로 돌아온다는 점이 틀리다.

include 지시어의 경우에는 컨테이너의 버전에 따라서 원래 페이지의 서블릿이 생성된 이후에 include 되는 페이지가 변동되었을 경우 그 변동을 원래 페이지가 반영하지 못하는 경우도 있다 따라서 **include 지시어는 일반적으로 정적인 페이지를 포함시킬 때 주로 사용하고 include 액션은 JSP 페이지처럼 동적인 페이지를 포함시키고자 할 때 주로 사용된다.**

이렇게 include 액션은 여러 페이지를 동적으로 하나의 페이지로 묶을 수 있으므로 각각의 페이지를 기능별로 모듈화 시켜서 하나의 페이지를 여러 모듈화된 페이지의 집합으로 표현하는 것이 가능하며 이러한 페이지를 템플릿 페이지라고 한다.

지시자 include는 소스 코드 형태로 삽입하므로 중복된 소스가 있는 경우 주의가 필요하다. 특히 변수의 선언이 중복되면 오류가 발생한다. 다음과 같이 지시자 include가 있는 페이지 includedirective.jsp에 변수 i와 n을 선언되었다고 가정하자.

```
<% int i = 12; %>
<% int n = 365; %>
<%@ include file="includesub.jsp" %>
```

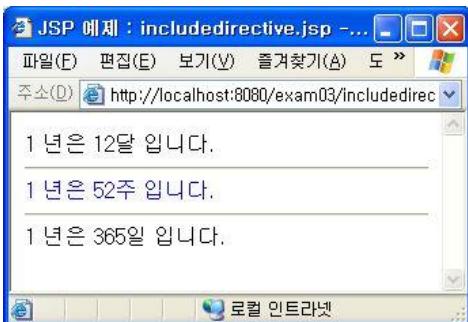
소스가 삽입되는 페이지 includesub.jsp에 다음과 같이 변수 n이 다시 선언되었다면 파일 includedirective.jsp에서 오류가 발생한다. 페이지 includesub.jsp 자체는 문제가 없으나 이 파일을 삽입하는 includedirective.jsp에 이름이 n인 변수를 선언하므로 중복된 지역변수 선언의 컴파일 오류가 발생한다.

```
<% int n = 52; %>
```

includedirective.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : includedirective.jsp</title>
</head>
<body>
    <% int i = 12; %>
    <% //int n = 365; %>
    <% int days = 365; %>
    1년은 <%=i %> 달입니다.
    <%@ include file="includesub.jsp" %>
    1년은 <%=days %> 일입니다.
</body>
</html>
```

[결과]



includesub.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : includesub.jsp </title>
</head>
<body>
    <% int n = 52; %>
    <hr><font color="blue">
        1년은 <%=n %>주 입니다.
    </font><hr>
</body>
</html>
```

위 예제 includedirective.jsp에서 변수 n 선언 문장에서 주석을 빼면 지시자에서 지역변수 n의 중복 선언 오류가 발생하는 것을 확인할 수 있다

```
12  <%@ include file="includesub.jsp" %>
13  1년은 <%=days %>일입니다.
```

지시자 include와 다르게 액션 태그 include를 이용했을 경우는 제어권 자체가 page로 설정한 페이지로 이동하기 때문에 지역 변수 중복 선언의 문제가 발생하지 않는다.

includeaction.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : includeaction.jsp </title>
</head>
<body>
    <% int i = 12; %>
    <% int n = 365; %>
    1년은 <%=i %>달 입니다.
    <jsp:include page="includesub.jsp" />
```

```
1년은 <%=n %>일입니다.
```

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

액션 태그 `<jsp:include ... />` 는 내장 객체 `pageContext`의 메소드 `include()`와 같은 기능을 수행한다.

```
<% pageContext.include(includeSub.jsp); %>  
<jsp:include page="includeSub.jsp" />
```

- include �렉티브의 활용

일반적으로 `<jsp:include />` 액션 태그는 레이아웃의 한 구성 요소를 모듈화하기 위해 사용되는 반면에, `include` 딕렉티브는 다음과 같이 두 가지 형태로 주로 사용된다

- 모든 JSP 페이지에서 사용되는 변수 지정
- 저작권 표시와 같은 간단하면서도 모든 페이지에서 중복되는 문장

그래서 `include` 딕렉티브를 사용하면 편리하게 공용 변수를 선언할 수 있다. 예를 들어 웹 애플리케이션을 구성하는 대다수의 JSP 페이지가 `session` 내장 객체에 저장된 속성값을 읽어와 사용한다고 가정하면 JSP 페이지 앞 부분에서 속성값을 읽어와 변수에 저장하는 코드를 추가하게 될 것이다. 이때 변수를 지정하는 부분을 별도의 파일에 작성한 후 그 파일을 `include` 딕렉티브로 포함시키는 것이 더 좋은 방법이다.

`include` 딕렉티브를 사용해서 포함하는 파일의 경우 일반 JSP 파일과 구분하기 위해서 확장자로 `.jspf`를 사용한다. 여기서 `.jspf`는 JSP Fragment 즉 JSP의 소스 코드 조각을 의미한다.

- 코드 조각 자동 포함 기능

JSP 2.0 버전부터 `include` 딕렉티브를 사용하지 않고도 JSP의 앞뒤에 지정한 파일을 삽입하는 기능을 제공하고 있다.

```
<%@ page contentType="text/html; charset=UTF-8" %>  
<%@ include file="/common/variable.jspf" %>  
<!DOCTYPE html>  
<html>  
...  
<%@ include file="/common/footer.jspf" %>  
</html>
```

다수의 JSP 페이지에서 앞 뒤로 같은 파일을 `include` 딕렉티브를 사용해서 삽입할 경우 여러 JSP에서 중복된 코드를 작성해 주어야 할 것이다. 중복되는 코드가 많다면 WEB-INF/web.xml 파일에 다음과 설정 정보를 추가해 줌으로써 코드 중복을 방지 할 수 있다.

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

```

http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
<jsp-config>
    <jsp-property-group>
        <url-pattern>/view/*</url-pattern>
        <include-prelude>/common/variable.jspf</include-prelude>
        <include-coda>/common/footer.jspf</include-coda>
    </jsp-property-group>
</jsp-config>
</web-app>

```

태그	설명
jsp-property-group	JSP의 프로퍼티를 지정함을 나타낸다.
url-pattern	프로퍼티를 적용할 JSP 파일에 해당하는 URL 패턴을 지정한다.
include-prelude	url-pattern 태그에 지정한 패턴에 해당하는 JSP 파일의 앞에 자동으로 삽입될 파일을 지정한다.
include-coda	url-pattern 태그에 지정한 패턴에 해당하는 JSP 파일의 뒤에 자동으로 삽입될 파일을 지정한다.

/common/variable.jspf

```

<%@ page contentType = "text/html; charset=UTF-8" %>
<%
    java.util.Date CURRENT_TIME = new java.util.Date();
%>

```

/common/footer.jspf

```

<%@ page contentType = "text/html; charset=UTF-8" %>
<div>Copyright © JSP CORPORATION ALL RIGHTS RESERVED.</div>

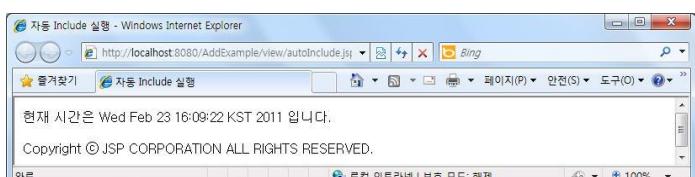
```

/view/autoInclude.jsp

```

<%@ page contentType = "text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html><head><meta charset="UTF-8"><title>자동 Include 실행</title></head>
<body>
    <p>현재 시간은 <%= CURRENT_TIME %> 입니다.</p>
</body>
</html>

```



10.4 액션 태그 forward

- 태그 정의 <jsp:forward page="filename" />

지시자 include와 다르게 액션 태그 include를 이용했을 경우는 결과 값이 포함되기 때문에 이러한 지역 변수 중복 선언의 문제가 발생하지 않는다.

```
<jsp:forward page="forwardsub.jsp" />
```

태그 forward에서 지정한 페이지를 호출하면 forward 태그가 있는 현재 페이지의 작업은 모두 중지되고, 이전에 출력한 버퍼링 내용도 모두 사라지게 되어 출력이 되지 않으며, 모든 제어가 page에 지정한 파일로 이동한다.

forward 액션은 현재 페이지의 요청과 응답에 관한 제어권을 page 속성에 지정된 이동할 페이지로 영구적으로 넘기는 기능을 한다. 이 때 이동하기 전의 페이지에 대한 모든 출력 버퍼의 내용은 무시(버퍼의 내용이 버려짐)되며 이동한 페이지가 요청을 처리하여 응답이 완료되면 원래 페이지로 제어권이 돌아가지 않고 그 상태에서 모든 응답이 종료된다.

요청과 응답에 관한 처리권이 넘어간다는 것은 원래 페이지에 의해 생성된 request 객체와 response 객체가 그대로 넘어간다는 것을 의미한다. 따라서 사용자가 request에 지정한 속성들은 포워딩된 페이지에서도 그대로 사용할 수 있다.

forward 액션의 page 속에서 지정되는 이동할 페이지의 주소는 동일한 웹 어플리케이션의 컨텍스트 루트를 기준으로 한 절대경로나 상대경로로만 지정이 가능하다. 즉 page 속성을 지정할 때는 프로토콜(http://)부분과 도메인(localhost), 그리고 포트 번호를 생략해야 한다.

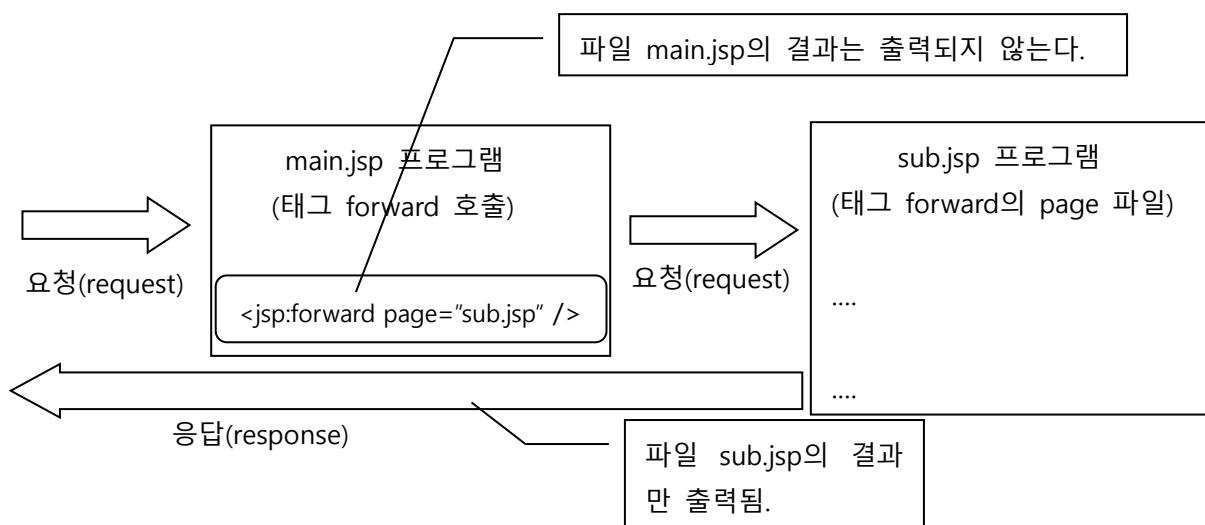
예를 들어 현재 페이지의 브라우저 주소창 경로가 http://localhost/study/test/forwardTest.jsp이고 포워딩 할 때 페이지의 경로가 http://localhost/study/forward/forward.jsp라면 다음과 같은 두 가지의 지정만을 유효하다.

```
<jsp:forward page="/test/forward/forward.jsp" />
```

해당 웹 어플리케이션의 컨텍스트 루트(/)의 절대 경로를 기준으로 페이지를 지정한 것이다.

```
<jsp:forward page="forward/forward.jsp" />
```

현재 페이지의 경로(/test)를 기준으로 상대경로로써 포워딩할 페이지를 지정한 것이다.



- 태그 forward와 include의 차이

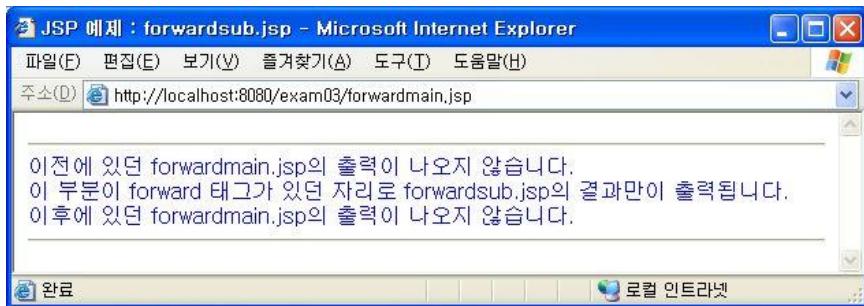
태그 `include`는 `page`에 지정된 페이지의 처리가 끝나면 다시 현재 페이지로 돌아와 처리를 진행해 나
가지만, 태그 `forward`는 `page` 속성에 지정된 페이지로 제어가 넘어가면 현재 페이지로 다시 돌아오지
않고 이동된 페이지에서 실행을 종료한다.

다음 예제 forwardmain.jsp 결과를 살펴보면 forwardmain.jsp의 자체에서 출력한 내용은 전혀 출력되지 않고 이동된 페이지 forwardsub.jsp의 내용만 출력되는 것을 볼 수 있다. 그러나 브라우저의 주소 부분에는 실행한 forwardmain.jsp 파일만이 표시되며, 브라우저의 캡션 부분에는 이동된 페이지인 forwardmain.jsp의 태그 `<title>`에 기술된 파일이름이 출력되고 있는 것을 볼 수 있다.

예제 forwardsub.jsp의 자체 실행 결과도 forwardmain.jsp의 실행 결과와 같다. 다만 예제 forwardsub.jsp의 실행에서는 브라우저의 주소가 forwardsub.jsp로 표시된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : forwardmain.jsp</title>
</head>
<body>
    <h2> forward 액션 태그 </h2>
    forwardmain.jsp 파일 시작 부분입니다.<br>
    forward 태그는 페이지 속성 파일로 제어를 넘깁니다.<br>
    forwardmain.jsp 페이지의 출력 내용은 하나도 출력되지 않습니다.<br>
    <jsp:forward page="forwardsub.jsp" />
    forwardmain.jsp 파일 끝 부분입니다.
</body>
</html>
```

[결과]



forwardsub.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : forwardsub.jsp</title>
```

```

</head>
<body>
    <hr><font color="blue">
        이전에 있던 forwardmain.jsp의 출력이 나오지 않습니다.<br>
        이 부분이 forward 태그가 있던 자리로 forwardsub.jsp의 결과만이 출력됩니다.<br>
        이후에 있던 forwardmain.jsp의 출력이 나오지 않습니다.
    </font><hr>
</body>
</html>

```

- 태그 forward와 같은 기능의 메소드 forward()

액션 태그 forward는 실제 JSP 서블릿 소스에서 내장 객체 pageContext의 메소드 forward()로 대체된다. 즉 다음 자바 소스 pageContext.forward()와 태그 <jsp:forward ... />는 같은 기능을 수행한다.

```

<% pageContext.forward("send.jsp"); %>
<jsp:forward page="send.jsp" />

```

다음 예제는 forward.html, forward.jsp, send.jsp로 구성되는 예제로 forward.html을 실행하여 접속할 사이트를 입력한 후 버튼을 누르면 지정한 사이트에 접속하는 기능을 수행한다.

forward.html

```

<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : forward.html</title>
</head>
<body>
    <h2>접속할 사이트를 입력하세요.</h2>
    <form method="post" name="test" action="forward.jsp" >
        URL : <input type="text" name="url">
        <input type="submit" value="보내기">
    </form>
</body>
</html>

```

forward.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : forward.jsp</title>
</head>
<body>
    <%

```

```

//pageContext.forward("send.jsp");
%>
<jsp:forward page="send.jsp" />
</body>
</html>

```

send.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : send.jsp</title>
</head>
<body>
    <% response.sendRedirect("http://" + request.getParameter("url")); %>
</body>
</html>

```

[결과]



태그 forward가 있는 JSP 서블릿 소스 [forward.jsp.java]를 살펴보면 다음과 같이 태그 forward가 내장 객체 pageContext의 메소드 forward()로 대체된 것을 확인할 수 있다.

```
_jspx_page_context.forward("send.jsp");
```

10.5 액션 태그 param

- **param** 태그 개요(파라미터 태그 `<jsp:param ... />`)

태그 param은 태그 `<jsp:param ... />` 와 함께 사용되어 page에 지정된 페이지로 필요한 파라미터의 이름(name)과 값(value)을 전송하는 역할을 수행한다.

그러므로 태그 param은 `<jsp:param name="param_name" value="param_value" />`와 같이 속성 name과 value를 제공한다. 태그 param을 지정하면 태그 `<jsp:include ... />`와 `<jsp:forward ... />`에서 page에 지정한 페이지로 내장 request에 대한 파라미터도 전달되고, 태그 param에 의한 파라미터도 함께 전달되는데, 이름의 같으면 태그 param에 의한 값이 전달된다.

태그 `<jsp:include ... />`에서 `<jsp:param ... />` 태그를 이용하는 소스는 다음과 같다. 태그 param을 이용할 때는 include 시작 태그의 마지막이 `/>`가 아니라 `>`임에 주의하자.

```
<jsp:include page="loginhandle.jsp" >
    <jsp:param name="userid" value="guest" />
    <jsp:param name="passwd" value="anonymous" />
</jsp:include>
```

태그 `<jsp:forward ... >`에서 `<jsp: param ... />` 태그를 이용하는 소스도 태그 `<jsp:include ... >`와 같으며, 태그 `<jsp:include ... >`와 마찬가지로 태그 `param`을 이용할 때는 forward 시작 태그의 마지막이 `>`임에 주의하자.

```
<jsp:forward page="forwardloginhandle.jsp" >
    <jsp:param name="userid" value="kdhong" />
    <jsp:param name="snum" value="2010-3459" />
</jsp:forward>
```

- 태그 include에서 param 태그 이용

액션 태그 `include`에서 삽입되는 페이지로 파라미터를 전송하려면 액션 태그 `<jsp:param name="name" value="value" />`를 이용한다. 액션 태그 `include`에서 태그 `<jsp:param ... />`을 이용하려면 종료 태그 `</jsp: include>`를 이용해야 한다.

태그 `include`에서 삽입되는 페이지에서는 태그 `param`으로 인자를 전송 받을 뿐만 아니라 기본적으로 내장 객체 `request`도 함께 전달되므로 `request.getParameter()`에 의해 인자도 함께 전송 받을 수 있다.

다음 예제는 `login.html`, `login.jsp`, `loginhandle.jsp` 3개의 파일로 구성된 프로그램이다.

이 예제는 `login.html`을 실행하여 로그인 버튼을 눌러 실행한다. 파일 `login.html`의 폼에서 입력 받은 자료를 내장 객체 `request`로 `login.jsp`로 전송한다. 프로그램 `login.jsp`에서는 아이디의 입력이 없으면 다시 `param` 태그의 인자 `userid`, `passwd`를 각각 "guest", "anonymous"로 수정하여 `loginhandle.jsp`를 `include` 시킨다. 만일 아이디의 입력이 있으면 `param` 태그를 이용하지 않으므로 그대로 `request`에 의해 폼의 입력 값이 전송된다.

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>로그인 : login.html</title>
</head>
<body>
    <h2>로그인 </h2>
    <form method="post" action="login.jsp">
        <table border="1">
            <tr><th>아이디</th><td><input type="text" name="userid"></td></tr>
            <tr><th>비밀번호</th><td><input type="password" name="passwd"></td></tr>
            <tr><td colspan="2" align="center"><input type="submit" value="로그인">
                <input type="reset" value="다시입력"> </td></tr>
        </table>
    </form>
</body>
```

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

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : login.jsp </title>
</head>
<body>
    <h2>로그인 예제 </h2>
    <%
        request.setCharacterEncoding("UTF-8");
        String userid = request.getParameter("userid");
        String passwd = request.getParameter("passwd");
    %>
    <%
        if (userid.equals("")) {
    %>
        <jsp:include page="loginhandle.jsp" >
            <jsp:param name="userid" value="guest" />
            <jsp:param name="passwd" value="anonymous" />
        </jsp:include>
    <%
        } else {
    %>
        <jsp:include page="loginhandle.jsp" />
    <%
        }
    %>
</body>
</html>
```

loginhandle.jsp

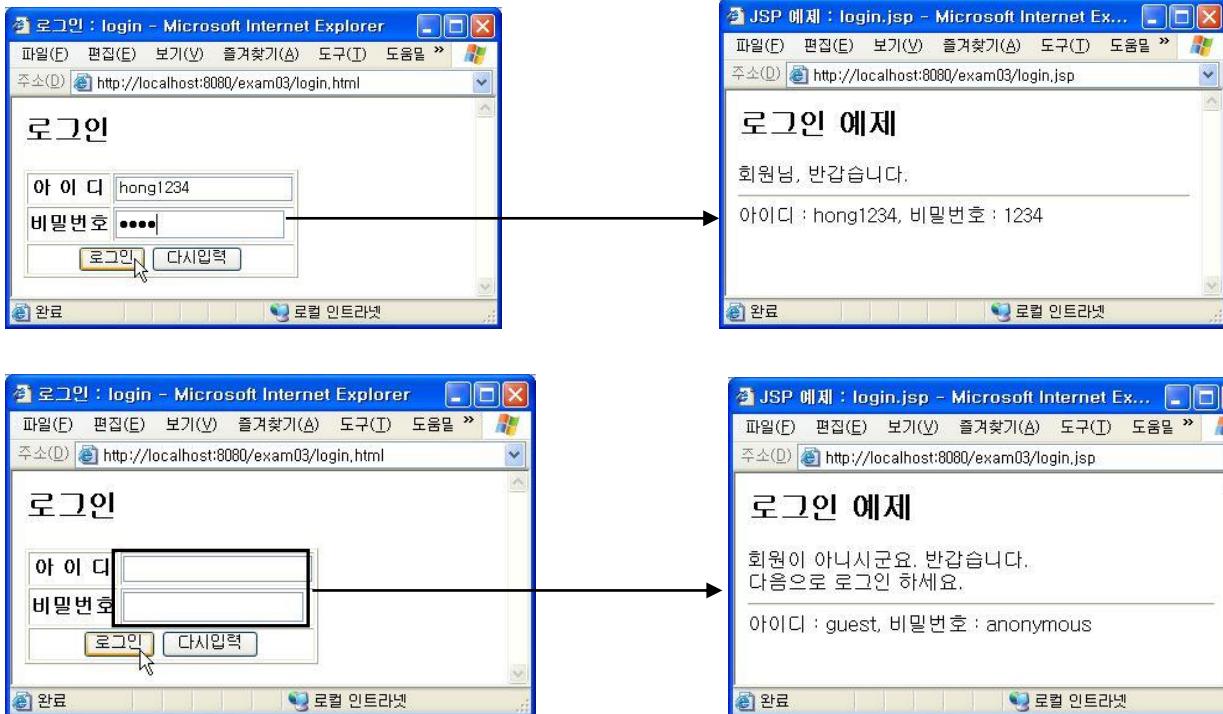
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : loginhandle.jsp </title>
</head>
```

```

<body>
<%
    request.setCharacterEncoding("UTF-8");
    String userid = request.getParameter("userid");
    String passwd = request.getParameter("passwd");
%>
<%
    if (userid.equals("guest")) {
        out.println("회원이 아니시군요. 반갑습니다.<br>");
        out.println("다음으로 로그인 하세요.<hr>");
    } else {
        out.println("회원님, 반갑습니다.<hr>");
    }
%>
아이디 : <%= userid %>,
비밀번호 : <%= passwd %>
</body>
</html>

```

[결과]



- 태그 forward에서 param 태그 이용(태그 forward에서 지정한 인자의 전송)

액션 태그 forward에서 삽입되는 페이지로 파라미터를 전송하려면 액션 태그 `<jsp:param name="name" value="value" />`를 이용한다. 액션 태그 forward에서 태그 `<jsp:param ... />`을 이용하려면 종료 태그 `</jsp:forward>`를 이용해야 한다.

태그 forward에서 삽입되는 페이지에서는 태그 param으로 인자를 전송받을 뿐만 아니라 기본적으로 내장 객체 request도 함께 전달되므로 request.getParameter()에 의해 인자를 전송 받을 수 있다.

다음 예제는 forwardlogin.html, forwardlogin.jsp, forwardloginhandle.jsp 3개의 파일로 구성된 프로그램이다. 이 예제는 forwardlogin.jsp를 실행하나 그 내부에 forward 태그가 있어 forwardloginhandle.jsp의 실행에서 출력되는 내용만 브라우저에 표시된다. **login.html를 forwardlogin.html로 변경 <form method="post" action=" forwardlogin.jsp"> 수정 후 실행**

forwardlogin.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : forwardlogin.jsp</title>
</head>
<body>
    <h2>forward 태그를 이용한 로그인 예제</h2>
    <%
        request.setCharacterEncoding("UTF-8");
        String userid = request.getParameter("userid");
        String passwd = request.getParameter("passwd");
    %>
    <%
        if ( userid == null && passwd == null ) {
    %>
        <jsp:forward page="forwardloginhandle.jsp" />
    <%
        } else {
    %>
        <jsp:forward page="forwardloginhandle.jsp" >
            <jsp:param name="snum" value="2010-3459" />
        </jsp:forward>
    <%
        }
    %>
</body>
</html>
```

forwardloginhandle.jsp

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

```

<head><meta charset="UTF-8">
<title>JSP 예제 : forwardloginhandle.jsp</title>
</head>
<body>
<%
request.setCharacterEncoding("UTF-8");
String userid = request.getParameter("userid");
String passwd = request.getParameter("passwd");
String studentnum = request.getParameter("snum");
%>
<%
if ( userid == null ? true : userid.equals("") ) {
%>
<h2>로그인 </h2>
<h2>아이디와 비밀번호를 입력하세요 </h2>
<form method="post" action="forwardlogin.jsp">
<table border="1">
<tr><th>아이디</th>
<td><input type="text" name="userid"></td></tr>
<tr><th>비밀번호</th>
<td><input type="password" name="passwd"></td></tr>
<tr><td colspan="2" align="center">
<input type="submit" value="로그인">
<input type="reset" value="다시입력"> </td></tr>
</table>
</form>
<%
} else {
%>
    아이디 : <%= userid %>,
    암호 : <%= passwd %>,
    학번 : <%= studentnum %>
    <hr> 회원님, 반갑습니다.
<%
}
%>
</body>
</html>

```

[결과]



10.6 액션 태그 plugin

- 자바 빈즈 또는 애플릿 삽입 태그(<jsp:plugin ... >)

액션 태그 plugin은 웹 브라우저에서 자바 빈즈 또는 애플릿을 플러그인하여 실행하는 태그이다. 태그 plugin은 각기 다른 웹 브라우저에서 인식할 수 있도록 마이크로소프트 사의 IE 경우일 때 object 태그로 만들어 주며, 넷스케이프 사의 경우일 때는 embed 형태의 태그로 만들어 준다.

현재 액션 태그 plugin에서 플러그인 할 수 있는 객체는 자바 빈즈와 애플릿만 가능하며 이에 따라 속성 type에는 bean 또는 applet만을 지정할 수 있다. 액션 태그 plugin에서 사용하는 다른 속성은 다음과 같다.

```
<jsp:plugin
    type = "bean | applet"
    code = "objectCode"
    codebase = "objectCodebase"
    align = "alignment"
    height = "height"
    hspace = "hspace"
    name = "componentName"
    vspace = "vspace"
    width = "width"
    archive = "archiveList"
    nspluginurl = "url"
    iepluginurl = "url" >
    <jsp:params>
        <jsp:param name="파라미터 이름" value="파라미터 값" />
    </jsp:params>
</jsp:plugin>
```

액션 태그 plugin의 속성 중에서 type, code, codebase만 필수적으로 입력하고 나머지는 선택적으로 사용한다.

속성	사용 용도
type	사용할 플러그인 종류로 applet 또는 bean
code	사용할 클래스 이름을 지정(불러올 클래스 파일)
codebase	사용할 클래스를 발견할 기본 경로를 지정(클래스 파일의 위치)

archive	jar 혹은 zip과 같은 압축 형태로 배포하는 경우. 플러그인에서 사용할 압축 파일을 지정
height	플러그인의 높이를 지정
width	플러그인의 너비를 지정
name	이름을 지정
hspace	좌우의 여백을 지정
vspace	상하 여백을 지정
nspluginurl	넷스케이프 네비게이터에서 자바 플러그인의 위치를 지정(넷스케이프 전용 플러그인의 다운로드 URL을 입력한다.)
iepluginurl	익스플로러에서 자바 플러그인의 위치를 지정(익스플로러 전용 플러그인의 다운로드 URL을 입력한다.)

- init() : 애플릿을 실행시키면 맨 처음 수행되는 메소드
- start() : 애플릿이 활성화될 때 마다 불려진다
- stop() : 애플릿이 비활성화 될 때마다 불려진다
- destroy() : 애플릿을 종료할 때 불려진다.
- paint() : 애플릿에 그림을 그릴 때 불려진다. paint()는 프로그래머가 직접 호출 할 수 없으며 init()에서 start() 호출 후에 자동으로 불려진다.
- update() : 애플릿의 그림을 모두 지우고, paint()를 호출한다.
- repaint() : update()를 호출한다. 프로그래머가 그림을 그려야 한다고 판단했을 때, repaint()를 호출하면 repaint()가 update()를 호출한다. 다시 update()는 paint() 호출하고 repaint()는 간접적으로 paint()를 호출한다. (애플릿의 실행 순서 : init() → start() → [paint()] → stop()→ destroy())

11. 액션 태그를 활용한 템플릿 페이지 작성

- 템플릿 페이지를 사용하는 이유

여러 사이트들을 돌아다녀보면 좌측과 상단의 메뉴는 고정되어 있고 메뉴에 따라 가운데 페이지가 매번 변하게 된다. 이렇게 반복되는 페이지의 사이트를 만들 때 템플릿 페이지를 사용하지 않는 것은 매우 비효율적이다. 템플릿 페이지를 사용하지 않을 경우 나중에 레이아웃을 변경할 경우 작성된 페이지를 모두 새로 작성해야 하기 때문이다.

이 같은 문제를 해결하기 위해서 사용하는 것이 바로 템플릿 페이지이다. 템플릿 페이지는 레이아웃을 표현하고 내용은 다른 페이지에서 가져오므로 나중에 페이지 수정 시에도 이 템플릿 페이지만 변환시키면 되므로 매우 편리하다.

- 일반 웹 페이지의 구조

위(TOP)	
왼쪽(LEFT)	가운데 (CENTER)
아래(BOTTOM)	

일반적인 웹 페이지의 구조이다. 크게 위(TOP), 아래(BOTTOM), 왼쪽(LEFT), 가운데(CENTER)로 네 부분으로 나누어져 있다. 이중 TOP, BOTTOM, LEFT 페이지의 경우는 메뉴로 사용되고 있으므로 매번 변경되는 페이지는 가운데 페이지뿐이다. 가운데 페이지는 메뉴에 의해 내용이 계속 바뀌게 되는데 이 레이아웃 구조대로 효율적으로 사이트를 개발하려면 템플릿 페이지를 사용하는 것이 좋다.

- 템플릿 페이지의 설계

템플릿 페이지의 설계에서 중요한 것은 레이아웃 구조를 파악하는 것이다. 템플릿 페이지 자체가 레이아웃 구조나 마찬가지이기 때문이다. 템플릿 페이지를 설계를 위해서는 기본적인 홈페이지의 틀이 결정되어 있어야 한다. 여기서는 앞의 일반적인 웹 페이지 구조처럼 TOP, BOTTOM, LEFT, CENTER 이렇게 네 부분으로 나누어 설계를 하도록 하겠다. 아래의 레이아웃 구조는 전형적인 웹 사이트의 구조이며 템플릿 페이지를 표현하기 위해 간단하게 작성해 볼 것이다.

- 액션 태그를 이용한 템플릿 페이지의 작성

여기서 작성한 템플릿 페이지는 위의 웹 페이지 레이아웃을 사용할 곳이다. 템플릿 페이지를 중심으로 설명할 것이므로, 위의 웹 페이지 구조에서 "신상품", "인기상품" 메뉴만 페이지화시키도록 하겠다. 파일 중에서 가장 주목해야 할 파일은 template.jsp이다. 이 파일이 TOP, BOTTOM, LEFT, CENTER의 레이아웃 구조를 담고 있을 파일이며, 메뉴에 따라 이동되는 페이지 또한 표시해 줄 페이지이다. 템플릿 페이지를 사용하는 이유와 특성을 익히기 위함이 목적이므로, 나머지 파일들은 간단하게 작성하도록 하겠다.

top.jsp(상단에 표시될 메뉴 파일 이름이다)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<a href=".template.jsp?page=login">Login</a> |
<a href=".template.jsp?page=join">Join</a>
```

bottom.jsp(하단에 표시될 메뉴 파일 이름이다)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<center>Since 2008</center>
```

left.jsp(좌측에 표시될 파일이다.)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<center>
<a href=".template.jsp?page=newitem">신상품 </a> <br> <br>
<a href=".template.jsp?page=bestitem">인기상품 </a> <br> <br>
</center>
```

newitem.jsp(신상품 페이지 파일 이름이다)

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

bestitem.jsp(인기상품 페이지 파일 이름이다)

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

template.jsp(템플릿 레이아웃 파일 이름이다)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%
    String pagefile = request.getParameter("page");
    if (pagefile==null){
        pagefile="newitem";
    }
    String pagefileex = ".jsp";
%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Template Test</title>
<style type="text/css">
* {margin:0; padding: 0; }

div#wrapper { margin:auto; width:960px; overflow:hidden; border:1px solid gray; }

div#header { float:left; width:100%; height:43px; border:1px solid gray; }

div#sidebar { float:left; width:15%; text-align:center; border:1px solid gray; }

div#contents { float:left; width:84%; text-align:center; }

div#footer { float:left; width:100%; height:40px; border:1px solid gray; }
```

```

</style>
</head>
<body>
<div id="wrapper">
    <div id="header">
        <jsp:include page="top.jsp"/>
    </div>
    <div id="sidebar">
        <jsp:include page="left.jsp"/>
    </div>
    <div id="contents">
        <jsp:include page="<% =pagefile+pagefileex %>" />
    </div>
    <div id="footer">
        <jsp:include page="bottom.jsp"/>
    </div>
</div>
</body>
</html>

```

[결과]



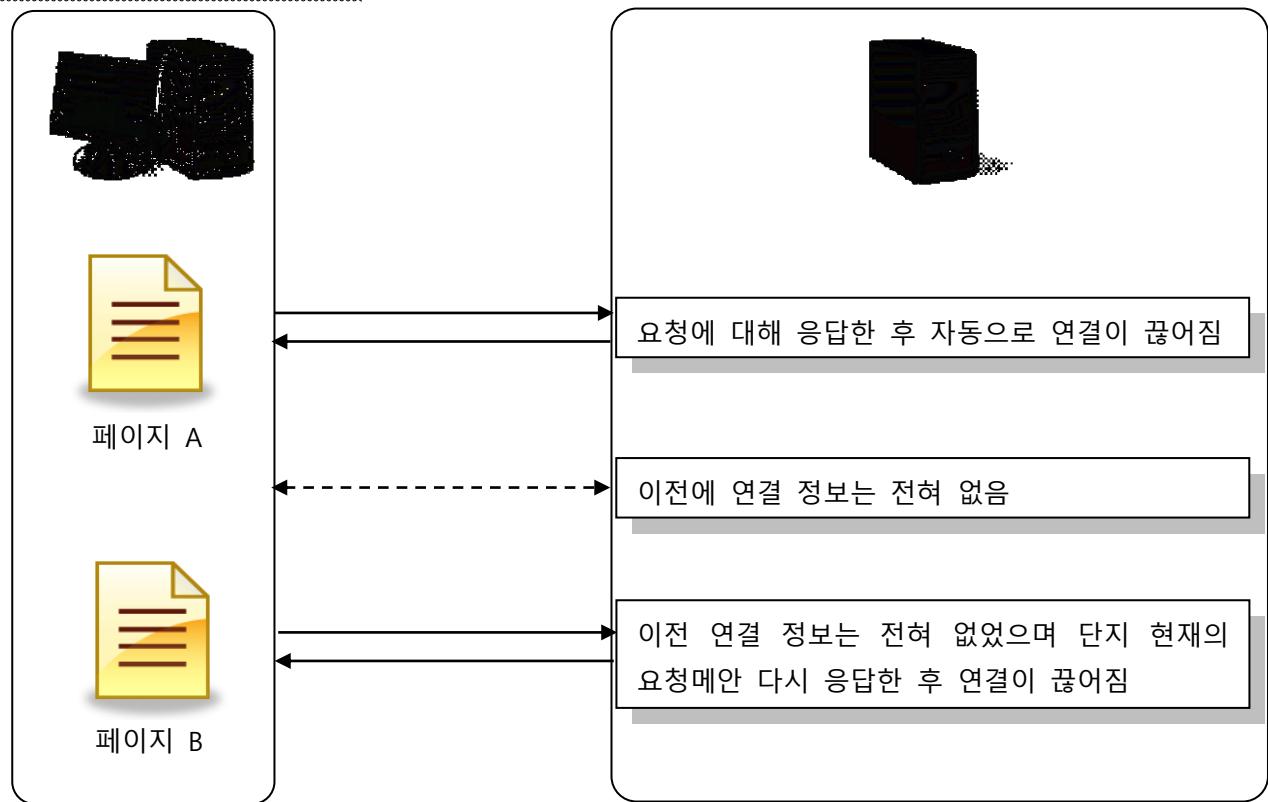
위의 결과 페이지를 보면 TOP, BOTTOM, LEFT 이 세부분은 항상 내용이 고정되어 있다. 신상품, 인기상품을 누르게 되면 그에 해당하는 페이지가 표시된다. 주소 표시줄을 보게 되면 '신상품'을 보고 있을 때는 page 파라미터에 newitem 값을 갖고, '인기상품'을 보고 있을 때는 page 파라미터에 'bestitem' 값을 갖는다. 템플릿 페이지를 이용하면 이와 같이 여러 페이지 파라미터 값을 사용하여 표시할 수 있다. 그리고 TOP, BOTTOM, LEFT 페이지 또한 가운데에 표시되는 각 메인 페이지마다 구현하지 않아도 템플릿 페이지에만 포함시켜놓으면 어떤 페이지를 표시하던지 항상 보여지게 된다. 이것을 JSP 페이지의 모듈화라고도 한다.

12. 웹의 비연결 특성

12.1 비연결 특성

- HTTP의 비연결 특성

웹에서 클라이언트가 요청을 하면 서버는 응답을 한다. 즉 한 페이지의 요청과 그 요청에 대한 응답이 있을 때만 클라이언트와 서버가 연결(Connection) 될 뿐 그 이후에는 연결이 자동으로 종료된다. 동일한 클라이언트가 다시 서버에 연결을 하더라도 서버는 그 이전 연결에 대한 클라이언트의 어떠한 상태(state) 정보도 가지고 있지 않은 상태에서 다시 연결과 종료가 이루어진다. 즉 웹 서비스에서 클라이언트와 서버는 웹 페이지들간 사이에서 서로 연관 없이 각각 독립적으로 연결이 이루어지므로 상태의 지속성을 유지할 수 없다. 이것은 웹을 지원하는 HTTP 통신 규약이 비 연결(Connection) 또는 무상태(stateless) 특성을 가지기 때문이다.



- 비연결성의 장단점

HTTP의 웹 브라우저에서 서버에 정보를 요청하여 그 결과를 얻어 오는 것이 모든 상태는 단절된다. 이러한 비연결성은 서로 연관 없는 페이지들을 접속할 때는 아무 문제가 되지 않으며, 오히려 서버에 접속한 클라이언트의 수가 많더라도 서버의 부담이 적고 서버의 자원을 효율적으로 사용할 수 있는 등의 장점으로 작용한다. 그러므로 웹의 비연결 특성은 초기에 웹 서비스를 빠르게 성장시킨 계기가 되었다. 그러나 웹 서비스를 이용해 다양하고 복잡한 정보 시스템을 구축하면서 이러한 웹의 비연결 특성은 많은 문제점을 야기 시킬 수 있다. 가장 간단한 예가 쇼핑몰의 장바구니이다. 인터넷 쇼핑몰에서 상품을 구매하는 경우, 사용자는 장바구니를 확인하면서 이미 선택한 상품도 보고 다시 다른 페이지로 이동하여 원하는 상품을 더 살 수 있으며 필요 없는 물건은 장바구니의 구매 목록에서 제거시킬 수도 있다. 이런 경우, 장바구니 페이지는 다른 페이지로 이동하더라도 현재 선택된 상품 목록과 관련 정보를 지속적으로 유지 관리해야 하는데, HTTP의 비연결성은 이러한 상태 정보의 관리를 어렵게 만든다.

12.2 세션과 쿠키

- 비연결성의 보완

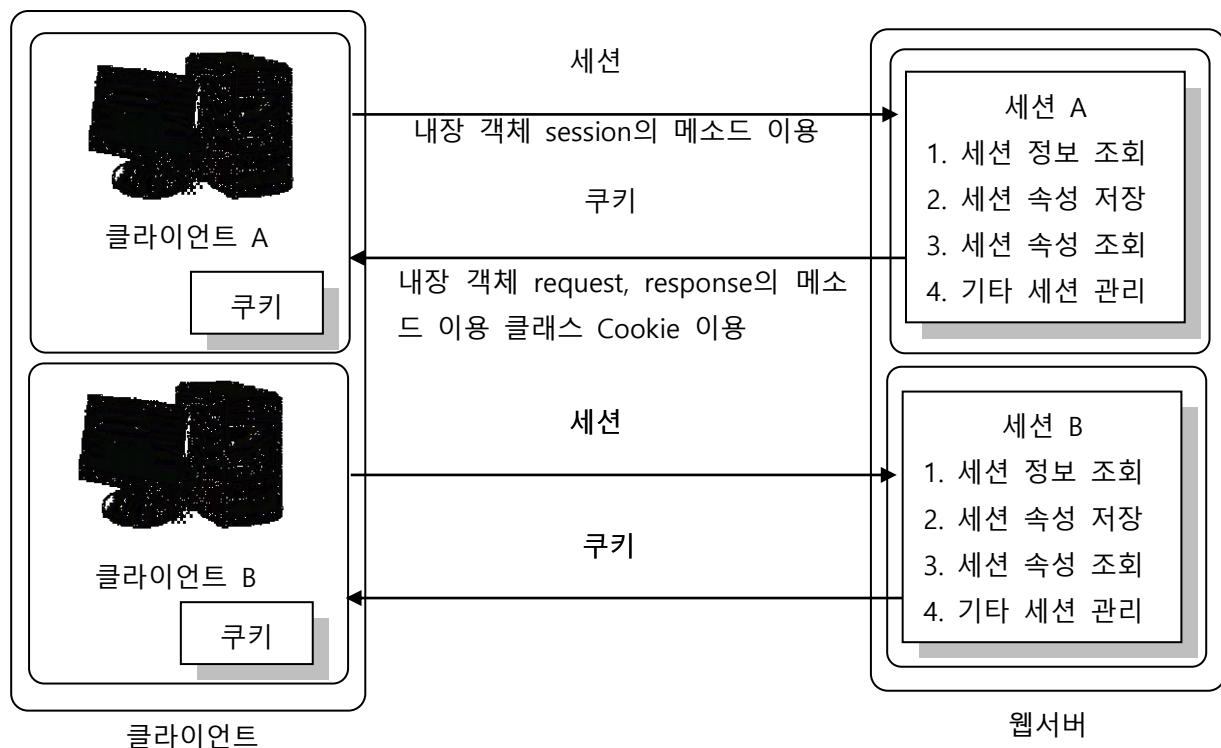
HTTP의 비연결 특성은 클라이언트가 이전에 한 작업의 정보를 알 수 없게 한다. 이러한 비연결성을 보완하고 페이지 간의 지속성 서비스를 제공하기 위한 기법이 쿠키(cookie)와 세션(session)이다.

쿠키는 넷스케이프에서 제안한 방식으로 CGI 프로그래밍 방식 때부터 사용하던 클라이언트 정보 관리 방법으로 클라이언트의 사용자 컴퓨터에 사용자 정보를 저장 관리한다. 그러므로 쿠키 방식은 서버에 부하를 주지 않으면서 사용자 정보를 관리할 수 있는 방법이다.

세션은 JSP에서 제공하는 클라이언트의 브라우저 정보 관리 방법으로 브라우저마다 각기 다른 사용자 정보를 서버에 저장하는 방법이다. 즉 세션은 클라이언트 사용자 별로 여러 페이지 이동을 인식하며, 필요한 정보를 서버에 저장하고 조회 할 수 있는 방법과 세션을 관리 할 수 있는 방법을 제공한다.

- 쿠키와 세션의 비교

JSP에서는 쿠키를 지원하기 위한 클래스 Cookie를 제공한다. 또한 JSP에서는 세션의 역할을 담당하는 내장 객체 session을 제공한다. 쿠키는 클라이언트 별 사용자 정보를 클라이언트 사용자 컴퓨터에 저장하나 세션은 서버에 저장한다. 그러므로 웹 서버가 종료되더라도 유효기간이 지나지 않은 쿠키는 조회가 가능하지만 세션 정보는 웹 서버가 종료되거나 일정한 시간 동안 서버에 반응을 하지 않으면 세션 정보가 삭제된다.



쿠키는 하나가 4K Byte 크기로 제한되고, 클라이언트당 쿠키의 최대 용량은 1.2M Byte로 제한이 있으며, 쿠키 값은 문자열을 지원하고 쿠키 정보는 클라이언트 컴퓨터에 파일 형태로 저장되어 다른 사람이나 시스템이 볼 수 있으므로 비밀유지가 어려운 단점이 있다. 반면에 세션은 클라이언트 별로 모든 객체 형태의 자료를 서버에 저장하므로 클라이언트 JSP 프로그램을 통해서만 참조가 가능하다. 그러므로 세션은 보안 유지에 강력한 장점이 있으며, 또한 서버의 용량만 가능하다면 저장 할 수 있는 세션 정보 크기의 한계도 없다.

쿠키

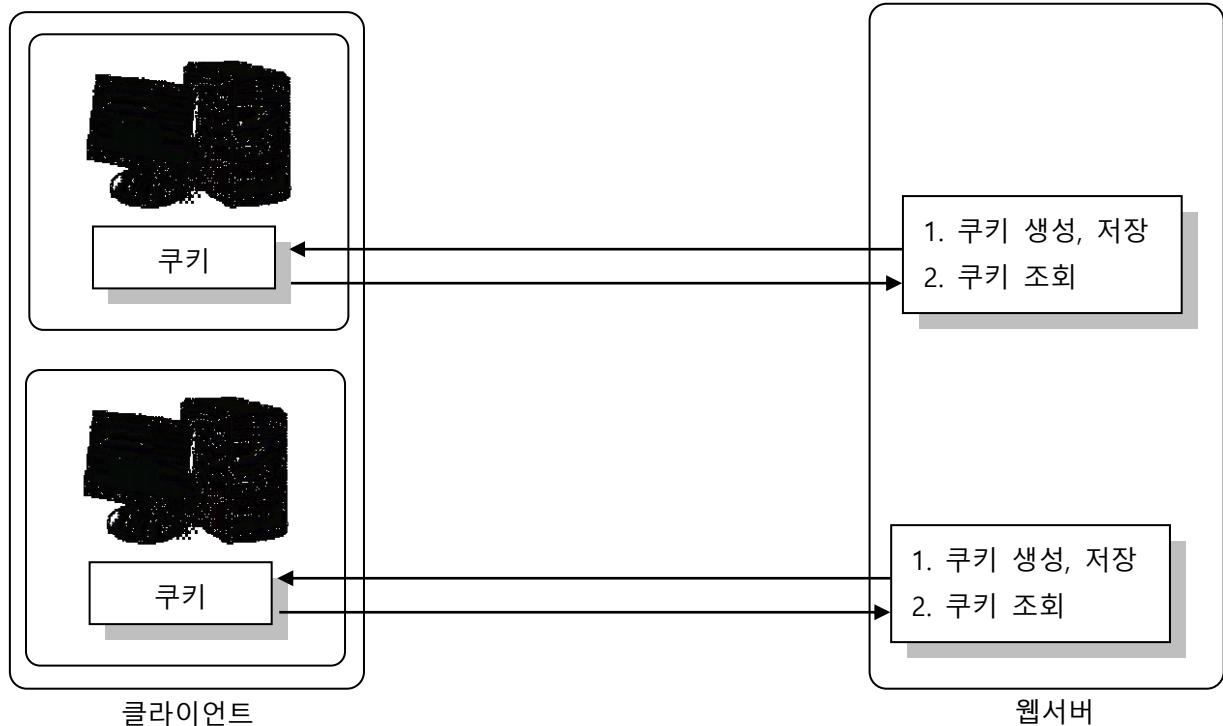
세션

사용 클래스 및 인터페이스	class javax.servlet.http.Cookie	interface javax.servlet.http.HttpSession
관련 내장 객체	response, request	session
저장 값 유형	문자열(String) 형태만 가능	자바의 모든 객체(Object)
저장 장소	클라이언트에 저장	서버에 저장
정보 크기	총 1.2M로 제한 있음	제한 없음
보안	어려움	강력함

12.3 쿠키 클래스

- 쿠키 정의

쿠키는 서버에서 만들어진 작은 정보의 단위로 서버에서 클라이언트의 브라우저로 전송되어 사용자의 컴퓨터에 저장된다. 이렇게 저장된 쿠키는 다시 해당하는 웹 페이지에 접속할 때 브라우저에서 서버로 전송될 수 있다.



쿠키는 사용자의 상태 정보를 클라이언트에서 관리하는 메커니즘을 의미한다. 클라이언트에 정보가 저장되기 때문에 서버의 부하가 크지 않지만 보안에 매우 취약하다. 쿠키는 웹 사이트의 도메인당 300개 까지 저장이 가능하다. 클라이언트에서 쿠키를 사용하지 못하도록 설정할 수 있기 때문에 제약이 있다. 쿠키는 클라이언트의 브라우저 메모리 및 OS 파일에 저장 가능하다. 기본 저장은 브라우저 메모리이기 때문에 브라우저를 종료하면 자동으로 쿠키 정보도 제거된다.

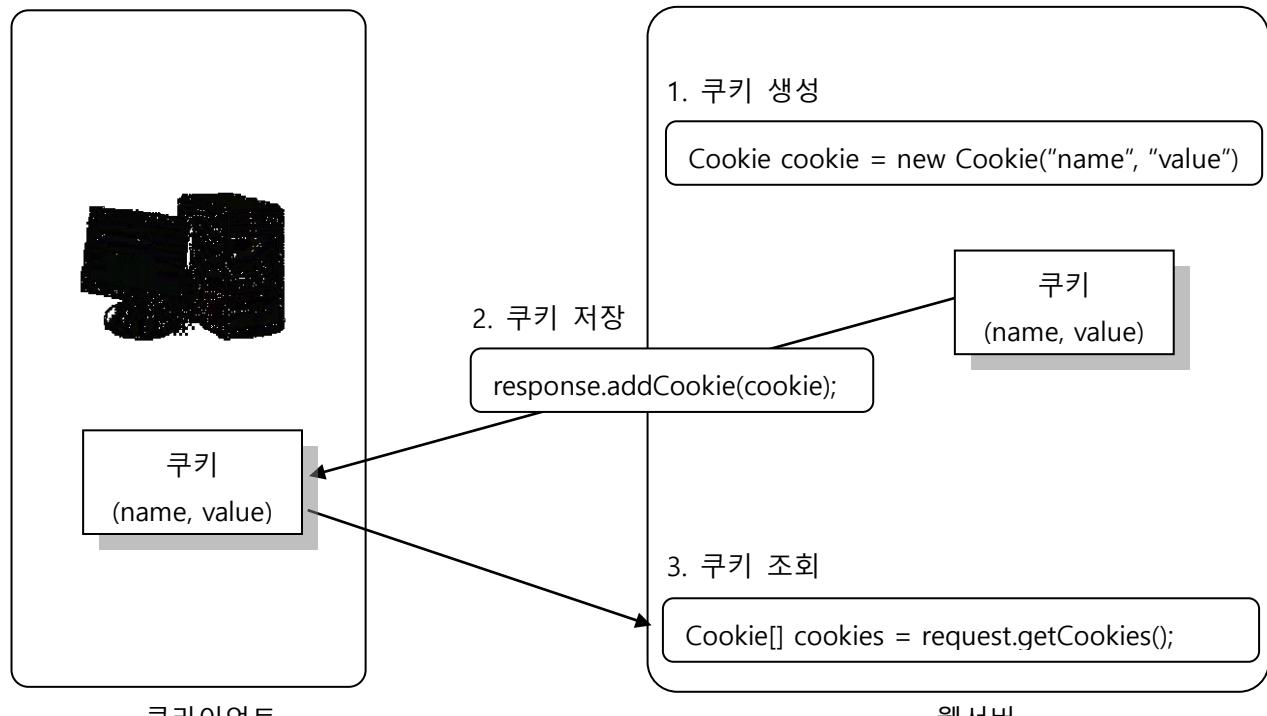
쿠키는 이름(name)과 값(value)으로 구성된 자료를 저장하며, 이러한 이름과 값 외에도 주석(comment), 경로(path), 유효기간(maxage 또는 expiry), 버전(version), 도메인(domain)과 같은 추가적인 정보를 저장할 수 있다.

- 클래스 Cookie

JSP에서는 쿠키를 사용하기 위해 javax.servlet.http.Cookie 클래스를 제공한다. 쿠키를 하나 생성하여 저장하려면, Cookie 객체를 하나 생성한 후 내장 객체 response의 addCookie() 메소드를 이용해서 생성

된 Cookie 객체를 인자로 클라이언트에게 쿠키를 전송하여 저장한다.

반대로 사용자 컴퓨터에 저장된 쿠키는 내장 객체 request의 getCookies() 메소드를 이용해서 모두 서버로 전달하여 조회할 수 있다.



쿠키는 그 수와 크기에 제한이 있는데, 하나의 쿠키는 4K Byte 크기로 제한되고, 브라우저는 각각의 웹사이트당 20개의 쿠키를 허용하며, 모든 웹 사이트를 합쳐 최대 300개를 허용한다. 그러므로 클라이언트당 쿠키의 최대 용량은 1.2M Byte이다.

클래스 Cookie는 패키지 javax.servlet.http에 속하며 다음과 같은 주요 메소드를 제공한다.

반환형	메소드 이름	메소드 기능
int	getMaxAge()	쿠키의 최대지속 시간을 초단위로 지정 -1일 경우 브라우저가 종료되면 쿠키도 만료
String	getName()	쿠키의 이름을 String으로 반환
String	getValue()	쿠키의 값을 String으로 반환
void	setMaxAge(int expiry)	쿠키의 만료시간을 초단위로 설정
void	setValue(String newValue)	쿠키에 새로운 값을 설정할 때 사용

12.4 쿠키의 사용

- 쿠키 추가

쿠키는 다음과 같이 문자열의 인자 2개로 생성하는데, 앞 인자는 쿠키의 이름(name)이고 뒤 인자는 쿠키의 값(value)이다. 즉 쿠키는 (이름, 값)의 한 쌍 정보를 입력하여 생성한다. 쿠키의 이름은 알파벳과 숫자로만 구성되고, 쿠키 값에는 공백, 괄호, 등호, 콤마, 콜론, 세미콜론 등을 포함할 수 없다.

```
Cookie cookie = new Cookie("user", "king");
```

생성된 쿠키는 메소드 setMaxAge()로 그 유효기간을 정할 수 있는데, 인자는 유효기간을 나타내는 초이다. 만일 유효기간을 0으로 지정하면 쿠키의 삭제를 의미하며, 음수를 지정하면 브라우저가 종료될 때

쿠키도 함께 삭제된다. 다음은 유효기간을 2분으로 지정하며, 1주일로 지정하려면 ($7 * 24 * 60 * 60$)로 한다.

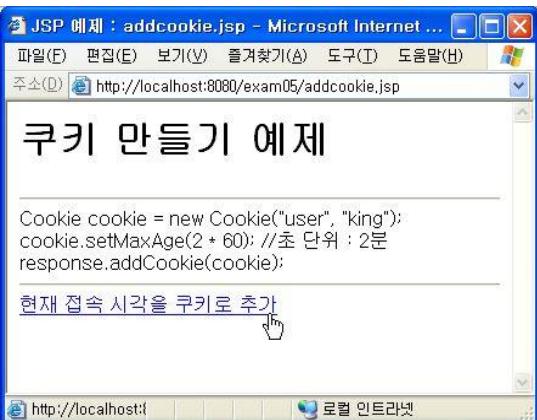
```
cookie.setMaxAge(2*60); //초 단위 :2분
```

쿠키는 내장 객체 response의 addCookie 메소드를 이용하여 클라이언트의 컴퓨터에 파일 형태로 저장한다.

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

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : addcookie.jsp</title>
</head>
<body>
    <h1>쿠키 만들기 예제</h1>
    <hr>
    Cookie cookie = new Cookie("user", "king"); <br>
    cookie.setMaxAge(2 * 60); //초 단위 :2분 <br>
    response.addCookie(cookie); <br>
    <%      Cookie cookie = new Cookie("user", "kang");
            cookie.setMaxAge(2 * 60); //초 단위 :2분
            response.addCookie(cookie); %>
    <hr><a href="addtimecookie.jsp">현재 접속 시각을 쿠키로 추가</a>
</body>
</html>
```

[결과]



다음은 현재의 시각 정보를 쿠키 이름 "lastconnect"로 저장하는 코드이다. 유효기간을 짧게 10초로 지정하면 10초 이후에 이 쿠키 정보가 삭제되는 것을 확인해 볼 수 있다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.Calendar" %>
<!DOCTYPE html>
```

```

<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : addtimecookie.jsp</title>
</head>
<body>
    <h1>현재 시각을 쿠키로 저장</h1>
    <hr>
    Calendar day = Calendar.getInstance();<br>
    String now = day.get(Calendar.YEAR)+ "-" +(day.get(Calendar.MONTH)+1);<br>
    now += "-" +day.get(Calendar.DAY_OF_MONTH);<br>
    Cookie cookie = new Cookie("lastconnect", now); <br>
    cookie.setMaxAge(10); //초 단위 : 10초 <br>
    response.addCookie(cookie); <br>
    <%
        Calendar day = Calendar.getInstance();
        String now = day.get(Calendar.YEAR)+ "-" +(day.get(Calendar.MONTH)+1);
        now += "-" +day.get(Calendar.DAY_OF_MONTH);
        Cookie cookie = new Cookie("lastconnect", now);
        cookie.setMaxAge(10); //초 단위 : 10초
        response.addCookie(cookie);
    %>
    <hr> <a href="getcookies.jsp">쿠키 조회</a>
</body>
</html>

```

[결과]



- 쿠키 조회

클라이언트에 저장된 쿠키를 조회하려면 내장 객체 request의 getCookies() 메소드를 이용한다. **메소드 getCookies()의 반환값은 저장된 모든 쿠키의 배열로, 쿠키가 없으면 null값이 반환된다.**

```

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

```

```

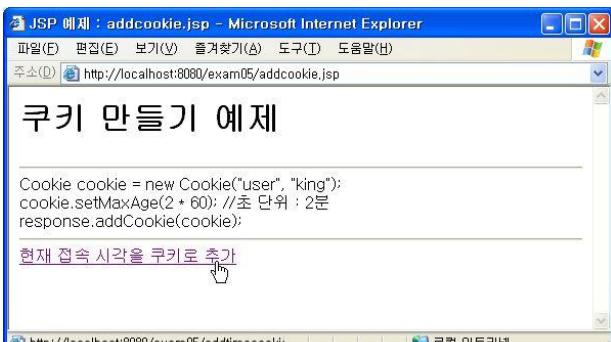
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : getcookies.jsp</title>
</head>
<body>
    <h1>쿠키 조회 예제</h1>
    <hr>
    <%
        Cookie[] cookies = request.getCookies();
        if (cookies == null) {
            out.println("쿠키가 없습니다.");
        } else {
            /*
            for (int i=0; i<cookies.length; ++i) {
                out.println("쿠키 이름(name) : " + cookies[i].getName() + ", ");
                out.println("쿠키 값(value) : " + cookies[i].getValue() + "<br>");
            }
            */
            for (Cookie c : cookies) {
                out.println("쿠키 이름(name) : " + c.getName() + ", ");
                out.println("쿠키 값(value) : " + c.getValue() + "<br>");
            }
        }
    %>
</body>
</html>

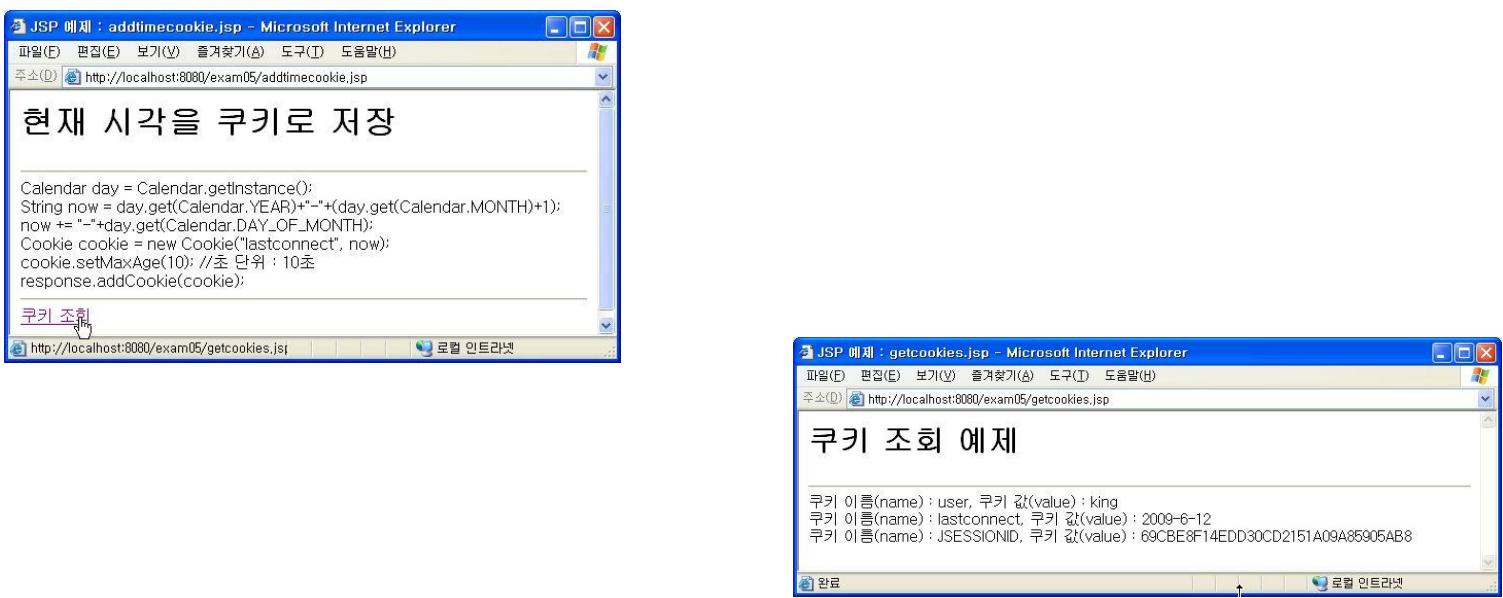
```

[결과]



addcookie를 실행하여 링크 [현재 접속 시각을 쿠키로 추가]를 연결하고, 다시 링크 [쿠키 조회]를 연결하면 3개의 쿠키가 조회되는 것을 볼 수 있다. 마지막에 조회되는 쿠키 [JSESSIONID]는 시스템이 남기는 쿠키로 브라우저가 종료될 때까지 유효하다.





위 결과에서 조회되는 쿠키 [lastconnect]와 [user] 중에서 10초가 경과된 후 getcookie를 다시 조회하면 다음과 같이 유효기간을 10초로 설정한 [lastconnect]는 삭제되고, [user]가 조회되고, 2분 정도가 경과된 후 다시 조회하면 유효기간이 2분인 [user]도 삭제되며, 시스템이 남긴 [JSESSION] 쿠키만 조회되는 것을 볼 수 있다.

coder.jsp

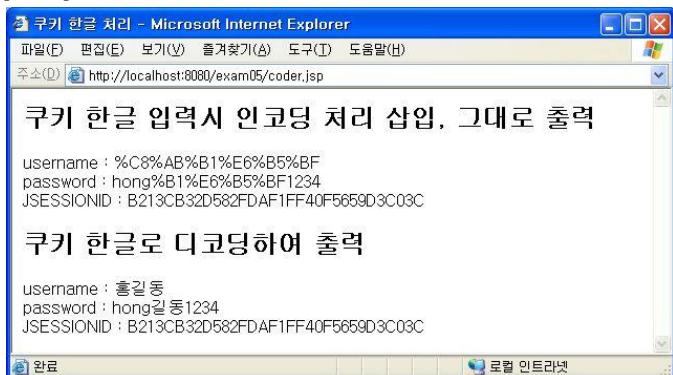
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.net.URLEncoder, java.net.URLDecoder" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>쿠키 한글 처리 : coder.jsp</title>
</head>
<body>
<h2>쿠키 한글 입력시 인코딩 처리 삽입, 그대로 출력</h2>
<%
Cookie user = new Cookie("username", URLEncoder.encode("홍길동", "UTF-8"));
Cookie pass = new Cookie("password", URLEncoder.encode("hong길동1234", "UTF-8"));
response.addCookie(user);
response.addCookie(pass);
Cookie[] cs = request.getCookies();
if(cs != null){
    for(Cookie cook : cs){
        out.println(cook.getName() + " : ");
        out.println(cook.getValue() + "<br>");
    }
}
```

```

    }
%>
<h2>쿠키 한글로 디코딩하여 출력</h2>
<%
if(cs != null){
    for(Cookie cook : cs){
        out.println(cook.getName() + " : ");
        out.println(URLDecoder.decode(cook.getValue(), "UTF-8") + "<br>");
    }
}
%>
</body>
</html>

```

[결과]



- 쿠키를 이용한 장바구니

product.html

```

<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>상품 리스트</title>
</head>
<body>
<h1>상품 리스트</h1>
<form action="CartSaveCookie" method="GET">
<input type="radio" name="product" value="BMW">BMW<br>
<input type="radio" name="product" value="SM5">SM5<br>
<input type="radio" name="product" value="K7">K7<br>
<input type="submit" value="카드저장">
</form>
</body>
</html>

```

장바구니 리스트

CartSaveCookieServlet.java

```
package com.controller;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/CartSaveCookie")
public class CartSaveCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 입력 파라미터 얻기
        String product = request.getParameter("product");
        //기존 쿠키 얻기
        Cookie [] cookies = request.getCookies();
        Cookie c = null;
        if(cookies == null || cookies.length == 0){
            c = new Cookie("product" , product );
        }else{
            c = new Cookie("product"+ ( cookies.length+1 ) , product );
        }
        //쿠키를 응답처리
        // c.setMaxAge(60*60);
        response.addCookie(c);
        out.print("<html> <body>");
        out.print("Product 추가");
        out.print("<a href='CartBasketCookie'>장바구니 보기</a>");
        //out.print("<a href="+response.encodeURL("CartBasketCookie")+">장바구니 보기2</a>");
        out.print("</body> </html>");
    }
}
```

```
}
```

CartBasketCookieServlet.java

```
package com.controller;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/CartBasketCookie")
public class CartBasketCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("장바구니 리스트<br>");
        //기존 쿠키 얻기
        Cookie [] cookies = request.getCookies();
        if( cookies != null ){
            for (Cookie c : cookies) {
                out.print(c.getName() + " :" + c.getValue() + "<br>");
            }
        }else{
            out.print("장바구니 비었음<br>");
        }
        out.print("<a href='product.html'>상품 선택 페이지</a><br>");
        out.print("<a href='CartDeleteCookie'>장바구니 비우기</a><br>");
        out.print("</body></html>");
    }
}
```

장바구니 리스트 비우기

CartDeleteCookieServlet.java

```
package com.controller;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/CartDeleteCookie")
public class CartDeleteCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html> <body> ");
        out.print("장바구니 비웠음!!");
        //기존 쿠키 얻기
        Cookie [] cookies = request.getCookies();
        if( cookies != null ){
            for (Cookie c : cookies) {
                c.setMaxAge(1);
                response.addCookie(c);
            }
            }else{
                out.print("장바구니 비었음<br>");
            }
            out.print("<a href='product.html'>상품 선택 페이지</a><br>");
            out.print(" </body> </html>");
        }
    }
```

12.5 세션의 사용

- 세션 개념과 HttpSession 클래스

클라이언트의 정보를 클라이언트 PC에 저장하는 것이 쿠키라면 클라이언트의 브라우저마다 각기 다른 정보를 서버에 저장하는 것이 세션이다. 즉 세션은 클라이언트 사용자 별로 여러 페이지 이동을 인식하며, 필요한 정보를 서버에 저장하고 조회할 수 있는 방법과 세션을 관리할 수 있는 방법을 제공한다. 일반적으로 사용되는 세션의 정의는 '서버와 클라이언트간의 지속적인 연결'을 의미한다. 연결을 통하여 클라이언트는 지속적으로 서버에 특정 동작을 요청할 수 있으며 서버는 실행 결과를 클라이언트에 응답할 수 있다.

데이터베이스를 사용하는 경우에도 클라이언트와 DB서버 간에 지속적인 연결을 의미하는 세션이 필요하다. HTTP 프로토콜을 기반으로 하는 웹 서비스는 동작 메커니즘이 다르다. 불특정 다수인 클라이언트와 지속적인 연결방식으로는 웹 서버의 부하가 매우 크기 때문이다. 동시 접속자가 100만 건이라고 가정했을 때, 100만 클라이언트와 지속적으로 연결된 서버가 동작하는 것은 불가능하다. 따라서 클라이언트가 웹 서버에 요청하고 응답 받으면 즉시 연결을 끊는 connectionless 방식으로 동작된다.

이것은 웹 브라우저의 각 페이지는 서로 간에 연결고리가 없다는 것을 의미한다. 즉, 첫 화면에서 선택한 물건을 장바구니에 담고 다음 페이지에서 결제할 때 이전 화면의 장바구니에 담긴 정보를 확인 할 수 없다는 것이다. 서비스되고 있는 많은 웹 사이트를 보면 장바구니 및 로그인 기능 같은 처리가 구현 HTTP 프로토콜의 문제점을 극복하기 위해서 사용자의 상태 정보를 관리하는 메커니즘이 필요한데 이것을 '세션 관리'라고 한다.

세션 관리의 방법

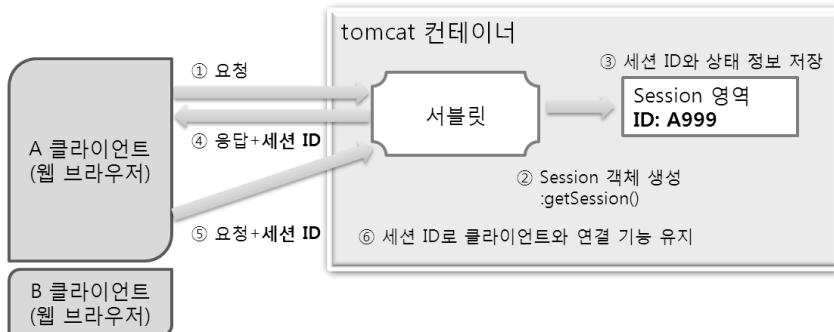
HttpSession 클래스를 이용한 세션 처리 방법

Cookie 클래스를 이용한 쿠키 처리 방법(앞에서 배웠다.)

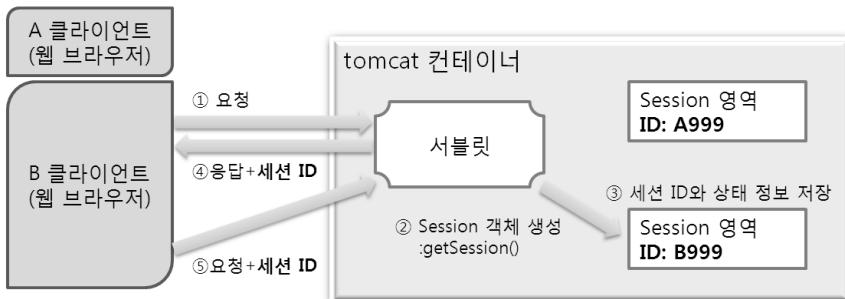
- HttpSession 클래스를 이용한 세션 처리 방법

세션(Session)이란 사용자의 상태 정보를 서버에서 관리하는 메커니즘을 의미한다. 세션의 정보는 클라이언트가 서버에 접속해서 종료될 때까지(브라우저를 종료할 때까지) 유지된다. 상태 정보가 서버에 저장되기 때문에 서버의 부하가 클 수 있다. time-out 제한을 두어 일정시간(기본 30분) 동안 요청이 없으면 서버는 세션 정보를 유지하지 않고 제거한다.

A 클라이언트가 서버에 요청하는 구조



B 클라이언트가 서버에 요청하는 구조



A 클라이언트(B 클라이언트)가 서블릿에 요청을 하면, 서블릿에서는 `getSession()` 메서드를 사용하여 session 영역을 생성하고 A 클라이언트의 고유 식별 값인 세션 ID를 생성하여 session 영역에 저장시킨다. 일반적으로 session 영역에는 장바구니 정보 및 로그인 정보 등을 저장한다.

서블릿의 실행 결과가 응답 처리될 때 자동으로 세션 ID 값이 포함되며, 동일한 브라우저에서 재요청이 발생되면 세션 ID 값을 포함하여 요청처리 된다. 서버는 재요청에 포함된 세션 ID 값을 이용하여 클라이언트와 연결 기능을 유지할 수 있다.

A 클라이언트가 일정시간(기본 30분) 동안 요청을 하지 않으면 서버는 클라이언트 정보를 제거할 목적으로 session 영역을 삭제한다. 로그아웃 같은 기능을 구현하기 위해서 session 영역을 즉시 삭제한다.

- 내장 객체 session

내장 객체 session이 세션을 지원하는 내장 객체이다. 내장 객체 session은 패키지 `javax.servlet.http`에 속하는 인터페이스 `HttpSession`이다. 내장 객체인 session은 세션 자체의 식별자와 생성 시간 정보를 제공하는 메소드인 `getId()`, `getCreationTime()` 등을 제공한다.

반환형	메소드 이름	메소드 기능
long	<code>getCreationTime()</code>	1970년 1월 1일 0시를 기준으로 하여 현재 세션이 생성된 시간까지 지난 시간을 계산하여 밀리세컨드로 반환
String	<code>getId()</code>	세션에 할당된 유일한 식별자(ID)를 String 타입으로 반환
int	<code>getMaxInactiveInterval()</code>	현재 생성된 세션을 유지하기 위해 설정된 최대 시간을 초의 정수형으로 반환. 지정하지 않으면 기본 값은 1800초 즉 30분이며, 기본값도 서버에서 설정 가능

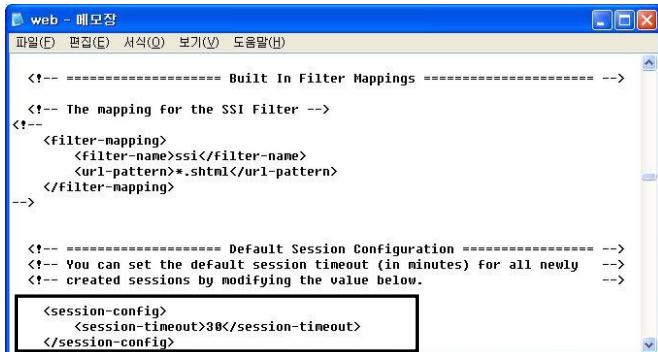
세션이 유지되는 동안에 필요한 속성 값은 session 객체의 `setAttribute(String name, Object value)`으로 저장할 수 있다. 세션의 속성값으로 저장할 수 있는 형태는 자바가 지원하는 객체이면 모두 가능하다. 저장된 속성값은 반대로 `getAttribute(String name)` 메소드를 이용해 조회할 수 있는데, 반환값 유형이 `Object`이므로 자료의 형변환이 필요하다. 내장 객체 session은 세션 처리 및 관리를 위한 다음과 같은 메소드를 제공한다.

반환형	메소드 이름	메소드 기능
<code>Object</code>	<code>getAttribute(String name)</code>	<code>name</code> 이란 이름에 해당되는 속성값을 <code>Object</code> 타입으로 반환. 해당되는 이름이 없을 경우에

		는 null을 반환.
Enumeration	getAttributeNames()	속성의 이름들을 Enumeration 타입으로 반환.
Object	getSession()	session 영역을 얻을 때 사용하며 session 영역이 없으면 새로 생성하고, 있으면 생성된 영역을 참조 request.getSession(true) 메서드와 동일한 기능 일반적으로 세션을 처음 생성하는 서블릿에서 지정
Object	getSession(false)	session 영역을 얻을 때 사용하며 session 영역이 없으면 null 값을 리턴하고, 있으면 생성된 영역을 참조 일반적으로 세션을 사용하는 서블릿에서 주로 사용
void	invalidate()	현재 생성된 세션을 삭제한다.
void	removeAttribute()	name으로 지정한 속성의 값을 제거
void	setAttribute(String name, Object value)	name으로 지정한 이름에 value값을 할당 Session scope를 따르기 때문에 브라우저를 종료하기 전까지 사용 가능 단, 기본으로 30분 이상 요청이 없을 시 제거
void	setMaxInactiveInterval(int interval)	세션의 최대 유지시간을 초 단위로 설정
boolean	isNew()	세션이 새로이 만들어졌으면 true, 이미 만들어진 세션이면 false를 반환.

- 세션의 주요 정보 조회

세션은 클라이언트의 브라우저가 서버에 접속하는 순간 생성되며 특별히 지정하지 않으면 세션의 유지 시간은 기본값으로 30분이 설정되어 있다. 세션의 유지 시간이란 서버에 접속한 후 서버에 어떠한 요청을 하지 않는 최대 시간으로, 30분 이상 서버에 전혀 반응을 보이지 않으면 세션이 자동으로 끊어진다. 이 세션 유지 시간은 서버에서 설정할 수 있는데, 톰캣에서 설치 폴더 하부 [conf]폴더에 web.xml을 살펴보면 <session-timeout>30</session-timeout>으로 30분이 지정되어 있는 것을 확인 할 수 있다.



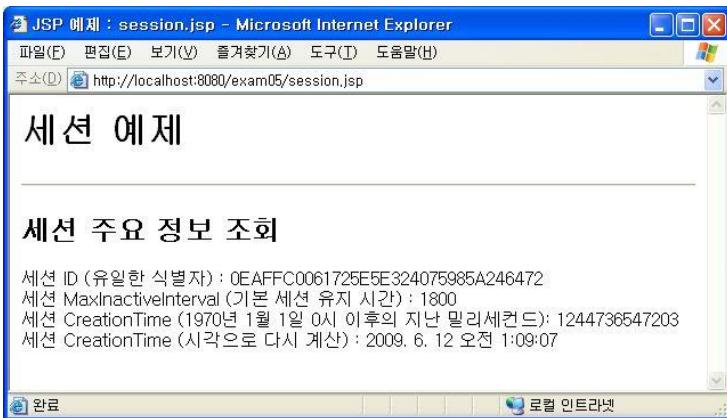
내장 객체 session 메소드 getCreationTime()은 현재 세션이 생성된 시간을 1970년 1월 1일 0시를 기준으로 지난 시간을 계산하여 결과를 밀리세컨드로 반환한다. 그러므로 이를 연, 월, 일, 시의 시간정보로 출력하려면 클래스 java.util.Date의 생성자 Date(long mseconds)를 이용하여 객체를 만든 후 출력해야

한다. 생성자 Date(long mseconds)는 인자인 밀리세컨드를 이용하여 1970년 1월 1일 0시를 기준으로 지난 시간 정보를 생성한다. 그러므로 Date(session.getCreationTime())는 세션이 생성된 시간을 연, 월, 일, 시의 시간정보의 객체를 생성한다.

session.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : session.jsp</title>
</head>
<body>
    <%@ page import="java.util.Enumeration, java.util.Date" %>
    <h1>세션 예제</h1>
    <hr><h2>세션 주요 정보 조회</h2>
    세션 ID (유일한 식별자) : <%= session.getId() %><br>
    세션 MaxInactiveInterval (기본 세션 유지 시간) :
        <%= session.getMaxInactiveInterval() %><br>
    <%
        long mseconds = session.getCreationTime();
        Date time = new Date(mseconds);
    %>
    세션 CreationTime (1970년 1월 1일 0시 이후의 지난 밀리세컨드): <%=mseconds %><br>
    세션 CreationTime (시각으로 다시 계산) : <%=time.toLocaleString() %>
</body>
</html>
```

[결과]



다시 refresh를 해도 동일한 세션이므로 모두 같은 정보를 출력한다. 그러나 브라우저를 바꾸거나 서버를 종료한 후 다시 실행하면 세션 ID와 생성 시간이 바뀌는 것을 확인할 수 있다.

- 세션에 주요 값 저장과 조회

내장 객체 session 메소드 `setAttribute(String name, Object value)`는 name과 value의 한 쌍으로 객체 Object를 저장하는 메소드로서, 세션이 유지되는 동안 저장이 필요한 자료를 저장한다. 다음 소스에서는 저장하는 세션 속성 값의 문자열이 어떠한 객체 유형이라도 가능하다.

```
session.setAttribute("id", "javajsp7");
session.setAttribute("passwd", "jdktomcat");
```

세션에 저장된 자료는 다시 `getAttribute(String name)` 메소드를 이용해 조회할 수 있는데, 반환값은 Object 유형이므로 저장된 객체로 자료유형 변환이 필요하다. 메소드 `setAttribute()`에 이용한 name인 "id"를 알고 있다면 쉽게 조회가 가능하다

```
String value = (String)session.getAttribute("id");
```

세션의 속성으로 지정한 이름을 모두 알기 위해서는 메소드 `getAttributeNames()`가 필요하다. 메소드 `getAttributeNames()`의 반환값은 인터페이스 Enumeration으로 패키지 java.util에 속한다.

메소드 `getAttributeNames()`의 반환값을 저장할 객체 e의 자료유형은 `Enumeration<String>`이다. 이와 같이 객체 e를 `Enumeration<String>`과 같이 일반화 유형으로 선언한다면 `e.nextElement()`의 반환값을 저장할 때 String을 자료유형 변환이 필요 없다는 장점이 있다. 자료유형 Enumeration은 여러 개의 내부 원소를 일렬로 저장한 구조를 지원하며, 내부 원소를 참조하기 위해 다음 두 메소드를 제공한다.

반환형	메소드 이름	메소드 기능
Boolean	<code>hasMoreElements()</code>	<code>enumeration</code> 의 내부에 더 이상 원소가 있는지 결과를 반환. 있다면 true, 없으면 false를 반환.
Object	<code>nextElement()</code>	<code>enumeration</code> 의 내부에 더 이상의 원소가 있다면 다음 원소를 반환 (요소 하나하나를 추출한다.)

sessionattribute.jsp

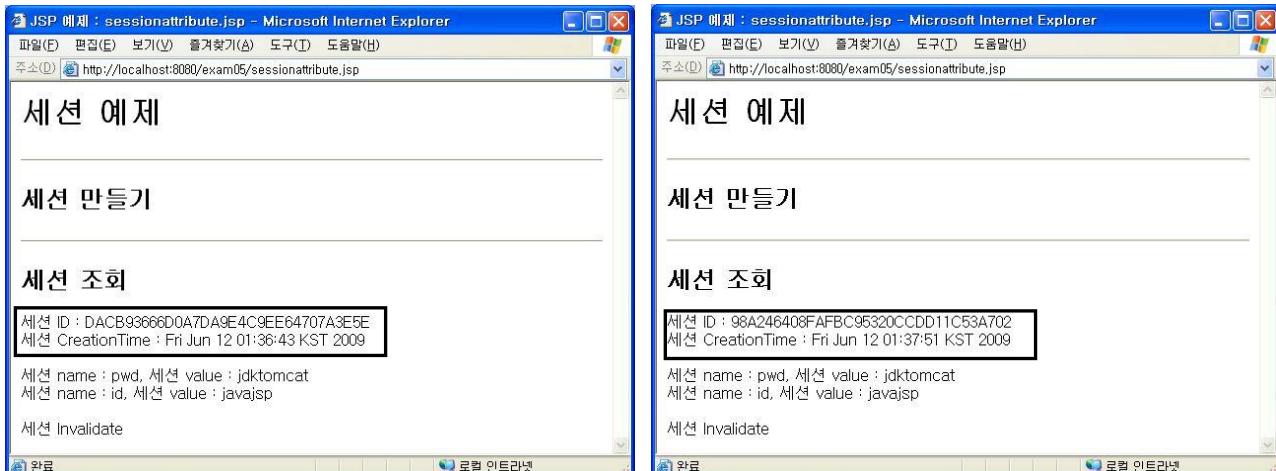
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : sessionattribute.jsp</title>
</head>
<body>
    <%@ page import="java.util.Enumeration, java.util.Date" %>
    <h1>세션 예제</h1>
    <hr><h2>세션 만들기</h2>
    <%
        session.setAttribute("id", "javajsp");
        session.setAttribute("pwd", "jdktomcat");
    %>
    <hr><h2>세션 조회</h2>
    세션 ID : <%= session.getId() %><br>
    세션 CreationTime : <%= new Date(session.getCreationTime()) %><br><br>
```

```

<%
    Enumeration<String> e = session.getAttributeNames();
    //session.getAttributeNames() 메소드는 세션의 속성 이름들을 배열 형태로 갖고있다
    while ( e.hasMoreElements() ) {
        String name = e.nextElement();
        String value = (String) session.getAttribute(name);
        out.println("세션 name : " + name + ", ");
        out.println("세션 value : " + value + "<br>");
    }
%>
<br>세션 Invalidate <% session.invalidate(); %>
</body>
</html>

```

[결과]



마지막 부분에 다음과 같이 `session.invalidate()`를 호출하므로 이전 세션을 무조건 무효화시킨다. 그러므로 다시 페이지를 refresh하면 항상 세션 ID와 생성시간이 바뀌는 것을 볼 수 있다.

- 세션 시간 설정(세션 timeout 시간 설정)과 속성 삭제

내장 객체인 `session`의 메소드 `setAttribute()`를 이용해 세션의 ID와 생성시간을 저장할 수 있으며, 메소드 `setMaxInactiveInterval(5)`를 이용해 5초 이상 서버에 반응을 하지 않으면 세션을 무효화 시킬 수 있다.

```

session.setAttribute("id", session.getId());
session.setAttribute("time", new Date(session.getCreationTime()));
session.setMaxInactiveInterval(5);

```

- 세션 속성 삭제

이미 지정된 세션 속성을 삭제하려면 메소드 `removeAttribute("id")`와 같이 속성 이름을 인자로 호출한다. 메소드 `isNew()`은 세션이 새로 만들어진 것인지 Boolean 유형으로 반환한다. 다음은 메소드 `isNew()`를 이용해 페이지에서 세션이 새로 만들어지면 세션 속성과 유효시간을 지정하며, 세션이 이미 만들어진 세션이라면 속성 이름 "id"의 속성값을 삭제하는 소스이다. 그리고 현재 페이지에서 서버에 반응을 보이

지 않은 시간을 계산하기 위해 현재 시간이 nowtime에서 이전에 참조한 시간이 저장된 beforetime을 뺀다. 여기서 beforetime은 이전에 참조한 시간을 저장하기 위한 변수이므로 소속 변수로 선언하고 페이지를 종료할 때 다시 nowtime을 beforetime에 저장한다.

- 세션을 제거하는 방법

time-out을 지정하여 제거하는 방법

setMaxInactiveInterval(second) 메서드 사용

```
session.setMaxInactiveInterval( 60*60*24 ); //24시간 유지
```

web.xml 사용

```
<web-app>
    <!-- 단위는 분(minute) -->
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>
</web-app>
```

- 즉시 제거하는 방법

invalidate() 메서드 사용

```
session.invalidate(); // 즉시 제거
```

- 세션에 저장된 특정 속성 값을 제거하는 방법

removeAttribute(name) 메서드 사용

```
session.removeAttribute(name); //name 값에 해당되는 속성 값만 제거
```

sessiontimeout.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : sessiontimeout.jsp</title>
</head>
<body>
    <%@ page import="java.util.Enumeration, java.util.Date" %>
    <h1>세션 예제</h1>
    <hr><h2>세션 만들기</h2>
    <!-- 이전 페이지 참조 시간을 저장하는 소속 변수 -->
    <%! long beforetime = new Date().getTime();%>
    <%
        long nowtime = new Date().getTime();
        if ( session.isNew() ) {
```

```

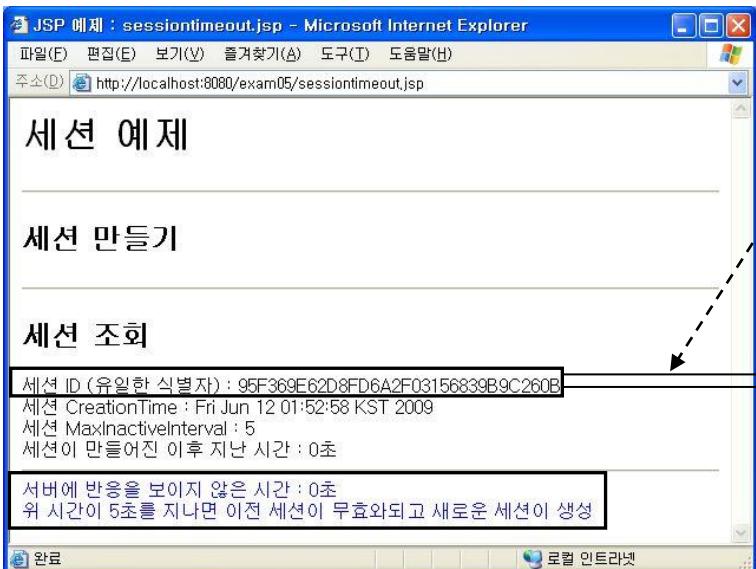
        session.setAttribute("id", session.getId());
        session.setAttribute("time", new Date(session.getCreationTime()));
        session.setMaxInactiveInterval(5);
    } else {
        session.removeAttribute("id");
    }
%>
<hr><h2>세션 조회</h2>
세션 ID (유일한 식별자) : <%= session.getAttribute("id") %><br>
세션 CreationTime : <%=session.getAttribute("time") %><br>
세션 MaxInactiveInterval : <%=session.getMaxInactiveInterval() %><br>
<% long sessiontime = nowtime - session.getCreationTime(); %>
세션이 만들어진 이후 지난 시간 : <%=sessiontime/1000 %>초
<font color=blue><hr>
<% long inactiveinterval = nowtime - beforetime; %>
서버에 반응을 보이지 않은 시간 : <%=inactiveinterval/1000 %>초 <br>
위 시간이 <%=session.getMaxInactiveInterval() %>초를 지나면
이전 세션이 무효화되고 새로운 세션이 생성</font><br>
<% beforetime = nowtime; %>
</body>
</html>

```

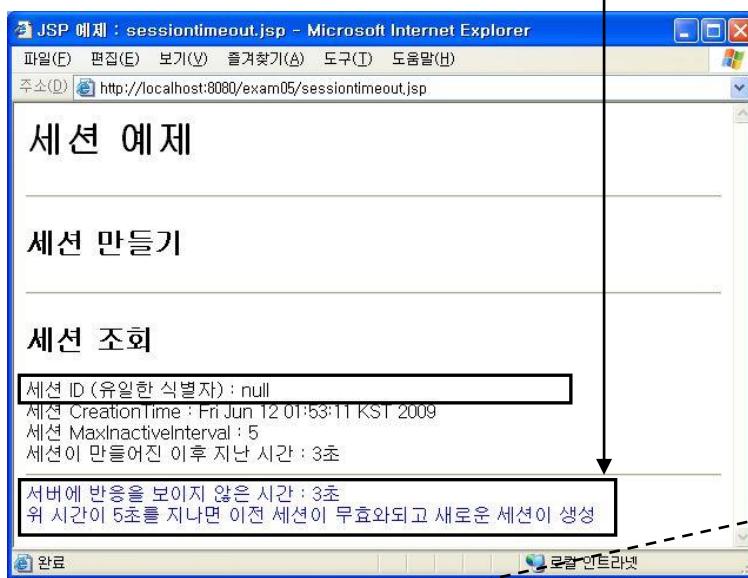
[결과]



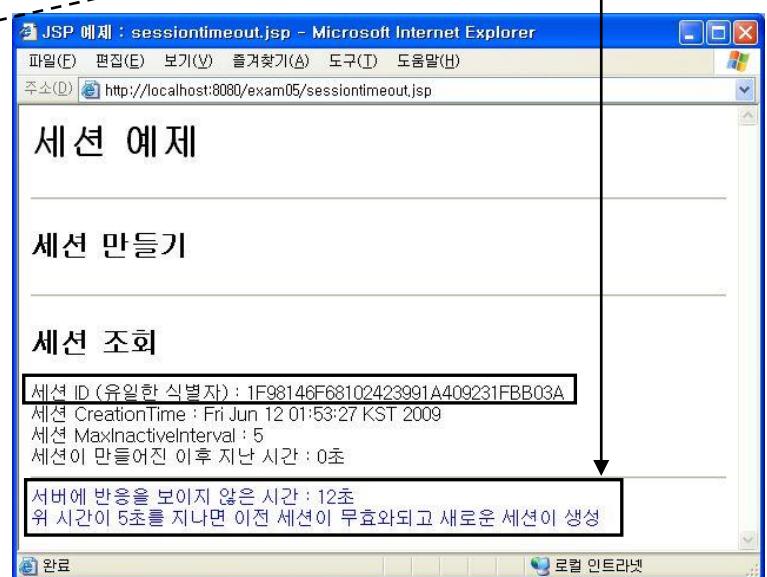
세션ID와 생성시간이 세션의 속성값으로 설정되고 조회되어 출력되는 것을 확인할 수 있다. 세션 유효 시간으로 설정한 5초 이전에 다시 페이지를 refresh하면 동일한 세션이므로 세션 ID가 삭제되어 null이 출력되는 것을 볼 수 있다. 만일 세션 유효시간으로 설정한 5초가 지난 후 페이지를 refresh하면 세션이 무효화되어 다시 새로운 세션이 설정되므로 조회되는 세션 ID가 이전 것과 다른 것을 볼 수 있다.



세션 유효시간으로 설정한 5초 이전에 다시 페이지를 refresh하면 동일한 세션이므로 세션 ID가 삭제되어 null이 출력되는 것을 볼 수 있다.



위에서 만일 세션 유효시간으로 설정한 5초가 지난 후 페이지를 refresh하면 세션이 무효화되어 다시 새로운 세션이 설정되므로 조회되는 세션 ID가 이전 것과 다른 것을 볼 수 있다.



12.6 쿠키와 세션 이용 비교

- 쿠키 이용

이미 생성하여 사용하고 있는 쿠키를 삭제하려면 setMaxAge()을 호출하여 다시 쿠키를 저장한다.

```
Cookie c = new Cookie("user", "hera");
Cookie[] cs = request.getCookies();
for(Cookie data : cs ){
    if(data.getName().equals("user")){
        c.setMaxAge(0);
        response.addCookie(c);
    }
}
```

다음은 지금까지 살펴 본 쿠키의 사용 방법을 정리한 표이다.

수행기능	이용 방법	이용 메소드
쿠키 생성	Cookie c = new Cookie("user","hera"); Cookie(String name, String value)	[Cookie 생성자]
쿠키 유효기간	c.setMaxAge(30);	void setMaxAge(int expiry)
저장 및 조회	int n = c.getMaxAge();	int getMaxAge()
쿠키 값 수정	c.setValue("king");	void setValue(String value)
쿠키 저장	response.addCookie(c)	void addCookie(Cookie c)
모든 쿠키 조회	cookie[] cs = request.getCookies();	Cookie[] getCookies()
개별 쿠키	cs[i].getValue()	String getValue()
이름 조회		
쿠키 삭제	c.setMaxAge(); response.addCookie(c);	void setMaxAge(int expiry)

- 세션 이용

세션에서 이름을 "time"으로 저장하여 사용하던 속성을 삭제하려면 메소드 removeAttribute("time") 호출 한다.

```
session.setAttribute("time", new Date());
session.removeAttribute("time")
```

수행기능	이용 방법	이용 메소드
세션 속성 설정	session.setAttribute("time", new Date());	void setAttribute(String name, Object value)
세션 속성 조회	Date d = (Date)session.getAttribute("time");	Object getAttribute(String name)
세션 유효기간	session.setMaxInactiveInterval(30)	void setMaxInactiveInterval(int n)
지정 및 조회	int n = session.getMaxInactiveInterval();	int getMaxInactiveInterval();
세션 모든 속성	Enumeration e =	Enumeration getAttributeNames();

의 이름 조회	session.getAttributeNames();	
Enumeration 처리	<pre>while(e.hasMoreElements()){ String name = (String)e.nextElement(); if(name.equals("time")) Date d = session.getAttribute(name); }</pre>	<p>[인터페이스 Enumeration 메소드]</p> <p>Boolean hasMoreElements();</p> <p>Object nextElement();</p>
세션 ID 조회	String id = session.getId();	String getId()
세션 생성 시간 조회	long ctime = session.getCreationTime()	long getCreationTime()
세션 종료	session.invalidate()	void invalidate()

-세션을 이용한 로그인 정보유지

sessionLogin.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Session Login</title>
</head>
<body>
<form action="sessionLogin2.jsp" method="post">
<table border=0 width=400 height=100>
    <tr bgcolor="yellow">
        <td align=right><font size=2>아이디 :</font></td>
        <td><input type="text" name="id" size=10></td>
    </tr>
    <tr bgcolor="yellow">
        <td align=right><font size=2>비밀번호 :</font></td>
        <td><input type="password" name="pass" size=12></td>
    </tr>
    <tr bgcolor="yellow">
        <td colspan=2 align=center>
            <input type="submit" value="로그인">
            <input type="reset" value="다시 작성">
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

sessionLogin2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<% session.setAttribute("id",request.getParameter("id")); %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
    <title>Session Login</title>
</head>
<body>
<center>
    <h3>로그인되었습니다.</h3>
    <h3>로그인 아이디 : <%=String(session.getAttribute("id")) %> </h3>
    <a href="sessionLogout.jsp">로그아웃</a>
</center>
</body>
</html>
```

sessionLogout.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<% session.removeAttribute("id"); %>
<h3>로그아웃 되었습니다.</h3>
<a href="sessionLogin.jsp">로그인 페이지로 이동</a>
```

- 세션을 이용한 장바구니 예제

productSession.html

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>상품 리스트</title>
</head>
<body>
<h1>상품 리스트</h1>
<form action="CartSave" method="GET">
    <input type="radio" name="product" value="BMW">BMW<br>
    <input type="radio" name="product" value="SM5">SM5<br>
    <input type="radio" name="product" value="K7">K7<br>
    <input type="submit" value="카드저장">
</form>
</body>
</html>
```

CartSaveServlet.java(선택한 물건 저장)

```
package com.controller;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
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("/CartSave")
public class CartSaveServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 입력 파라미터 얻기
        String product = request.getParameter("product");
        //세션객체 얻기
        HttpSession session = request.getSession();
        ArrayList<String> list = (ArrayList<String>)session.getAttribute("product");
        if( list == null ){
            list = new ArrayList<String>();
            list.add(product);
            session.setAttribute("product", list);
        }else{
            list.add(product);
        }
        out.print("<html><body>");
        out.print("Product 추가");
        out.print("<a href='CartBasket'>장바구니 보기</a>");
        out.print("</body></html>");
    }
}
```

CartBasketServlet.java(장바구니 리스트)

```
package com.controller;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
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("/CartBasket")
public class CartBasketServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("장바구니 리스트");
        //세션객체 얻기
        HttpSession session = request.getSession(false);
        session.setMaxInactiveInterval(200);
        if( session != null ){
            ArrayList<String> list = (ArrayList<String>)session.getAttribute("product");
            out.print("상품: " + list + "<br>");
        }else{
            out.print("세션 없음" + "<br>");
        }
        out.print("<a href='productSession.html'>상품 선택 페이지 </a><br>");
        out.print("<a href='CartDelete'>장바구니 비우기 </a><br>");
        out.print("</body></html>");
    }
}
```

CartDeleteServlet.java(장바구니 삭제)

```
package com.controller;

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;

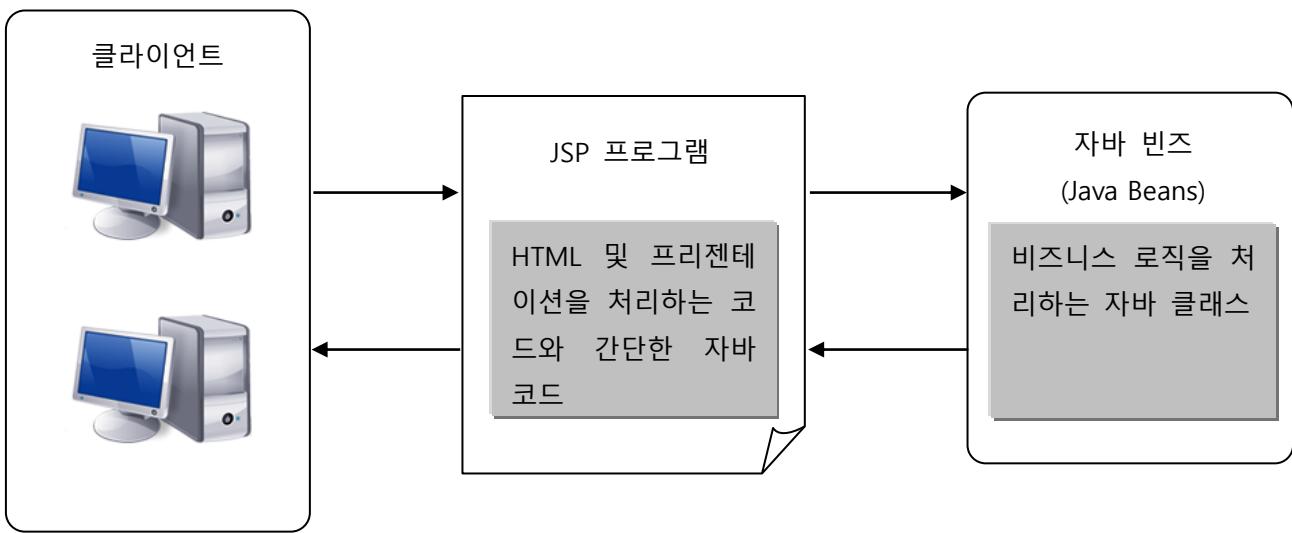
@WebServlet("/CartDelete")
public class CartDeleteServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("장바구니 비웠음!!!");
        //세션객체 얻기
        HttpSession session = request.getSession(false);
        if( session != null ){
            session.invalidate(); //세션 즉시 제거
        }else{
            out.print("세션 없음" + "<br>");
        }
        out.print("<a href='productSession.html'>상품 선택 페이지 </a><br>");
        out.print("</body></html>");
    }
}
```

13. 자바 빈즈

13.1 자바 빈즈 개요

- 자바빈즈란? 데이터를 표현하는 것을 목적으로 하는 자바 클래스이다.

JSP 프로그램의 장점 중의 하나는 비즈니스 로직 부분과 프리젠테이션 부분을 나누어 코딩할 수 있다는 점이다. 그러나 지금까지의 JSP 프로그램은 하나의 JSP 프로그램 내부에 로직 부분의 자바 코드와 프리젠테이션 부분의 HTML 코드가 복잡하게 구성된 것이 사실이다. 자바 빈즈는 프로그램의 비즈니스 로직 부분과 프리젠테이션 부분을 분리해서, 비즈니스 로직 부분을 담당하는 자바 프로그램 단위라 할 수 있다. 그러므로 자바 빈즈를 이용하면 JSP 페이지가 복잡한 자바 코드로 구성되는 것을 피하고, JSP 페이지에는 HTML 코드와 쉽고 간단한 자바 코드만을 구성할 수 있다.



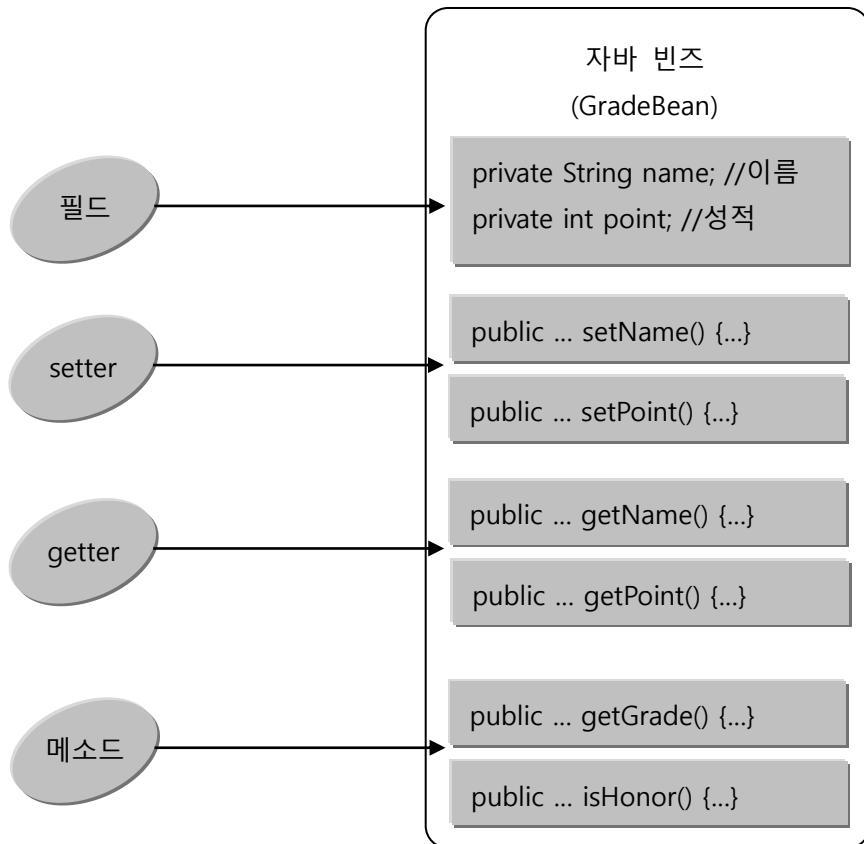
자바 빈즈(Java Beans)는 자바 프로그램에서 특정한 작업인 비즈니스 로직을 독립적으로 수행하는 하나의 프로그램 단위이다. 자바뿐만 아니라 일반 프로그래밍 분야 중 하나의 큰 프로그램에서 독립적으로 수행되는 하나의 작은 프로그램 부품을 컴포넌트(Component)라고 부른다. 그러므로 자바 빈즈는 자바 프로그램에서의 컴포넌트이며, 넓은 의미로 자바 빈즈는 자바의 모든 클래스를 의미할 수도 있다. 이러한 자바 빈즈를 잘 활용한다면 한 번 작성된 자바 빈즈를 여러 응용 프로그램에서 재사용하여 프로그램의 개발 기간도 단축할 수 있는 장점을 가져올 수 있다.

전자제품과 같이 소프트웨어(프로그램)도 부품화하여 개발할 수 있는데 이와 같은 개념을 컴포넌트 프로그래밍 방식이라고 한다. 이를 자바에서 빈(Bean)이라고 하고 일반적으로 컴포넌트라고 한다. 하나의 큰 시스템을 구축할 경우 처음부터 끝까지 모두 다 개발하는 방식이 아닌 기능 모듈별로 프로그램을 작게 만들어 하나씩, 하나씩 조립을 하는 방식을 통해 큰 시스템을 구축할 수 있다. 이와 같이 함으로써 또 다른 시스템을 개발하려고 한다면 시간 단축 및 비슷한 작업의 반복, 버그 발생 확률 등의 비효율적인 부분을 해결할 수 있다. 그러므로 하나의 시스템을 구축할 때 다른 개발시에도 유용하게 사용할 기능을 빈(컴포넌트)으로 작성하여 사용하게 되면 다음 개발에도 그 빈을 활용하게 되므로 효율적으로 개발 할 수 있을 것이다.

- 자바 빈즈의 구성

자바 빈즈는 일반 자바 클래스이다. 자바 빈즈는 소속 변수(member variables)인 필드와 메소드로 구성

된다. 자바 빈즈의 필드는 일반적으로 외부에서 참조할 수 없도록 private로 선언되며, 외부에서 자바 빈즈의 필드를 참조하기 위해서 public으로 선언된 setter와 getter를 제공한다. 즉 메소드 중에서 특히 필드에 값을 저장하는 메소드를 setter라 하고, 필드에 저장된 값을 반환하는 메소드를 getter라 한다. 자바 빈즈는 getter와 setter 외에 필요하면 다양한 메소드를 제공할 수 있다. 자바 빈즈의 필드를 참조하기 위한 메소드를 setter, getter라고 부르는 이유는, 한 예로 자바 빈즈의 필드 userid를 참조하는 setter, getter를 각각 setUserId(), getUserId()로 명명하기 때문이다.



13.2 자바 빈즈 태그

자바 빈즈를 이용하는 액션 태그는 XML 태그와 같은 형식이며 다음과 같이 3개를 제공한다.

액션	내용
<jsp:useBean id="login" class="LoginBean" />	JSP와 연관시켜 자바 빈즈를 생성
<jsp:setProperty name="login" property="pass" />	생성된 자바 빈즈의 객체를 이용해 setter에 속성 값을 전달
<jsp:getProperty name="login" property="pass" />	생성된 자바 빈즈의 객체를 이용해 getter로 속성 값을 반환

태그 <jsp:useBean ... />에서 이용하는 속성은 id, class, scope가 있으며, <jsp:setProperty ... />에서 이용하는 속성은 name, property, param, value가 사용되고, <jsp:getProperty ... />에서 이용하는 속성은 name, property 2개이다.

액션	속성	값유형	설명
	id	문자열	JSP 페이지내에서 자바 빈즈의 참조 변수를 저장하는 변수 이름을 지정
	class	문자열	생성할 자바 빈즈의 클래스 이름
<jsp:useBean ... />	scope	page request session application	자바 빈즈의 유효 범위를 나타내며 지정하지 않으면 기본값은 page
	name	문자열	<jsp:useBean>에서 지정한 id로 지정
	property	문자열	자바 빈즈의 setter()의 이름 setName()에서 set를 제거한 name으로 지정하며, 값이 "*"이면 파라미터의 모든 값을 지정하는 의미.
<jsp:setProperty ... />	param	문자열	속성 property와 함께 쓰이며, 지정된 파라미터로 전달받은 파라미터의 이름을 지정
	value	문자열	속성 property와 함께 쓰이며, 자바 빈즈의 setter()의 setName(value)에 지정하는 인자(매개변수)값인 value를 지정
	name	문자열	<jsp:useBean>에서 지정한 id로 지정
<jsp:getProperty ... />	property	문자열	자바 빈즈의 getter() 이름 getName()에서 name으로 지정

태그 <jsp:useBean ... />에서 속성 scope는 자바 빈즈의 유효 범위를 나타내는데 page, request, session, application 중에 하나의 값을 가지며, 지정하지 않으면 기본값은 page이다.

액션	내용
page	자바 빈즈가 현재 JSP 페이지 내에서만 사용 가능하며, 기본 값이므로 특별히 지정하지 않으면 이 옵션이 적용. 가장 좁은 범위 scope값.
request	JSP 페이지는 request 객체가 영향을 미치는 모든 JSP 페이지까지 자바 빈즈 이용 가능
session	세션이 유효한 페이지까지 자바 빈즈 이용 가능
application	응용 프로그램의 모든 페이지에서 자바 빈즈 객체의 사용이 가능하며, 이 값의 장 넓은 범위 scope 값

- 자바 빈즈 태그의 이용

태그 <jsp:useBean ... />은 JSP 프로그램에서 자바 빈즈를 이용하려는 선언 문장에 해당한다. 태그 <jsp:useBean ... />은 적어도 속성 id와 class가 있어야 하는데, id는 객체 참조를 저장하는 변수 이름이며 class는 객체 참조의 클래스 이름이다.

```
<jsp:useBean id="test" class="ClassName" />
<% ClassName test = new ClassName(); %>
```

위와 같이 scope를 지정하지 않으면 기본 값으로 page를 말한다.

```
<jsp:useBean id="test" class="ClassName" scope="page" />
```

자바 빈즈의 이용 범위를 모든 응용 프로그램 범위로 지정하려면 다음과 같이 속성 scope를 application 으로 지정한다.

```
<jsp:useBean id="test" class="ClassName" scope="application" />  
<jsp:setProperty name="test" property="name" value="홍길동" />
```

태크 <jsp:setProperty ... />는 이미 선언된 자바 빈즈에서 속성 property로 지정된 이름을 갖는 메소드 setter를 호출하는 문장이다. 태그 <jsp:setProperty ... />는 적어도 속성 name과 property는 있어야 하며 속성 name은 반드시 태그 <jsp:setProperty id="test" ... />에서 이미 지정한 id값과 일치해야 한다. 태그 <jsp:setProperty ... />의 속성 property는 호출할 setter 이름이 setName()이라면 property="name"으로 지정하며, 속성 value는 메소드 setter를 호출할 때의 인자 값이다.

```
<jsp:setProperty name="test" property="name" value="홍길동" />  
<% test.setName("홍길동"); %>
```

태그 <jsp:setProperty ... />에서 속성 property는 다음 4개 중에 하나의 형태로 이용할 수 있다.

```
<jsp:setProperty name="test" property="*" />  
<jsp:setProperty name="test" property="name" />  
<jsp:setProperty name="test" property="name" value="username" />  
<jsp:setProperty name="test" property="name" value="홍길동" />
```

<jsp:setProperty ... />에서 속성 name과 property가 있으면 property로 지정된 같은 이름으로 파라미터 인자를 이용하는 문장이다.

```
<jsp:setProperty name="test" property="name" />  
<% test.setName( request.getParameter("name") ); %>
```

<jsp:setProperty ... />에서 속성 name과 property, param이 모두 있으면 지정된 param으로 파라미터 인자를 이용하는 문장이다.

```
<jsp:setProperty name="test" property="name" param="username" />  
<% test.setName( request.getParameter("username") ); %>
```

<jsp:getProperty ... />는 2개의 속성 name과 property가 모두 있어야 하며 속성 name은 반드시 태그 <jsp:useBean id="test" ... />에서 지정한 id 값과 일치해야 한다. 태그 <jsp:getProperty ... />의 속성

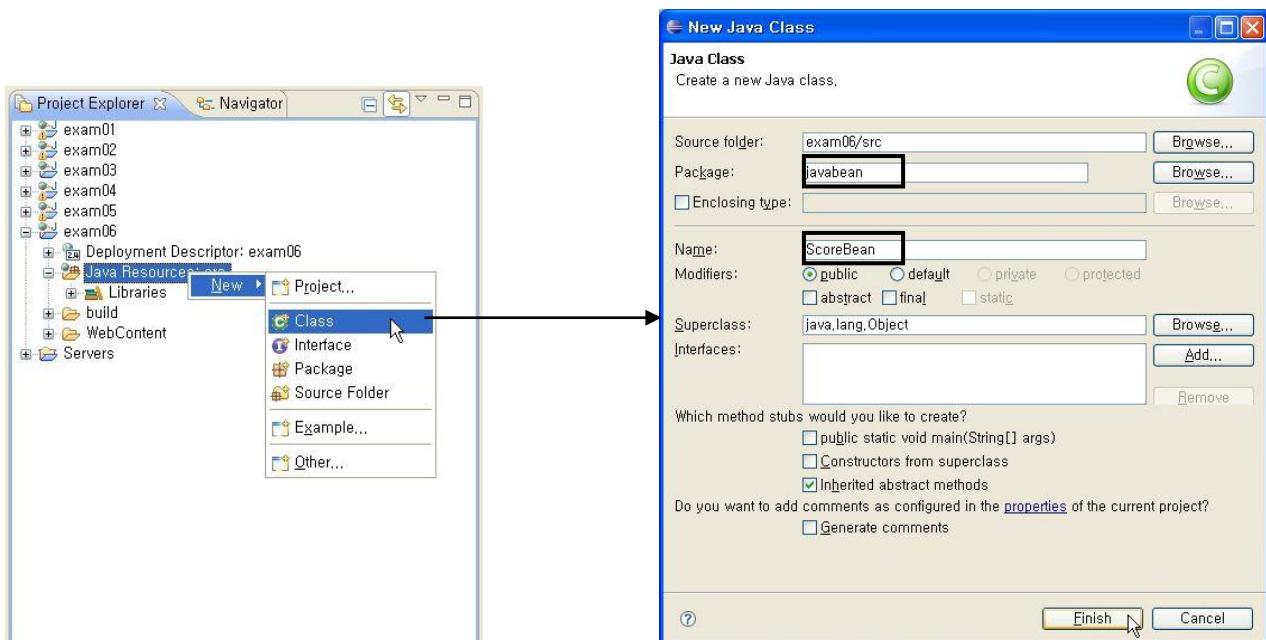
property는 호출할 getter이름이 getName()이라면 property="name"으로 지정한다.

```
<jsp:useBean id="test" class="ClassName" scope="application" />  
  
<jsp:getProperty name="test" property="name" />  
  
<%=test.getName(); %>
```

13.3 자바 빈즈 사용

- 자바 빈즈를 이용한 자료 값의 저장과 처리

자바 빈즈를 개발하고 자바 빈즈 태그를 이용하여 학생의 이름과 성적 정보를 저장하여 조회하는 프로그램을 작성해보자. 그리고 자바 빈즈는 일반 자바 프로그램으로, 이를립스에서 클래스로 생성한다. 자바 빈즈의 클래스를 만들기 위해서는 프로젝트 [exam06] 하부 [Java Resources: src]에서 오른쪽 마우스 클릭 후 메뉴 [new]/[Class]를 선택한다. 대화상자 [New Java Class]에서 [Package]와 [Name]에 각각 원하는 패키지 이름과 클래스 이름을 입력한다. 패키지는 관련된 클래스가 모여있는 폴더로 주로 소문자로 이름을 작성해 주면 된다.



```
package javabean;  
  
public class ScoreBean {  
    private String name;      //이름  
    private int point;        //성적  
}
```

이제 자바 빈즈의 정보를 저장, 조회하는 getter와 setter를 만든다. getter는 메소드 이름 getXxxx()으로 만들며 setter는 setXxxx(type xxxx)으로 작성하는데, 메소드 이름 xxxx는 필드의 이름을 말한다. 메소드 이름에서 get과 set 다음에 나오는 첫 글자는 대문자로 작성하는 것이 관례이다. getter와 setter는 소속 변수 중에서 저장과 조회가 필요한 필드에 대하여 생성한다.

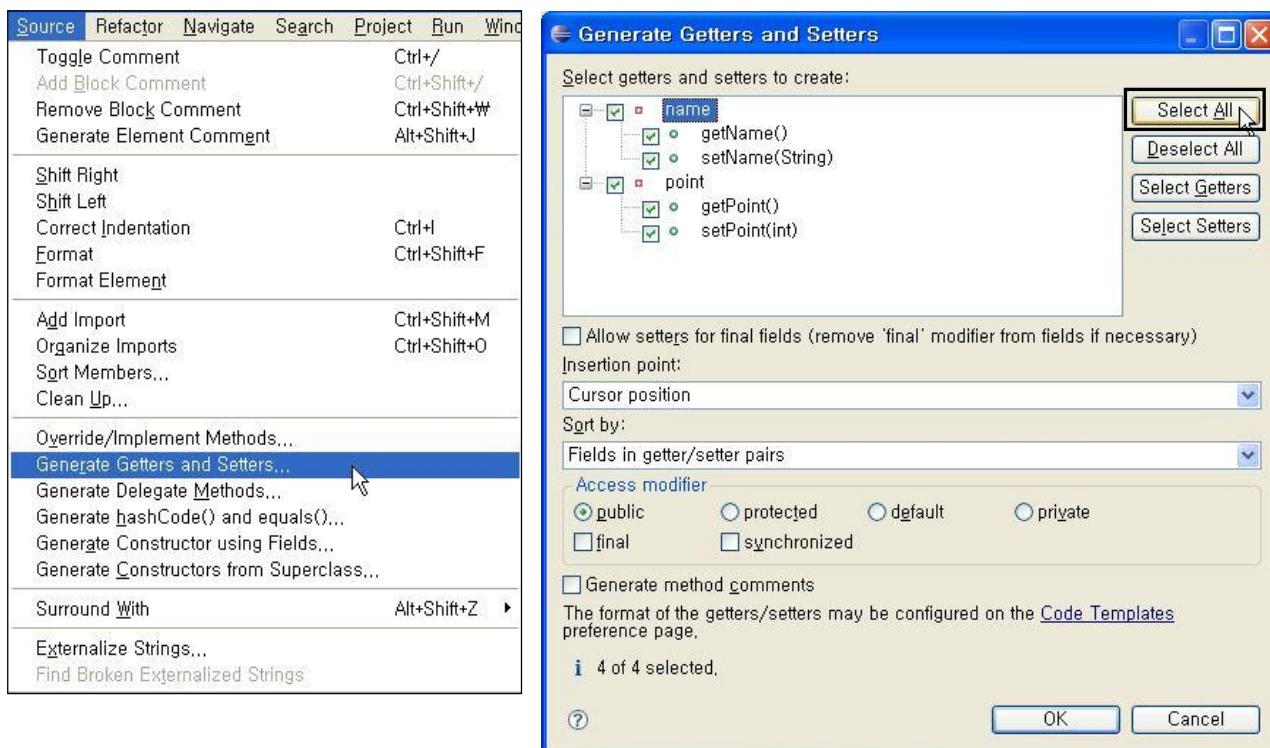
```

public String getName() {
    return name;
}

public void setName(String name){
    this.name = name;
}

```

getter와 setter는 이클립스에서 메뉴 [source]/[Generate Getter and Setter ...]를 이용하여 일괄적으로 생성할 수 있다.



```

package javabean;

public class ScoreBean {
    private String name;          //이름
    private int point;           //성적

    //메뉴 [source]/[Generate Getter and Setters ...]를 이용 자동 생성
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPoint() {
        return point;
    }
}

```

```

public void setPoint(int point) {
    this.point = point;
}
}

```

- 자바 빈즈에 정보를 저장, 조회하는 JSP 프로그램

태그 <jsp:useBean ... />를 이용하여 작성된 자바 빈즈를 생성할 수 있다. 속성 id는 적절한 이름으로 명명하고, class는 패키지 이름을 포함한 자바 빈즈의 클래스 이름으로 지정하며, scope는 4가지 중의 하나를 지정한다.

```

<jsp:useBean id="score" class="javabean.ScoreBean" scope="page" />
<% javabean.ScoreBean score = new javabean.ScoreBean() %>

```

위와 같은 <jsp:useBean ... /> 태그는 자료유형 javabean.ScoreBean으로 변수 이름 score를 선언하는 의미와 같다. 그러므로 id="score"로 지정한 score는 javabean.ScoreBean 객체를 저장한 참조 변수로서 계속 이용이 가능하다.

자바 빈즈에 정보를 저장하려면 태그 <jsp:setProperty ... />를 이용한다. 즉 태그 <jsp:setProperty ... />는 자바 빈즈의 setter를 호출하는 방법이다. 속성은 name은 <jsp:useBean ... /> 태그의 id에 지정한 score를 반드시 지정해야 하며, 속성 property는 호출할 setter의 이름인 setName()에서 set을 제거한 "name"으로 지정한다. 마지막으로 속성 value는 setter에 지정할 매개 변수의 값으로 value="홍길동"이면 setName("홍길동")을 호출하는 것을 의미한다. 즉 다음 태그는 자바 빈즈인 javabean.ScoreBean 객체에서 메소드 score.setName("홍길동")을 호출하는 것과 같다.

```

<jsp:setProperty name="score" property="name" value="홍길동" />
<% score.setName("홍길동"); %>

```

마찬가지로 점수를 자바 빈즈에 저장하려면 다음과 같이 property를 "point"로 value를 원하는 점수로 지정한다. 다음 <jsp:setProperty ... /> 태그는 다음과 같은 자바 소스를 의미한다.

```

<jsp:setProperty name="score" property="point" value="85" />
<% score.setPoint(85); %>

```

자바 빈즈에 정보를 조회하려면 태그 <jsp:getProperty ... />를 이용한다. 즉 태그 <jsp:getProperty ... />는 자바 빈즈의 getter를 호출하는 방법이다. 속성 name은 <jsp:useBean ... /> 태그의 id에 지정한 score를 반드시 지정해야 한다. 또한 속성 property는 자바 빈즈에 구현한 getter()인 getName()이름에서 get 다음의 이름인 "name"으로 지정한다. 다음 <jsp:getProperty ... /> 태그는 다음과 같은 자바 소스를 의미한다.

```

<jsp:getProperty name="score" property="name" />
    ↓      ↓
<% out.println(score.getName()); %> 또는 <%= score.getName(); %>

```

마찬가지로 점수를 자바 빈즈에 저장된 점수와 학점을 출력하려면 위와 같은 방법으로 작성하면 된다.

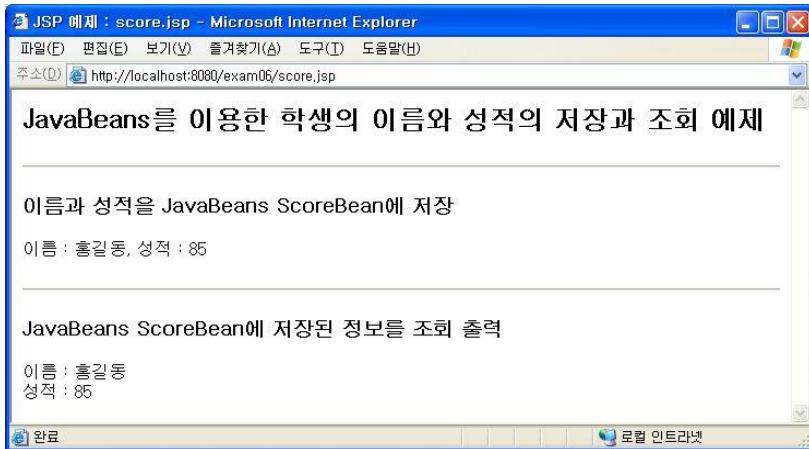
score.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<jsp:useBean id="score" class="javabean.ScoreBean" scope="page" />
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : score.jsp</title>
</head>
<body>
    <h2>JavaBeans를 이용한 학생의 이름과 성적의 저장과 조회 예제</h2>
    <hr>
    <h3>이름과 성적을 JavaBeans ScoreBean에 저장</h3><p>
        이름 : <%= "홍길동"%>,
        성적 : <%= "85" %><p>
        <jsp:setProperty name="score" property="name" value="홍길동"/>
        <jsp:setProperty name="score" property="point" value="85"/>
        <hr>
        <h3>JavaBeans ScoreBean에 저장된 정보를 조회 출력</h3><p>
        이름 : <jsp:getProperty name="score" property="name" /><br>
        성적 : <jsp:getProperty name="score" property="point" /><br>
    </body>
</html>

```

[결과]



- 자바 빈즈 태그를 사용하지 않고 자바 빈즈의 이용

자바 빈즈 관련 태그를 이용하지 않고 자바 빈즈를 사용 할 수 있다. 그러나 자바 빈즈 관련 태그를 이용하지 않으면 자바 빈즈의 사용 범위인 page, request, session, application을 선택할 수 없다.

scorenotag.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : scorenotag.jsp</title>
</head>
<body>
    <h2>태그를 사용하지 않는 방법으로 JavaBeans를 이용하는 예제</h2>
    <% javabean.ScoreBean score = new javabean.ScoreBean(); %>
    <hr>
    <h3>이름과 성적을 JavaBeans ScoreBean에 저장</h3><p>
        이름 : <%= "홍길동"%>,
        성적 : <%= "85" %><p>
        <% score.setName("홍길동"); %>
        <% score.setPoint(85); %>
        <hr>
        <h3>JavaBeans ScoreBean에 저장된 정보를 조회 출력</h3><p>
        이름 : <% out.println(score.getName()); %><br>
        성적 : <% out.println(score.getPoint()); %>
    </body>
</html>
```

13.4 자바 빈즈를 이용한 폼 입력 처리

- 폼의 입력 자료를 자바 빈즈에 저장(폼 구성과 프로그램의 내용)

예제 score.jsp와 자바 빈즈 javabean.ScoreBean를 수정하고, 이름과 점수를 폼으로 입력 받아 처리하는 프로그램을 작성해 보자. 이름과 점수를 입력 폼을 구성하는 프로그램은 grade.html로 하고, grade.jsp에서 입력받은 폼 정보를 다시 자바 빈즈에 전달하는 프로그램은 grade.jsp로 하며, 이름과 점수를 저장하여 그 점수에 해당하는 학점을 반환하는 자바 빈즈 프로그램은 javabean.GradeBean으로 구성한다.

- 자바 빈즈 및 프로그램 구현(사용자 입력 폼 작성)

폼으로 이름과 점수를 입력 받는 HTML 문서를 작성하자. 폼에서 이름과 점수의 [name] 속성값으로 지정한 name과 point를 잘 기억하여 자바빈즈를 작성할 때 소속 변수의 이름을 name과 point로 명명한다.

grade.html

```
<!DOCTYPE html>
<html>
```

```

<head>
<meta charset="UTF-8">
<title>예제 grade.html</title>
</head>
<body>
    <h2> 이름과 점수를 입력하세요 </h2>
    <form method="post" action="grade.jsp">
        이름 : <input type="text" name="name" size="16"><br>
        점수 : <input type="text" name="point" size="3"><p>
        <input type="submit" value="입력완료" >
        <input type="reset" value="다시쓰기" >
    </form>
</body>
</html>

```

- 자바 빈즈 작성

클래스 GradeBean은 패키지 javabean으로, 이름과 점수 정보를 저장할 필드는 name과 point로 선언한다. 필드 이름 name과 point는 이름과 점수를 입력 받는 폼에서 사용한 [name] 속성 값인 name과 point와 동일하게 명명한다. 이러한 필드 이름의 사용 관례는 반드시 지켜야 할 규정은 아니나 소속 변수 getter와 setter를 편하게 만들기 위한 방법이다.

```

package javabean;
public class GradeBean {
    private String name;           //이름
    private int point;             //성적
}

//메뉴 [source]/[Generate Getter and Setters ...]를 이용 자동 생성
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getPoint() {
    return point;
}

public void setPoint(int point) {
    this.point = point;
}

//성적의 학점을 계산하는 메소드는 직접 구현
public String getGrade() {
}

```

```

String grade = "";
if (point >= 90)
    grade = "A";
else if (point >= 80)
    grade = "B";
else if (point >= 70)
    grade = "C";
else if (point >= 60)
    grade = "D";
else
    grade = "F";
return grade;
}
}

```

- 자바 빈즈를 이용한 정보의 저장 및 조회

태크 <jsp:useBean ... />를 이용하여 작성된 자바 빈즈를 생성할 수 있다. 속성 id는 score로 명명하고, class는 패키지 이름을 포함한 자바 빈즈의 클래스 이름으로 지정하며, scope는 4가지 중의 하나를 지정한다.

```
<jsp:useBean id="score" class="javabean.GradeBean" scope="page" />
```

자바 빈즈에 정보를 저장하려면 태크 <jsp:setProperty ... />를 이용한다. 속성 name은 <jsp:useBean ... /> 태그의 id에 지정한 score를 반드시 지정해야 하며, 속성 property는 자바 빈즈 객체에서 호출할 메소드 이름인 setName()과 setPoint()에서 set을 제거한 "name"과 "point"로 지정한다. 마지막으로 속성 param은 폼에서 지정한 입력항목의 name 값인 "name"과 "point"로 지정한다.

```
<jsp:setProperty name="score" property="name" param="name" />
<jsp:setProperty name="score" property="point" param="point" />
```

위 태그는 다음과 같은 자바 소스를 의미한다.

```
score.setName(request.getParameter("name"));
score.setPoint(request.getParameter("point"))
```

위 소스에서 속성 **property**와 **param**값이 같으므로 다음과 같이 속성 **property** 하나만을 기술해도 상관없다. 만일 폼의 이름과 필드의 이름이 다르면 태그 <jsp:setProperty ... />에서 폼의 이름은 param에, 필드의 이름은 property에 기술한다.

```
<jsp:setProperty name="score" property="name" />
<jsp:setProperty name="score" property="point" />
```

자바 빈즈에 정보를 조회하려면 태그 <jsp:getProperty ... />를 이용한다. 호출해야 할 메소드가 각각 getName(), getPoint()이므로 속성 property를 각각 name과 point로 지정한다.

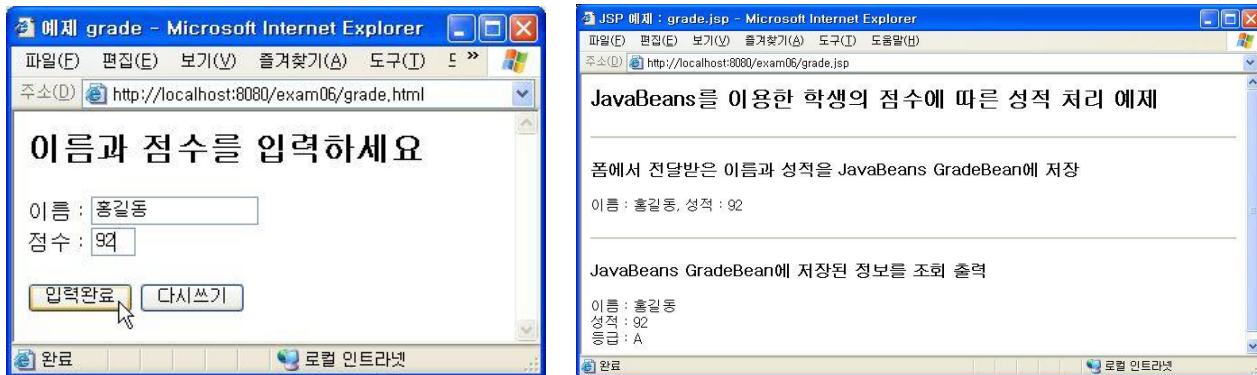
```
<jsp:getProperty name="score" property="name" />
```

```
<jsp:getProperty name="score" property="point" />
```

grade.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : grade.jsp</title>
</head>
<body>
    <h2>JavaBeans를 이용한 학생의 점수에 따른 성적 처리 예제</h2>
    <% request.setCharacterEncoding("UTF-8"); %>
    <jsp:useBean id="score" class="javabean.GradeBean" scope="page" />
    <hr>
    <h3>폼에서 전달받은 이름과 성적을 JavaBeans GradeBean에 저장</h3><p>
        이름 : <%= request.getParameter("name") %>,
        성적 : <%= request.getParameter("point") %><p>
        <jsp:setProperty name="score" property="name" param="name" />
        <jsp:setProperty name="score" property="point" param="point" />
        <HR>
        <h3>JavaBeans GradeBean에 저장된 정보를 조회 출력</h3><p>
        이름 : <jsp:getProperty name="score" property="name" /><br>
        성적 : <jsp:getProperty name="score" property="point" /><br>
        등급 : <jsp:getProperty name="score" property="grade" /><br>
    </body>
</html>
```

[결과]



- 태그 **<jsp:setProperty name="score" property="*" />**

여기서 폼을 구성하는 입력이 하나 이상이라면 property 속성 값 "*"으로 표현하면 된다.

gradeall.jsp

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

```

<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : gradeall.jsp</title>
</head>
<body>
    <h2>JavaBeans를 이용한 학생의 점수에 따른 성적 처리 예제</h2>
    <% request.setCharacterEncoding("UTF-8"); %>
    <jsp:useBean id="score" class="javabean.GradeBean" scope="page" />
    <hr>
    <h3>폼에서 전달받은 이름과 성적을 JavaBeans GradeBean에 저장</h3><p>
    이름 : <%= request.getParameter("name") %>,
    성적 : <%= request.getParameter("point") %><p>
    <b><jsp:setProperty name="score" property="*" /></b>
    <HR>
    <h3>JavaBeans GradeBean에 저장된 정보를 조회 출력</h3><p>
    이름 : <jsp:getProperty name="score" property="name" /><br>
    성적 : <jsp:getProperty name="score" property="point" /><br>
    등급 : <jsp:getProperty name="score" property="grade" /><br>
</body>
</html>

```

13.5 학생 정보 처리 자바 빈즈

학생 아이디, 학생 이름, 학생 번호, 태어난 해, 암호, 이메일 주소 6개의 정보를 입력 받아 자바 빈즈 javabean.StudentBean에 정보를 저장하는 조회하는 프로그램을 작성하자.

- student.html

```

<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 : StudentBean 이용</title>
<style type="text/css">
body, table, input{font:9pt 돋움;}
</style>
</head>
<body>
<form name="student" method="post" action="student.jsp">
<table cellspacing="1" cellpadding="2" border="1" bordercolor="#000000" align="center">
    <tr bgcolor="yellow">
        <th align="center" colspan="2">학생 정보</th>
    </tr>

```

```

<tr>
    <td>학생 아이디 </td>
    <td><input type="text" name="id" > </td>
</tr>
<tr>
    <td style="letter-spacing:3px">학생 이름 </td>
    <td><input type="text" name="name" > </td>
</tr>
<tr>
    <td style="letter-spacing:3px">학생 번호 </td>
    <td><input type="text" name="snum" > </td>
</tr>
<tr>
    <td style="letter-spacing:3px">태어난 해 </td>
    <td>
        <select name="year">
            <option value="1970" selected>1970</option>
            <option value="1971">1971</option>
            <option value="1972">1972</option>
            <option value="1973">1973</option>
            <option value="1974">1974</option>
            <option value="1975">1975</option>
            <option value="1976">1976</option>
        </select>
    </td>
</tr>
<tr>
    <td style="letter-spacing:3px">비밀 번호 </td>
    <td><input type="password" name="pass" > </td>
</tr>
<tr>
    <td style="letter-spacing:5px">이 메 일 </td>
    <td><input type="text" name="email" > </td>
</tr>
<tr>
    <td align="center" colspan="2" > <input type="submit" value="입력완료" >
        <input type="reset" value="다시쓰기" > </td>
    </td>
</tr>
</table>
</form>

```

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

- StudentBean.java

```
package javabean;  
import java.util.Calendar;  
public class StudentBean {  
    private String id;          //ID  
    private String name;        //이름  
    private String snum;        //학번  
    private int year;           //생년  
    private String pass;        //암호  
    private String email;       //전자메일  
    //메뉴 [source]/[Generate Getter and Setters ...]를 이용 자동 생성  
    public String getId() {  
        return id;  
    }  
    public void setId(String id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getSnum() {  
        return snum;  
    }  
    public void setSnum(String snum) {  
        this.snum = snum;  
    }  
    public int getYear() {  
        return year;  
    }  
    public void setYear(int year) {  
        this.year = year;  
    }  
    public String getPass() {  
        return pass;
```

```

    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    //태어난 해를 이용하여 현재의 나이를 반환하는 메소드는 직접 구현
    public int getAge() {
        int curyear = Calendar.getInstance().get(Calendar.YEAR);
        System.out.println(curyear);
        return curyear - year + 1;
    }
}

```

- student.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSP 예제 stdudent.jsp</title>
</head>
<body>
    <h2>JavaBean StudentBean을 이용한 예제</h2>
    <% request.setCharacterEncoding("UTF-8"); %>
    <jsp:useBean id="student" class="javabean.StudentBean" scope="page" />
    <hr>
    <h3>폼에서 전달받은 학생정보를 그대로 자바빈 StudentBean에 저장</h3><p>
    <jsp:setProperty name="student" property="*" />
    <hr>
    <h3>JavaBean StudentBean에 저장된 정보를 조회 출력</h3><p>
    학생 ID : <jsp:getProperty name="student" property="id" /><br>
    학생 이름 : <jsp:getProperty name="student" property="name" /><br>
    학생 번호 : <jsp:getProperty name="student" property="snum" /><br>
    나이(생년) : <%=student.getAge() %>(<jsp:getProperty name="student"
                                                property="year" />)<br>
    암호 : <jsp:getProperty name="student" property="pass" /><br>

```

```
전자메일 : <jsp:getProperty name="student" property="email" /> <br>
</body>
</html>
```

[결과]

The screenshot displays two Microsoft Internet Explorer windows side-by-side.

Left Window (JSP 예제 : StudentBean 이용 - Microsoft Internet Explorer):

- Title bar: JSP 예제 : StudentBean 이용 - Microsoft Internet Explorer
- Address bar: http://localhost:8080/exam06/student.html
- Content area: A form titled "학생 정보" (Student Information) with the following fields:
 - 학생 아이디: language
 - 학 生 이 름: 홍길동
 - 학 生 번 호: 90010001
 - 태 어 난 해: 1970
 - 비 밀 번 호: ****
 - 이 메 일: lang1234@naver.com
- Buttons at the bottom: "입력완료" (Input Complete) and "다시쓰기" (Reuse)

Right Window (JSP 예제 student.jsp - Microsoft Internet Explorer):

- Title bar: JSP 예제 student.jsp - Microsoft Internet Explorer
- Address bar: http://localhost:8080/exam06/student.jsp
- Content area:

JavaBean StudentBean을 이용한 예제

폼에서 전달받은 학생정보를 그대로 자바빈 StudentBean에 저장

JavaBean StudentBean에 저장된 정보를 조회 출력

```
학생 ID : language
학생 이름 : 홍길동
학생 번호 : 90010001
나이(생년) : 40(1970)
암호 : 1234
전자메일 : lang1234@naver.com
```

14. 파일 업로드

14.1. 파일 업로드의 원리

파일 업로드는 클라이언트에서 서버 측으로 데이터를 보내는 것으로 데이터를 받는 곳인 서버에서 파일을 받을 수 있도록 처리해 주어야 한다.

- 파일 전송 방식

일반 파라미터를 전송할 때 사용하는 인코딩과 파일 업로드 할 때 사용하는 인코딩은 서로 다르다. 앞서 HTTP의 데이터 전송 방식은 크게 GET 방식과 POST 방식이 존재한다고 했었는데 이 두방식의 차이는 파라미터 데이터를 요청 URL로 전송하느냐 아니면 스트림으로 전송하느냐의 차이였다. 스트림 기반의 전송 방식인 POST 방식은 또 다시 다음의 두가지 인코딩 방식에 따라서 전송하는 데이터 형식이 달라진다.

- application/x-www-form-urlencoded
- multipart/form-data

지금까지 사용한 예제들은 application/x-www-form-urlencoded 인코딩을 사용해서 데이터를 전송했는데, 파일 업로드 하기 위해서는 multipart/form-data 인코딩을 사용해야만 한다.

업로드 페이지의 태그 형식

```
<form method="post" enctype="multipart/form-data">
    <input type="file" name="filename"/>
</form>
```

폼 태그의 enctype 속성이 multipart/form-data 로 설정되어 있어야 한다.

multipart/form-data가 설정되지 않으면 데이터를 보내는데 get과 post 방식에 용량이 제한되기 때문에 큰 데이터의 파일은 전송할 수 없다. 이 속성을 지정하면 데이터도 파일 형태로 넘어가며 큰 용량의 파일도 전송할 수 있다.

파일 다운로드의 경우 브라우저에서 링크된 서버측의 파일을 클릭하기만 하면 브라우저에서 알아서 클라이언트로 다운로드 기능을 담당하기 때문에 아무처리 없이 가능하다.

서버에서는 전송된 데이터를 받아서 파일로 만드느 역할을 하는데 이 때 필요한 것이 업로드 모듈이다. 가장 널리 사용되는 업로드 컴포넌트인 COS 라이브러리를 이용하여 파일 업로드를 구현한다.

<http://www.servlets.com> 에서 com.oreilly.servlet 메뉴 클릭 Download에서 cos-26Dec2008.zip 다운 받아 압축을 푼다.

The screenshot shows the Servlets.com website's download page for the COS library. At the top, there's a navigation bar with links like 'Search', 'Home', 'What's New?', 'Secret Polls', 'Mailing Lists', 'List Archives', 'Servlet Engines', 'Servlet TSPs', 'Servlet Tools', 'Documentation', and 'Online Articles'. Below the navigation, there's a 'April 19, 2010' section with news about 'DOM 1.1 Released'. A red arrow points to the 'What's New?' link. To the right of the news, there's a 'Polling' section asking 'Which XML object model do you use most?'. Below that is a 'New COS Release' section with news about 'DOM 1.1 Released'. Another red arrow points to the 'New COS Release' link. At the bottom of the page, there's a 'Download' section with a table showing a single file entry: 'cos-26Dec2008.zip'. A red arrow points to this file entry. The table has columns for 'Version' and 'Comments'. The 'Comments' column for the file entry contains a bulleted list of improvements.

Version	Comments
cos-26Dec2008.zip	<ul style="list-style-type: none">Added support for Servlets 2.4 and Java 5.Added an ExceededException type to make catching easier.Added support for EBCDIC machines.Added a workaround for browsers that send Content-Length of -1.Added a workaround for Opera missing parameter names.

lib 폴더에 있는 cos.jar 파일을 이클립스의 라이브러리 폴더(WebContent\WEB-INF\lib)에 복사한다.

MultipartRequest 의 특징

MultipartRequest는 COS 라이브러리에서 가장 중심적인 역할을 하는 클래스이다.

이 클래스가 파일 업로드를 직접적으로 담당하는 클래스이다.

MultipartRequest 클래스는 안전성도 높고 파일 중복 처리 인터페이스가 있어 쉽게 처리가 가능하다.

MultipartRequest 생성자

```
MultipartRequest(javax.servlet.http.HttpServletRequest request,  
java.lang.String saveDirectory,  
int maxPostSize,  
java.lang.String encoding,  
FileRenamePolicy policy)
```

인자	설명
request	MultipartRequest와 연결할 request 객체를 의미한다.
saveDirectory	서버 측에 저장될 경로를 의미한다.
maxPostSize	최대 파일 크기를 의미한다.
encoding	파일의 인코딩 방식을 의미한다.
policy	파일 중복 처리를 위한 인자를 의미한다.

MultipartRequest의 메소드

메소드명	설명
Enumeration getParameterNames()	폼에서 전송된 file type을 제외한 파라미터 이름들을 Enumeration 타입으로 반환한다.
String[] getParameterValues(String paramName)	해당하는 파라미터의 값을 String[] 타입으로 반환한다.
String getParameter(String paramName)	해당하는 파라미터의 값을 String 타입으로 반환한다.
Enumeration getFileNames()	폼에서 전송된 file type의 파라미터 이름들을 Enumeration 타입으로 반환한다.
String getFileSystemName(String paramName)	클라이언트가 업로드한 파일의 실제적으로 업로드된 이름을 반환한다.
String getOriginalFileName(String paramName)	클라이언트가 업로드한 파일의 원래이름을 반환한다.
String getContentType(String paramName)	업로드된 파일의 마임타입을 반환한다.
File getFile(String paramName)	업로드된 파일 객체를 반환한다.

파일 업로드 폼

FileUploadForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head><meta charset="UTF-8">  
<title>FileUpload Form</title>  
</head>
```

```

<body>
<center>
<form action="fileUpload.jsp" method="post" enctype="multipart/form-data"

```

파일 업로드 폼	
올린 사람 :	<input type="text"/>
제목 :	<input type="text"/>
파일명1 :	<input type="file"/> <input type="button" value="찾아보기..."/>
파일명2 :	<input type="file"/> <input type="button" value="찾아보기..."/>
<input type="button" value="전송"/>	

파일 업로드 폼	
올린 사람 :	<input type="text" value="홍길동"/>
제목 :	<input type="text" value="자료"/>
파일명1 :	<input type="file" value="C:\Users\jws\Desktop"/> <input type="button" value="찾아보기..."/>
파일명2 :	<input type="file" value="C:\Users\jws\Desktop"/> <input type="button" value="찾아보기..."/>
<input type="button" value="전송"/>	

업로드 페이지 작성

fileUpload.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%@ page import="com.oreilly.servlet.MultipartRequest" %>
<%@ page import="com.oreilly.servlet.multipart.DefaultFileRenamePolicy" %>

```

```

<%@ page import="java.util.*" %>
<%
    String uploadPath=request.getRealPath("/upload"); //업로드 경로를 지정
    int size = 10*1024*1024;
    String name="";
    String subject="";
    String filename1="";
    String filename2="";
    try{
        MultipartRequest multi=new MultipartRequest(request,
uploadPath,
size,
"UTF-8",
new DefaultFileRenamePolicy());
        name=multi.getParameter("name");
        subject=multi.getParameter("subject");
        Enumeration files=multi.getFileNames();
        String file1 =(String)files.nextElement();
        filename1=multi.getFilesystenName(file1);
        String file2 =(String)files.nextElement();
        filename2=multi.getFilesystenName(file2);
    }catch(Exception e){
        e.printStackTrace();
    }
%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<body>
<form name="filecheck" action="fileCheck.jsp" method="post">
    <input type="hidden" name="name" value="<%="name%>">
    <input type="hidden" name="subject" value="<%="subject%>">
    <input type="hidden" name="filename1" value="<%="filename1%>">
    <input type="hidden" name="filename2" value="<%="filename2%>">
</form>
<a href="#" onclick="javascript:filecheck.submit()">업로드 확인 및 다운로드 페이지 이동</a>
</body>
</html>

```



업로드 확인 및 다운로드 페이지

fileCheck.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    String name=request.getParameter("name");
    String subject=request.getParameter("subject");
    String filename1=request.getParameter("filename1");
    String filename2=request.getParameter("filename2");
%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>파일 업로드 확인 및 다운로드</title>
</head>
<body>
    올린 사람 : <%=name %><br>
    제목 : <%=subject %><br>
    파일명1 : <a href="upload/<%=filename1 %>"><%=filename1 %></a><br>
        
    파일명2 : <a href="upload/<%=filename2 %>"><%=filename2 %></a><p>
        
</body>
</html>
```



14.2 @MultipartConfig 애노테이션을 이용한 파일 업로드

가장 많이 사용되고 알려진 것은 아파치 그룹에서 제공하는 Commons FileUpload 라이브러리이다.

Spring 프레임워크 및 Struts2 프레임워크 같은 유명한 프레임워크에서도 사용되는 매우 안정적인 라이브러리이다.

하지만 서블릿 3.0 버전부터는 @MultipartConfig 어노테이션과 javax.servlet.http.Part 인터페이스를 사용하여 보다 쉽게 파일 업로드 기능을 구현할 수 있다.

@MultipartConfig 어노테이션에서 사용 가능한 속성 목록

속성	설명
maxFileSize	업로드 파일의 최대 크기 값. 기본 값은 -1(크기 제한 없음)
maxRequestSize	HTTP 요청의 최대 크기 값. 기본 값은 -1(크기 제한 없음)
location	파일 저장 경로. 파일은 Part의 write 메서드가 호출될 때 저장된다.

사용 가능한 Part 인터페이스의 메서드 목록이며, 폼 태그 요청의 모든 속성은 Part로 변환되어 처리

메서드	설명
String getName()	HTML 태그의 폼 태그 이름을 리턴한다. 태그명이 파트의 이름이 된다.
String contentType()	파일의 contentType을 리턴
Collection getHeaderNames()	Part의 모든 헤더명을 리턴
getHeader(name)	설정한 헤더의 값을 리턴
write(path)	업로드한 파일을 출력
delete()	파일과 임시 파일을 삭제
InputStream getInputStream()	업로드한 파일의 내용을 InputStream 객체로 리턴

다음은 사용자가 파일을 업로드 한 경우의 Part의 헤더 정보이다.

헤더 정보에서 문자열을 파싱하여 업로드한 파일명을 얻을 수 있다.

```
content-type: 값
content-disposition:form-data; name="폼태그명"; filename="파일명"
```

- 'c:/upload' 업로드 폴더를 만든다.

uploadForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>FileUpload 실습 </title>
</head>
<body>
<h1>FileUpload 실습 </h1>
<form action="Upload" method="post" enctype="multipart/form-data">
작성자 <input type="text" name="theAuthor" > <br>
파일 <input type="file" name="theFile" > <br>
<input type="submit" value="업로드" >
</form>
</body>
</html>
```

UploadServlet.java

```
package com.controller;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;
@WebServlet("/Upload")
@MultipartConfig( maxFileSize= 1024*1024*2 , location ="c:\upload" )
public class UploadServlet extends HttpServlet {
    // 파일명 얻기
    private String getFilename(Part part){
        String fileName = null;
        String contentDispositionHeader = part.getHeader("content-disposition");
        String [] elements = contentDispositionHeader.split(";");
        for(String element: elements){
            if( element.trim().startsWith("filename")){
                fileName = element.substring(element.indexOf('=') + 1);
                fileName = fileName.trim().replace("\"", "");
            }
        }
        return fileName;
    }
    //end getfilename
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        Part part = request.getPart("theFile");
        String fileName = getFilename(part);
        if( fileName != null && !fileName.isEmpty()){
            // part.write("c:\upload\"+ fileName);
            part.write(fileName);
        }
        String author = request.getParameter("theAuthor");
        author = new String( author.getBytes("iso-8859-1") , "UTF-8");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
    }
}
```

```

        out.print("작성자:" + author + "<br>");
        out.print("파일명:" + fileName+ "<br>");
        out.print("파일크기:" + part.getSize() + "<br>");
    }//end doPost
}//end class

```

- 파일 다운로드

파일 업로드 기능을 하는 라이브러리는 제공되지만 다운로드 기능을 구현한 라이브러리는 없기 때문에 직접 프로그래밍 작업으로 처리해야 한다.

구현 방법은 다운로드에서 사용했던 Content-Disposition 응답 헤더를 추가하고, 값을 'attachment; filename=파일명' 형식으로 지정한다. 마지막으로 OutputStream 객체를 사용하여 웹 브라우저로 출력한다.

먼저 UploadServlet.java 파일을 다음과 같이 수정한다. 파일을 다운로드하기 위해서는 파일을 선택해야 되기 때문에 링크를 추가한다.

```

        out.print("작성자:" + author + "<br>");
out.print("파일명:" + fileName+ "<br>");
        out.print("파일크기:" + part.getSize() + "<br>");
```

수정 내용

```

        out.print("작성자:" + author + "<br>");
out.print("파일명:<a href='FileDown?file_name=" + fileName + "'>" + fileName + "</a><br>");
        out.print("파일크기:" + part.getSize() + "<br>");
```

FileDownServlet.java

```

package com.controller;
import java.io.FileInputStream;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/FileDown")
public class FileDownServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        String fileName = request.getParameter( "file_name" );
        String sDownloadPath = "C:\upload";
        String sFilePath = sDownloadPath + fileName;
```

```

byte b[] = new byte[4096];
FileInputStream in = new FileInputStream(sFilePath);
String sMimeType = getServletContext().getMimeType(sFilePath);
System.out.println("sMimeType>>"+sMimeType );
// octet-stream 은 8비트로 된 일련의 데이터를 뜻합니다. 지정되지 않은 파일 형식을 의미합니다.
if(sMimeType == null) sMimeType = "application/octet-stream";
response.setContentType(sMimeType);
//한글 업로드
String sEncoding = new String(fileName.getBytes("UTF-8"),"8859_1");
response.setHeader("Content-Disposition", "attachment; filename= " + sEncoding);
ServletOutputStream out = response.getOutputStream();
int numRead;
// 바이트 배열b의 0번 부터 numRead번 까지 브라우저로 출력
while((numRead = in.read(b, 0, b.length)) != -1) {
    out.write(b, 0, numRead);
}
out.flush();
out.close();
in.close();
}//end
}//end class

```

14.3 자카르타 프로젝트

아파치 소프트웨어 재단(AFC: Apache Foundation Consortium)의 하위 기관

-아파치 소프트웨어 라이센스를 바탕으로 하는 공동 개발방식을 장려

-다양한 오픈 소스 기반의 자바 솔류션

자카르타 프로젝트 종류

-톰캣

-JSTL도 Taglibs 이름의 프로젝트에서 개발한 제품

-commons

The Commons Proper	
The Commons Proper is dedicated to one principal goal: creating and maintaining reusable Java components for reuse across projects. The components are designed to be easy to use, yet powerful enough to be useful in a wide variety of applications. They are developed by a community of developers from around the world, and are released under the Apache License.	
Commons developers will make an effort to ensure that their components have minimal dependencies or can be deployed easily. In addition, Commons components will keep their interfaces as stable as possible, so that they can implement these components without having to worry about changes in the future.	
The article Components gives an overview of (some of) the components which can be found here. We welcome participation from all that are interested, at all skill levels. Coding, documentation and test development process. If you are interested in participating in any of these aspects, please join us!	
Components	
Attributes	Runtime API to metadata attributes such as doclet.
BeanUtils	Easy-to-use wrappers around the Java reflection and introspection APIs.
Betwixt	Places XML and Java objects to XML documents, and vice versa.
Chain	Chain Responsibility pattern implementation.
CLI	Command Line arguments parser.
Codec	General encoding/decoding algorithms (for example phonetic, base64, URL).
Collections	Extends or augments the Java Collections Framework.
Compress	Defines an API for working with tar, zip and bz2 files.
Configuration	Reading of configuration/preferences files in various formats.
Daemon	Alternative invocation mechanism for unix-daemon-like java code.
DBCP	Database connection pooling services.
DbUtils	JDBC helper library.
Digester	XML-to-Java-object mapping utility.
Discovery	Tools for locating resources by mapping service/reference names to resource names.
EL	Interpreter for the Expression Language defined by the JSP 2.0 specification.
Email	Library for sending e-mail from Java.
Executor	API for dealing with external process execution and environment management in Java.
FileUpload	File upload capability for your servlets and web applications.
IO	Collection of I/O utilities.
JCI	Java Compiler Interface
Jelly	XML based scripting and processing engine.

<http://commons.apache.org>에서 FileUpload와 IO에서

Downloading

Full Releases

- FileUpload 1.2.1** - 18 January 2008
 - Download the binary and source distributions from a mirror site [here](#)
- FileUpload 1.2** - 13 February 2007
 - Download the binary and source distributions from a mirror site [here](#)
- FileUpload 1.1.1** - 08 June 2006
 - Download the binary and source distributions from a mirror site [here](#)
- FileUpload 1.1** - 22 Dec 2005
 - Download the binary and source distributions from a mirror site [here](#)
- FileUpload 1.0** - 24 Jun 2003
 - Download the binary and source distributions from a mirror site [here](#)

Releases

The latest version is v1.4. - [Download now!](#)

The [upgrade notes](#) are also available.

For previous releases, see the [Apache Archive](#)

Support

The [commons mailing lists](#) act as the main support for development discussion. Please remember that the lists are public and may be monitored by third parties.

Download Commons FileUpload

Using a Mirror

We recommend you use a mirror to download our release builds, from our main distribution directories. Recent releases (48 hours) You are currently using <http://apache.tt.co.kr>. If you encounter any problems, please try one of the backup mirrors (at the end of the mirrors list) that should be available.

Other mirrors: <http://apache.tt.co.kr>

The [KEYS](#) link links to the code signing keys used to sign the JAR files. The [MD5](#) link downloads the checksum from the main site.

Commons FileUpload 1.2.1

Binaries

[commons-fileupload-1.2.1-bin.tar.gz](#)
[commons-fileupload-1.2.1-bin.zip](#)

Commons IO 1.4

Binaries

[commons-io-1.4-bin.tar.gz](#)
[commons-io-1.4-bin.zip](#)

Source

[commons-io-1.4-src.tar.gz](#)
[commons-io-1.4-src.zip](#)

commons-fileupload-1.2.1-bin.zip와 commons-io-1.4-bin.zip 을 다운 받아 압축을 푼 후에 commons-fileupload-1.2.1.jar, commons-io-1.4.jar 파일을 이클립스의 라이브러리 폴더(¶WebContent\WEB-INF\lib)에 복사한다.

파일 업로드 품

fileUpload.html

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>파일 업로드</title>
</head>
<body>
<h2> 파일 업로드 예제 </h2><hr>
<form method="post" action="fileUpload1.jsp" enctype="multipart/form-data">
  사용자: <input type="text" name="user"> <br> <hr>
  첨부파일1: <input type="file" size=50 name="file1"> <br>
  첨부파일2: <input type="file" size=50 name="file2"> <br>
  <input type="submit" value="전송">
</form>
</body>
</html>
```

파일 업로드 예제

사용자:

첨부파일1: [찾아보기...](#)

첨부파일2: [찾아보기...](#)

파일 업로드 예제

사용자: 홍길동

첨부파일1: C:\Users\jws\Desktop\VC++\복차.txt [찾아보기...](#)

첨부파일2: C:\Users\jws\Desktop\Oracle 경로.txt [찾아보기...](#)

파일 업로드 처리

fileUpload1.jsp

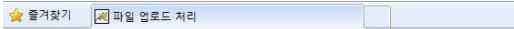
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>파일 업로드 처리 </title>
</head>
<body>
<h2> 파일 업로드 예제 </h2><hr>
<%@ page import = "org.apache.commons.fileupload.disk.DiskFileItemFactory" %>
<%@ page import = "org.apache.commons.fileupload.servlet.ServletFileUpload" %>
<%@ page import = "org.apache.commons.fileupload.FileItem" %>
<%@ page import = "java.io.File" %>
<%@ page import = "java.io.IOException" %>
<%@ page import = "java.util.List" %>
<%
    //업로드된 파일이 저장되는 폴더
    String strSaveDir = getServletContext().getRealPath("/uploadStorage");
    File saveDir = new File(strSaveDir);
    out.println("업로드되는 파일이 저장될 폴더 : <br>" + saveDir.getPath() + "<p>");
    //업로드에 필요한 임시 폴더
    String strTempDir = getServletContext().getRealPath("temp");
    File tempDir = new File(strTempDir);
    if ( !tempDir.exists() ) tempDir.mkdir();
    out.println("업로드를 위한 임시 폴더 : <br>" + tempDir.getPath() + "<br><hr>");
    if (ServletFileUpload.isMultipartContent(request)) {
        // 임시 폴더에 저장 할 파일의 크기 설정
        // DiskFileItemFactory factory = new DiskFileItemFactory(1024*100, tempDir);
        DiskFileItemFactory factory = new DiskFileItemFactory();
        factory.setSizeThreshold(1024*100);
        factory.setRepository(tempDir);
        // 파일 업로드 처리기
        ServletFileUpload upload = new ServletFileUpload(factory);
        // 크기 제약 설정
        upload.setSizeMax(1024*1024);
        // request를 분석해 각 항목으로 처리
        List<FileItem> items = upload.parseRequest(request);
        out.println("<h3> 업로드 처리 결과 </h3> <p><hr>");
        for ( FileItem fileItem : items ) {
```

```

//일반 인자와 파일업로드 인자를 구분하여 처리
if (fileItem.isFormField()) {
    // 파일 이외의 파라미터 내용 출력
    out.println(fileItem.getFieldName() + " :" + fileItem.getString("UTF-8") + "<p><hr>");
} else {
    // 업로드한 파일이 존재하는 경우
    if ( fileItem.getSize() > 0 ) {
        //파일 이름만 추출하여 fileName에 저장
        String fileName = new File( fileItem.getName() ).getName();
        //업로드파일 저장 폴더 생성
        if ( !saveDir.exists() ) saveDir.mkdir();
        try {
            File uploadedFile = new File(saveDir, fileName);
            //같은 이름이 이미 있으면 현재 시간정보를 뒤에 붙인 파일 이름으로 저장
            if ( uploadedFile.exists() ) {
                java.util.Date now = new java.util.Date();
                String newFileName = fileName + "-" + now.getTime();
                uploadedFile = new File(saveDir, newFileName);
            }
            out.println("이름이 같은 파일이 이미 있어 다음 파일 이름으로 수정하였습니다. <br>");
            out.println("이전 파일 이름 :" + fileName + ", ");
            out.println("수정 파일 이름 :" + newFileName + "<p><hr>");
        }
        //업로드 파일 저장
        fileItem.write(uploadedFile);
        out.println("업로드 폴더 위치 :" + saveDir.getPath() + "<br>");
        out.println("업로드 파일 이름 :" + uploadedFile.getName() + "<p><hr>");
    } catch(IOException e) {
        // 예외 처리
        out.println(e.toString());
    }
}
}
}
}

%>
</body>
</html>

```



파일 업로드 예제

업로드되는 파일이 저장될 폴더 :
C:\Tomcat6\wwwapps\fileupload\uploadStorage

업로드를 위한 임시 폴더 :
C:\Tomcat6\wwwapps\fileupload\temp

업로드 처리 결과

user : 흥길동

업로드 폴더 위치 : C:\Tomcat6\wwwapps\fileupload\uploadStorage
업로드 파일 이름 : C++복자.txt

업로드 폴더 위치 : C:\Tomcat6\wwwapps\fileupload\uploadStorage
업로드 파일 이름 : oracle 경로.txt

14.3 셈네일 이미지

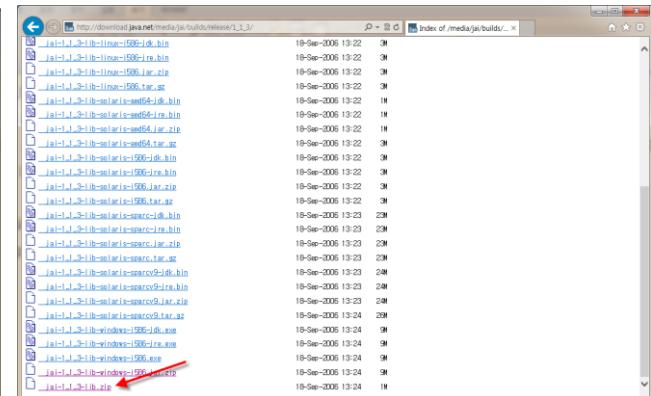
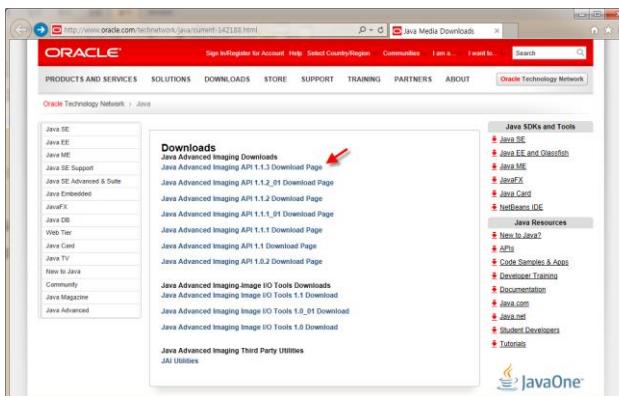
1) JAI(Java Advanced Imaging) API

JAI API는 SUN사에서 제공하는 강력한 이미지 편집 기능을 가진 API이다. JAI API는 다양한 이미지 타입과 다양한 효과를 적용시킬수 있다.

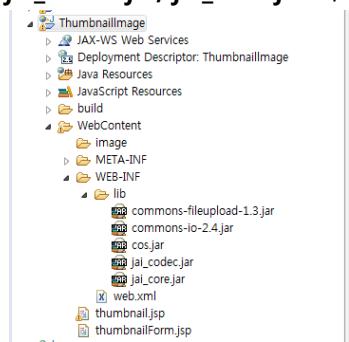
2) JAI API 다운로드 및 설치

<http://www.oracle.com/technetwork/java/current-142188.html>

jai-1_1_3-lib.zip 파일 다운로드 후 압축 해제



jai_codec.jar, jai_core.jar 두 개의 파일을 lib 폴더에 복사한다.



thumbnailForm.jsp

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

```

<head><meta charset="UTF-8">
<title>썸네일 이미지 품</title>
</head>
<body>
<center>
<h3>썸네일 이미지 품 예제</h3>
<form action="thumbnail.jsp" method="post" enctype="multipart/form-data">
이미지 파일 : <input type="file" name="filename"> <p>
<input type="submit" value="전송">
</form>
</center>
</body>
</html>

```

thumbnail.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.awt.Graphics2D" %>
<%@ page import="java.awt.image.renderable.ParameterBlock" %>
<%@ page import="java.awt.image.BufferedImage" %>
<%@ page import="javax.media.jai.JAI" %>
<%@ page import="javax.media.jai.RenderedOp" %>
<%@ page import="javax.imageio.ImageIO" %>
<%@ page import="com.oreilly.servlet.MultipartRequest" %>
<%@ page import="com.oreilly.servlet.multipart.DefaultFileRenamePolicy" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%
    String imagePath=request.getRealPath("/image");
    int size = 1*1024*1024 ;
    String filename="";
    try{
        MultipartRequest multi= new MultipartRequest(request,
                                                       imagePath,
                                                       size,
                                                       "UTF-8",
                                                       new DefaultFileRenamePolicy());
        Enumeration files=multi.getFileNames();
        String file =(String)files.nextElement();
        filename=multi.getFilesystemName(file);
    }catch(Exception e){

```

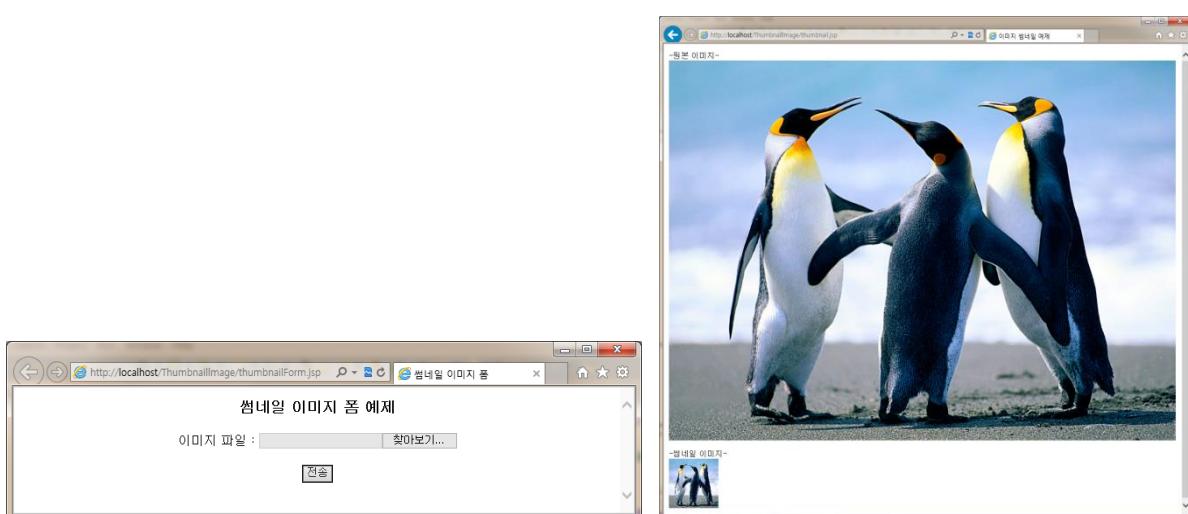
```

        e.printStackTrace();
    }

    ParameterBlock pb=new ParameterBlock();
    pb.add(imagePath+"/"+filename);
    RenderedOp rOp=JAI.create("fileload",pb);
    BufferedImage bi= rOp.getAsBufferedImage();
    BufferedImage thumb=new BufferedImage(100,100,BufferedImage.TYPE_INT_RGB);
    Graphics2D g=thumb.createGraphics();
    g.drawImage(bi,0,0,100,100,null);
    File file=new File(imagePath+"/sm_"+filename);
    ImageIO.write(thumb,"jpg",file);

%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>이미지 썸네일 예제</title>
</head>
<body>
    -원본 이미지-<br>
     <p>
    -썸네일 이미지-<br>
    
</body>
</html>

```



- JDBC 드라이버 설치

- JDBC 드라이버 선택

- JDBC 드라이버는 사용하고자 하는 데이터베이스 벤더 별로 제공 된다.

- 오라클 JDBC드라이버

- C:\app\oracle11\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar

- 설치 디렉터리(다음 중 한 가지를 이용함)

- JDK설치디렉터리\re\lib\ext\에 복사하는 방법.

- 톰캣설치디렉터리\lib 폴더에 복사하는 방법(이 방법만 사용)**

- 이클립스 프로젝트의 WebContent\WEB-INF\lib 폴더에 복사하는 방법

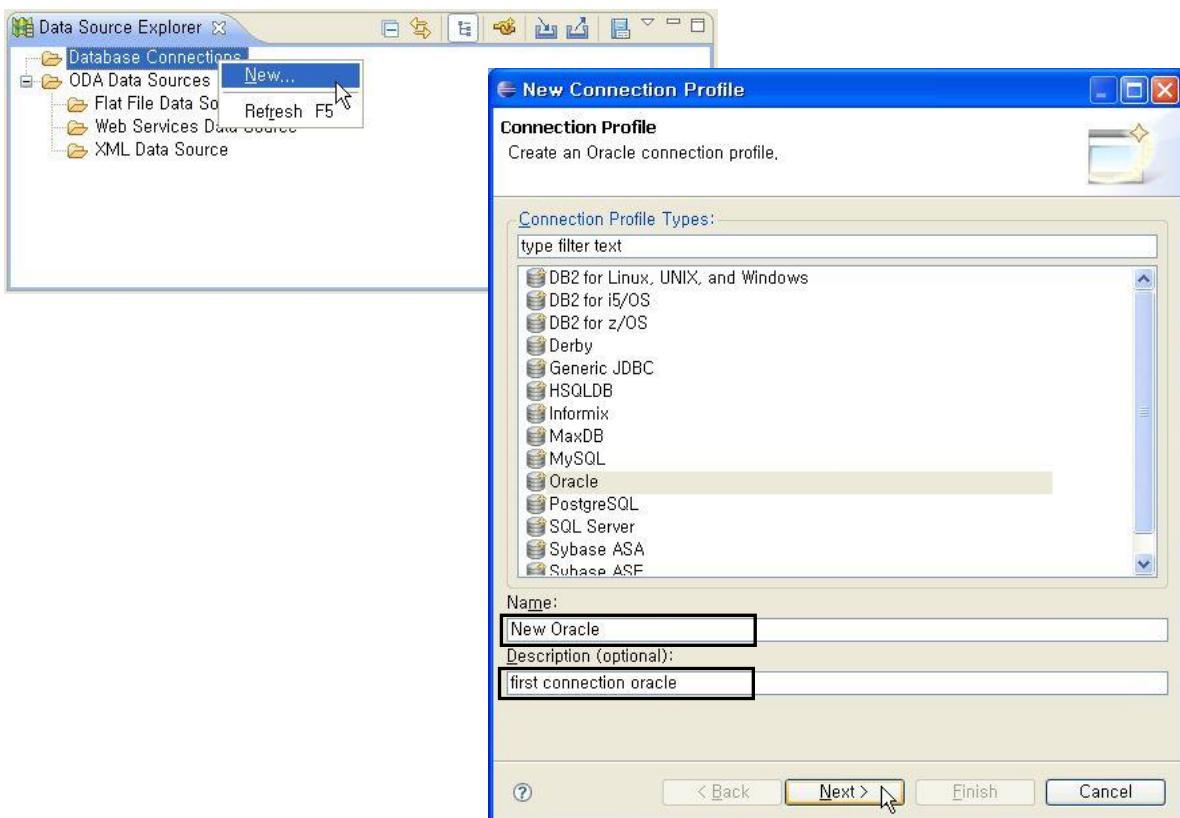
v

WebContent\WEB-INF\lib 폴더에 설치

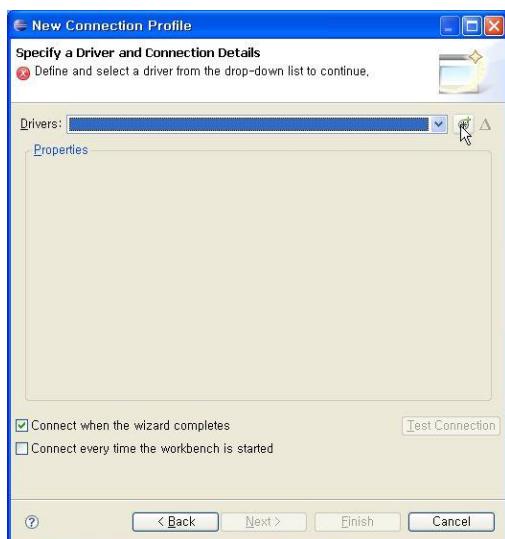
15. 커넥션 프로파일 만들기

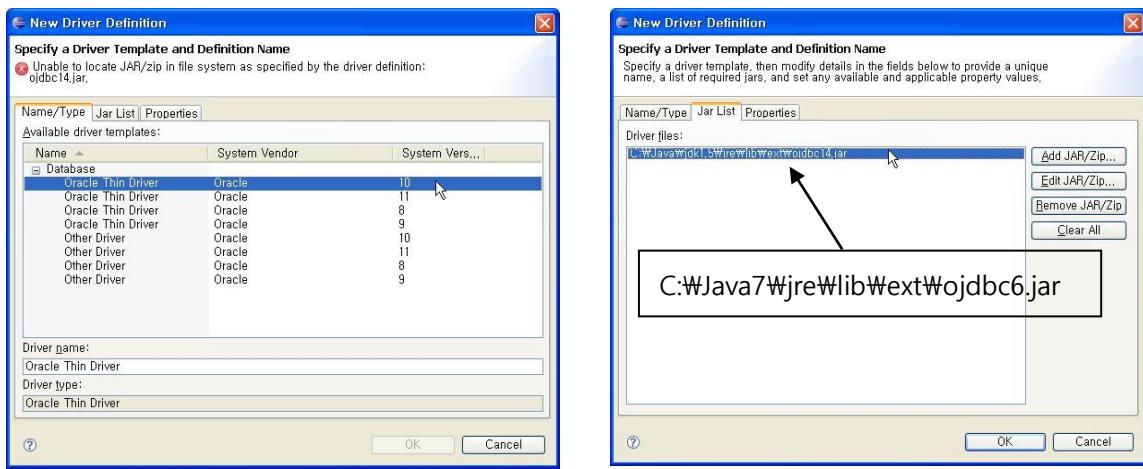
이클립스에서 데이터베이스 작업을 하려면 [Data Source Explorer] 뷰를 이용한다. [Java EE] 퍼스텍티브에서 [Data Source Explorer] 뷰는 전체화면의 하단에 위치가 보이지 않으면 메뉴 [Window]/[Show View]/[Data Source Explorer]로 실행할 수 있다.

이제 데이터베이스에 연결하기 위해 [커넥션 프로파일]을 하나 만들자. [Data Source Explorer] 뷰의 [Database]의 왼쪽 메뉴 [New ...]를 선택하거나 상단 [New Connection profile] 아이콘을 클릭한다.



생성된 [New Connection profile] 첫 대화상자에서 [Connection Profile Types]은 연결하려는 DBMS인 Oracle을 지정한다. 또한 커넥션 프로파일 생성 후 구분을 위해 [Name]과 [Description]을 적절히 입력한 후 [Next >] 버튼을 누른다. 속성 [Name]에는 접속하는 DBMS 이름이 들어가도록 한다.





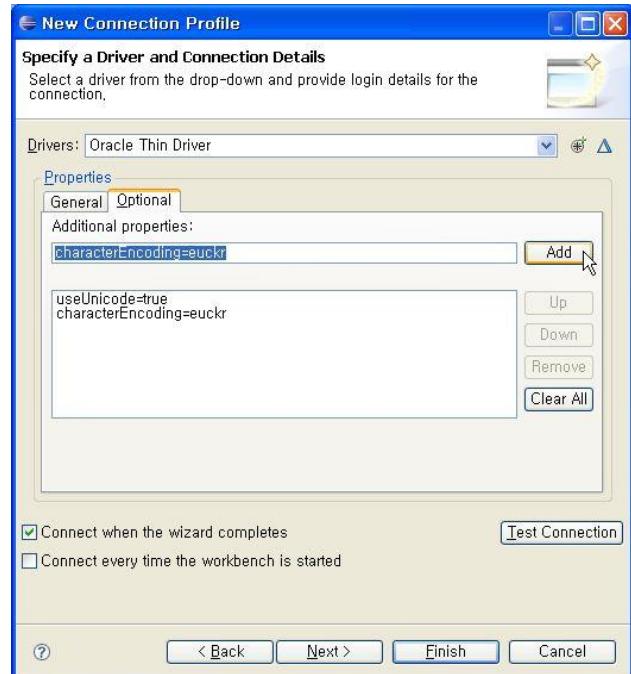
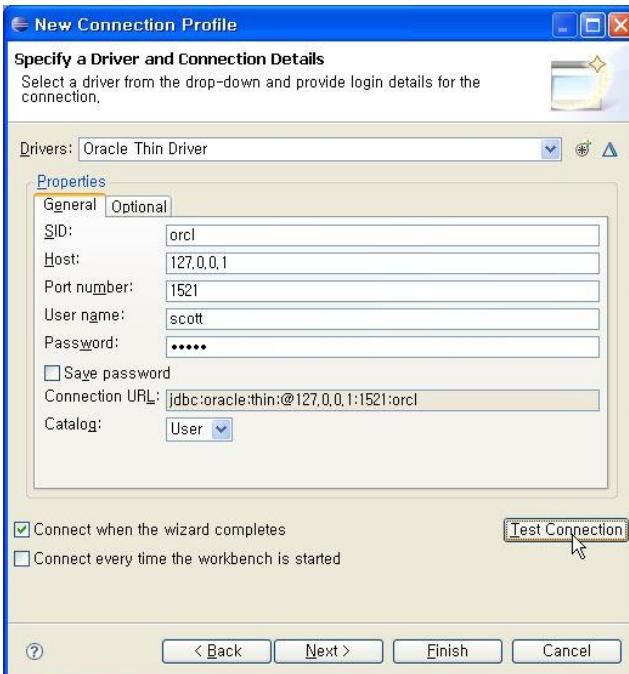
처음에는 위 화면에서 [Drivers]를 비롯해 대부분의 내용이 비어있는데, 우측 상단의 [New Driver Definition]을 이용해 원하는 데이터베이스 드라이버를 지정해야 한다. 생성된 [New Driver Definition] 대화상자의 [Jar List] 탭에서 오른쪽 [Add Jar/Zip] 버튼을 눌러 폴더 [java 설치 폴더]/[jre]/[ext]에 이미 저장해 놓은 oracle의 jdbc 드라이버인 [ojdbc6.jar] 파일을 선택한다.

[New Driver Definition] 대화상자에서 [ok]를 누르면 다시 돌아온 [New Driver Definition] 대화 상자에서 [Properties]의 [General]과 [Optional]을 보고 연결할 DB인 Oracle의 속성값으로 적절히 입력한다. 이 중에서 가장 중요한 속성은 URL로 마지막 부분이 접속하려는 데이터베이스 이름인데 이 부분에 접속을 원하는 데이터베이스 이름을 넣어야 한다. 여기서는 DBMS Oracle에 SID인 orcl로 접속해야 하므로 다음과 같이 입력해야 한다.

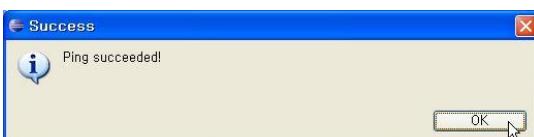
```
jdbc:oracle:thin:@localhost:1521:orcl
```

탭구분	속성	값	의미
General	SID (System Identifier, 시스템 식별자)	orcl	oracle를 설치할 때 설정하는 것인데 대부분 일정하지 않다. SID는 오라클 데이터베이스가 아닌 오라클 인스턴스를 식별하기 위한 유일한 값이고, 그 값이 데이터베이스 이름이 된다.
	Host	localhost / 127.0.0.1	접속하고자 하는 오라클의 위치
	Port Number	1521	oracle의 포트를 작성하는데 일반적으로 설치 하였다면 1521 이다.
	User Name	scott	접속할 사용자 이름
	Password	tiger	접속할 사용자의 비밀번호
	Connection URL	jdbc:oracle:thin: @localhost:1521:orcl	접속하고자 하는 데이터베이스
Catalog	User	사용자 / DBA	
Optional	useUnicode	useUnicode=true	유니코드 사용 여부

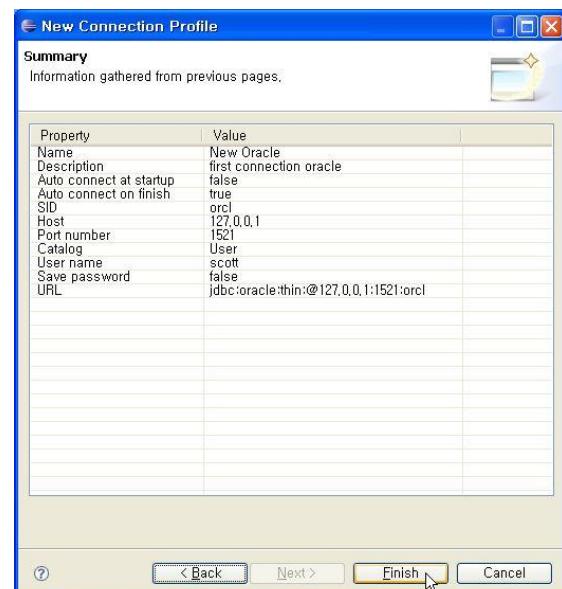
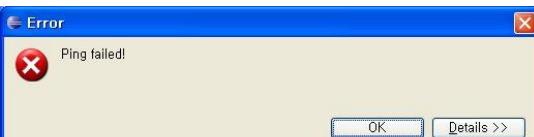
속성 [Optional]에서 [useUnicode=true]와 [characterEncoding=euckr]을 입력하여 [Add]를 누르고 다시 한 번 [URL]과 다른 부분의 속성을 확인한 후 우측 하단의 [Test Connection]을 눌러 DB 접속 검사를 한다. DB 접속에 성공하면 다음과 같이 [success] 대화상자가 나타난다. 만일 속성에 문제가 있으면 [failed] 대화상자가 표시될 것이다.



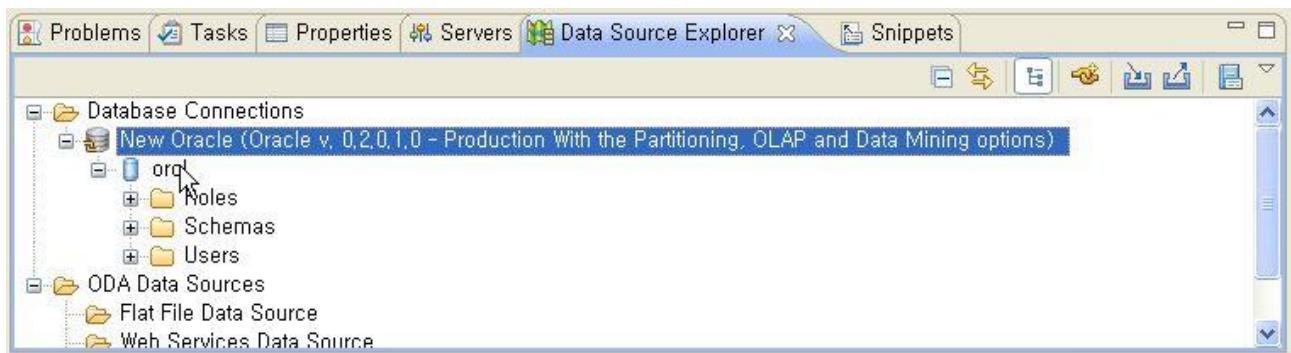
[테스트 결과 – 성공시]



[테스트 결과 – 실패시]

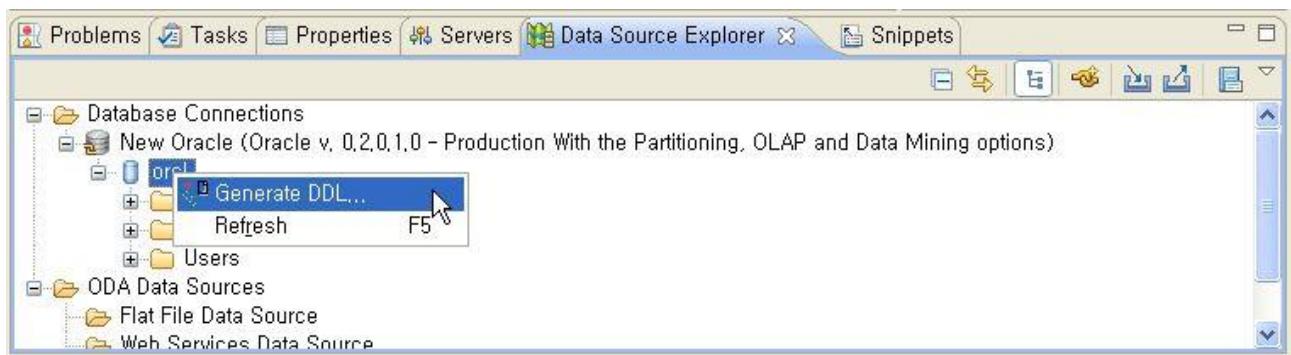


DB 접속 점검에 성공 한 후 [Next >] 클릭 후 [Finish]를 누르면 [New Connection Profile] 대화상자에서 지정한 [Name]으로 [New Connection Profile] 아이콘이 생기며, 그 하부에는 속성 [Properties]의 [SID]에 지정한 이름으로 접속한 데이터베이스 아이콘이 생긴다.



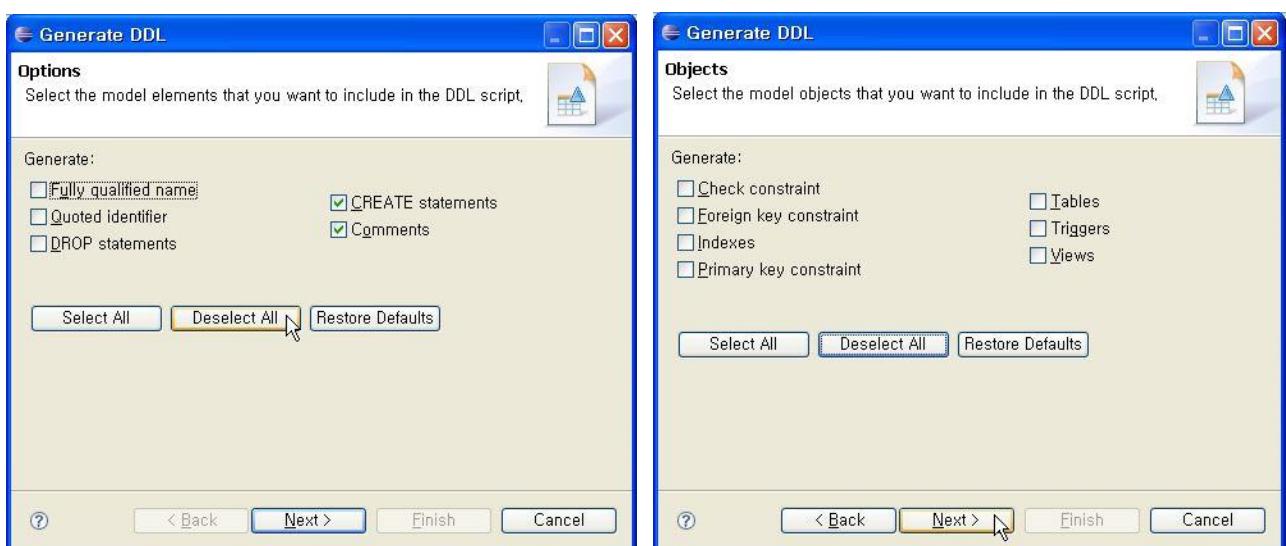
- 이클립스에서 테이블 생성

데이터베이스를 생성할 DDM SQL 문장을 실행하는 [질의 뷰]를 하나 생성한다. 이를 위하여 다음 그림처럼 앞에서 만든 커넥션 프로파일 [New Oracle]의 하부 전역 데이터베이스 [orcl]에서 메뉴 [Generate DDL...]를 선택한다.



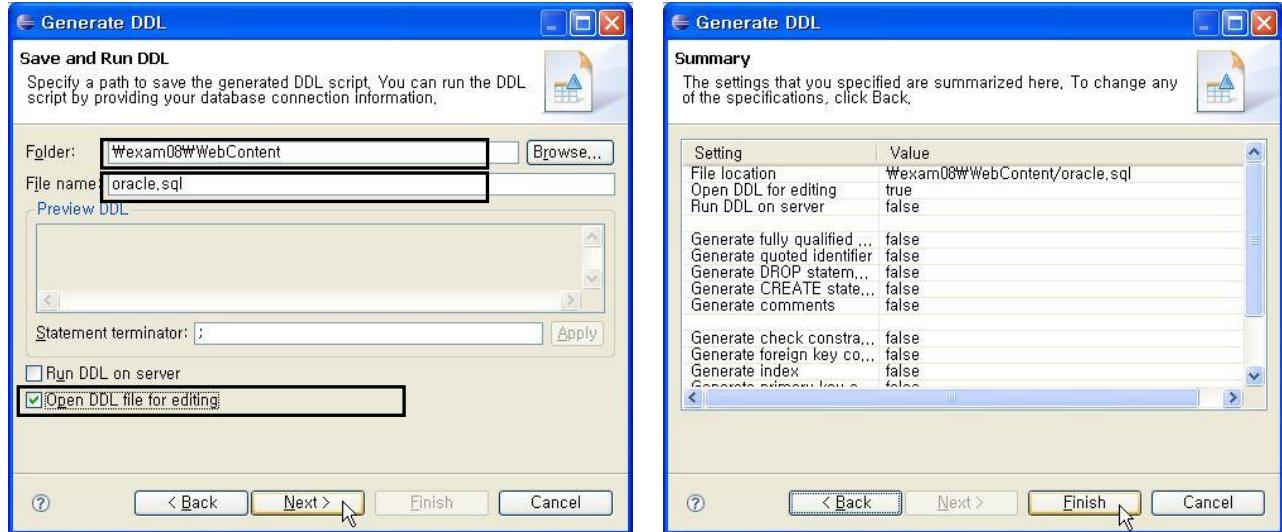
[Generate DDL...] 대화상자에서 문장이 없는 [질의 뷰]를 만들기 위해 [Deselect All]을 눌러 선택된 체크박스가 없도록 하고 [Next >]버튼을 눌러 마지막 대화상자에 도달하도록 한다.

DDL문장을 생성하고 싶다면 적당한 관련 문장의 체크박스에 체크를 할 수 있다.



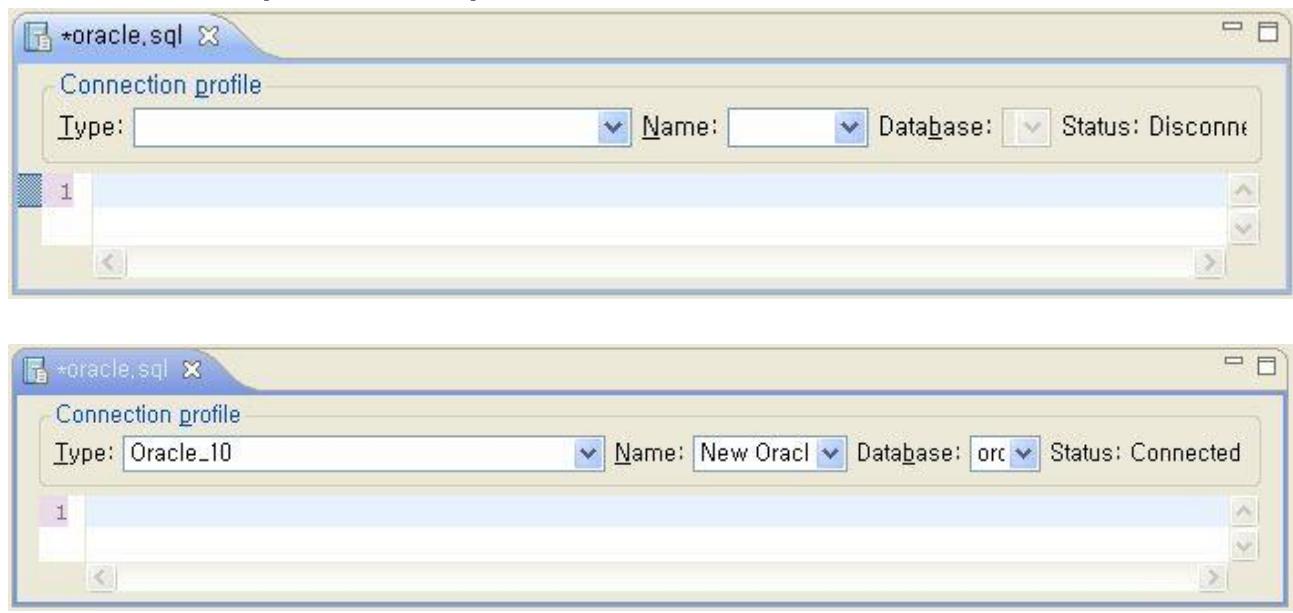
다음 [Generate DDL...] 대화상자에서 편의를 위하여 폴더를 현재 프로젝트 하부 [WebContent]로 지정하

고 파일 이름을 적당히 [oracle.sql]로 지정하며, [질의뷰]를 생성한 후, 바로 편집기에서 열기 위해 [Open DDL file for editing] 체크를 체크한다. [Next]버튼을 눌러 다음으로 이동하여 내용을 확인한 후, [Finish]를 눌러 [질의 뷰]를 생성한다.



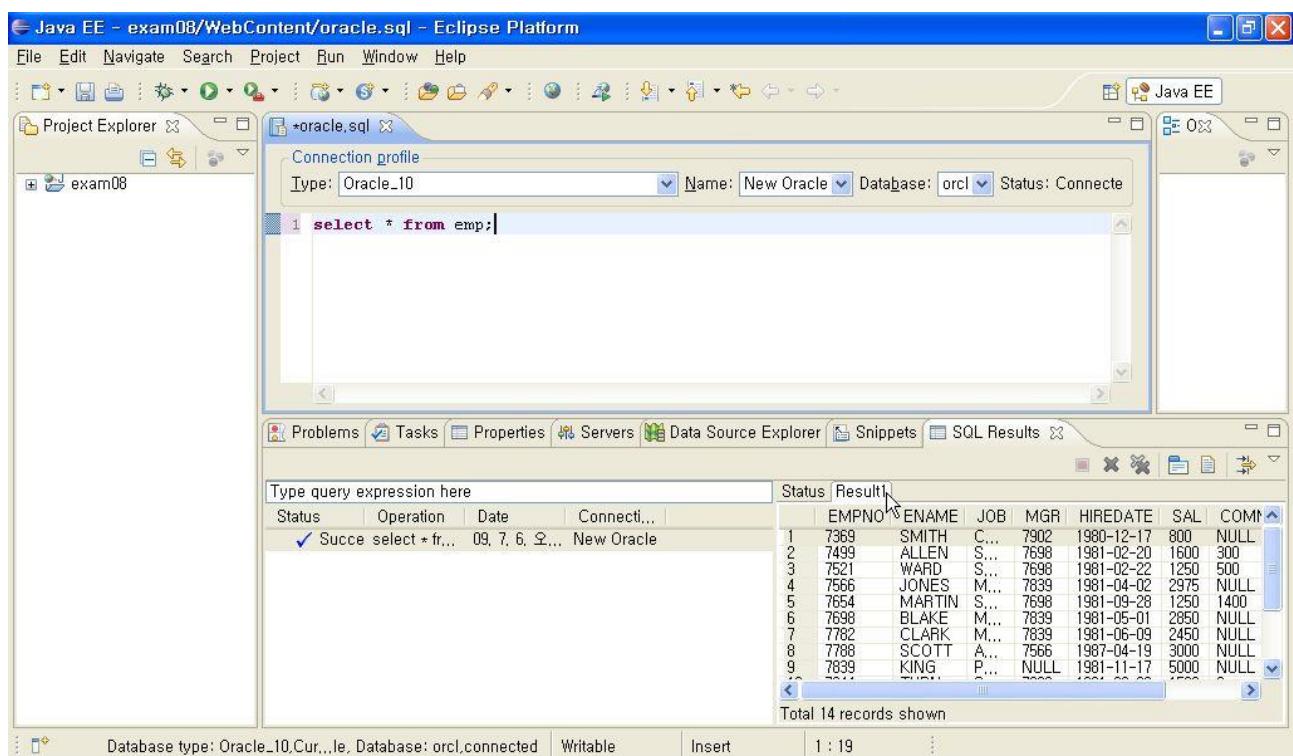
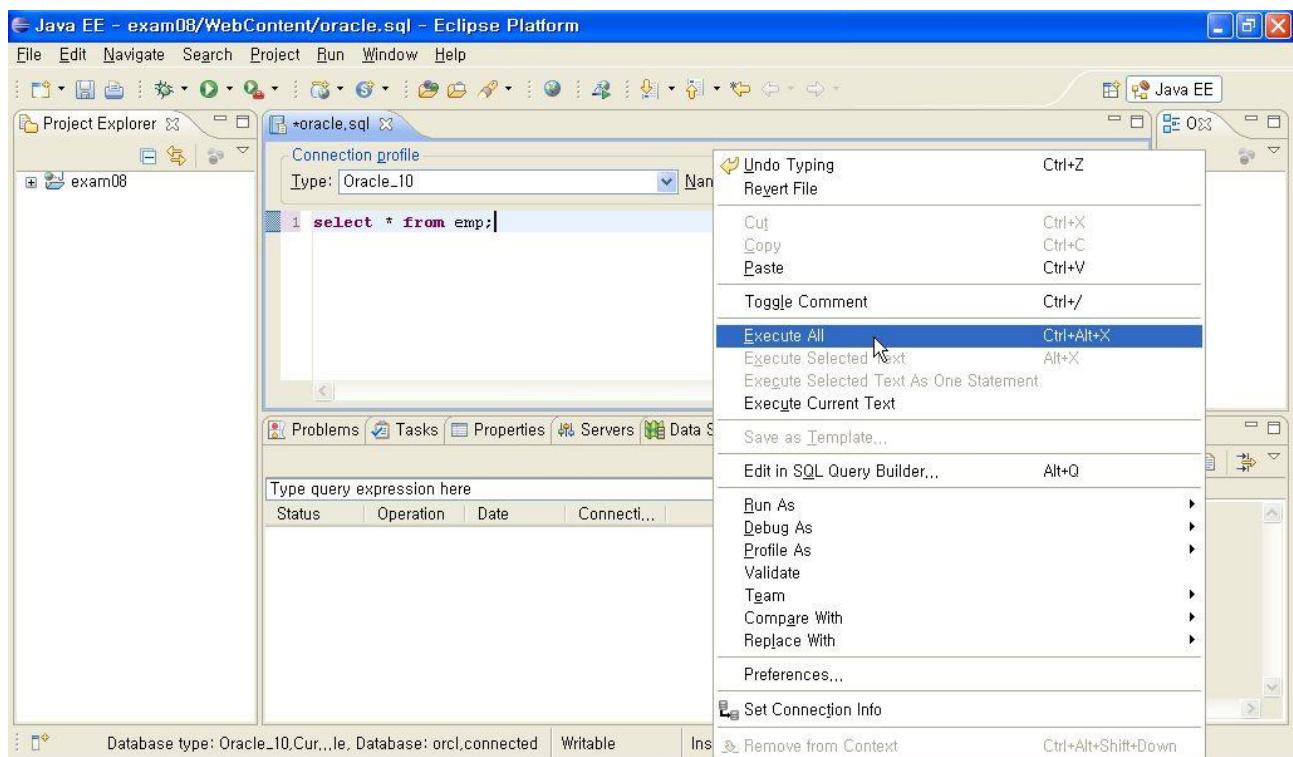
처음 생성된 [질의 뷰]에서는 데이터베이스 연결 정보가 하나도 지정되어 있지 않다. 즉 [질의 뷰]상단의 [Connection profile]을 보면 Type, Name, Database 3개의 데이터베이스 연결 정보가 비어 있는 것을 볼 수 있다. 이 3개의 속성은 다음과 같은 데이터베이스 연결 정보를 나타낸다. 생성된 [질의 뷰]에서 SQL 문을 실행하려면 이 3개의 속성을 연결하려는 데이터베이스의 정보로 선택해 주어야 한다

- Type : DBMS의 종류
- Name : 이용할 [커넥션 프로파일] 이름
- Database : 연결할 [커넥션 프로파일]의 데이터베이스 이름



[질의뷰] 상단의 Type, Name, Database 3개의 데이터베이스 연결 정보를 각각 [Oracle_11g], [New Oracle], [orcl]로 지정한 후, [질의뷰] 편집기에서 [SELECT * FROM EMP]를 기술한다. SQL문장 실행을 위한 메뉴 [Execute All]을 선택한다.

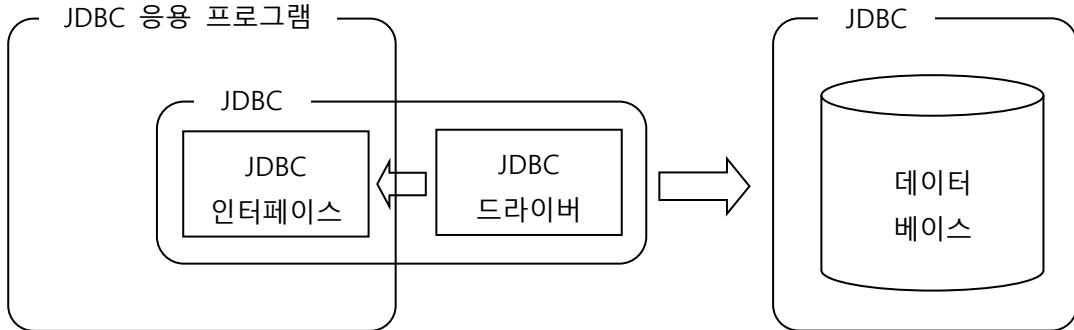
```
select * from emp;
```



16. JDBC 개요

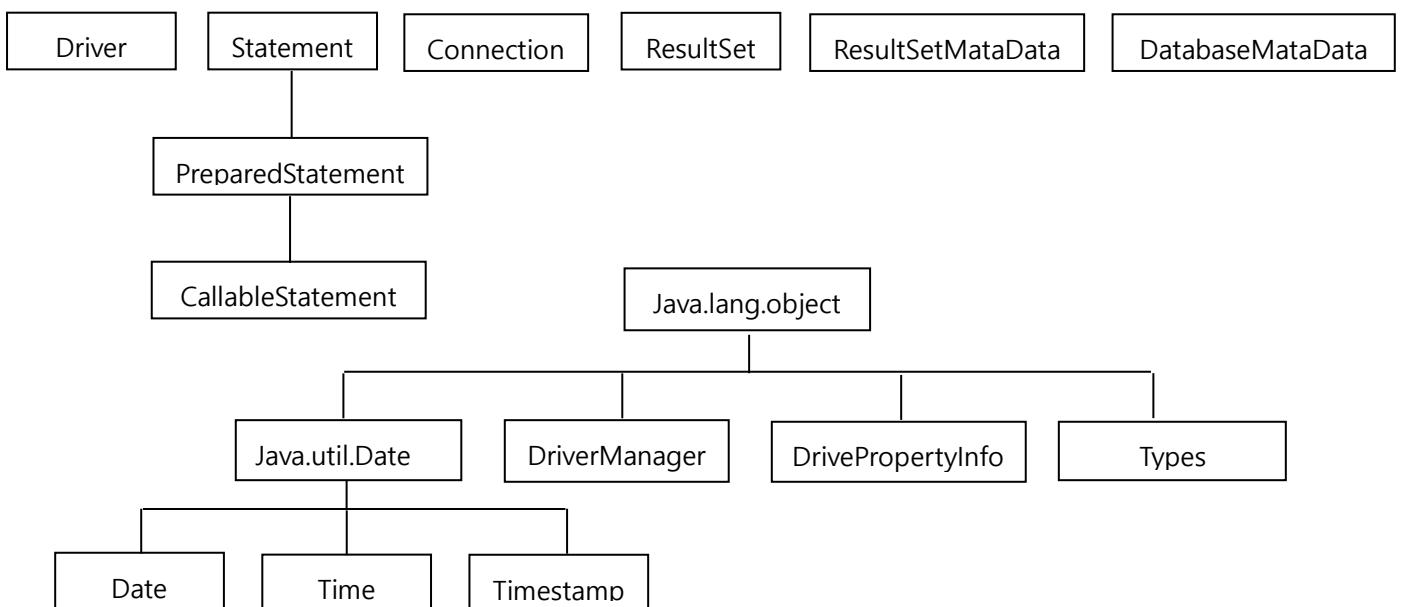
16.1 JDBC의 이해

JDBC(Java DataBase Connectivity)는 자바 프로그램에서 데이터베이스와 연결하여 데이터베이스 관련 작업을 할 수 있도록 해주는 자바 프로그래밍 인터페이스를 위한 API(Application Programming Interface) 규격이다



JDBC는 Driver, DriverManager, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, ResultSetMetaData, Types, DataSource 등 여러 개의 클래스와 인터페이스로 구성된 패키지 java.sql과 javax.sql로 구성되어 있다. 즉 JDBC는 다음과 같은 데이터베이스 기능을 지원하기 위한 표준 API를 제공하고 있다.

- 데이터베이스를 연결하여 테이블 형태의 자료를 참조
- SQL문을 질의
- SQL문의 결과를 처리



- JDBC의 역할

ODBC(Open DataBase Connectivity)는 JDBC보다 먼저 마이크로소프트사가 개발한 것으로, C 또는 C++ 등의 언어를 이용하여 DBMS에 독립적으로 데이터베이스 프로그래밍을 가능하도록 하는 API 규격이다. JDBC도 ODBC와 마찬가지로 DBMS의 종류에 상관없이 쉽게 SQL문을 수행하고 그 결과를 처리할 수 있도록 설계되어 있다. 즉 한번 JDBC로 작성된 프로그램은 오라클, MySQL, SQLServer, DB2 등 어떤 DBMS를 사용하든지 소소의 수정을 최소화하여 바로 실행할 수 있다. 물론 이러한 DBMS에 독립적인 프로그래밍을 가능하도록 하려면 JDBC와 함께 JDBC 드라이버도 필요하다.

- JDBC 드라이버

JDBC드라이버는 JDBC 인터페이스에 맞추어 해당 DBMS에서 JDBC 관련 API 호출이 가능하도록 만든 업체나 다른 연구 기관에서 JDBC 드라이버를 유형1, 2, 3, 4로 크게 4가지로 분류하며, 각각의 JDBC 드라이버의 특징을 기술하면 다음 표와 같다.

- JDBC-ODBC 브릿지 드라이버
- Native-API 드라이버
- Net-Protocol 드라이버
- Native-Protocol 드라이버

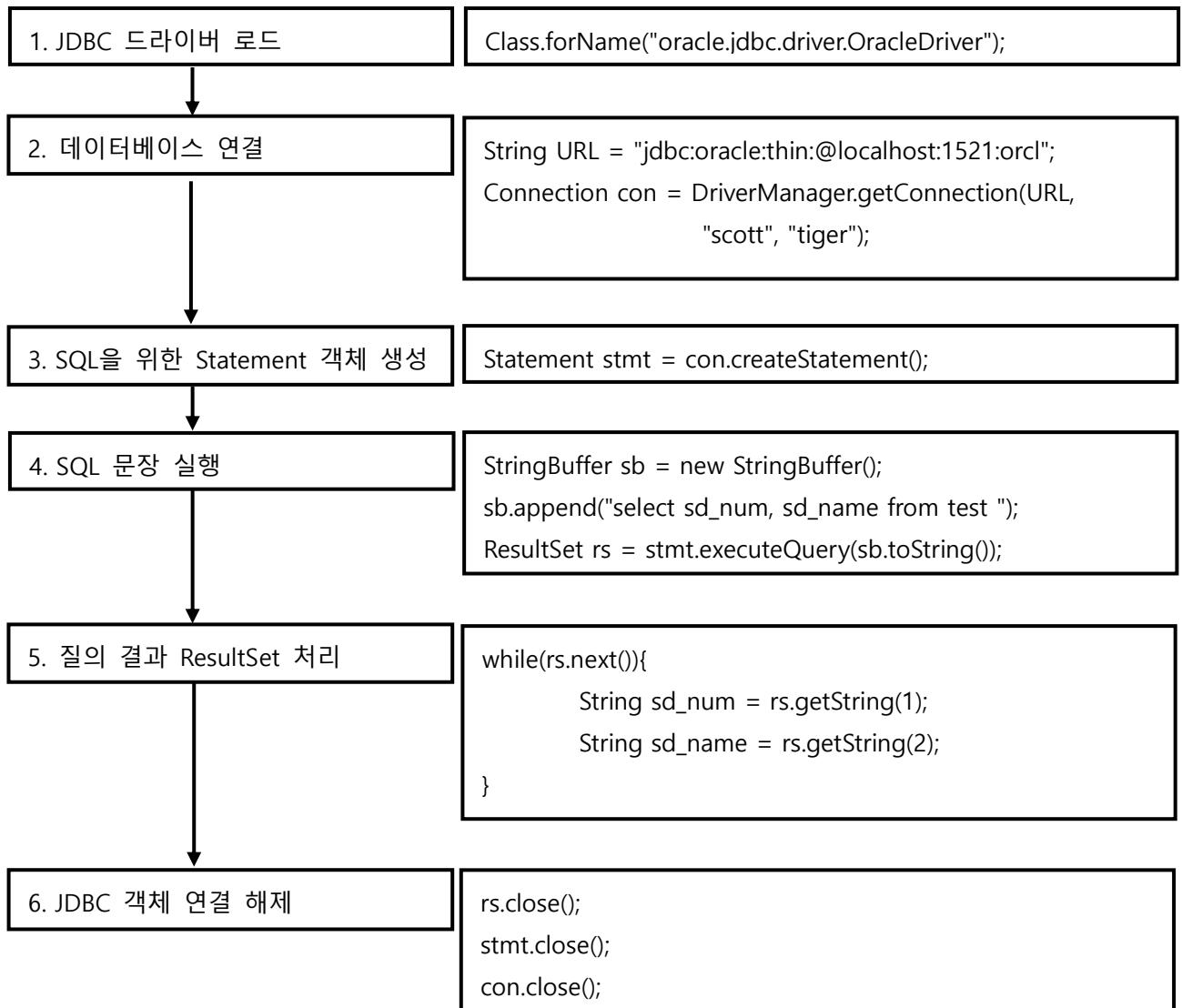
JDBC 유형	유형 이름	유형
유형 1	JDBC-ODBC 브릿지 드라이버	자바 소프트에서 제공하고 있는 JDBC 드라이버를 위한 JDBC-ODBC 브릿지 드라이버를 말한다. 즉 데이터베이스에 연결하기 위해 JDBC-ODBC 브릿지 드라이버는 단지 ODBC를 호출하는 역할만 수행한다. 이 드라이버를 이용하려면 JDBC-ODBC 브릿지 드라이버와 ODBC 드라이버가 설치되어 있어야 한다.
유형 2	Native-API 드라이버 (Partial Java JDBC Driver)	JDBC에서 데이터베이스의 API를 직접 호출하는 방식으로, 자바 코드와 일부의 C/C++의 API를 이용하는 드라이버이다
유형 3	Net-Protocol 드라이버 (Pure Java JDBC Driver With middleware)	JDBC와 데이터베이스 사이에 미들웨어(middleware)를 설치하여 처리하는 방식으로 JDBC 호출을 데이터베이스에 독립적인 프로토콜을 이용하여 해당 데이터베이스와 통신하는 드라이버로 순수 자바로 만들어진다
유형 4	Native-Protocol 드라이버 (Pure Java JDBC Driver)	순수 자바 기반의 드라이버 JDBC 호출을 데이터베이스에 직접 전달하는 방식이다. 그러므로 가장 처리 속도가 빠르고, 드라이버 외에는 추가적인 라이브러리가 필요 없다. 다만 각각의 해당 데이터베이스 회사에서 드라이버를 각각 제공해야 하고, 데이터베이스가 바뀌면 다시 그 데이터베이스 전용 드라이버를 설치해야 한다.

16.2. JDBC 프로그래밍 개요

- JDBC 프로그래밍 절차(JDBC 프로그래밍 절차 6단계)

일반적으로 JDBC 자바 프로그래밍 절차는 6단계로 구성된다.

첫번째 단계 [JDBC 드라이버 로드]에서부터 마지막 여섯 번째 단계 [JDBC 객체 연결 해제]까지 6단계를 거쳐 JDBC 프로그래밍 과정이 이루어진다.



- JDBC 관련 기본 클래스

JDBC 프로그래밍에 이용되는 클래스로는 가장 먼저 JDBC 드라이버를 로드하는 클래스 `java.lang.Class`가 있으며, 패키지 `java.sql`에 소속된 클래스 `DriverManager`, 인터페이스 `Connection`, `Statement`, `ResultSet` 등이 있다. 이러한 클래스의 용도와 이용 메소드를 정리하면 다음과 같다.

패키지	인터페이스(클래스)	클래스 용도	이용 메소드
<code>java.lang</code>	<code>클래스 Class</code>	지정된 JDBC 드라이버를 실행시간동안 메모리에 로드	<code>forName()</code>
<code>java.sql</code>	<code>클래스 DriverManager</code>	여러 JDBC 드라이버를 관리하는 클래스로 데이터베이스를 접속하여 연결 객체 반환	<code>getConnection()</code>
	<code>인터페이스 Connection</code>	특정한 데이터베이스 연결 상태를 표현하는 클래스로 질의할 문장 객체 반환	<code>createStatement(); close();</code>
	<code>인터페이스 Statement</code>	데이터베이스에 SQL 질의한 문장을 질의하여 그 결과인 결과집합 객체를 반환	<code>executeQuery(); close();</code>
	<code>인터페이스 ResultSet</code>	질의 결과의 자료를 저장하며 테이블 구조	<code>next(); getString(); getInt(); close();</code>

16.3 JDBC 프로그래밍 절차 6단계

1단계 : JDBC 드라이버 로드

JDBC 프로그래밍을 하기 위해 가장 먼저 해야 할 일은 JDBC 드라이버를 로드하는 일이다. 이미 설치한 JDBC 드라이버 내부를 살펴보면 드라이버의 파일 이름이 [OracleDriver]이고, [oracle.jdbc.driver]인 것을 알 수 있다.

그러므로 로드할 드라이버의 클래스 이름을 [oracle.jdbc.driver.OracleDriver]로 지정하여 문장 Class.forName()을 호출한다. 즉 문장 Class.forName()은 동적으로 JDBC 드라이버 클래스를 로드하는 것으로, 지정한 드라이버 클래스 객체화 되고 객체화 되고 객체화와 동시에 자동적으로 DriverManager.registerDriver()를 호출하여 DriverManager에서 관리하는 드라이버 리스트에 등록이 이루어진다. 이제 DriverManager를 이용해서 데이터베이스에 연결을 할 수 있는 환경을 갖춘 것이다.

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

다음은 여러 DBMS에 따라서 이용하는 드라이버 클래스 이름을 정리한 표이다.

DBMS 종류	JDBC 드라이버 로드 문장
ORACLE	Class.forName("oracle.jdbc.driver.OracleDriver");
MS SQL Server	Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
mSQL	Class.forName("com.imaginay.sql.msql.MsqlDriver");
MySQL	Class.forName("org.gjt.mm.mysql.Driver"); Class.forName("com.mysql.jdbc.Driver");
ODBC	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

2단계 : 데이터베이스 연결

1단계 이미 로드된 드라이버 클래스에게 지정한 데이터베이스로 연결을 요구하는 단계로 DriverManager.getConnection()을 호출한다. 클래스 DriverManager의 static 메소드인 getConnection()을 호출하면 등록된 드라이버 중에서 주어진 URL로 데이터베이스에 연결할 수 있는 드라이버를 찾아서 그 Driver 클래스의 메소드 connect()를 호출하고, 결과인 Connection 객체를 반환하게 된다. 메소드인 getConnection()은 필요 시 인자로 데이터베이스 URL, 데이터베이스 사용자 이름, 암호를 이용한다.

```
String URL = "jdbc:oracle:thin:@localhost:1521:orcl";
Connection con = DriverManager.getConnection(URL, "scott", "tiger");
```

반환된 Connection 객체의 역할은 데이터베이스와 애플리케이션 간의 연결을 유지시켜주는 것이다. 데이터베이스의 연결은 Connection 객체의 close 메소드가 호출될 때까지 지속된다.

드라이버로부터 Connection 객체를 가져오기 위해서는 실제 DBMS가 있는 장소를 알려주어야 하는데, 이는 문자열 형태의 데이터베이스 URL 정보로 표현된다.

URL - "jdbc:oracle:thin:@localhost:1521:orcl"

IP - oracle이 설치된 ip를 작성한다.

PORT - oracle의 포트를 작성하는데 일반적으로 설치 하였다면 1521 이다.

ORACLE_SID – oracle을 설치할 때 설정하는 것인데 대부분 일정하지 않다. SID(System Identifier, 시스템

식별자는 오라클 데이터베이스가 아닌 오라클 인스턴스를 식별하기 위한 유일한 값이고, 그 값이 데이터베이스 이름이 된다.

user - oracle의 user를 말한다.

password – oracle user에 대한 password를 말한다.

여러 DBMS에 따른 데이터베이스 URL을 살펴보면 다음과 같다.

DBMS 종류	JDBC URL
ORACLE	"jdbc:oracle:thin:@localhost:1521:orcl"
MS SQL Server	"jdbc:microsoft:sqlserver://localhost:1433"
mSQL	"jdbc:mysql://localhost:1114/univdb"
MySQL	"jdbc:mysql://localhost:3306/univdb"
ODBC	"jdbc:odbc:mydb"

3단계 : SQL을 위한 Statement 객체 생성

이제 반환된 Connection 객체를 이용해서 SQL문을 실행하고 그 결과를 반환 받을 수 있는 Statement, PreparedStatement, CallableStatement 객체를 생성하는 단계이다.

인터페이스인 Statement, PreparedStatement, CallableStatement는 질의 문장인 SQL문을 추상화시킨 인터페이스이다. 이들은 Connection 객체인 메소드 createStatement()를 호출하여 Statement 객체를 얻어온다.

```
Statement stmt = con.createStatement();
```

마찬가지로 Connection 객체의 다음 메소드 prepareStatement(), prepareCall()를 이용해 PreparedStatement, CallableStatement도 얻을 수 있다. 다만 이 메소드를 호출할 때는 SQL문장을 인자로 이용해야 한다.

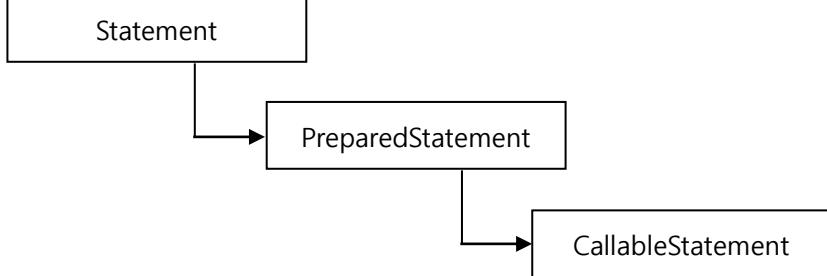
```
PreparedStatement pstmt = con.prepareStatement(SQL);
CallableStatement cst = con.prepareCall(SQL);
```

JDBC는 Statement보다 그 기능이 향상된 PreparedStatement와 CallableStatement를 제공한다.

PreparedStatement는 순전히 자바 언어로만 Statement의 단점을 극복한 인터페이스이고,

CallableStatement는 DBMS의 저장 함수인 Stored Procedure를 사용하기 위한 인터페이스이다.

PreparedStatement는 Statement를 상속받고, CallableStatement는 PreparedStatement를 상속받은 관계이다.



위 3개의 Statement 중에서 가장 쉽게 이용할 수 있는 인터페이스가 Statement이나 성능이 떨어지는

단점이 있다. PreparedStatement는 SQL 문장에서 일정 부분만 변화면서 여러 번 반복해서 사용되는 SQL을 이용할 때 상당히 편리하고 효율적이다.

질의문장을 위한 인터페이스 종류	특징
Statement	Connection 객체에서 createStatement() 메소드 호출로 생성. 단순한 SQL 문장을 보낼 때 사용되며, 성능이다 효율성에서 가장 낮음
PreparedStatement	Connection 객체에서 prepareStatement() 메소드를 호출로 생성. 주로 PreparedStatement 클래스는 한번 사용되고 마는 SQL문이 아니라 여러 번 반복해서 사용되는 SQL을 다룰 때 편리. 컴파일할 때 에러를 체크하기 때문에 좀 더 효율적이며 처리하는 속도 역시 훨씬 빠름.
CallableStatement	Connection 객체에서 prepareCall() 메소드의 호출로 생성. CallableStatement 객체는 저장 프로시저를 호출할 때 사용

4단계 : SQL문장 실행

질의 문장을 실행하기 위해, 객체 Statement의 메소드 executeQuery(SQL)의 인자로 SQL 문장을 넣어 호출한다. DML의 한 종류인 SELECT문장은 질의 결과로 테이블 형태의 결과를 반환한다. 메소드 executeQuery()도 SELECT문장과 같이 질의 결과로 테이블 형태의 결과를 반환하는데, 이 반환 유형이 인터페이스 ResultSet이다.

```
ResultSet rs = stmt.executeQuery("select * from student");
```

즉 메소드 **executeQuery()**는 데이터베이스 구조와 테이블의 내용에 영향을 미치지 않는 select 문 등의 질의에 적합하다. 또한 객체 Statement는 메소드 executeUpdate()를 제공하는데 이는 create 또는 drop 과 같은 DDL이나 insert, delete, update와 같이 테이블의 내용을 변경하는 DML 문장에 사용한다. 메소드 **executeUpdate()**는 질의 문장이 DML인 경우 변경된 행의 수인 정수를 반환하고, DML인 경우 0을 반환한다.

```
int rowCount= stmt.executeUpdate("delete from student where sd_name='김정수' ");
```

객체 Statement는 메소드 execute()도 제공하는데, execute()는 메소드 executeQuery()와 executeUpdate(), 둘 중에서 어느 것을 사용해야 할 지 모를 때 유용하다. 메소드 execute()는 DML, DDL등의 모든 질의 문장에 사용할 수 있으며, 테이블 형태의 결과인 ResultSet을 얻기 위해서는 메소드 호출 후 별도로 마련된 메소드 getResultSet()을 사용해야 한다. 마찬가지로 질의 결과로 수정된 행 수를 얻기 위해서는 메소드 호출 후, 메소드 getCount()을 사용한다.

```
stmt.execute("select * from student");
```

```
ResultSet rs = stmt.getResultSet();
```

```
stmt.execute("delete from student where sd_name='김정수' ");
```

```
int updateCount = stmt.getUpdateCount();
```

객체 Statement에서 SQL 질의 문장을 실행하는 메소드 executeQuery()와 executeUpdate(), execute() 3가지를 정리하면 다음과 같다.

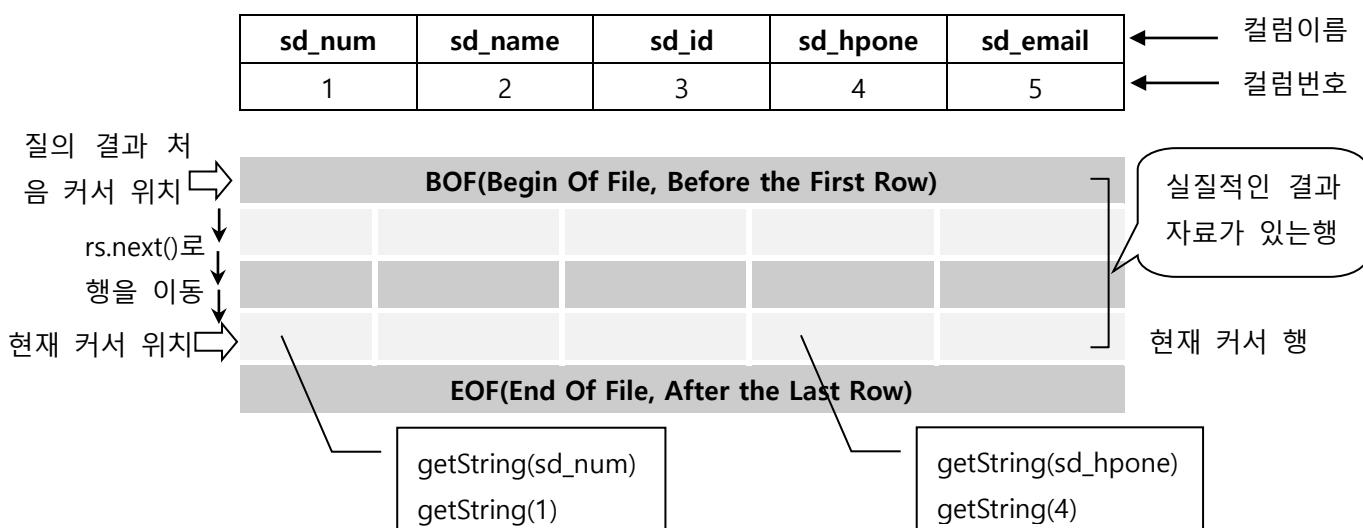
질의 메소드 종류	반환 자료유형	특징
executeQuery()	ResultSet	주로 select와 같이 데이터베이스에 변경을 주지 않는 SQL문을 실행할 경우에 사용하며, 그 결과로 하나의 ResultSet 객체를 반환.
executeUpdate()	int	데이터베이스의 값 또는 구조를 변경시키는 MDL인 insert, update, delete와 같은 질의와 create, drop과 같은 DDL 구문을 사용할 때 주로 이용하며, 질의 수행 후, 영향을 받은 테이블의 행 수인 정수를 반환하는데, DDL인 경우 0을 반환.
execute()	boolean 첫 결과가 ResultSet이면 true, 결과가 행 수 또는 없으면 false	실행해야 할 SQL문이 어떠한 종류의 것인지를 모를 경우에 유용하며, Statement 객체를 실행한 결과가 하나 이상의 ResultSet 객체를 반환하거나 ResultSet 객체와 데이터에 영향을 끼친 행의 수 등이 함께 반환될 경우에 사용. 또한 반환될 객체가 어떤 형태인지 예측할 수 없을 경우에도 사용

5단계 : 질의 결과 ResultSet

객체 Statement의 메소드 executeQuery()를 호출하여 반환 받은 객체 ResultSet은 테이블 형태의 결과를 추상화한 인터페이스이다. 메소드 executeQuery()는 질의 결과가 없더라도 null을 반환하지 않는다.

```
ResultSet rs = stmt.executeQuery("select * from student");
```

ResultSet에는 결과의 현재 행(row)을 가리키는 커서(cursor)라는 개념이 있으며, 이 커서를 다음 행으로 이동시키는 메소드가 next()이다. ResultSet은 실질적으로 질의 결과의 자료가 있는 영역과 함께 첫 행 자료 이전(Before the First Row)에 BOF(Begin Of File)행과 마지막 행 자료 이후(After the Last Row)에 EOF(End Of File)라는 실제 자료가 없는 영역이 있다. 각각의 행에서 각 컬럼은 select 문에서 이용한 칼럼이름 또는 번호 순으로 식별할 수 있는데, 편의를 위하여 번호는 1번부터 시작한다.



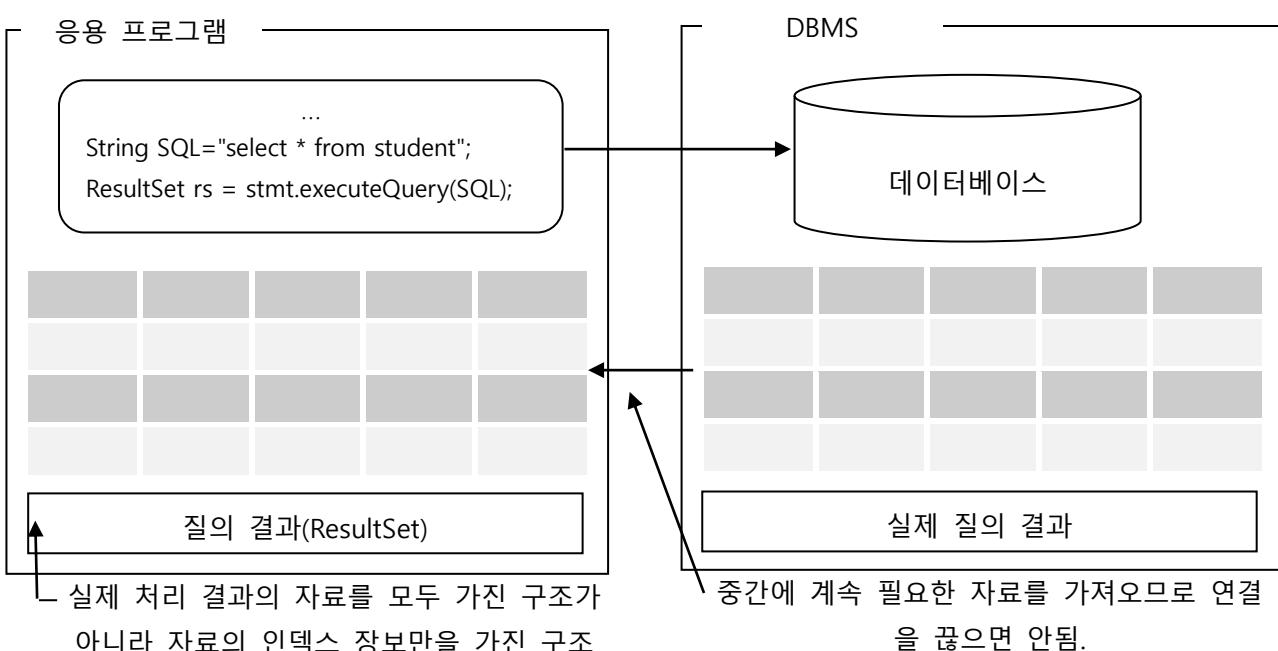
메소드 `executeQuery()`로부터 `ResultSet`을 받은 경우, 제일 처음에는 커서가 바로 BOF 행에 위치한다. 메소드 `next()`는 커서를 다음 행으로 한 행 이동하며 이동된 행이 실질적인 자료가 있는 행이면 `true`를 반환하고, 그렇지 않고 BOF나 EOF과 같이 실질적인 자료가 없는 행이면 `false`를 반환한다. 그러므로 다음과 같이 `while` 문장을 이용하면, `ResultSet`에서 결과 자료가 있는 1행부터 마지막 행까지 순회할 수 있다.

```
while(rs.next()){
    //현재 커서가 있는 행의 컬럼을 처리
    ...
}
```

커서가 있는 행에서 칼럼 자료를 참조하기 위해 `ResultSet`이 제공하는 메소드 `getString()`을 이용한다. `getString()`의 인자는 칼럼 이름을 문자열로 직접 쓰거나 또는 칼럼 번호를 이용할 수 있다. 칼럼 값의 자료유형에 따라 메소드 `getString()` 뿐만 아니라 `getInt()`, `getDouble()`, `getDate()` 등 다양한 칼럼 반환 메소드를 제공한다. JDBC 드라이버는 `getXXX()` 메소드를 사용하여 특정 칼럼 값을 가져올 때 데이터베이스의 자료 유형을 해당하는 가장 유사한 자바 유형으로 변환하여 반환한다.

```
<% while(rs.next()){ %>
<%= rs.getString(1) %>
<%= rs.getString("sd_name") %>
<%= rs.getString("sb_id") %>
<%= rs.getString(4) %>
<%= rs.getString(5) %>
<% } %>
```

SQL 질의 결과에서 `ResultSet` 객체가 실제로 처리결과 자료 자체를 가지고 있는 것은 아니며, `ResultSet` 객체는 해당 자료에 대한 포인터의 역할을 수행하는 인덱스만을 가지고 있다. 그러므로 `ResultSet`은 커서를 이동하면서 필요한 자료를 그때 그때 가져오므로 `ResultSet`의 작업이 모두 종료한 이후에 연결을 해제해야 한다.



클래스 ResultSet에서 이용되는 주요 메소드를 정리하면 다음과 같다.

반환 자료 유형	메소드	특징
boolean	absolute(int row)	
boolean	next()	
boolean	previous()	
boolean	first()	
boolean	last()	
boolean	afterLast()	
boolean	beforeFirst()	
int	findColumn(String cname)	인자인 칼럼의 이름이 ResultSet 객체의 몇 번째 열인지 위치 값을 반환
Date	getDate()	
String	getString()	
int	getInt()	
long	getLong()	
float	getFloat()	
double	getDouble()	
ResultSetMetaData	getMetaData()	ResultSet의 Column에 대한 자료유형과 속성에 대한 정보를 얻어오기 위해 ResultSetMetaData 객체를 반환
void	close()	ResultSet 객체의 연결 해제

6단계 : JDBC 객체 연결 해제

JDBC 프로그래밍의 마지막 단계는 이미 사용한 JDBC 객체의 연결을 해제하는 일이다. 특히 데이터베이스 연결을 의미하는 Connection 객체의 연결 해제는 메모리와 서버의 부하의 관점에서 중요하다. 데이터베이스 작업이 종료되었으면 Connection 객체 변수 con에서 메소드 close()를 이용하여 연결을 해제한다.

```
con.close();
```

Connection 객체의 연결을 해제하기 전에 객체 ResultSet과 Statement도 메소드 close()를 호출하여 명시적으로 연결을 해제함으로써 시스템 자원을 효율적으로 이용하도록 하는 것이 좋다.

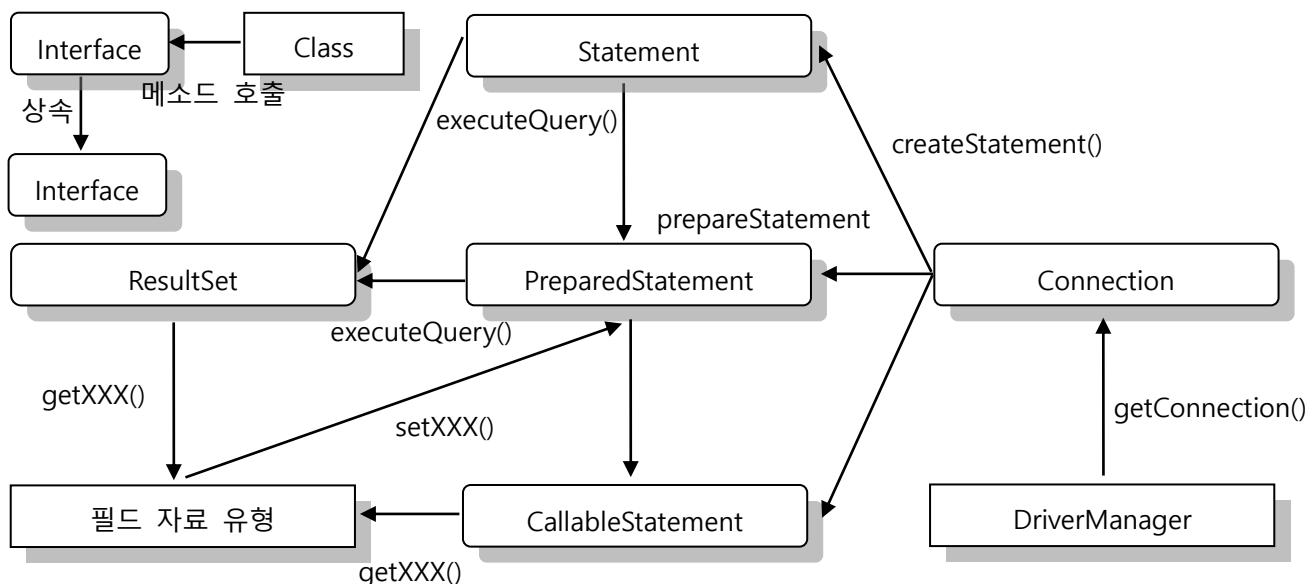
```
rs.close();
stmt.close();
```

JDBC 객체의 연결 해제는 관련 작업의 종료를 의미하므로 데이터 처리 시점과 연결 해제 시점을 잘 고려해야 할 것이다.

16.4 JDBC 관련 인터페이스와 클래스

- 패키지 java.sql

JDBC 프로그래밍을 하기 위해 지금까지 살펴온 JDBC 관련 클래스와 인터페이스 중에서 패키지 java.sql에 소속된 인터페이스와 클래스에서 메소드 호출과 상속 관계를 표현하면 다음과 같다.



dbconnect.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>데이터베이스 예제 : dbconnect.jsp</title>
</head>
<body>

<%@ page import="java.sql.*" %>

<h2>데이터베이스 드라이버와 DB 연결 점검 프로그램 </h2>
<%
try {
    String driverName = "oracle.jdbc.driver.OracleDriver";
    String dbURL = "jdbc:oracle:thin:@localhost:1521:orcl";

    Class.forName(driverName);
    Connection con = DriverManager.getConnection(dbURL, "scott", "tiger");
    out.println("Oracle 서버에 성공적으로 접속했습니다");
    con.close();
}
%>
```

```

catch (Exception e) {
    out.println("Oracle 서버에 접속하는데 문제가 있습니다. <hr>");
    out.println(e.getMessage());
    e.printStackTrace();
}
%>

</body>
</html>

```

- 테이블 조회 프로그램

JDBC 인터페이스를 이용하여 이미 만든 테이블 student의 모든 내용을 질의하여 브라우저에 출력하는 프로그램을 작성하자. 질의를 위하여 가장 간단한 인터페이스 Statement를 이용하며, 테이블 조회의 결과 처리를 위하여 인터페이스 ResultSet을 이용한다. 가장 먼저 드라이버를 로드하고 데이터베이스를 연결한다.

다음으로 데이터베이스 연결에 성공한 Connection 객체의 메소드 createStatement()를 호출하여 객체 Statement를 반환 받아 변수 stmt에 저장한다. 객체 Statement에서 질의문 "select * from student"를 인자로 메소드 executeQuery()를 호출하여, 그 결과를 ResultSet 객체 변수 result에 저장한다.

마지막으로 테이블 조회의 결과가 저장된 ResultSet 객체 변수 result에서 메소드 next()를 호출하며 while 반복을 수행한다. 클래스 ResultSet의 메소드 next()는 조회 결과의 테이블에서 모든 행을 순회하도록, 행의 조회가 완료되면 false를 반환하여 while 반복을 종료한다. 조회 결과 테이블의 행에서 각각의 필드를 조회하기 위한 클래스 ResultSet의 메소드 getString()을 이용한다. 메소드 getString()의 인자는 필드 이름이나 필드 번호를 사용하며 필드 번호는 만들어진 순서에 따라 1번부터 순서대로 정해진다. 다음은 테이블 student의 모든 내용을 질의하여 브라우저에 출력하는 예제 selectdb의 소스와 결과이다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>데이터베이스 예제 : 테이블 student.jsp 조회</title>
</head>
<body>
<%@ page import="java.sql.*" %>
<h2> 테이블 student 조회 프로그램 </h2>
<hr><center>
<h2>학생정보 조회</h2>
<%
Connection con = null;
Statement stmt = null;
String driverName = "oracle.jdbc.driver.OracleDriver";
String dbURL = "jdbc:oracle:thin:@localhost:1521:orcl";

```

```

try {
    Class.forName(driverName);
    con = DriverManager.getConnection(dbURL, "scott", "tiger");
    stmt = con.createStatement();
    ResultSet result = stmt.executeQuery("select sd_num, sd_name, sd_id, sd_passwd, s_num,
                                         sd_jumin, sd_hpone, sd_address, sd_email, sd_date from student");
%>
<table width="100%" border="2" cellpadding="1">
    <tbody align="center">
        <tr>
            <th>학번</th>
            <th>이름</th>
            <th>아이디</th>
            <th>비밀번호</th>
            <th>학과번호</th>
            <th>주민등록번호</th>
            <th>휴대폰</th>
            <th>주소</th>
            <th>이메일</th>
            <th>등록일자</th>
        </tr>
<%>
        while (result.next()) {
%>
        <tr>
            <td><%= result.getString(1) %></td>
            <td><%= result.getString(2) %></td>
            <td><%= result.getString(3) %></td>
            <td><%= result.getString(4) %></td>
            <td><%= result.getString(5) %></td>
            <td><%= result.getString(6) %></td>
            <td><%= result.getString(7) %></td>
            <td><%= result.getString(8) %></td>
            <td><%= result.getString(9) %></td>
            <td><%= result.getString(10) %></td>
        </tr>
<%>
        }
        result.close();
    }
}

```

```

        catch(Exception e) {
            out.println("student 테이블을 조회하는데 문제가 있습니다. <hr>");
            out.println(e.toString());
            e.printStackTrace();
        }
    finally {
        if(stmt != null) stmt.close();
        if(con != null) con.close();
    }
%>
</tbody>
</table>
</center>
</body>
</html>

```

- 인터페이스 PreparedStatement를 이용한 검색 프로그램

(PreparedStatement는 SQL문의 구조는 동일하나 조건이 다른 문장을 변수 처리함으로써 항상 SQL문을 동일하게 처리할 수 있는 인터페이스이다)

인터페이스 PreparedStatement는 미리 컴파일된 SQL 문장을 포함함으로써 Statement 객체보다 실행 속도가 빠르며 여러 번 실행할 SQL문장의 실행에 효율적이다. SQL문의 조건에 사용할 값이나 필드가 사용자의 선택이나 특정 작업 결과에 따라서 결정되어야 하는 경우, 인터페이스 PreparedStatement에 사용할 SQL문장에는 표시[?]인 매개변수를 포함하고, 이후에 setXXX() 메소드로 그 매개변수에 값을 지정함으로써 SQL문장을 완성시켜 실행한다

```

String sql = "select * from student where sd_num=?";
String num = request.getParameter("snum");
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setInt(1, num);
ResultSet result = pstmt.executeQuery();

```

다음은 인터페이스 PreparedStatement를 이용하여 테이블 student에서 학생 이름으로 검색하는 예제로, 그 결과를 브라우저에 출력하는 프로그램을 작성하자. 예제 selectname.html에서 학생 이름을 입력 받을 폼을 구성하여 [보내기]버튼을 누르면, 예제 selectname.jsp 를 실행하여 테이블 student에서 학생 이름으로 검색하도록 한다. 검색은 입력된 문자열과 시작하는 모든 이름을 검색하도록 하며 결과는 학생 정보를 모두 출력한다.

```

<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>테이블 student.html 에서 이름으로 조회</title>
</head>

```

```

<body>
    <h2>테이블 student에서 이름으로 조회하는 프로그램 </h2><hr>
    <h3>조회할 이름을 입력하세요.</h3>
    <form method="post" name="test" action="selectname.jsp" >
        이름 : <input type="text" name="sname">
        <input type="submit" value="보내기">
    </form>
</body>
</html>

```

인터페이스 PreparedStatement를 사용하기 위해 질의문 내부에 검색에 따라 변하는 부분을 [?]로 삽입한다. 그리고 질의문 [?]부분에 대입할 문자열을 [request.getParameter("sname")+"%"]로 구성한다. 입력된 문자열로 시작하는 모든 이름의 학생을 찾는 검색 조건을 만족시키기 위해, 질의문에서 where절은 like 문장으로 하고, 폼에서 입력한 이름 뒤에 [%]를 붙여 입력한 문자열로 시작하는 모든 이름을 검색하도록 한다.

인터페이스 PreparedStatement 객체를 얻기 위하여, 위에서 만든 질의문 SQL을 인자로 Connection 객체 변수 con의 메소드 prepareStatement(SQL)를 호출한다. 다음으로 아직 완성하지 않는 질의문[?]부분에 지정할 값을 대입하기 위해 객체 변수 pstmt로 메소드 setString(1, name)을 호출한다. 이때 인자 1은 첫번째 미완성 위치인 [?] 부분을 의미하며, 인자 name은 지정된 [?]부분에 대체될 자료값을 의미한다.

만일 변수 name의 문자열 값이 '김%'라면 문장 pstmt.setString(1, name)에 의하여 다음과 같은 질의문이 완성된다. 객체 변수 pstmt에서 인자 없이 메소드 executeQuery()를 호출하여 그 결과를 ResultSet 객체 변수 result에 대입한다.

```

"select * from student where name like ?";
    ↑
    pstmt.setString(1, name);
    ↓
"select * from student where name like '김%' ";

```

selectname.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>데이터베이스 예제 : 테이블 student name 으로 조회</title>
</head>
<body>
<%@ page import="java.sql.*" %>
<% request.setCharacterEncoding("UTF-8"); %>
<h2>테이블 student에서 이름으로 조회하는 프로그램 </h2>

```

```

<hr><center>
<h2>학생정보 조회</h2>
<%
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet result = null;
    String driverName = "oracle.jdbc.driver.OracleDriver";
    String dbURL = "jdbc:oracle:thin:@localhost:1521:orcl";
    String sql = "select no, sd_num, sd_name, sd_id, sd_passwd, s_num, sd_jumin, sd_hpone,
                 sd_address, sd_email, sd_date from student where sd_name like ?";
    String name = request.getParameter("sname") + "%";
    int rowCount = 0;
    try {
        Class.forName(driverName);
        con = DriverManager.getConnection(dbURL, "scott", "tiger");
    }catch(ClassNotFoundException ex1) {
        out.println("드라이버 로딩 에러 :" + ex1.toString());
        return;
    }catch(Exception ex2) {
        out.println("DB 접속 실패 :" + ex2.toString());
        return;
    }
    try{
        pstmt = con.prepareStatement(sql);
        pstmt.setString(1, name);
        result = pstmt.executeQuery();
    }>
    <table width="100%" border="2" cellpadding="1">
        <tbody align="center">
            <tr>
                <th>학번</th>
                <th>이름</th>
                <th>아이디</th>
                <th>비밀번호</th>
                <th>학과번호</th>
                <th>주민등록번호</th>
                <th>휴대폰</th>
                <th>주소</th>
                <th>이메일</th>
                <th>등록일자</th>

```

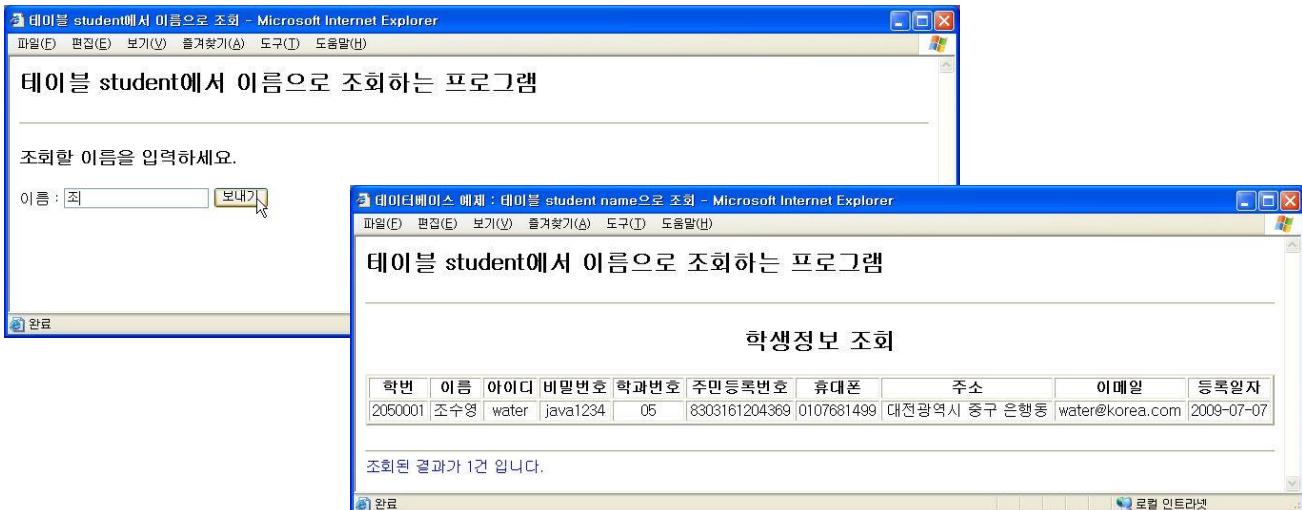
```

</tr>
<%
    while (result.next()) {
%>
<tr>
<td><%= result.getInt("sd_num") %></td>
<td><%= result.getString("sd_name") %></td>
<td><%= result.getString("sd_id") %></td>
<td><%= result.getString("sd_passwd") %></td>
<td><%= result.getString("s_num") %></td>
<td><%= result.getString("sd_jumin") %></td>
<td><%= result.getString("sd_hpone") %></td>
<td><%= result.getString("sd_address") %></td>
<td><%= result.getString("sd_email") %></td>
<td><%= result.getDate("sd_date") %></td>
</tr>
<%
    rowCount++;
}
}
catch(Exception e) {
    out.println("student 테이블에 데이터를 검색하는데 문제가 있습니다. <hr>");
    out.println(e.toString());
    e.printStackTrace();
}
finally {
    if(result != null) result.close();
    if(pstmt != null) pstmt.close();
    if(con != null) con.close();
}
%>
</tbody>
</table>
</center>
<p><hr><font color="blue">
<%
    if (rowCount == 0)
        out.println("조회된 결과가 없습니다.");
    else
        out.println("조회된 결과가 " + rowCount + "건 입니다.");

```

```
%>
</font>
</body>
</html>
```

[결과]



- 메타데이터 조회

메타데이터는 데이터를 위한 데이터를 말한다. 데이터베이스에서 메타데이터란 데이터베이스 자체에 대한 정보 또는 테이블 자체 및 칼럼에 대한 정보를 말한다. 이러한 메타데이터 정보를 지원하기 위해 JDBC는 인터페이스 DatabaseMetaData와 ResultSetMetaData를 제공한다.

인터페이스 ResultSetMetaData의 주요 메소드를 살펴보면 다음과 같다.

반환 유형	메소드 이름	기능
int	getColumnCount()	ResultSet의 칼럼 수를 반환
int	getColumnDisplaySize(int col)	인자 col에 지정한 칼럼의 문자 최대 표현 크기를 반환
String	getColumnName(int col)	인자 col에 지정한 칼럼의 제목을 반환
String	getColumnName(int col)	인자 col에 지정한 칼럼의 이름을 반환
int	getColumnType(int col)	인자 col에 지정한 칼럼의 유형을 반환, 반환값은 java.sql.Types에 정의된 정수 상수로 반환
String	getColumnTypeName(int col)	인자 col에 지정한 칼럼의 자료유형 이름을 반환
String	getTableName(int col)	인자 col이 있는 테이블 이름을 반환
boolean	isAutoIncrement(int col)	인자 col에 지정한 칼럼이 autoincrement 여부를 반환

인터페이스 DatabaseMetaData에서 드라이버와 데이터베이스에 관한 주요 메소드를 살펴보면 다음과 같다.

반환 유형	메소드 이름	기능
int	getDriverMajorVersion()	드라이버의 주 버전을 반환
int	getDriverMinorVersion()	드라이버의 부 버전을 반환

String	getDriverName()	드라이버 이름을 반환
String	getDriverVersion()	드라이버 버전을 반환
int	getDatabaseMajorVersion()	데이터베이스의 주 버전을 반환
int	getDatabaseMinorVersion()	데이터베이스의 부 버전을 반환
String	getDatabaseProductName()	데이터베이스 상품 이름을 반환
String	getDatabaseProductVersion()	데이터베이스의 상품 버전을 반환

- ResultSetMetaData

인터페이스 ResultSetMetaData를 이용하여 테이블 student의 칼럼이름, 칼럼유형, 칼럼크기 정보 등의 메타정보를 출력하는 프로그램을 작성하자. 데이터베이스를 연결하는 모듈은 지금까지의 프로그램과 동일하며 인터페이스 ResultSetMetaData의 객체를 얻기 위해 테이블 student를 모두 조회하는 SQL 문장으로 ResultSet객체를 반환하여 저장한다.

인터페이스 ResultSet의 메소드 getMetaData()를 호출하여 ResultSetMetaData의 객체를 변수 rsmd에 저장한다. 인터페이스 ResultSetMetaData의 메소드 getColumnCount()는 ResultSet의 칼럼 수를 반환한다.

```
result = stmt.executeQuery("select no, sd_num, sd_name, sd_id, sd_passwd, s_num, sd_jumin, sd_hpone,
                           sd_address, sd_email, sd_date from student");
rsmd = result.getMetaData();
int cCount = rsmd.getColumnCount();
```

칼럼 수만큼 반복을 하면서 ResultSetMetaData의 메소드 getColumnLabel()을 이용하여 칼럼의 이름을 반환받아 출력한다.

```
for ( int i = 1; i <= cCount; i++ ) { %>
    <td><%= rsmd.getColumnLabel(i) %></td>
<% } %>
```

마찬가지로 칼럼 수만큼 반복을 하면서 ResultSetMetaData의 메소드 getColumnTypeName()을 이용하여 칼럼의 유형을 반환받아 출력한다.

```
<td><%= rsmd.getColumnTypeName(i) %></td>
```

또한 ResultSetMetaData의 메소드 getPrecision()을 이용하여 칼럼의 크기를 반환받아 출력한다.

```
<td><%= rsmd.getPrecision(i) %></td>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>인터페이스 ResultSetMetaData.jsp 조회</title>
</head>
<body>
<%@ page import="java.sql.*" %>
```

```

<h2>테이블 student의 테이블 메타데이터 조회 프로그램 </h2>
<hr><center>
<h2>인터페이스 ResultSetMetaData 이용</h2>
<%
Connection con = null;
Statement stmt = null;
ResultSet result = null;
ResultSetMetaData rsmd = null;
String driverName = "oracle.jdbc.driver.OracleDriver";
String dbURL = "jdbc:oracle:thin:@localhost:1521:orcl";
String dbuser = "scott";
String dbpasswd = "tiger";
try {
    Class.forName(driverName);
    con = DriverManager.getConnection(dbURL, dbuser, dbpasswd);
}catch(ClassNotFoundException ex1) {
    out.println("드라이버 로딩 에러 :" + ex1.toString());
    return;
}catch(Exception ex2) {
    out.println("DB 접속 실패 :" + ex2.toString());
    return;
}
try{
    stmt = con.createStatement();
    result = stmt.executeQuery("select no, sd_num, sd_name, sd_id, sd_passwd, s_num, sd_jumin,
                               sd_hpone, sd_address, sd_email, sd_date from student");
    rsmd = result.getMetaData();
    int cCount = rsmd.getColumnCount();
%>


| 일련번호 | 학번 | 이름 | 아이디 | 비밀번호 | 학과번호 | 주민등록번호 | 휴대폰 |
|------|----|----|-----|------|------|--------|-----|
|------|----|----|-----|------|------|--------|-----|


```

```

<th>주소</th>
<th>이메일</th>
<th>등록일자</th>
</tr>
<tr>
<%
    for ( int i = 1; i <= cCount; i++ ) {
%>
    <td><%= rsmd.getColumnName(i) %></td>
<%
    }
%>
</tr>
<tr>
<%
    for ( int i = 1; i <= cCount; i++ ) {
%>
    <td><%= rsmd.getColumnTypeName(i) %></td>
<%
    }
%>
</tr>
<tr>
<%
    for ( int i = 1; i <= cCount; i++ ) {
%>
    <td><%= rsmd.getPrecision(i) %></td>
<%
    }
%>
</tr>
<%
}
catch(Exception e) {
    out.println("student 테이블에 데이터를 조회하는데 문제가 있습니다.. <hr>");
    out.println(e.toString());
    e.printStackTrace();
}
finally {
    if(result != null) result.close();
}

```

```

if(stmt != null) stmt.close();
if(con != null) con.close();
}

%>
</tbody>
</table>
</center>
</body>
</html>

```

[결과]

The screenshot shows a Microsoft Internet Explorer window with the title "인터페이스 ResultSetMetaData 조회 - Microsoft Internet Explorer". The main content area displays the heading "테이블 student의 테이블 메타데이터 조회 프로그램" and "인터페이스 ResultSetMetaData 이용". Below this, there is a table showing the structure of the student table:

일련번호 호	학번	이름	아이디	비밀번호	학과번호	주민등록 번호	휴대폰	주소	이메일	등록일 자
NO NUMBER	SD_NUM VARCHAR2	SD_NAME VARCHAR2	SD_ID VARCHAR2	SD_PASSWD VARCHAR2	S_NUM VARCHAR2	SD_JUMIN CHAR	SD_HPONE VARCHAR2	SD_ADDRESS VARCHAR2	SD_EMAIL VARCHAR2	SD_DATE DATE
0	8	12	12	12	2	13	15	80	40	0

17. 커넥션 풀

17.1 커넥션 풀 개념

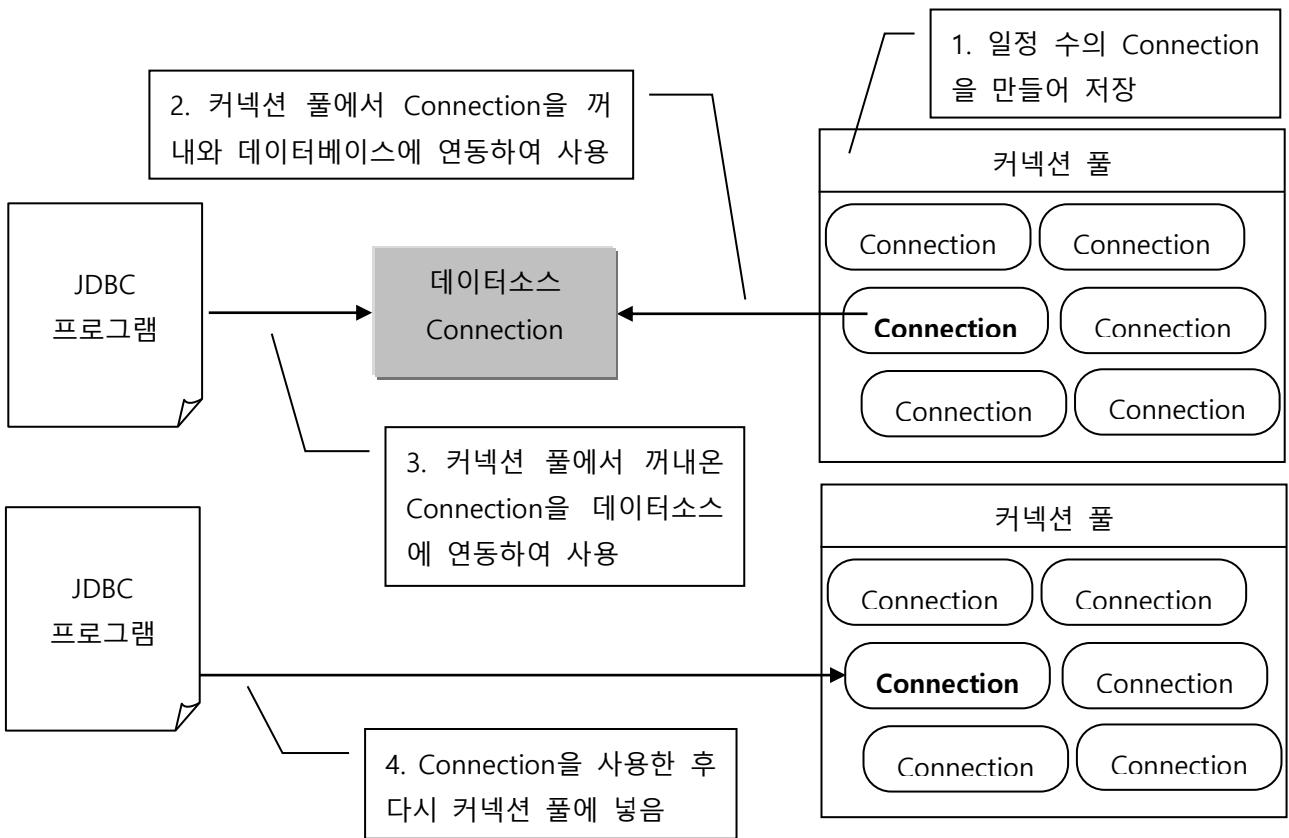
- 일반적인 데이터베이스 연결 프로그램의 문제점

일반적으로 JDBC 프로그래밍 절차의 단계 1단계에서 6단계까지 모두를 하나의 JSP 페이지 내부에 실행하였다. 만일 하나의 프로젝트를 수행한다고 보면 수 많은 JSP 페이지에서 이러한 데이터베이스 연결 모듈을 구현하고, 구현된 페이지수는 수 많은 클라이언트의 요청에 따라 데이터베이스 연결 과정과 연결 해제 과정을 끊임없이 반복할 것이다. 즉 클라이언트의 요청이 있을때마다 DriverManager 객체로부터 Connection 객체를 얻어와 데이터베이스 작업을 수행한 후 다시 Connection 객체를 해제한다.

이러한 데이터베이스 연결 작업은 서버의 자원을 이용하는 작업으로, 계속적으로 발생한다면 시스템에 상당히 부하를 주는 요소이다. 그러므로 대규모 시스템일수록 데이터베이스의 커넥션 연결은 매우 중요하며, 일관된 커넥션 관리가 필요하다. 이러한 커넥션 문제를 해결하기 위해 고안된 방식이 데이터베이스 커넥션 풀(Database Connection Pool) 관리 기법이다

- 커넥션 풀(Connection Pool) 정의

커넥션 풀(Connection Pool)이란 미리 여러 개의 데이터베이스 커넥션을 만들어 확보해 놓고 클라이언트의 요청이 있는 경우, 커넥션을 서비스해 주는 커넥션 관리 기법이다.



- 커넥션 풀(Connection Pool) 기능

JDBC를 사용하여 데이터베이스와 연동할 때 가장 많은 자원을 낭비하는 것이 Connection을 연결하는 작업이다. DAO 클래스의 각 메서드 내에서 매번 Connection을 맺고 CLOSE 처리한다. thread-safe 하기 위해서는 반드시 지켜져야 할 사항이다.

Connection Pool 기능은 많은 자원을 낭비하는 Connection을 사용자가 요청할 때마다 매번 연결하지 않고, 미리 일정 개수만큼 Connection을 맺어 필요한 DAO 클래스에서는 빌려 사용하고 반환하는 방법이다. 과거에는 Connection Pool 기능을 구현한 클래스를 직접 작성하여 사용하였으나, 현재에는 Connection Pool 기능이 포함된 라이브러리를 웹 사이트에서 무료로 다운받아 사용하거나 tomcat7 컨테이너에서 제공하는 Pool 기능을 사용한다.

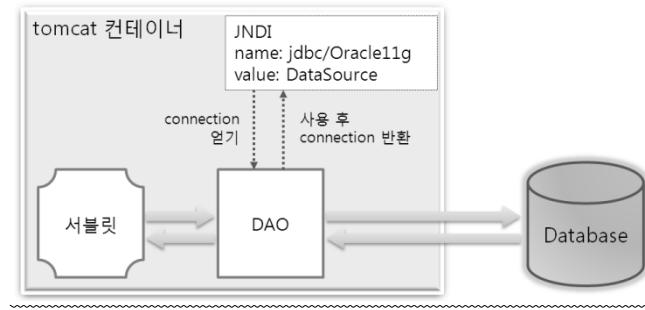
Pool 기능을 포함한 API 객체는 javax.sql.DataSource이며 JNDI 기법을 사용하여 tomcat 컨테이너에 등록되어 있다. DataSource의 getConnection() 메서드를 사용하여 Connection을 얻고 CLOSE 메서드를 사용하여 Pool에 반납한다.

17.2 DBCP

- DBCP(Database Connection Pool)

DBCP를 이용하는데 JNDI(Java Naming and Directory Interface) 기술을 이용해 접근한다.

JNDI는 명명 및 디렉토리 서비스를 사용하기 위한 API이다. 필요한 데이터(클래스, 파일등)를 name/value 형식으로 저장하고 필요한 데이터는 name값을 이용하여 value값을 얻는 형태이다. JNDI의 대표적인 경우가 DNS(Domain Name Server)이다. 웹 브라우저에서 영문 URL이 name값이고 IP가 value값이 된다.



- 파일 [context.xml]의 컨텍스트에 리소스 설정

톰캣의 JSP 프로그램에서 DBCP를 사용하려면 톰캣에서 DBCP를 위한 JNDI 데이터소스 환경 설정을 해야 한다.

컨테이너를 자원 관리자를
기술할 수 있는데 Application
혹은 Container가 속성값에
온다.

웹에서 수행 되어질 때 웹상에서 자원의
이름을 찾아면 type 속성의 값에 기술된
클래스로 내보내진다(변환된다)
jdbc/Oracle11g라는 이름을 찾으면 이름
에 해당하는 객체의 타입이 javax.sql.Data
Source으로 리턴된다는 것이다

```
<Resource name="jdbc/Oracle11g" auth="Container" type="javax.sql.DataSource"
maxActive="100" maxIdle="30" maxWait="10000"
username="scott" password="tiger" driverClassName="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />
```

데이터베이스 JDBC 드라이버 클래스 이름

데이터베이스 최대 유휴 연결 개수, 즉 클라이언트 사용이 없을 때에도 항상 유지할 연결의 최대 개수를 의미한다. -1인 경우 무제한이다. 생략 가능

데이터베이스 최대 연결 개수, 즉 클라이언트 요청에 따라 커넥션을 생성할 최대 개수이다. maxActive 이상의 연결이 동시에 발생할 경우 우선순위에 따라 연결 처리를 하므로, 일부 클라이언트 연결에서는 지연이 발생할 수 있다. 0인 경우 무제한으로 시스템 성능 및 데이터베이스 서버 설정에 따라 연결이 가능하다.

```

<!-->
<Resource name="jdbc/Oracle11g" auth="Container" type="javax.sql.DataSource"
maxActive="100" maxIdle="30" maxWait="10000" username="scott" password="tiger"
driverClassName="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />

```

17.3 DBCP 이용 프로그래밍

- DBCP를 이용한 데이터베이스 프로그래밍 방법을 알아보자.

DBCP 프로그램에서 클래스 javax.naming.InitialContext, 인터페이스 javax.sql.DataSource의 사용을 위하여 다음과 같은 import 페이지 지시자가 필요하다.

```
<%@ page import="java.sql.* , javax.sql.* , javax.naming.*" %>
```

다음으로 가장 먼저 InitialContext 객체가 필요하므로 객체를 하나 생성한다.

클래스 javax.naming.InitialContext는 JNDI 서비스를 하기 위해 객체 InitialContext를 생성하기 위한 클래스이며, 객체 InitialContext가 생성되면 이미 저장한 DBCP를 위한 리소스를 참조하게 된다.

```
InitialContext ctx = new InitialContext();
```

다음은 위 문장으로 얻은 InitialContext 객체 변수 ctx의 메소드 lookup을 이용해 DataSource 객체를 찾는 부분이다. 인터페이스 javax.sql.DataSource는 커넥션 풀을 위해 만든 인터페이스로 DBMS 커넥션 풀을 구현한 객체를 표현한다. 즉 툴캣의 JNDI 설정에서 우리가 지정한 "jdbc/Oracle11g"이름으로 커넥션 풀을 위한 객체를 찾아 DataSource 객체 변수 ds에 저장한다.

```
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/Oracle11g");
```

메소드 lookup()의 인자를 살펴보면 "java:comp/env"는 그대로 입력하고 마지막 문자열 [jdbc/Oracle11g]은 환경 설정에서 파일 [context.xml]의 태그<Resource>에서 속성 name="jdbc/Oracle11g"으로 기술한 name 속성 값이다. 인터페이스 DataSource도 패키지 javax.sql에 속하므로 import가 필요하다.

```
"java:comp/env/jdbc/Oracle11g"
```

지금까지 살펴본 문장은 다음과 같이 2개의 문장이다.

```
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/Oracle11g");
```

이제 마지막으로 DataSource 객체의 메소드 getConnection()를 호출하여 Connection 객체 con을 얻는다.

```
con = ds.getConnection();
```

이로써 JDBC 프로그래밍을 위한 6단계 중에서 DBCP를 이용한 JDBC 드라이버 로드와 데이터베이스 연결이 완료되었다. 이제 이 이후에 필요한 JDBC 프로그래밍 절차인 SQL을 위한 Statement 객체 생성, SQL 문장 실행, 질의 결과 ResultSet 처리, JDBC 객체 연결 해제는 일반적인 JDBC 프로그래밍과 동일하다.

dbconnectwithdbcp.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>Jakarta DBCP</title>
```

```

</head>
<body>
<%@ page import="java.sql.* , javax.sql.* , javax.naming.*" %>
<h2>Jakarta DBCP를 이용한 DB 연결 점검 프로그램 </h2>
<%
try {
    InitialContext ctx = new InitialContext();
    DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/Oracle11g");
    Connection con = ds.getConnection();
    /*
    Context initCtx = new InitialContext();
    Context env = (Context) initCtx.lookup("java:comp/env/");
    DataSource ds = (DataSource) env.lookup("jdbc/Oracle11g");
    Connection con = ds.getConnection();
    */
    out.println("ORACLE 서버에 성공적으로 접속했습니다");
    con.close();
} catch (Exception e) {
    out.println("ORACLE 서버에 접속하는데 문제가 있습니다. <hr>");
    out.println(e.getMessage());
    e.printStackTrace();
}
%>
</body>
</html>

```

[결과]



- DBCP를 이용한 레코드 삽입 프로그램

DBCP를 이용해 테이블 student에 레코드 하나를 삽입하는 프로그램을 작성해 보자.

레코드 삽입을 위해 SQL 문에서 변수를 효과적으로 처리하는 인터페이스 PreparedStatement를 이용하고 SQL문장을 위해 클래스 StringBuffer를 이용한다. 물론 String으로 이용해도 상관없다.

클래스 StringBuffer의 메소드 append()를 이용해 SQL문장을 계속 추가할 수 있다. SQL문장은 PreparedStatement의 사용을 위해 실제 삽입할 자료 값 부분을 [...]로 구성한다.

```
PreparedStatement pstmt = null;
StringBuffer SQL = new StringBuffer("insert into student(no, sd_num, sd_name, sd_id, sd_passwd,
s_num, sd_jumin, sd_hpone, sd_address, sd_email )");
SQL.append("values(student_seq.nextval, ?, ?, ?, ?, ?, ?, ?, ?)");
```

DBCP를 이용해 데이터베이스에 연결하는 모듈이다.

```
Context initCtx = new InitialContext();
Context env = (Context) initCtx.lookup("java:comp/env/");
DataSource ds = (DataSource) env.lookup("jdbc/Oracle11g");
Connection con = ds.getConnection();
```

Connection 객체의 prepareStatement()메소드를 호출하여 PreparedStatement 객체를 얻는다. 이때 인자로 SQL문이 저장된 StringBuffer 객체 SQL을 이용하는데, 인자의 유형이 String이므로 SQL.toString()을 호출한다. PreparedStatement의 SQL문에서 아직 완성되지 않는 [...] 부분을 지정하기 위해 setString()과 setInt()를 이용한다.

```
pstmt = con.prepareStatement(SQL.toString());
//삽입할 학생 레코드 데이터 입력
pstmt.setString(1, "90030001");
pstmt.setString(2, name);
...
pstmt.setString(9, "dbcp@dreamwiz.com");
```

PreparedStatement 객체의 메소드 executeUpdate()를 호출하여 SQL문을 실행한다. 메소드 executeUpdate()는 SQL문을 실행하여 수정된 레코드(행) 수를 반환하므로 그 반환 값을 다음과 같이 이용할 수 있다.

```
int rowCount = pstmt.executeUpdate();
if (rowCount == 1)
    out.println("<hr>학생 [" + name+ "] 레코드 하나가 성공적으로 삽입 되었습니다.<hr>");
else
    out.println("학생 레코드 삽입에 문제가 있습니다.");
```

studentDBCP.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>DBCP 이용 : 테이블 student 레코드 삽입</title>
</head>
<body>
```

```

<%@ page import="java.sql.* , javax.sql.* , javax.naming.*" %>
<h2>테이블 student에 학생 삽입 프로그램 </h2>
<hr><center>
<h2>학생 삽입</h2>
<%
    Connection con = null;
    PreparedStatement pstmt = null;
    Statement stmt = null;
    ResultSet result = null;
    StringBuffer SQL = new StringBuffer("insert into student(no, sd_num, sd_name, sd_id, sd_passwd,
    s_num, sd_jumin, sd_phone, sd_address, sd_email, sd_date)");
    SQL.append(" values (student_seq.nextval,?,?,?,?,?, ?,?, sysdate)");
    String name = "전미정";
    try {
        Context initCtx = new InitialContext();
        Context env = (Context) initCtx.lookup("java:comp/env/");
        DataSource ds = (DataSource) env.lookup("jdbc/Oracle11g");
        con = ds.getConnection();
        pstmt = con.prepareStatement(SQL.toString());
        //삽입할 학생 레코드 데이터 입력
        pstmt.setString(1, "90030001");
        pstmt.setString(2, name);
        pstmt.setString(3, "dbcp");
        pstmt.setString(4, "dbcp1234");
        pstmt.setString(5, "03");
        pstmt.setString(6, "7106192345623");
        pstmt.setString(7, "0111294567");
        pstmt.setString(8, "서울시 영등포구 당산동");
        pstmt.setString(9, "dbcp@dreamwiz.com");
        int rowCount = pstmt.executeUpdate();
        if (rowCount == 1)
            out.println("<hr>학생 [" + name+ "] 레코드 하나가 성공적으로 삽입 되었습니다.<hr>");
        else
            out.println("학생 레코드 삽입에 문제가 있습니다.");
        //다시 학생 조회
        stmt = con.createStatement();
        result = stmt.executeQuery("select sd_num, sd_name, sd_id, sd_passwd, s_num, sd_jumin,
        sd_hpone, sd_address, sd_email, sd_date from student");
    }
    %>
    <table width="100%" border="2" cellpadding="1">

```

```

create sequence student_seq
start with 1
increment by 1
nocycle;

```

```

<tbody align="center">
<tr>
    <th>학번</th>
    <th>이름</th>
    <th>아이디</th>
    <th>비밀번호</th>
    <th>학과번호</th>
    <th>주민등록번호</th>
    <th>휴대폰</th>
    <th>주소</th>
    <th>이메일</th>
    <th>등록일자</th>
</tr>
<%
    while (result.next()) {
%>
<tr>
    <td><%= result.getInt("sd_num") %></td>
    <td><%= result.getString("sd_name") %></td>
    <td><%= result.getString("sd_id") %></td>
    <td><%= result.getString("sd_passwd") %></td>
    <td><%= result.getString("s_num") %></td>
    <td><%= result.getString("sd_jumin") %></td>
    <td><%= result.getString("sd_hpone") %></td>
    <td><%= result.getString("sd_address") %></td>
    <td><%= result.getString("sd_email") %></td>
    <td><%= result.getDate("sd_date") %></td>
</tr>
<%
    }
}
catch(Exception e) {
    out.println("student 테이블에 삽입 또는 조회에 문제가 있습니다. <hr>");
    out.println(e.toString());
    e.printStackTrace();
}
finally {
    if(result != null) result.close();
    if(pstmt != null) pstmt.close();
    if(stmt != null) stmt.close();
}

```

```

if(con != null) con.close();
}

%>
</tbody>
</table>
</center>
</body>
</html>

```

18. JDBC를 위한 자바 빈즈

18.1 데이터베이스 연동 자바 빈즈

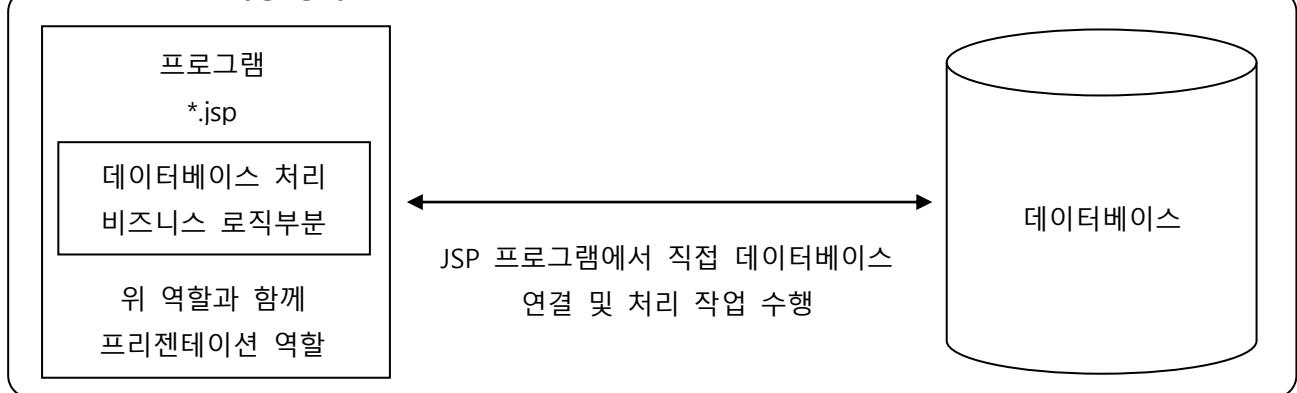
- 데이터베이스 처리 작업과 화면 표현 작업의 분리

JSP 프로그램 장점 중의 하나가 비즈니스 로직 처리와 프리젠테이션 처리를 분리하여 개발할 수 있다는 것이다. JDBC 프로그래밍에서는 데이터베이스 연결 및 처리 작업과 데이터베이스에서 질의 결과로 얻은 자료를 출력하는 부분이 모두 JSP 프로그램 내부에서 구현되었다

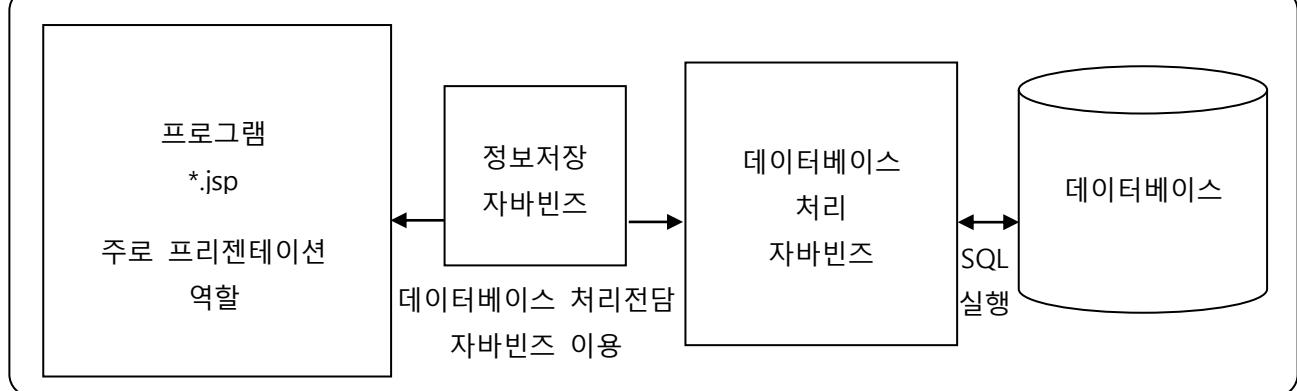
이러한 개발 방식과는 달리 자바빈즈를 이용한 JDBC 프로그래밍 방식은 비즈니스 로직 처리와 프리젠테이션 처리를 분리하여 개발할 수 있어 보다 발전된 프로그래밍 방식이다.

즉 자바빈즈를 이용한 JDBC 프로그래밍 방식은 자바 빈즈의 목적을 살려 데이터베이스 연동 자바빈즈에서 데이터베이스 연결 및 처리 작업을 하고 그 결과인 자료 출력은 JSP 프로그램에서 수행하는 프로그램 방식이다.

JDBC 프로그래밍 방식



자바빈즈를 이용한 JDBC 프로그래밍 방식



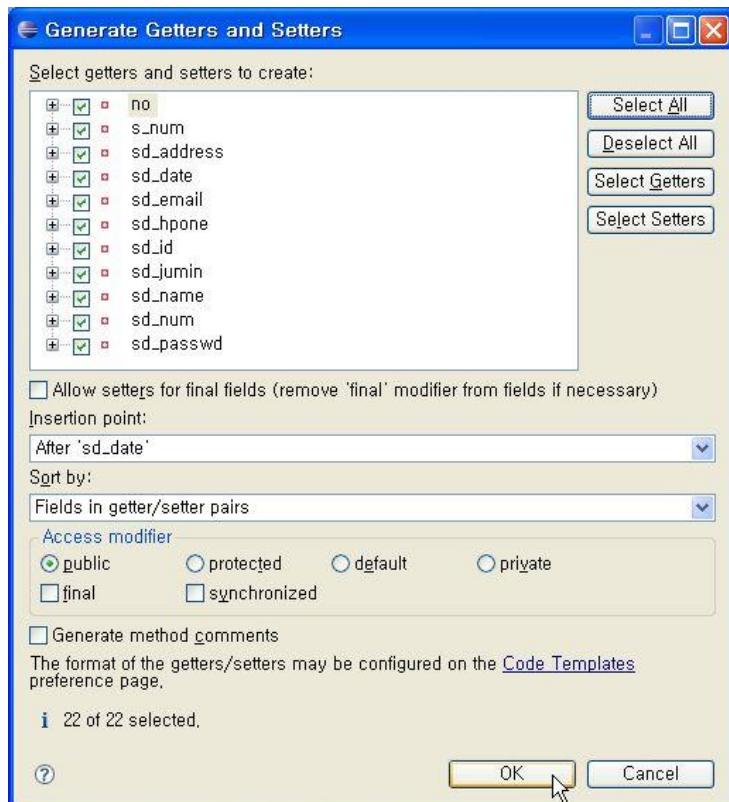
자바빈즈 JDBC 프로그램에서 사용되는 자바빈즈는 크게 2가지로 나뉘는데, 그 하나는 정보 저장용 자바빈즈로 JSP 프로그램에서 데이터베이스로 자료를 전달하고, 다시 반대로 데이터베이스로부터 자료를 전달 받아 JSP 프로그램으로 전달하고 자바빈즈가 필요하다. 또 하나는 데이터베이스 연결과 각종 SQL문을 직접 실행 처리하는 데이터베이스 처리용 자바빈즈이다.

- 학생 정보를 저장하는 자바빈즈

학생테이블 student의 모든 학생을 출력하는 프로그램 selectdb.jsp를 자바빈즈를 이용한 JDBC 프로그래밍 방식으로 개선해 보자.

테이블 student에서 1명의 학생 정보를 저장하는 자바빈즈 StudentEntity를 작성하자.

이클립스에서 현재 작업 프로젝트 폴더 하부 [Java Resources: src]하부에서 클래스 StudentEntity를 생성 한다. 클래스 StudentEntity는 패키지 [studentmgr]로 하며, 접근 제어자로 private로 테이블 student의 필드 이름과 동일하게 소속 변수를 선언한다. 이클립스에서 [메뉴] [source]/[Generate Getters and Setters ...]를 이용하여, 대화상자 [Generate Getters and Setters]에서 9개의 모든 필드에 대한 getter와 setter를 자동으로 생성한다.



```
package studentmgr;
public class StudentEntity {
    private int no;
    private String sd_num;
    private String sd_name;
    private String sd_id;
    private String sd_passwd;
    private String s_num;
```

```
private String sd_jumin;
private String sd_phone;
private String sd_address;
private String sd_email;
private String sd_date;
public int getNo() {
    return no;
}
public void setNo(int no) {
    this.no = no;
}
public String getSd_num() {
    return sd_num;
}
public void setSd_num(String sd_num) {
    this.sd_num = sd_num;
}
public String getSd_name() {
    return sd_name;
}
public void setSd_name(String sd_name) {
    this.sd_name = sd_name;
}
public String getSd_id() {
    return sd_id;
}
public void setSd_id(String sd_id) {
    this.sd_id = sd_id;
}
public String getSd_passwd() {
    return sd_passwd;
}
public void setSd_passwd(String sd_passwd) {
    this.sd_passwd = sd_passwd;
}
public String getS_num() {
    return s_num;
}
public void setS_num(String s_num) {
    this.s_num = s_num;
```

```

    }
    public String getSd_jumin() {
        return sd_jumin;
    }
    public void setSd_jumin(String sd_jumin) {
        this.sd_jumin = sd_jumin;
    }
    public String getSd_hpone() {
        return sd_hpone;
    }
    public void setSd_hpone(String sd_hpone) {
        this.sd_hpone = sd_hpone;
    }
    public String getSd_address() {
        return sd_address;
    }
    public void setSd_address(String sd_address) {
        this.sd_address = sd_address;
    }
    public String getSd_email() {
        return sd_email;
    }
    public void setSd_email(String sd_email) {
        this.sd_email = sd_email;
    }
    public String getSd_date() {
        return sd_date;
    }
    public void setSd_date(String sd_date) {
        this.sd_date = sd_date;
    }
}

```

- 데이터베이스 연동을 위한 자바빈즈

데이터베이스 연동을 위한 자바빈즈는 JDBC 프로그래밍 절차를 모두 구현하는 프로그램이다.

학생 테이블 student의 모든 레코드를 선택해오는 질의 수행을 위한 데이터베이스 연동 자바빈즈 프로그램 StudentDatabase.java를 작성해 보자.

프로그램 StudentDatabase에서 가장 먼저 데이터베이스 연결을 위한 드라이버이름, 연결URL, 사용자, 암호를 저장하기 위한 상수를 선언한다.

```
// 데이터베이스 연결 관련 상수 선언
```

```

private static final String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver";
private static final String JDBC_URL = "jdbc:oracle:thin:@127.0.0.1:1521:ordl";
private static final String USER = "scott";
private static final String PASSWD = "tiger";

```

다음으로 데이터베이스 연결 관련 주요 객체 Connection, Statement를 저장할 변수 con과 stmt를 멤버 변수로 선언한다.

```

// 데이터베이스 연결 관련 변수 선언
private Connection con = null;
private Statement stmt = null;

```

데이터베이스 연동 자바빈즈 StudentDatabase는 기본 생성자에서 JDBC 드라이버를 로드하며, 3개의 메소드에서 데이터베이스 작업을 수행한다.

반환 유형	메소드 이름	기능
생성자	StudentDatabase()	데이터베이스 연결을 위한 데이터베이스 드라이버를 로드
void	connect()	DriverManager를 이용하여 데이터베이스에 연결. Connection 객체를 멤버변수 con에 저장.
void	disconnect()	데이터베이스 관련 객체의 연결을 해제
ArrayList<StudentDatabase>	getStudentList()	테이블 student의 각 행을 StudentEntity에 저장하여 모든 학생의 StudentEntity를 추가한 ArrayList를 반환

```

package studentmgr;
import java.sql.*;
import java.util.ArrayList;
// 테이블 student를 위한 데이터베이스 연동 자바빈즈 프로그램
public class StudentDatabase {
    // 데이터베이스 연결 관련 상수 선언
    private static final String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver";
    private static final String JDBC_URL = "jdbc:oracle:thin:@127.0.0.1:1521:ordl";
    private static final String USER = "scott";
    private static final String PASSWD = "tiger";
    // 데이터베이스 연결 관련 변수 선언
    private Connection con = null;
    private Statement stmt = null;
    // JDBC 드라이버를 로드하는 생성자
    public StudentDatabase() {
        // JDBC 드라이버 로드

```

```

        try {
            Class.forName(JDBC_DRIVER);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // 데이터베이스 연결 메소드
    public void connect() {
        try {
            // 데이터베이스에 연결, Connection 객체 저장
            con = DriverManager.getConnection(JDBC_URL, USER, PASSWD);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // 데이터베이스 연결 해제 메소드
    public void disconnect() {
        if(stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(con != null) {
            try {
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    // 게시판의 모든 레코드를 반환하는 메소드
    public ArrayList<StudentEntity> getStudentList() {
        connect();
        // 질의 결과를 저장할 ArrayList를 선언
        // ArrayList 내부에는 학생정보를 저장한 StudentEntity가 삽입
        ArrayList<StudentEntity> list = new ArrayList<StudentEntity>();
        String SQL = "select sd_num, sd_name, sd_id, sd_passwd, s_num, sd_jumin, sd_hpone,
sd_address, sd_email, sd_date from student";
    }

```

```

try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(SQL);
    //ResultSet의 결과에서 모든 행을 각각의 StudentEntity 객체에 저장
    while (rs.next()) {
        //한 행의 학생정보를 저장할 학생을 위한 빈즈 객체 생성
        StudentEntity stu = new StudentEntity();
        //한 행의 학생정보를 자바 빈즈 객체에 저장
        stu.setSd_num ( rs.getString("sd_num") );
        stu.setSd_name ( rs.getString("sd_name") );
        stu.setSd_id ( rs.getString("sd_id") );
        stu.setSd_passwd ( rs.getString("sd_passwd") );
        stu.setS_num ( rs.getString("s_num") );
        stu.setSd_jumin ( rs.getString("sd_jumin") );
        stu.setSd_hpone ( rs.getString("sd_phone") );
        stu.setSd_address ( rs.getString("sd_address") );
        stu.setSd_email ( rs.getString("sd_email") );
        stu.setSd_date ( rs.getString("sd_date") );
        //ArrayList에 학생정보 StudentEntity 객체를 추가
        list.add(stu);
    }
    rs.close();
} catch (SQLException e) {
    e.printStackTrace();
}
finally {
    disconnect();
}
//완성된 ArrayList 객체를 반환
return list;
}
}

```

- 학생 조회 프로그램

위에서 구현한 자바빈즈 StudentEntity와 StudentDatabase를 이용하여 테이블 student의 학생을 조회하여 출력하는 JSP 프로그램을 구현하자.

useBean 태그를 이용하여 데이터베이스 연동 자바빈즈 StudentDatabase를 변수 stdtdb로 사용한다.

```
<jsp:useBean id="stdtdb" class="studentmgr.StudentDatabase" scope="page" />
```

자바빈즈 StudentDatabase의 객체 변수 stdtdb의 메소드 getStudentList()를 호출하여 출력하려는 모든

학생의 정보를 ArrayList 객체 변수 list에 저장하고, 전체 학생 수를 출력하기 위해 변수 counter에 list.size()의 반환값을 저장한다.

```
ArrayList<StudentEntity> list = stdtdb.getStudentList();
int counter = list.size();
```

for문을 이용하여 객체 변수 list를 순회하면서 배열 형태로 저장되어 있는 StatementEntity 객체를 하나씩 조회하여 브라우저에 출력한다.

```
<%
for( StudentEntity stdt : list ) {
%>
<tr>
<td><%= stdt.getSd_num () %></td>
<td><%= stdt.getSd_name() %></td>
...
</tr>
<% } %>
```

selectStudentbean.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>데이터베이스 자바 빈즈 예제 : 테이블 student 조회</title>
</head>
<body>
<%@ page import="java.util.ArrayList, studentmgr.StudentEntity" %>
<h2> 자바 빈즈 StudentDatabase를 이용한 테이블 student 조회 프로그램 </h2>
<hr><center>
<h2>학생정보 조회</h2>
<jsp:useBean id="stdtdb" class="studentmgr.StudentDatabase" scope="page" />
<%
ArrayList<StudentEntity> list = stdtdb.getStudentList();
int counter = list.size();
if (counter > 0) {
%>
<table width="100%" border="2" cellpadding="1">
<tbody align="center">
<tr>
<th>학번</th>
<th>이름</th>
<th>아이디</th>
```

```

<th>비밀번호</th>
<th>학과번호</th>
<th>주민등록번호</th>
<th>휴대폰</th>
<th>주소</th>
<th>이메일</th>
<th>등록일자</th>
</tr>
<%
for( StudentEntity stdt : list ) {
%>
<tr>
<td><%= stdt.getSd_num() %></td>
<td><%= stdt.getSd_name() %></td>
<td><%= stdt.getSd_id() %></td>
<td><%= stdt.getSd_passwd() %></td>
<td><%= stdt.getS_num() %></td>
<td><%= stdt.getSd_jumin() %></td>
<td><%= stdt.getSd_hpone() %></td>
<td><%= stdt.getSd_address() %></td>
<td><%= stdt.getSd_email() %></td>
<td><%= stdt.getSd_date() %></td>
</tr>
<%
        }
    }
%>
</tbody>
</table>
</center>
<p><hr>조회된 학생 수가 <%=counter%>명 입니다.
</body>
</html>

```

[결과]

The screenshot shows a Microsoft Internet Explorer window with the title "데이터베이스 자바 빈즈 예제 : 태이블 student 조회 - Microsoft Internet Explorer". The address bar shows "http://localhost:8080/exam01/selectstudentbean.jsp". The main content is a table titled "학생정보 조회" (Student Information Inquiry) with the following data:

학번	이름	아이디	비밀번호	학과 번호	주민등록번호	휴대폰	주소	이메일	등록일자
06010001	김정수	javajsp	java1234	01	8710101653872	01012345678	서울시 서대문구 창전동	java12@naver.com	2009-07-07 22:51:28.0
95010002	김수현	jdbcmania	mania12	01	7505052129875	0113452468	서울시 서초구 양재동	jdbcmania@naver.com	2009-07-07 22:51:28.0
98040001	공지영	gonji	mania12	04	7812242250875	01612657455	부산광역시 해운대구 반송동	gonji@nate.com	2009-07-07 22:51:28.0
02050001	조수영	water	java1234	05	8303161204369	0107681499	대전광역시 중구 은행동	water@korea.com	2009-07-07 22:51:28.0
94040002	최경란	novel	novel2468	04	7410091893562	0119872455	경기도 수원시 장안구 이복동	novel@naver.com	2009-07-07 22:51:28.0
08020001	안익태	korea	korea99	02	8908151734120	0168452345	서울시 마포구 대흥동	korea@nate.com	2009-07-07 22:51:29.0
90030001	전미정	dbcp	dbcp1234	03	7106192345623	0111294567	서울시 영등포구 당산동	dbcp@dreamwiz.com	2009-07-08 11:40:11.0

조회된 학생 수가 7명입니다.

18.2 DBCP를 이용한 데이터베이스 연동 자바빈즈

데이터베이스 연동 StudnetDatabase를 수정하여, DBCP를 이용한 데이터베이스 연동 자바빈즈 프로그램을 작성해보자.

- 설정 파일 수정

DBCP를 이용하려면 설정파일 [context.xml]을 수정해야 한다. 파일 [context.xml]에는 이용할 커넥션 풀의 이용 정보 리소스를 추가한다..

```
<Resource name="jdbc/Oracle11g" auth="Container" type="javax.sql.DataSource"
    maxActive="100" maxIdle="30" maxWait="10000"
    password="tiger" driverClassName="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" username="scott" />
```

- 데이터베이스 연동을 위한 자바빈즈

데이터베이스 연동 자바빈즈인 StudentDatabase는 데이터베이스 연결을 우리가 직접 구현한 자바빈즈다. 이제 DBCP 커넥션 풀을 이용한 데이터베이스 연동 자바빈즈인 StudentDatabaseCP.java를 작성해 보자.

DBCP 커넥션 풀을 이용한 데이터베이스 연동 자바빈즈 프로그램인 StudentDatabaseCP의 패키지는 studentmgr로 하고, DBCP 커넥션 풀을 위한 인터페이스 DataSource와 클래스 InitialContext의 import문을 기술한다.

그리고 보다 효율적인 질의를 위해 인터페이스 Statement 대신에 인터페이스 PreparedStatement를 이용하며, 커넥션 풀을 위한 인터페이스 DataSource를 필드로 선언한다.

생성자에서는 등력한 DBCP를 찾아 DataSource 객체 ds에 저장하는 작업을 수행한다.

```
public StudentDatabaseCP() {
```

```

...
InitialContext ctx = new InitialContext();
ds = (DataSource) ctx.lookup("java:comp/env/jdbc/Oracle11g");
}

```

메소드 connect()에서 DataSource객체 ds를 통해 데이터베이스 연결, Connection 객체 con에 저장한다.

```

void connect() {
    ...
    con = ds.getConnection();
}

```

메소드 getStudentList()는 자바빈즈 StudentDatabase에서 수행한 작업과 동일하다. 다만 여기서는 보다 효율적인 질의 수행을 Statement대신에 인터페이스 PreparedStatement를 이용한다.

즉 con.createStatement()대신에 con.prepareStatement(SQL)를 호출하여 호출 인자로 SQL을 직접 사용한다. SQL문장에 매개변수 [...]부분이 없으므로 바로 pstmt.executeQuery()를 실행하여 결과를 얻는다.

```

package studentmgr;
import java.sql.*;
import javax.sql.DataSource;
import java.util.ArrayList;
import javax.naming.InitialContext;
//DBCP를 이용한 테이블 student 처리 데이터베이스 연동 자바빈즈 프로그램
public class StudentDatabaseCP {
    // 데이터베이스 연결관련 변수 선언
    private Connection con = null;
    private PreparedStatement pstmt = null;
    private DataSource ds = null;
    // 등록한 DBCP 데이터소스 찾아 저장하는 생성자
    public StudentDatabaseCP() {
        try {
            InitialContext ctx = new InitialContext();
            ds = (DataSource) ctx.lookup("java:comp/env/jdbc/Oracle11g");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 데이터소스를 통해 데이터베이스에 연결, Connection 객체에 저장하는 메소드
    void connect() {
        try {
            con = ds.getConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

        }
    }

// 데이터베이스 연결 해제 메소드
void disconnect() {
    if(pstmt != null) {
        try {
            pstmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if(con != null) {
        try {
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

// 게시판의 모든 레코드를 반환하는 메소드
public ArrayList<StudentEntity> getStudentList() {
    connect();
    ArrayList<StudentEntity> list = new ArrayList<StudentEntity>();
    String SQL = "select sd_num, sd_name, sd_id, sd_passwd, s_num, sd_jumin, sd_hpone, sd_address, sd_email, sd_date from student";
    try {
        pstmt = con.prepareStatement(SQL);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            //한 행의 학생정보를 저장할 학생을 위한 빈즈 객체 생성
            StudentEntity stu = new StudentEntity();
            //한 행의 학생정보를 자바 빈즈 객체에 저장
            stu.setSd_num ( rs.getString("sd_num") );
            stu.setSd_name ( rs.getString("sd_name") );
            stu.setSd_id ( rs.getString("sd_id") );
            stu.setSd_passwd ( rs.getString("sd_passwd") );
            stu.setS_num ( rs.getString("s_num") );
            stu.setSd_jumin ( rs.getString("sd_jumin") );
            stu.setSd_hpone ( rs.getString("sd_hpone") );
            stu.setSd_address ( rs.getString("sd_address") );
        }
    }
}

```

```

        stu.setSd_email ( rs.getString("sd_email") );
        stu.setSd_date ( rs.getString("sd_date") );
        //리스트에 추가
        list.add(stu);
    }
    rs.close();
} catch (SQLException e) {
    e.printStackTrace();
}
finally {
    disconnect();
}
return list;
}
}

```

- 학생 조회 프로그램

위에서 작성한 StudentDatabaseCP를 이용하여 학생을 조회 • 출력하는 JSP 프로그램 selectstudentCP bean.jsp를 작성하자. 가장 먼저 useBean 태그를 이용하여 클래스 studentmgr.StudentDatabaseCP를 객체변수 stdtbd로 지정한 후, 학생정보를 모두 조회하는 메소드 getStudentDatabaseCP를 객체변수 stdtbd로 지정한 후, 학생정보를 모두 조회하는 메소드 getStudentList()를 호출한다.

selectStudentbeanCP.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>데이터베이스 자바 빈즈 예제 : CP 이용 selectStudentbeanCP.jsp</title>
</head>
<body>
<%@ page import="java.util.ArrayList, studentmgr.StudentEntity" %>
<h2> 자바 빈즈 StudentDatabaseCP를 이용한 테이블 student 조회 프로그램 </h2>
<hr><center>
<h2>학생정보 조회</h2>
    <jsp:useBean id="stdtbd" class="studentmgr.StudentDatabaseCP" scope="page" />
<%
    ArrayList<StudentEntity> list = stdtbd.getStudentList();
    int counter = list.size();
    if (counter > 0) {
%>
    <table width="100%" border="2" cellpadding="1">

```

```

<tbody align="center">
<tr>
    <th>학번</th>
    <th>이름</th>
    <th>아이디</th>
    <th>비밀번호</th>
    <th>학과번호</th>
    <th>주민등록번호</th>
    <th>휴대폰</th>
    <th>주소</th>
    <th>이메일</th>
    <th>등록일자</th>
</tr>
<%
    for( StudentEntity stdt : list ) {
%>
<tr>
    <td><%= stdt.getSd_num() %></td>
    <td><%= stdt.getSd_name() %></td>
    <td><%= stdt.getSd_id() %></td>
    <td><%= stdt.getSd_passwd() %></td>
    <td><%= stdt.getS_num() %></td>
    <td><%= stdt.getSd_jumin() %></td>
    <td><%= stdt.getSd_hpone() %></td>
    <td><%= stdt.getSd_address() %></td>
    <td><%= stdt.getSd_email() %></td>
    <td><%= stdt.getSd_date() %></td>
</tr>
<%
    }
%>
</tbody>
</table>
<%
    }
%>
</center>
<p><hr> 조회된 학생 수가 <%=counter%> 명 입니다.
</body>
</html>

```

[결과]

자바 빈즈 StudentDatabaseCP를 이용한 테이블 student 조회 프로그램

학생정보 조회

학번	이름	아이디	비밀번호	학과번호	주민등록번호	휴대폰	주소	이메일	등록일자
06010001	김정수	javajsp	java1234	01	8710101653872	01012345678	서울시 서대문구 창전동	java12@naver.com	2009-07-07 22:51:28.0
95010002	김수현	jdbcmania	mania12	01	7505052129875	0113452468	서울시 서초구 양재동	jdbcmania@naver.com	2009-07-07 22:51:28.0
98040001	공지영	gonji	mania12	04	7812242250875	01612657455	부산광역시 해운대구 반송동	gonji@nate.com	2009-07-07 22:51:28.0
02050001	조수영	water	java1234	05	8303161204369	0107681499	대전광역시 중구 은행동	water@korea.com	2009-07-07 22:51:28.0
94040002	최경란	novel	novel2468	04	7410091893562	0119872455	경기도 수원시 장안구 이북동	novel@naver.com	2009-07-07 22:51:28.0
08020001	안익태	korea	korea99	02	8908151734120	0168452345	서울시 마포구 대흥동	korea@nate.com	2009-07-07 22:51:29.0
90030001	전미정	dbcp	dbcp1234	03	7106192345623	0111294567	서울시 영등포구 당산동	dbcp@dreamwiz.com	2009-07-08 11:40:11.0

조회된 학생 수가 7명입니다.

18.3 기본 게시판을 위한 데이터베이스와 자바빈즈

1) 기본 게시판을 위한 테이블 생성 및 프로그램 구성

자바빈즈를 이용한 JDBC 프로그래밍 방법에 익숙해지도록 간단한 게시판을 직접 만들어보자.

게시판은 BBS(Bulletin Board System)라고 부르며 대부분의 사람이 사용해 본 경험이 있으므로 우리가 직접 구현해 보기에 적합한 예제이다. 게시판도 상업용으로 사용할 수 있는 전문 게시판을 만들려면 그 구조가 단순하지 않으나 여기에서는 가능한 간단한 구조의 게시판을 만들어 보도록 하자.

- 테이블구조

게시판을 위한 테이블 구조를 다음과 같이 정의한다.

필드번호	이름	내용	유형	크기	주키	NULL
1	no	일련번호	number		pk	NO
2	name	이름	varchar2	12		NO
3	passwd	암호	varchar2	12		NO
4	title	제목	varchar2	50		NO
5	email	이메일	varchar2	40		NO
6	regdate	등록일자	date		기본값설정	YES
7	content	내용	varcar2	4000		NO

sql

```
create table board(
    no number not null primary key,
    name varchar2(12) not null,
```

```

passwd varchar2(12) not null,
title varchar2(50) not null,
email varchar2(40) not null,
regdate date default sysdate,
content varchar2(4000) not null
);
create sequence board_seq
start with 1
increment by 1
nocycle;

```

- 프로그램 구성과 실행

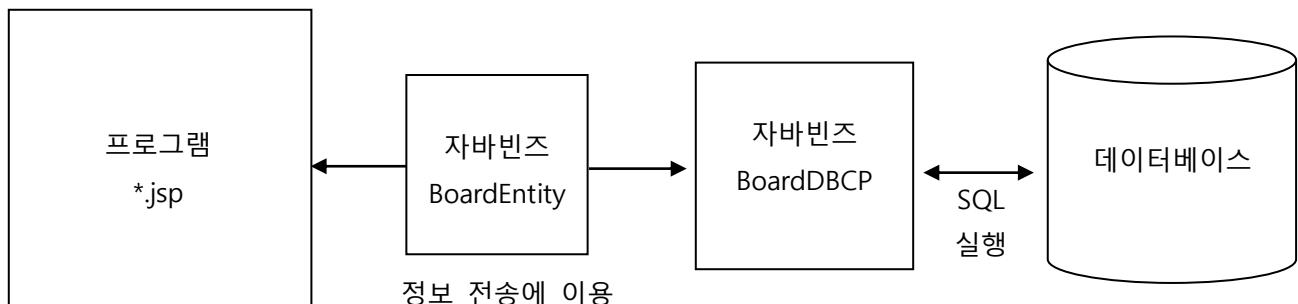
기본 게시판을 위한 프로그램은, 자바빈즈 프로그램 2개와 JSP 프로그램 3개, 자바스크립트 1개의 파일로 구성된다.

종류	파일	기능
자바빈즈	BoardEntity	JSP 프로그램과 자바빈즈 사이에서 자료 전송으로 이용되는 자바빈즈
	BoardDBCP	DBCP를 이용하여 데이터베이스 연결 및 작업을 수행하는 자바빈즈
JSP 프로그램	listboard.jsp	모든 게시 목록을 표시하는 프로그램
	editboard.jsp	게시물의 등록, 수정, 삭제를 위해 입력 폼 처리를 수행하는 프로그램
	processboard.jsp	게시물의 등록, 수정, 삭제를 수행하는 프로그램
자바스크립트	boardform.js	프로그램 editboard.jsp 에 구현된 폼에서 입력 값 검사를 위해 사용하는 자바스크립트 함수 구현.

2) 게시판 구현을 위한 자바빈즈

- 자바빈즈 프로그램 구성

자바빈즈 BoardEntity은 게시 테이블 board의 한 행의 정보를 저장하는 자바빈즈로 JSP 프로그램과 BoardDBCP 자바빈즈 사이에 정보 전송 매개변수 역할을 수행한다. 자바빈즈 BoardDBCP는 DBCP를 이용하여 실제 데이터베이스에 연결해 등록, 수정, 삭제의 SQL 수행 작업을 실행한다.



- 게시물 정보 저장 자바빈즈

게시판 테이블 board의 한 행을 저장할 수 있는 자바빈즈 BoardEntity를 생성한다. 자바빈즈 BoardEntity에서 패키지 board로 하며, 테이블 board의 필드 이름과 동일한 이름으로 7개의 필드를 만든다.

```
package board;
import java.util.Date;
public class BoardEntity {
    private int no;
    private String name;
    private String passwd;
    private String title;
    private String email;
    private Date regdate;
    private String content;
    //자동으로 생성된, 모든 필드에 대한 getter와 setter
    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPasswd() {
        return passwd;
    }
    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getEmail() {
        return email;
    }
}
```

```

    }
    public void setEmail(String email) {
        this.email = email;
    }
    public Date getRegdate() {
        return regdate;
    }
    public void setRegdate(Date regdate) {
        this.regdate = regdate;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
}

```

- 데이터베이스 연동을 위한 자바 빈즈

게시판 테이블 board의 데이터베이스 연동을 위한 자바빈즈 프로그램을 작성해 보자.

자바빈즈 BoardDBCP.java는 다음과 같이 1개의 생성자와 8개의 메소드로 구성된다.

반환유형	메소드이름	기능
생성자	BoardDBCP()	데이터소스를 이용하여 커넥션 풀을 생성
void	connect()	데이터소스의 커넥션 풀에서 Connection 객체를 얻어 멤버변수 con에 저장
void	disconnect()	데이터베이스 관련 객체의 연결을 해제
Vector<BoardEntity>	getBoardList()	테이블 board의 모든 레코드를 Vector에 저장하여 반환
BoardEntity	getBoard(int no)	인자 no가 주키인 레코드를 검색하여 반환
boolean	insertDB(BoardEntity board)	인자 board로 테이블 board에 삽입
boolean	updateDB(BoardEntity board)	인자 board로 테이블 board에 수정
boolean	deleteDB(int no)	인자 no가 주키인 레코드를 검색하여 삭제
boolean	isPasswd(int no, String passwd)	인자 no와 passwd가 테이블 board에서 일치하는지 검사

반환유형	이용 SQL 문장	기능
getBoardList()	select * from board	모든 행 검색
getBoard(int no)	select * from board where no=?	지정한 no행을 검색
insertDB(BoardEntity board)	insert into board(no, name, passwd,	지정한 행으로 삽입

	title, email, content) values(board_seq.nextval, ?, ?, ?, ?, ?)	
updateDB(BoardEntity board)	update board set title=?, email=?, content=? where no=?	지정한 행을 수정
deleteDB(int no)	delete from board where no=?	지정한 no 행을 삭제
isPasswd(int no, String passwd)	select passwd from board where no=?	지정한 no행의 passwd를 검색

자바빈즈 BoardDBCP.java는 Connection, PreparedStatement, DataSource의 객체를 멤버변수로 갖는다.

주키 no를 이용하여 board 테이블의 한행을 BoardEntity에 저장하여 반환하는 메소드 getBoard(int no)는 다음과 같이 select 질의 문장에서 인자로 들어 온 no를 매개변수 [?]에 지정하여 질의한다.

```
public BoardEntity getBoard(int no) {
    connect();
    String SQL = "select no, name, passwd, title, email, regdate, content from board where no = ?";
    BoardEntity brd = new BoardEntity();
    try {
        pstmt = con.prepareStatement(SQL);
        pstmt.setInt(1, no);
        ResultSet rs = pstmt.executeQuery();
        ...
    } catch (SQLException e) {
        e.printStackTrace();
    }
    finally {
        disconnect();
    }
    return brd;
}
```

질의 결과인 ResultSet 객체 rs에서 각 필드를 받아오기 위해 필드 유형에 맞도록 각각 rs.getInt("no"), rs.getString("name")를 이용하도록 한다. 특히 테이블 board의 필드 regdate는 유형이 date이므로 메소드 getTimestamp()를 이용해야 등록 날짜와 시간까지 모두 반환 받을 수 있다.

```
brd.setNo ( rs.getInt("no") );
brd.setName ( rs.getString("name") );
...
brd.setRegdate ( rs.getTimestamp("regdate") );
brd.setContent ( rs.getString("content") );
```

메소드 insertDB(BoardEntity board)에서 PreparedStatement에 이용할 SQL문을 다음과 같이 정의한다.

```
String sql = "insert into board(no,name,passwd,title,email,content) values(board_seq.nextval, ?, ?, ?, ?, ?)"
```

메소드 updateDB(BoardEntity board)에서 PreparedStatement에 이용할 SQL문은 매개변수 [?] 부분이 5개이며, 각각의 매개변수를 다음과 같이 지정하여 질의한다.

```
String sql ="update board set name=?, title=?, email=?, content=? where no=?";  
...  
stmt = con.prepareStatement(sql);  
stmt.setString(1, board.getName());  
stmt.setString(2, board.getTitle());  
stmt.setString(3, board.getEmail());  
stmt.setString(4, board.getContent());  
stmt.setInt(5, board.getNo());  
int rowUdt = stmt.executeUpdate();
```

메소드 deleteDB(int no)에서는 다음 SQL문과 같이 where 조건절 no의 값을 인자로 하여 들어온 no로 지정해 질의한다.

```
String sql ="delete from board where no=?";  
...  
stmt = con.prepareStatement(sql);  
stmt.setInt(1, no);  
stmt.executeUpdate();
```

사용자가 게시 항목을 수정 또는 삭제하려면 이미 저장된 암호와 사용자가 입력한 암호가 일치해야 한다. 이때 사용하는 메소드 isPasswd(int no, String passwd)로 현재 게시물의 no와 사용자가 입력한 passwd를 인자로 호출한다. 메소드 isPasswd(int no, String passwd) 구현에서 우선 다음 SQL문으로 호출 시 들어 온 인자 no를 이용하여 테이블 board에 이미 저장된 암호를 얻어와 변수 orgPasswd에 저장한다.

```
String sql ="select passwd from board where no=?";  
...  
stmt = con.prepareStatement(sql);  
stmt.setInt(1, no);  
ResultSet rs = stmt.executeQuery();  
rs.next();  
String orgPasswd = rs.getString(1);
```

메소드 isPasswd(int no, String passwd)를 호출할 때 인자로 준 passwd와 실제 암호인 orgPasswd를 비교하여 같으면 true를 반환하도록 변수 success에 저장하여 반환한다.

```
if ( passwd.equals(orgPasswd) ) success = true;  
...  
return success;
```

BoardDBCP.java

```
package board;
import java.sql.*;
import java.util.*;
import javax.sql.*;
import javax.naming.*;
//DBCP를 이용한 테이블 board 처리 데이터베이스 연동 자바빈즈 프로그램
public class BoardDBCP {
    // 데이터베이스 연결관련 변수 선언
    private Connection con = null;
    private PreparedStatement pstmt = null;
    private DataSource ds = null;
    // JDBC 드라이버 로드 메소드
    public BoardDBCP() {
        try {
            InitialContext ctx = new InitialContext();
            ds = (DataSource) ctx.lookup("java:comp/env/jdbc/Oracle11g");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 데이터베이스 연결 메소드
    public void connect() {
        try {
            con = ds.getConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 데이터베이스 연결 해제 메소드
    public void disconnect() {
        if(pstmt != null) {
            try {
                pstmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(con != null) {
            try {
```

```

        con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// 게시판의 모든 레코드를 반환 메서드
public Vector<BoardEntity> getBoardList() {
    connect();
    Vector<BoardEntity> list = new Vector<BoardEntity>();
    String SQL = "select no, name, passwd, title, email, regdate, content from board";
    try {
        pstmt = con.prepareStatement(SQL);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            BoardEntity brd = new BoardEntity();
            brd.setNo ( rs.getInt("no") );
            brd.setName ( rs.getString("name") );
            brd.setPasswd ( rs.getString("passwd") );
            brd.setTitle ( rs.getString("title") );
            brd.setEmail ( rs.getString("email") );
            brd.setRegdate ( rs.getTimestamp("regdate") );
            brd.setContent ( rs.getString("content") );
            //리스트에 추가
            list.add(brd);
        }
        rs.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    finally {
        disconnect();
    }
    return list;
}

// 주 키 no의 레코드를 반환하는 메서드
public BoardEntity getBoard(int no) {
    connect();
    String SQL = "select no, name, passwd, title, email, regdate, content from board
where no = ?";

```

```

BoardEntity brd = new BoardEntity();
try {
    pstmt = con.prepareStatement(SQL);
    pstmt.setInt(1, no);
    ResultSet rs = pstmt.executeQuery();
    rs.next();
    brd.setNo ( rs.getInt("no") );
    brd.setName ( rs.getString("name") );
    brd.setPasswd ( rs.getString("passwd") );
    brd.setTitle ( rs.getString("title") );
    brd.setEmail ( rs.getString("email") );
    brd.setRegdate ( rs.getTimestamp("regdate") );
    brd.setContent ( rs.getString("content") );
    rs.close();
} catch (SQLException e) {
    e.printStackTrace();
}
finally {
    disconnect();
}
return brd;
}

// 게시물 등록 메서드
public boolean insertDB(BoardEntity board) {
    boolean success = false;
    connect();
    String sql ="insert into board(no, name, passwd, title, email, content)
                 values(board_seq.nextval,?, ?, ?, ?, ?)";
    try {
        pstmt = con.prepareStatement(sql);
        pstmt.setString(1, board.getName());
        pstmt.setString(2, board.getPasswd());
        pstmt.setString(3, board.getTitle());
        pstmt.setString(4, board.getEmail());
        pstmt.setString(5, board.getContent());
        pstmt.executeUpdate();
        success = true;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return success;
}

```

```

        }
    finally {
        disconnect();
    }
    return success;
}

// 데이터 수정을 위한 메서드
public boolean updateDB(BoardEntity board) {
    boolean success = false;
    connect();
    String sql ="update board set name=?, title=?, email=?, content=? where no=?";
    try {
        pstmt = con.prepareStatement(sql);
        //인자로 받은 board객체를 이용해 사용자가 수정한 값을 가져와 SQL문 완성
        pstmt.setString(1, board.getName());
        pstmt.setString(2, board.getTitle());
        pstmt.setString(3, board.getEmail());
        pstmt.setString(4, board.getContent());
        pstmt.setInt(5, board.getNo());
        int rowUdt = pstmt.executeUpdate();
        //System.out.println(rowUdt);
        if (rowUdt == 1) success = true;
    } catch (SQLException e) {
        e.printStackTrace();
        return success;
    }
    finally {
        disconnect();
    }
    return success;
}

// 게시물 삭제를 위한 메서드
public boolean deleteDB(int no) {
    boolean success = false;
    connect();
    String sql ="delete from board where no=?";
    try {
        pstmt = con.prepareStatement(sql);
        // 인자로 받은 주 키인 no 값을 이용해 삭제
        pstmt.setInt(1, no);

```

```

        pstmt.executeUpdate();
        success = true;
    } catch (SQLException e) {
        e.printStackTrace();
        return success;
    }
    finally {
        disconnect();
    }
    return success;
}

// 데이터베이스에서 인자인 no와 passwd가 일치하는지 검사하는 메서드
public boolean isPasswd(int no, String passwd) {
    boolean success = false;
    connect();
    String sql ="select passwd from board where no=?";
    try {
        pstmt = con.prepareStatement(sql);
        pstmt.setInt(1, no);
        ResultSet rs = pstmt.executeQuery();
        rs.next();
        String orgPasswd = rs.getString(1);
        if ( passwd.equals(orgPasswd) ) success = true;
        System.out.println(success);
        rs.close();
    } catch (SQLException e) {
        e.printStackTrace();
        return success;
    }
    finally {
        disconnect();
    }
    return success;
}
}

```

- listboard.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">

```

```

<title>게시판 기본 예제 : 게시 목록 표시</title>
<style type="text/css">
    body, table {font-size:9pt;font-family:굴림;}
    th.cl{color: #0000CC;}
</style>
</head>
<body>
<h2 align="center">게시판 목록 표시 프로그램 </h2>
<hr>
<center>
    <%@ page import="java.util.Vector, board.BoardEntity, java.text.SimpleDateFormat" %>
    <jsp:useBean id="brddb" class="board.BoardDBCP" scope="page" />
    <%
        //게시 목록을 위한 배열리스트를 자바빈즈를 이용하여 확보
        Vector<BoardEntity> list = brddb.getBoardList();
        int counter = list.size();
        int row = 0;
        if (counter > 0) {
    %>
    <table width="800" border="0" cellpadding="1" cellspacing="3">
    <tr>
        <th class="cl">번호</th>
        <th class="cl">제목</th>
        <th class="cl">작성자</th>
        <th class="cl">작성일</th>
        <th class="cl">전자메일</th>
    </tr>
    <%
        //게시 등록일을 2010-3-15 10:33:21 형태로 출력하기 위한 클래스
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        for( BoardEntity brd : list ) {
            //홀짝으로 다르게 색상 지정
            String color = "papayawhip";
            if ( ++row % 2 == 0 ) color = "white";
    %>
    <tr bgcolor=<%=color %> onmouseover="this.style.backgroundColor='SkyBlue'">
        <!-- 수정과 삭제를 위한 링크로 no를 전송 -->
        <td align="center"><%= brd.getNo()%></td>
        <td align="left">

```

```

        <a href="editboard.jsp?no=<%= brd.getNo()%>"><%= brd.getTitle() %></a></td>
        <td align="center"><%= brd.getName() %></td>
        <!-- 게시 작성일을 2010-3-15 10:33:21 형태로 출력 -->
        <td align="center"><%= df.format(brd.getRegdate()) %></td>
        <td align="center"><%= brd.getEmail() %></td>
    </tr>
    <%
    }
%>
</table>
<% } %>
<hr width="90%">
<p>조회된 게시판 목록 수가 <%=counter%>개 입니다.
</center>
<hr>
<center>
<form name="form" method="post" action="editboard.jsp">
    <input type="submit" value="게시등록">
</form>
</center>
</body>
</html>

```

[결과]



- boardform.js

```

function deletecheck() {
    if ( document.boardform.passwd.value.replace(/\Ws/g,"") === "" ) {
        alert("암호를 입력해 주세요.");
        document.boardform.passwd.focus();
        return;
    }
}

```

```

        }

        ok = confirm("삭제하시겠습니까?");

        if (ok) {
            document.boardform.menu.value='delete';
            document.boardform.submit();
        } else {
            return;
        }
    }

function insertcheck() {
    with(document.boardform){
        if (name.value.replace(/\s/g,"")=="") {
            alert("이름을 입력해 주세요.");
            name.focus();
            return false;
        }
        if(email.value.replace(/\s/g,"")==""){
            alert("이메일을 입력해 주세요.");
            email.focus();
            return false;
        }
        if (title.value.replace(/\s/g,"")=="") {
            alert("제목을 입력해 주세요.");
            title.focus();
            return false;
        }
        if (content.value.replace(/\s/g,"")=="") {
            alert("내용을 입력해 주세요.");
            content.focus();
            return false;
        }
        if (passwd.value.replace(/\s/g,"")=="") {
            alert("비밀번호를 입력하세요.");
            passwd.focus();
            return false;
        }
        if (passwd.value.search(/\W/) >=0) {
            alert("비밀번호는 알파벳과 숫자로 구성되어야 합니다.");
            passwd.value="";
            passwd.focus();
        }
    }
}

```

```

        return false;
    }

    if (passwd.value.length<4 || passwd.value.length>8) {
        alert("비밀번호는 4자이상 8자이하로 작성되어야 합니다.");
        passwd.value="";
        passwd.focus();
        return false;
    }
}

document.boardform.menu.value='insert';
document.boardform.submit();
}

function updatecheck() {
    with(document.boardform){
        if (name.value.replace(/\s/g,"")=="") {
            alert("이름을 입력해 주세요.");
            name.focus();
            return false;
        }
        if(email.value.replace(/\s/g,"")==""){
            alert("이메일을 입력해 주세요.");
            email.focus();
            return false;
        }
        if (title.value.replace(/\s/g,"")=="") {
            alert("제목을 입력하세요.");
            title.focus();
            return false;
        }
        if (passwd.value.replace(/\s/g,"")=="") {
            alert("비밀번호를 입력하세요.");
            passwd.focus();
            return false;
        }
        if (passwd.value.search(/\W/)>=0) {
            alert("비밀번호는 알파벳과 숫자로 구성되어야 합니다.");
            passwd.value="";
            passwd.focus();
            return false;
        }
    }
}

```

- editboard.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head> <meta charset="UTF-8">
<title>게시판 기본 예제 : 게시 품 작성</title>
<style type="text/css">
    body, table {font-size:9pt;font-family:굴림;}
</style>
<!-- 파일 boardform.js : 품의 각 입력 값이 있는지를 검토하는 함수 구현 -->
<script type="text/javascript" src="boardform.js"> </script>
</head>
<body>
    <%@ page import="board.*" %>
    <%
        String name = "";
        String email = "";
        String title = "";
    %>
```

```

String content = "";
String headline = "등록";
String no = request.getParameter("no");
if (no != null) {
    //등록이 아닌 경우, 출력을 위해 선택한 게시의 각 필드 내용을 저장
    int nonum = Integer.parseInt(no);
    BoardDBCP brddb = new BoardDBCP();
    BoardEntity brd = brddb.getBoard(nonum);
    name = brd.getName();
    email = brd.getEmail();
    title = brd.getTitle();
    content = brd.getContent();
    headline = "수정 삭제";
}
%>

<center><form name="boardform" method="post" action="processboard.jsp" >
<!-- menu : 등록, 수정 또는 삭제 구분을 위한 매개변수로 이용 -->
<input type="hidden" name="menu" value="insert">
<!-- 수정 또는 삭제를 위한 게시 id를 hidden으로 전송 -->
<input type="hidden" name="no" value="<%={no} %>">
<table width="800" border="0" cellspacing="0" cellpadding="7">
<tr><td colspan="4">
    <h2 align="center">게시판 <%={headline} %> 프로그램 </h2>
    <hr></td>
</tr>
<tr>
    <td width="100">이 름 : </td>
    <td width="300">
        <input type="text" name="name" value="<%={name} %>" size="30" maxlength="10"></td>
    <td width="100">전자메일 :</td>
    <td width="300">
        <input type="text" name="email" size="30" value="<%={email} %>" maxlength="30"></td>
    </tr>
    <tr >
        <td width="50">제 목 : </td>
        <td colspan="3">
            <input type="text" name="title" size="80" value="<%={title} %>" maxlength="80"></td>
        </tr>
        <tr><td colspan="4">
            <textarea name="content" rows="10" cols="105"><%={content} %></textarea> <br>

```

```

<fieldset style="width:97%">
    <script type="text/javascript">
        v();
    </script>
</fieldset>
</td>
</tr>
<tr>
    <td colspan="4">비밀번호 :
        <input type="password" name="passwd" size="20" maxlength="10"><font color="red">
            현재 게시 내용을 수정 또는 삭제하려면 이미 등록한 비밀번호가 필요합니다.</font></td>
    </tr>
    <tr>
        <td colspan="4" height="5"><hr size="2"></td>
    </tr>
    <tr>
        <td colspan="4">
            <% if (no == null) { %>
                <!-- 버튼을 누르면 boardform.js의 함수를 실행하여 processboard.jsp로 이동 -->
                <input type="button" value="등록" onClick="insertcheck()">
            <% } else { %>
                <!-- 버튼을 누르면 boardform.js의 각 함수를 실행하여 processboard.jsp로 이동 -->
                <input type="button" value="수정완료" onClick="updatecheck()">
                <input type="button" value="삭제" onClick="deletecheck()">
            <% } %>
                <!-- 목록보기 버튼은 listboard.jsp로 이동 -->
                <input type="button" value="목록보기" onClick="location.href='listboard.jsp'">
                <input type="reset" value="취소">
            </td>
        </tr>
        </table>
    </form>
</center>
</body>
</html>

```

[결과]

개시판 기본 예제 : 게시 품 작성 - Microsoft Internet Explorer

파일(E) 편집(E) 보기(V) 즐겨찾기(A) 도구(I) 도움말(H)

주소(D) http://localhost:8080/exam01/editboard.jsp

제목 : 기본 게시판의 프로그램 구성

내용 : 기본 게시판을 위한 프로그램으로, 자바빈즈 프로그램 2개의 파일과 JSP 프로그램 3개의 파일, 자바스크립트 1 개의 파일로 구성되었습니다. (^_~)

비밀번호 : **** 현재 게시 내용을 수정 또는 삭제하려면 이미 등록한 비밀번호가 필요합니다.

등록 목록보기 취소

로컬 인트라넷

개시판 기본 예제 : 게시 품 작성 - Microsoft Internet Explorer

파일(E) 편집(E) 보기(V) 즐겨찾기(A) 도구(I) 도움말(H)

주소(D) http://localhost:8080/exam01/editboard.jsp?no=1

제목 : 기본 게시판의 프로그램 구성

내용 : 기본 게시판을 위한 프로그램으로, 자바빈즈 프로그램 2개의 파일과 JSP 프로그램 3개의 파일, 자바스크립트 1 개의 파일로 구성되었습니다. (^_~)

비밀번호 : **** 현재 게시 내용을 수정 또는 삭제하려면 이미 등록한 비밀번호가 필요합니다.

수정완료 삭제 목록보기 취소

로컬 인트라넷

- processboard.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>게시판 기본 예제 : 게시 등록 수정 삭제 처리</title>
</head>
<body>
```

```

<!-- 게시의 등록, 수정, 삭제를 위한 자바빈즈 이용 선언-->
<jsp:useBean id="brd" class="board.BoardEntity" scope="page" />
<jsp:useBean id="brddb" class="board.BoardDBCP" scope="page" />
<%
    //한글 처리를 위해 문자인코딩 지정
    request.setCharacterEncoding("UTF-8");
    //등록(insert), 수정(update), 삭제(delete) 중 하나를 저장
    String menu = request.getParameter("menu");
    // 등록 또는 수정 처리 모듈
    if ( menu.equals("delete") || menu.equals("update") ) {
        String no = request.getParameter("no");
        String passwd = request.getParameter("passwd");
        int nonum = Integer.parseInt(no);
        //데이터베이스 자바빈즈에 구현된 메소드 isPasswd()로
        //id와 암호가 일치하는지 검사
        if ( !brddb.isPasswd(nonum, passwd) ) {
            %>
                <!-- 암호가 틀리면 이전 화면으로 이동 -->
                <script>alert("비밀번호가 다릅니다."); history.go(-1);</script>
            <%
        } else {
            if ( menu.equals("delete") ) {
                //삭제를 위해 데이터베이스 자바빈즈에
                //구현된 메소드 deleteDB() 실행
                brddb.deleteDB(nonum);
            } else if ( menu.equals("update") ) {
                %>
                    <!-- 수정 시 BoardEntity에 지정해야 하는 필드 id -->
                    <jsp:setProperty name="brd" property="no" />
                    <jsp:setProperty name="brd" property="name" />
                    <jsp:setProperty name="brd" property="title" />
                    <jsp:setProperty name="brd" property="email" />
                    <jsp:setProperty name="brd" property="content" />
            <%
                //수정을 위해 데이터베이스 자바빈즈에
                //구현된 메소드 updateDB() 실행
                brddb.updateDB(brd);
            }
            //기능 수행 후 다시 게시 목록 보기로 이동
            response.sendRedirect("listboard.jsp");
        }
    }
}

```

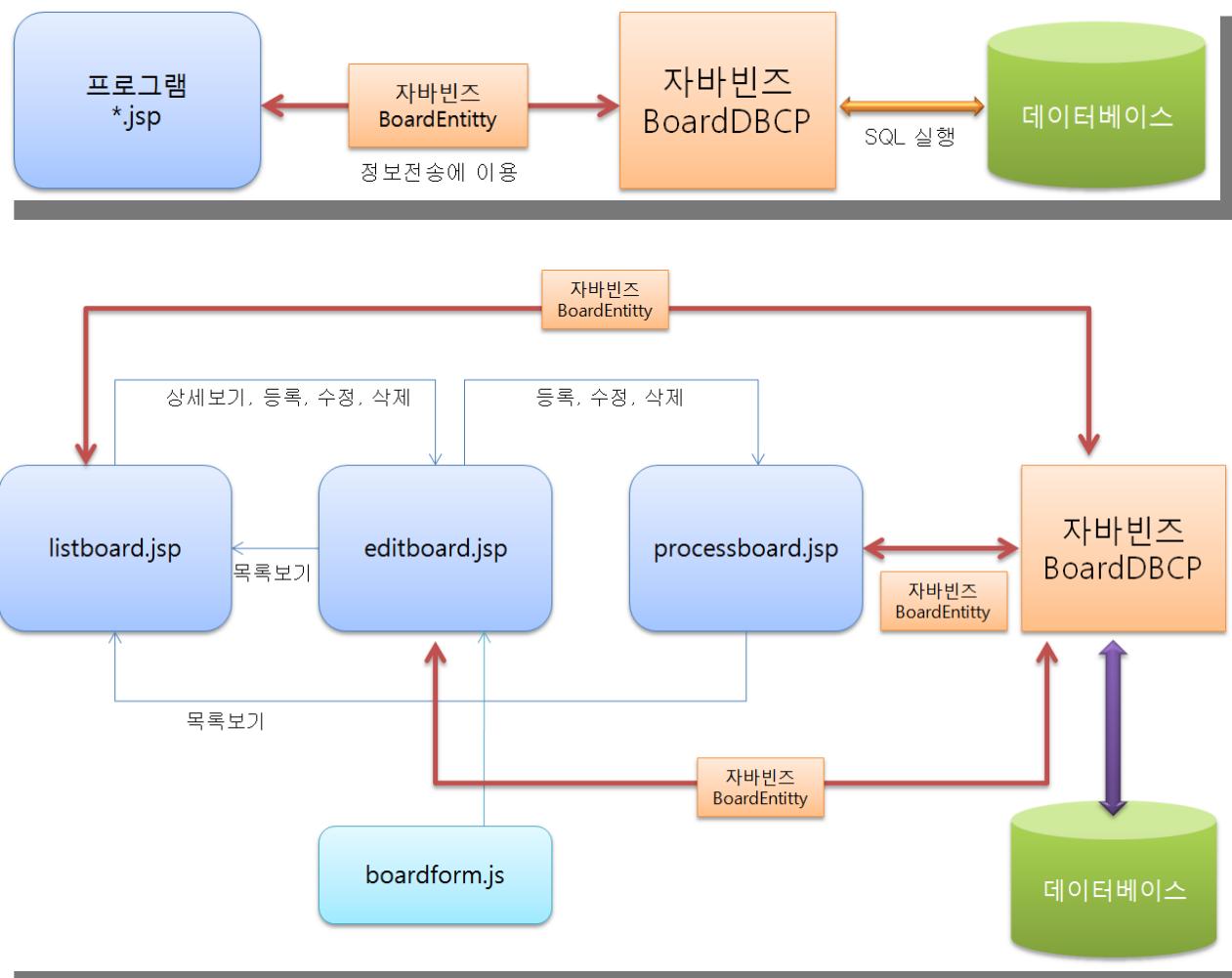
```

        }
    } else if ( menu.equals("insert") ) {
%>
    <!-- 등록 시 BoardEntity에 지정해야 하는 필드 passwd -->
    <jsp:setProperty name="brd" property="name" />
    <jsp:setProperty name="brd" property="title" />
    <jsp:setProperty name="brd" property="email" />
    <jsp:setProperty name="brd" property="content" />
    <jsp:setProperty name="brd" property="passwd" />
<%
//등록을 위해 데이터베이스 자바빈즈에 구현된 메소드 insertDB() 실행
brddb.insertDB(brd);
//기능 수행 후 다시 게시 목록 보기로 이동
response.sendRedirect("listboard.jsp");
%

```

</body>

</html>



19. 표현 언어 개요

19-1. 표현언어란?

- 객체의 간단한 표현언어

JSP에서 브라우저의 출력은 주로 표현식 태그를 이용한다.

```
<%= request.getParameter("userid") %>
```

위와 같은 태그를 간단히 줄이는 방법으로, 다음과 같이 표현언어(Expression Language)를 이용하는 방법이 있다.

```
 ${ param.userid }  
 ${ param['userid'] }  
 ${ param["userid"] }
```

표현언어는 자바 표준 라이브러리(JSTL: Java Standard Tag Library)1.0 규약에 처음 소개된 내용으로, JSP 2.0 규약에서는 편의를 위하여 이러한 표현언어를 새롭게 추가하게 되었다. 표현언어는 원리 SPEL(Simplest Possible Expression Language)로 불렸으며, JSP 페이지에서 이용되는 여러 외부 데이터 객체를 쉽게 간편하게 참조하기 위한 언어이다. 즉 표현언어는 <%= %>인 표현식 대신에 사용하거나 내장 객체 또는 액션태그에 저장된 자료를 쉽게 참조하기 위해 만들어진 언어이다.

표현 언어의 주요 기능

- JSP의 네 가지 기본 객체가 제공하는 영역의 속성 사용
- 집합 객체에 대한 접근 방법 제공
- 수치 연산, 관계 연산, 논리 연산자 제공
- 자바 클래스 메서드 호출 기능 제공
- 표현언어만의 기본 객체 제공

※ 스크립트 요소(스크립트릿, 표현식, 선언부)를 제외한 나머지 부분에서 사용한다.

표현언어는 반드시 \${}로 시작하고, \${}와 {} 사이에 빈 공간은 없어야 하며, }으로 종료된다. 즉 간단히 다음과 같은 구문으로 exp의 결과를 브라우저에 출력한다. 여기서 하나 주의할 점은 다음의 exp가 JSP 스크립트릿이나 선언에서 사용하는 자바 변수가 아니라는 것이다.

```
 ${ exp }
```

표현언어의 기본적인 문법을 정리하면 다음과 같다.

- 표현언어는 \${}로 시작한다.
- 표현언어의 문자구조는 \${ exp }와 같다.
- 표현식 exp에서는 산술, 관계, 논리와 같은 기본적인 연산이 가능하다.
- 표현식은 [객체명](브라켓 연산자 []를 사용해도 된다), [객체명.속성명], [객체명[첨자]], [객체명["속성명"]], [객체명['속성명']]과 같은 구조로 구성된다.

- 표현언어의 자료유형과 상수

표현언어에서 사용하는 자료유형은 정수형, 실수형, 문자열형, 논리형, null 5가지이다.

문자열형은 "JSP", 'JSP'와 같이 큰따옴표("), 작은따옴표(') 둘 다 이용할 수 있다.

- 정수형
- 실수형
- 문자열형
- true, false의 논리형(boolean)형
- null형

마찬가지로 표현언어에서 이용되는 상수는 다음과 같다.

- 논리값 true, false
- 자바에서 이용되는 정수형으로 1, 2, -5
- 자바에서 이용되는 실수형으로 3.1, 4.5E4
- 문자열은 "Java", 'Java'와 같이 큰따옴표("), 작은따옴표(') 모두 이용 가능
- 아무 것도 없다는 의미의 null

- 표현언어의 변수

표현언어에서 이용되는 변수는 pageContext.getAttribute(변수이름)으로 조회되는 변수의 평가값으로 사용된다. 만일 다음과 같은 표현식이 있다고 가정하자.

```
{${ product }}
```

JSP 엔진은 page, session, application 범위에서 등록된 product를 평가하여 그 값을 반환한다. 만일 product를 찾을 수 없다면 null을 반환한다. 물론 변수가 표현언어 내장 객체라면 그 내장 객체의 값을 반환한다.

19.2 표현언어 연산자

- 다양한 연산자

표현언어의 연산자는 자바언어에서 지원하는 연산자인 산술, 관계, 논리, 조건, 팔호 연산자 등을 지원한다. 또한 산술연산자(+, -, *, /)에서 나누기인 div, 나머지인 mod도 지원하며, 논리 연산자(&&, ||, !)에서는 and, or, not 관계연산자(<, <=, ==, !=, >=, >) 단어의 첫자를 띤 lt(less than), le(less than or equal), eq(equal), ne(not equal), ge(greater than or equal), gt(greater than)도 지원한다.

연산자분류	연산자종류
이항 산술 연산자	+ - * / div % mod
이항 관계 연산자	< <= == != >= > lt le eq ne ge gt
첨자 연산자	. []
이항 논리 연산자	&& and or
단항 논리 연산자	! not
단항 산술 연산자	-
empty 연산자	empty

삼항 조건 연산자	? :
괄호 연산자	()

표현언어 연산자의 연산 우선순위를 살펴보면, [] . 연산자의 우선순위가 가장 높으며, 다음으로 괄호연산자 (), 그리고 조건 연산자 ?: 이 연산순위가 가장 낮다

연산자	우선순위
[] .	1
()	2
- ! not empty	3
* / div % mod	4
+ -	5
< <= > = > lt le ge gt	6
== != eq ne	7
&& and	8
or	9
? :	10

표현식 exp를 출력하는 표현언어 문장은 \${ exp }인데, \$가 아닌 \$\$는 표현언어로 인식하지 않으므로 그대로 문자 \$를 브라우저에 출력한다. 즉 \$\$1+2는 표현언어의 결과가 아닌 문자열 \${1+2}를 그대로 출력하며, \${1+2}는 표현언어의 결과 3을 출력한다.

```
<td>$$1+2</td>
<td>${1+2}</td>
```

조건연산자인 [exp1 ? result1 : result2]는 exp1을 검사하여 true이면 연산의 결과는 result1이고 false이면 연산 결과가 result2인 연산자이다.

```
<td>${(1==2) ? 3 : 4}</td>
```

관계연산자인 피연산자가 정수, 실수뿐만 아니라 문자열에도 쓰이며 이 경우 문자열 순서대로 유니코드 값의 비교이다.

```
<td>${(1==2) ? 3 : 4}</td>
```

다음 비교연산에서 피연산자인 'hit'와 'hip'에서 앞의 두 문자인 'hi'는 같으므로 그 다음 문자인 't'와 'p'를 비교한다. 그러므로 'hit'는 'hip'보다 크다. 다음 연산의 결과는 (!true)이므로 false가 출력된다.

```
<td>${ !( 'hit' gt 'hip' ) }</td>
```

다음 연산에서 관계 연산자인 >와 !=는 우선순위가 논리연산자인 and보다 빠르므로,

```
<td>${ 5>4 and 10 != 3*10 }</td>
```

괄호연산자를 사용하면 다음과 같고, 그 결과는 (true and true)이므로 true이다.

```
<td>${ (5 > 4) and (10 != 3*10) }</td>
```

- basicEL.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>표현언어 연산자</title>
</head>
<body>
<h2>JSP 2.0 표현언어(Expression Language) 연산자</h2>
<hr><br>
<table border="1">
<tr>
<td><b>EL Expression</b></td>
<td><b>Result</b></td>
</tr>
<tr>
<td>₩${1}</td>
<td>${1}</td>
</tr>
<tr>
<td>₩${1 + 2}</td>
<td>${1 + 2}</td>
</tr>
<tr>
<td>₩${1.2 + 2.3}</td>
<td>${1.2 + 2.3}</td>
</tr>
<tr>
<td>₩${1.2E4 + 1.4}</td>
<td>${1.2E4 + 1.4}</td>
</tr>
<tr>
<td>₩${-4 - 2}</td>
<td>${-4 - 2}</td>
</tr>
<tr>
<td>₩${21 * 2}</td>
<td>${21 * 2}</td>
</tr>
<tr>
```

```

<td>${3 / 4}</td>
<td>${3 / 4}</td>
</tr>
<tr>
    <td>${3 div 4}</td>
    <td>${3 div 4}</td>
</tr>
<tr>
    <td>${3 / 0}</td>
    <td>${3 / 0}</td>
</tr>
<tr>
    <td>${10 % 4}</td>
    <td>${10 % 4}</td>
</tr>
<tr>
    <td>${10 mod 4}</td>
    <td>${10 mod 4}</td>
</tr>
<tr>
    <td>${(1==2) ? 3 : 4}</td>
    <td>${(1==2) ? 3 : 4}</td>
</tr>
<tr>
    <td>${'a' < 'b'}</td>
    <td>${'a' < 'b'}</td>
</tr>
<tr>
    <td>${!('hit' gt 'hip')}</td>
    <td>${!('hit' gt 'hip')}</td>
</tr>
<tr>
    <td>
        ${5 > 4 and 10 != 3*10}</td>
    <td>
        ${5 > 4 and 10 != 3*10}</td>
    </tr>
</table>
</body>
</html>

```

JSP 2.0 표현언어(Expression Language) 연산자

EL Expression	Result
\${1}	1
\${1 + 2}	3
\${1.2 + 2.3}	3.5
\${1.2E4 + 1.4}	12001.4
\${-4 - 2}	-6
\${21 * 2}	42
\${3 / 4}	0.75
\${3 div 4}	0.75
\${3 / 0}	Infinity
\${10 % 4}	2
\${10 mod 4}	2
\${(1==2) ? 3 : 4}	4
\${'a' < 'b'}	true
\${!('hit' gt 'hip')}	false
\${5 > 4 and 10 != 3*10}	true

- empty 연산자

표현언어의 독특한 연산자인 empty 연산자는 empty 다음의 피연산자인 표현식이 null인가를 검사하여 true 또는 false를 반환하는 연산자이다. 정의되지 않은 표현식 n 또는 상수 null을 표현언어로 출력하면 아무것도 출력되지 않는다.

```
 ${ null }  
 ${n}
```

바로 이런한 표현식 n을 empty 연산자로 \${ empty n}하면 그 결과는 true이다. 상수 null도 같은 결과이다.

```
 ${ empty null}  
 ${ empty n}
```

즉 다음과 같은 표현언어에서 exp가 null 또는 null을 의미하는 값이면 true이고, 그렇지 않으면 false이다.

```
 ${ empty exp }
```

- null
- 정의되지 않은 객체
- 빈 문자열
- 길이가 0인 배열
- 비어있는 집합체

표현언어 내장 객체 param을 이용하여 param.user를 empty 연산자로 검사하면, 매개변수 user가 정의되어 값이 지정되면 false이고, 그렇지 않으면 true이다.

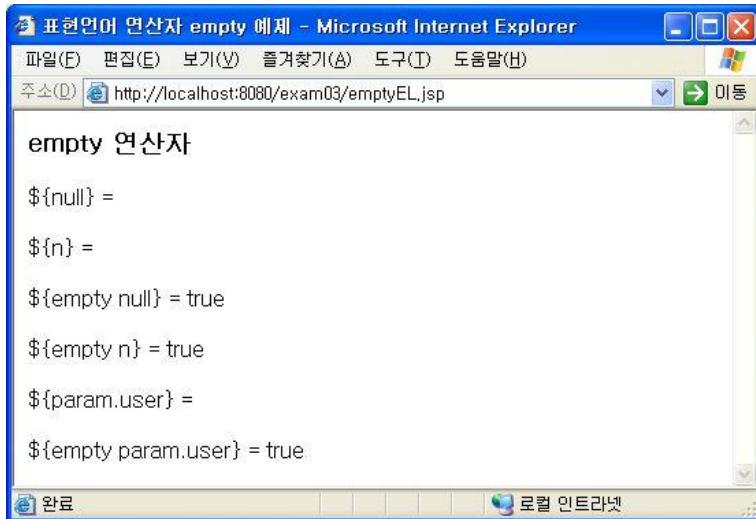
```
 ${ empty param.user}
```

- emptyEL.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head><meta charset="UTF-8">  
<title>표현언어 연산자 empty 예제</title>  
</head>  
<body>  
 ${null} = ${ null } <p>  
 ${n} = ${n} <p>  
 ${empty null} = ${empty null} <p>  
 ${empty n} = ${empty n} <p>  
 ${param.user} = ${param.user} <p>  
 ${empty param.user} = ${empty param.user}
```

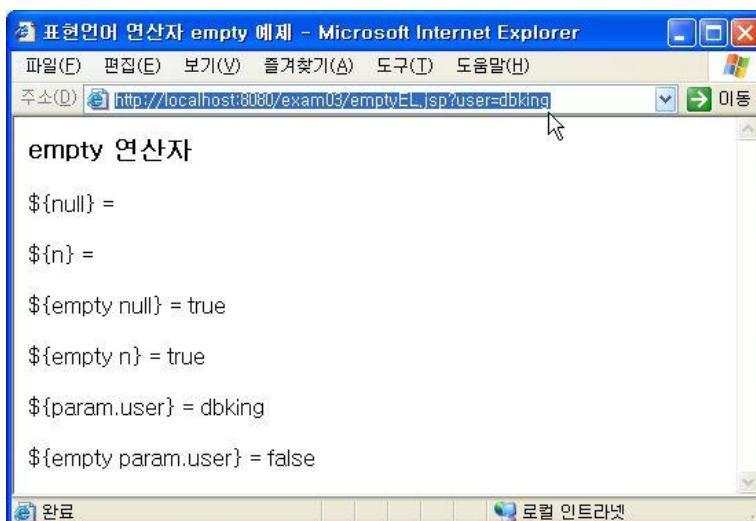
```
</body>  
</html>
```

[결과]



주소줄 URL에 [?user=dbking]으로 직접 매개변수 user를 추가하면, \${param.user}의 결과는 dbking이며, \${ empty param.user }의 결과는 false인 것을 알 수 있다.

```
http://localhost:8080/exam03/emptyEL.jsp?user=dbking
```



19.3 표현 언어 내장 객체

19.3.1 표현 언어 내장 객체 개요

- 표현언어 내장 객체 종류

표현언어는 다양한 내장 객체를 제공하여 웹 응용 프로그램에서 필요한 정보를 쉽게 참조할 수 있도록 한다. 표현언어는 다음과 같이 JSP page 객체, 범위, 요청 매개변수, 요청헤더, 초기화 매개변수, 쿠키 등 6가지 분류에서 총 11가지의 내장 객체를 제공한다.

분류	내장 객체	자료유형	기능
JSP page 객체	pageContext	javax.servlet.jsp. PageContext	JSP 페이지 기본 객체로서, servletContext, session, request, response등의 여러 객체를 참조 가능
범위	pageScope	java.util.Map	page 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체, \${pageScope.속성}으로 값을 참조
	requestScope	java.util.Map	request 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체, \${requestScope.속성}으로 값을 참조
	sessionScope	java.util.Map	session 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체, \${sessionScope.속성}으로 값을 참조
	applicationScope	java.util.Map	application 기본 객체에 저장된 속성의 <속성, 값>을 저장한 Map 객체, \${applicationScope.속성}으로 값을 참조
요청 매개변 수	param	java.util.Map	요청 매개변수 <매개변수이름, 값>을 저장한 Map 객체, \${param.name}은 request.getParameter(name)을 대체
	paramValues	java.util.Map	요청 매개변수 <매개변수이름, 값>을 저장한 Map 객체, request.getParameterValues()처리와 동일
요청 헤더	header	java.util.Map	요청 정보의 <헤더이름, 값>을 저장한 Map 객체, \${ header["name"] }은 request.getHeader(헤더이름)와 같음.
	headerValues	java.util.Map	요청 정보 배열을 <헤더이름, 값>을 저장한 Map 객체, request.getHeaders()의 처리와 동일
초기화 매개변 수	initParam	java.util.Map	초기화 매개변수의 <이름, 값>을 저장한 Map 객체, \${initParam.name}은 application.getInitParameter(name)을 대체
쿠키	cookie	java.util.Map	쿠키 정보의 배열을 <쿠키이름, 값>을 저장한 Map 객체, request.getCookies()의 Cookies 배열의 이름과 값으로 Map을 생성.

- 표현언어에서 Map의 이용

위 표현언어 내장 객체의 자료유형을 살펴보면 pageContext를 제외하고는 모두 java.util.Map 인터페이스이다. Map 인터페이스는 자바에서 지원하는 집합체의 한 종류로 Map<key, value>의 형태로 이용되는데, key는 키로 사용될 객체이며, value는 키로 저장하는 값으로 사용될 객체이다. 즉 하나의 key에 하나의 value를 대응시켜 저장하고, 저장된 key로 value를 조회하는 구조이다. Map에서 key는 절대로 중복(duplicate) 될 수 없다.

Map에 <키, 값>의 쌍을 저장하려면 메소드 put(키, 값)을 이용하며, 키값으로 조회하려면 get(키)를 이용한다. Map에서 이용되는 주요 메소드 중에는 키와 값이 Map에 있는지 검사하는 메소드 containsKey() 와 containsValue()가 있다.

표현언어에서 a.b는 일반적으로 a["b"]와 같다. 표현식 a[b]의 평가는 다음의 절차를 따른다.

- 만일 a가 Map이면 a.get(b)이다. 그러나 !a.containsKey(b)이면 null을 반환한다.
- 만일 a가 List나 배열이면 b를 정수의 첨자로 변환하여 a.get(b)를 평가하거나 또는 적절히 Array.get(a, b)를 평가한다. 이 과정에서 첨자의 IndexOutOfBoundsException이 발생하면 null을 반환하며, 다른 예외가 발생하면 오류가 발생한다.
- 만일 a가 자바빈즈 객체이면 b를 문자열로 변환하여, 속성 b에 해당하는 getter인 객체의 메소드 getB() 호출 결과를 반환하며, 이 과정에서 예외가 발생하면 오류가 발생한다.

표현식 a.b에서 만일 a가 표현언어의 내장 객체라면 pageContext를 제외한 모든 내장 객체는 Map이므로, 모두 위의 첫번째의 평가 과정을 거쳐 결과를 반환한다.

- 표현언어 내장 객체 이용

```
<%  
    request.setAttribute("univ", "한국대학교");  
%>
```

표현언어에서 위 속성 "univ"를 참조하여 출력하려면 표현언어 내장 객체 requestScope 다음에 점 연산자를 이용하여 속성 univ를 바로 참조할 수 있다.

```
 ${requestScope.univ}
```

위 구문에서 점 연산자는 내장 객체명['속성명']과 대괄호 연산자 []로도 가능하며, 속성명은 큰따옴표("") 또는 작은따옴표('') 둘 다 이용할 수 있다.

```
 ${requestScope['univ']}  
 ${requestScope["univ"]}
```

다음과 같이 JSP 내장 객체 application으로 속성 "name"을 저장했다면, 표현언어 내장 객체 applicationScope.name으로 application 속성 "name"을 참조할 수 있다.

```
application.setAttribute("name", "홍길동");  
...  
 ${applicationScope.name}
```

- implicitObjectEL.jsp

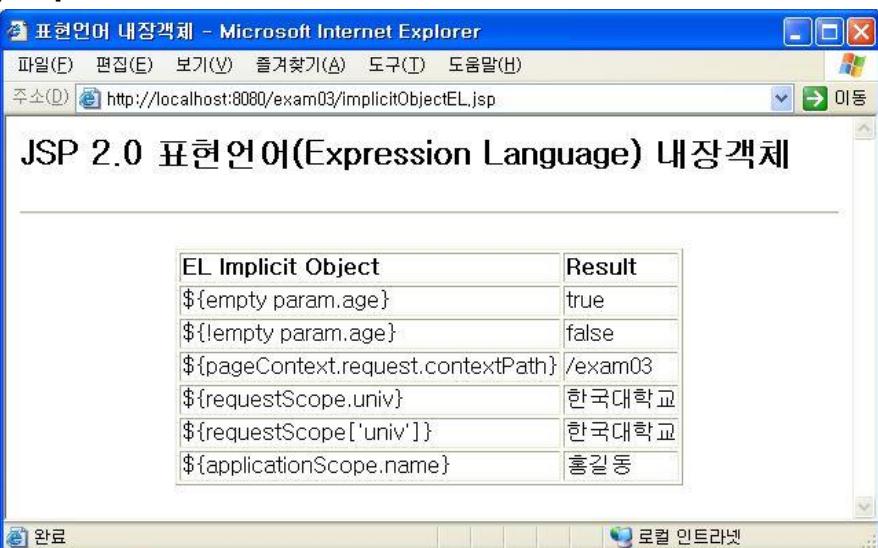
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>표현언어 내장객체</title>
</head>
<body>
    <h2>JSP 2.0 표현언어(Expression Language) 내장객체</h2>
<%
    request.setAttribute("univ", "한국대학교");
    application.setAttribute("name", "홍길동");
%>
<hr><br>
<table border="1" align="center" >
<tr>
    <td><b>EL Implicit Object</b></td>
    <td><b>Result</b></td>
</tr>
<tr>
    <td>${empty param.age}</td>
    <td>${empty param.age}</td>
</tr>
<tr>
    <td>${!empty param.age}</td>
    <td>${!empty param.age}</td>
</tr>
<tr>
    <td>${pageContext.request.contextPath}</td>
    <td>${pageContext.request.contextPath}</td>
</tr>
<tr>
    <td>${requestScope.univ}</td>
    <td>${requestScope.univ}</td>
</tr>
<tr>
    <td>${requestScope['univ']}</td>
    <td>${requestScope['univ']}</td>
</tr>
<tr>
```

```

<td>${applicationScope.name}</td>
<td>${applicationScope.name}</td>
</tr>
</table>
</body>
</html>

```

[결과]



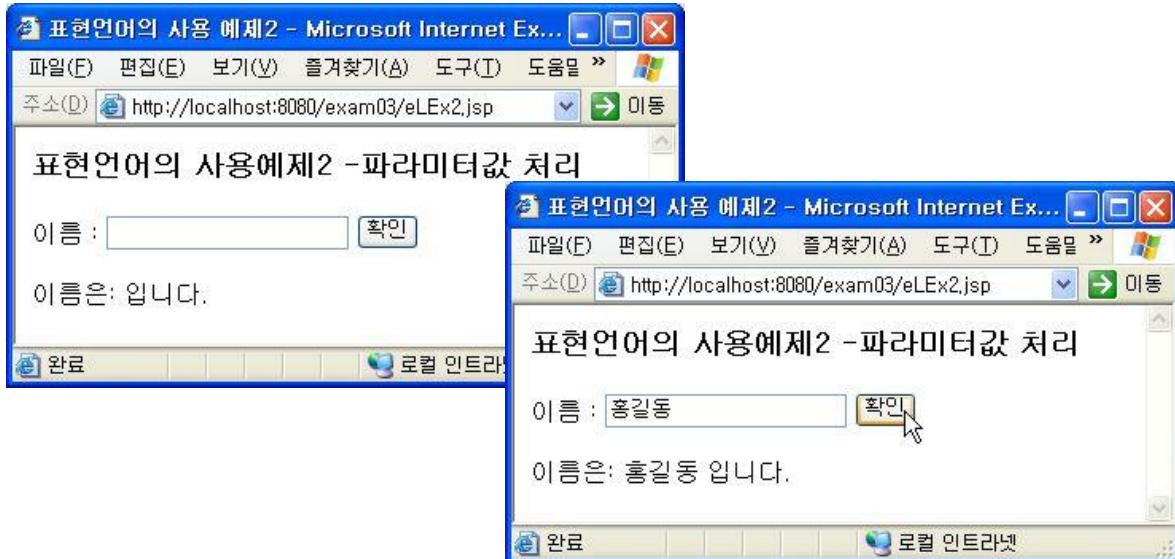
eLEx2.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<% request.setCharacterEncoding("UTF-8");%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>표현언어의 사용 예제2</title>
</head>
<body>
<h3>표현언어의 사용예제2 -파라미터값 처리</h3>
<p>
<form action="eLEx2.jsp" method="post">
    이름 : <input type="text" name="name" value="${param['name']}'>
    <input type="submit" value="확인">
</form>
<p>
이름은: ${param.name} 입니다.
</body>
</html>

```

[결과]



19.3.2 주요 표현 언어 내장 객체

- 표현언어 내장 객체 Cookie

JSP 페이지에서 쿠키 객체 c를 "user" 이름으로 생성한 후, 내장 객체 response의 메소드 addCookie(c)를 호출하여 쿠키 c를 저장했다고 가정한다.

```
Cookie c = new Cookie("userid", "king");
response.addCookie(c);
```

등록된 쿠키는 다음과 같이 간단한 표현언어 내장 객체인 cookie를 이용해 cookie.userid.value로 참조할 수 있다. 즉 쿠키를 생성할 때 쿠키이름인 "userid"를 이용하여 **cookie.쿠키이름.value**로 한다.

```
${ cookie.userid.value }
```

다음과 같이 쿠키를 생성할 때 쿠키이름이 "CookieID"라면 표현언어에서는 cookie.CookieID.value로 쿠키 값인 "cookievalue"를 참조할 수 있다.

```
Cookie c = new Cookie("CookieID", "cookievalue");
${ cookie.CookieID.value }
```

만일 위의 쿠키를 JSP 스크립트릿에서 참조하여 출력하려면 등록된 쿠키가 저장된 쿠키 배열을 참조한 후, 반복문을 이용해 원하는 쿠키를 검사한 후 참조해야 함으로 여러 줄의 코딩이 필요하나, 표현언어를 이용하면 위와 같이 매우 간단한 것을 알 수 있다. 즉 아래의 문장을 \${ cookie.userid.value }으로 대체할 수 있으니 바로 이것이 표현언어를 사용하는 장점이다.

```
Cookie[] cs = request.getCookie();
for(Cookie ck : cs) {
    if(ck.getName().equals("userid"))
        out.println(ck.getValue());
}
```

- cookieObjectEL.jsp(cookieObjectEL.jsp?userid)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>표현언어 내장객체 cookie</title>
</head>
<body>
<h2>표현언어 내장객체 cookie 이용</h2>
<%
    Cookie c = new Cookie("userid", "king");
    response.addCookie(c);
%>
W${cookie.userid.value} = ${cookie.userid.value}
<p><hr><h2>스크립트릿에서 직접 Cookie 이용</h2>
<%
    Cookie[] cs = request.getCookies();
    if (!(cs == null)) {
        for (Cookie ck : cs) {
            if ( ck.getName().equals("userid") )
                out.println(ck.getValue());
        }
    }
%>
</body>
</html>
```

[결과]



- 표현언어 내장 객체 header와 headerValues

표현언어 내장 객체 header를 이용하여 host와 user-agent의 속성 출력이 가능하다. 속성 host는 서버컴

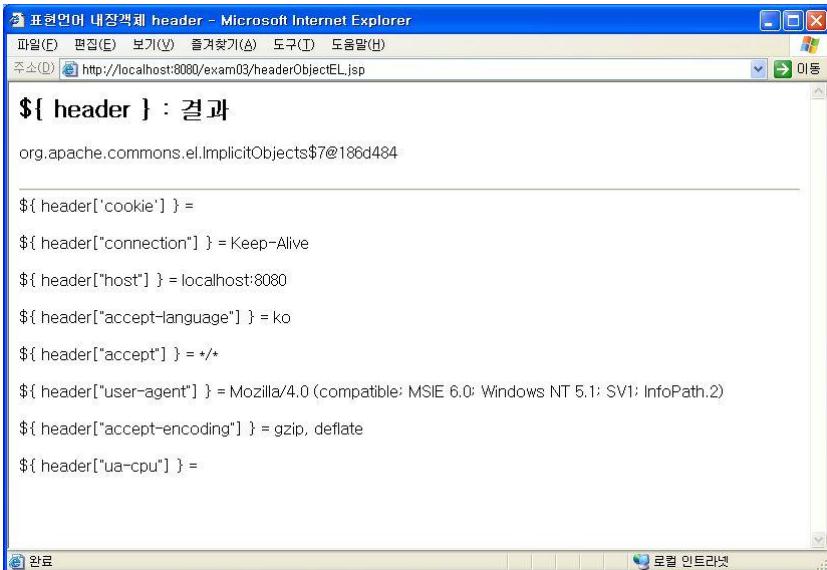
퓨터 이름이며, 속성 user-agent는 클라이언트 브라우저의 정보를 나타낸다.

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

- headerObjectEL.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head><meta charset="UTF-8">  
<title>표현언어 내장객체 header</title>  
</head>  
<body>  
<h2>${ header } : 결과 </h2>  
${ header } <p>  
<hr>  
${ header['cookie'] } = ${ header['cookie'] } <p>  
${ header["connection"] } = ${ header["connection"] } <p>  
${ header["host"] } = ${ header["host"] } <p>  
${ header["accept-language"] } = ${ header["accept-language"] } <p>  
${ header["accept"] } = ${ header["accept"] } <p>  
${ header["user-agent"] } = ${ header["user-agent"] } <p>  
${ header["accept-encoding"] } = ${ header["accept-encoding"] } <p>  
${ header["ua-cpu"] } = ${ header["ua-cpu"] } <p>  
</body>  
</html>
```

[결과]



표현언어 내장 객체 headerValues는 request.getHeaders()의 결과의 배열 형태를 Map에 결합시킨 객체로 다음과 같이 이름과 첨자를 이용하여 사용한다.

```
 ${ headerValues.cookie[0] }  
 ${ headerValues.connection[0] }
```

내장객체 headerValues의 점 연산자와 이용되는 이름으로는 위 예제에서 보듯이 cookie, connection, host, accept, accept-language, user-agent, accept-encoding, ua-cpu 등이 있다. 여기서 한 단어로 구성되는 속성은 위와 같이 점 연산자를 이용할 수 있으나, 속성이름에 -이 들어 있는 accept-language와 같은 이름은 점 연산자를 사용할 수 없으며, 다음과 같이 []연산자를 이용하여 \${ headerValues["accept-language"] [0] }와 같이 사용해야 한다.

```
 ${ headerValues["accept-language"] [0] }  
 ${ headerValues["user-agent"] [0] }
```

19.4 액션태그와 표현언어

19.4.1 <jsp:useBean ...>태그의 객체 이용

- ArrayList의 배열 객체 이용

액션 태그 <jsp:useBean ...>을 이용하여 클래스 java.util.ArrayList의 객체 color를 정의하자. 클래스 java.util.ArrayList의 메소드 add()를 이용하여 필요한 색상을 5개 추가한다.

```
<jsp:useBean id="color" class="java.util.ArrayList">  
<%  
    color.add("red");  
    color.add("orange");  
    color.add("green");  
    color.add("blue");  
    color.add("violet");  
%>  
</jsp:useBean>
```

액션태그 <jsp:useBean ...>로 등록한 객체 color는 배열 객체이므로 첨자를 이용한 스타일인 표현언어 \${ color[0] }와 같이 참조할 수 있다.

```
<font color="${color[0]}> <li> 이 색상은 ${color[0]}색입니다. </li> </font>
```

- actiontagEL.jsp

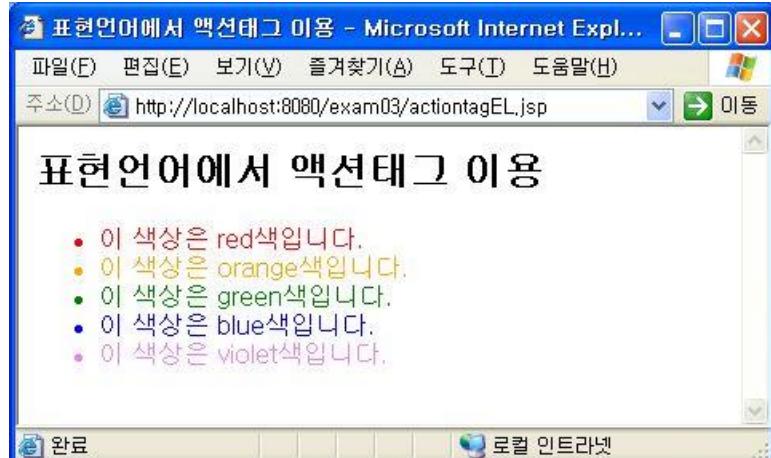
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head><meta charset="UTF-8">  
<title>표현언어에서 액션태그 이용</title>  
</head>  
<body>  
  
<h2> 표현언어에서 액션태그 이용 </h2>
```

```

<jsp:useBean id="color" class="java.util.ArrayList">
<%
    color.add("red");
    color.add("orange");
    color.add("green");
    color.add("blue");
    color.add("violet");
%>
</jsp:useBean>
<ul>
    <font color="${color[0]}"> <li> 이 색상은 ${color[0]}색입니다. </li> </font>
    <font color="${color[1]}"> <li> 이 색상은 ${color[1]}색입니다. </li> </font>
    <font color="${color[2]}"> <li> 이 색상은 ${color[2]}색입니다. </li> </font>
    <font color="${color[3]}"> <li> 이 색상은 ${color[3]}색입니다. </li> </font>
    <font color="${color[4]}"> <li> 이 색상은 ${color[4]}색입니다. </li> </font>
</ul>
</body>
</html>

```

[결과]



- 자바빈즈의 getter 호출

자바빈즈 membertest.User를 만든 후, 액션 태그<jsp:useBean ...>을 이용하여 먼저 만든 클래스 member.User의 객체 user를 생성한다. 자바빈즈 객체 user에 태그 <jsp:setProperty ...>를 이용하여 필드 3개의 값을 저장한다.

```

<jsp:useBean id="user" class="membertest.User" scope="page">
    <jsp:setProperty name="user" property="uname" value="홍길동" />
    <jsp:setProperty name="user" property="uid" value="king" />
    <jsp:setProperty name="user" property="uage" value="20" />
</jsp:useBean>

```

자바빈즈 user를 이용한 표현언어 \${user.uname}는 자바빈즈 user.getUname()과 같은 의미이다. 또한 표

현언어 \${ user.uname }은 표현언어 \${user["uname"]} 또는 \${user['uname']}로도 가능하다.

```
₩${ user.uname } = ${ user.uname } <br>
₩${ user.uid } = ${ user.uid } <br>
₩${ user.uage } = ${ user.uage } <br>
₩${ ${user["uname"]} } = ${user["uname"]} <br>
₩${ ${user["uid"]} } = ${user["uid"]} <br>
₩${ ${user["uage"]} } = ${user["uage"]} <br>
```

- User.java

```
package membertest;
public class User {
    private String uname;
    private String uid;
    private int uage;
    public String getUsername() {
        return uname;
    }
    public void setUsername(String uname) {
        this.uname = uname;
    }
    public String getUid() {
        return uid;
    }
    public void setUid(String uid) {
        this.uid = uid;
    }
    public int getUage() {
        return uage;
    }
    public void setUage(int uage) {
        this.uage = uage;
    }
}
```

- membertest.User를 자바빈즈로 정의하여 표현언어에서 자바빈즈 객체의 getter를 이용하는 프로그램과 그 결과이다.

userEL.jsp

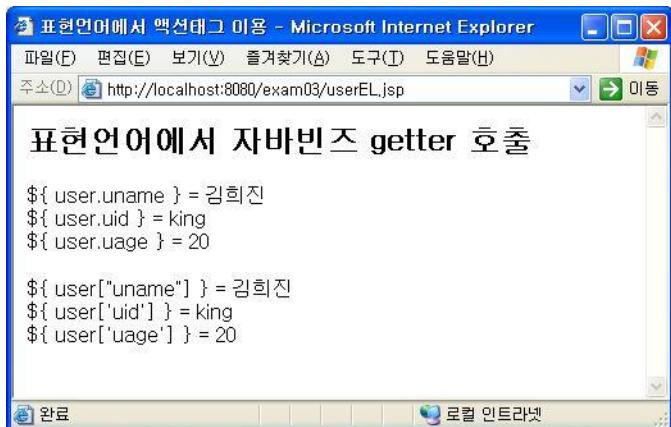
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
```

```

<head><meta charset="UTF-8">
<title>표현언어에서 액션태그 이용</title>
</head>
<body>
<h2> 표현언어에서 자바빈즈 getter 호출 </h2>
<jsp:useBean id="user" class="membertest.User">
    <jsp:setProperty name="user" property="uname" value="김희진"/>
    <jsp:setProperty name="user" property="uid" value="king"/>
    <jsp:setProperty name="user" property="uage" value="20"/>
</jsp:useBean>
${user.uname} = ${user.uname} <br>
${user.uid} = ${user.uid} <br>
${user.uage} = ${user.uage} <br>
<p>
${user["uname"]} = ${user["uname"]} <br>
${user['uid']} = ${user['uid']} <br>
${user['uage']} = ${user['uage']} <br>
</body>
</html>

```

[결과]



19.5 표현언어에서 클래스에 정의된 메소드 이용

18.5.1 표현언어에서 이용할 함수 만들기

- 3개의 파일 필요

표현언어에서 일반적인 자바코드를 사용할 수 없다. 이러한 제한을 다소 해소하기 위한 방법을 JSP에서 제공하는데, 자바 클래스의 메소드를 태그로 정의하여 사용하는 방법이다. 즉 클래스에 정의한 메소드를 표현언어에서 다음과 같이 호출하려면, 먼저 접두어 prefixname으로 태그를 선언해야 한다.

```
{prefixname:functionname()}
```

표현언어에서 함수를 이용하려면 다음과 같이 3가지 작업을 수행한 뒤, JSP 프로그램에서 표현언어의 함

수를 호출할 수 있다.

순서	작업	파일이 저장되는 폴더	파일
1	클래스 작성	[Java Resources: src]/[패키지]	ELDateFormat.java
2	TLD 파일 작성	[WebContent]/[WEB-INF]/[tld]	ELfunction.tld
3	JSP 파일 작성	[WebContent]	function.jsp

JSP의 과거 버전에서는 web.xml 파일에서 TLD 파일에 대한 정보를 지정하는 작업이 필요했으나 이제는 그 작업이 필요 없다.

19.5.2 클래스 작성

- 클래스 ELDateFormat toFormat() 메소드 작성

작성할 클래스 ELDateFormat은 클래스 SimpleDateFormat을 이용하여 인자로 입력된 Date 자료유형의 날짜를 [2010-11-22(월) 16:33:44] 형태로 출력하는 함수 toFormat()을 정의한다.

클래스 SimpleDateFormat에서 날짜를 지정하는 패턴을 살펴보면, yyyy는 연도, MM은 월, dd는 날짜이며, E는 요일이고, HH는 시간, mm은 분, ss는 초를 나타낸다. SimpleDateFormat의 객체 df를 생성하는데 정적(static) 메소드에서 변수 df를 사용하려면 메소드를 정적(static)으로 선언해야 한다.

```
private static SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd(E) HH:mm:ss");
```

JSP 프로그램의 표현언어에서 호출할 수 있는 클래스의 메소드는 정적인 메소드만 가능하다. 그러므로 클래스 ELDateFormat의 정적인 메소드 toFormat()을 다음과 같이 정의한다. 메소드 toFormat()이 정적이므로 그 메소드 내부에서 사용하는 필드인 객체변수 df도 정적으로 선언해야 한다.

```
public static String toFormat(Date date) {
    return df.format(date);
}
```

- ELDateFormat.java

```
package form;
import java.text.SimpleDateFormat;
import java.util.Date;
public class ELDateFormat {
    private static SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd(E) HH:mm:ss");
    public static String toFormat(Date date) {
        return df.format(date);
    }
}
```

19.5.3 태그 라이브러리 디스크립터(TLD) 파일 작성

- [WebContent]/[WEB-INF]/[tld]에 TLD 파일 생성

태그 라이브러리 디스크립트(Tag Library Descriptor)는 말 그대로 라이브러리의 정보를 기술하는 파일로 줄여서 TLD라고 한다. 위에서 작성한 클래스 ELDateFormat의 메소드 toFormat()을 표현언어에서 이용하려면, 클래스 ELDateFormat의 메소드 toFormat()을 태그로 이용한다는 정보를 TLD 파일에 저장해야 한

다. 저장할 TLD 파일의 확장자는 tld이며 일반적으로 폴더 [WEB-INF] 하부에 폴더 [tld]를 만들어 저장한다. 이제 클래스 ELDateFormat의 메소드 toFormat()을 태그로 이용할 정보를 저장하는 TLD 파일 ELfunction.tld를 작성하자.

이클립스 프로젝트 폴더 하부 [WebContent]/[WEB-INF]에 TLD파일이 위치할 폴더 [tld]를 만든다. 폴더 [WebContent]/[WEB-INF]/[tld] 하부에 태그 라이브러리 디스크립트(TLD) 파일 [ELfunction.tld]를 만든다. TLD파일의 처음은 호환성을 위한 태그와 현재 xml문서의 규칙을 저장하고 있는 스키마 파일의 위치를 지정하는 태그이다. 이 부분은 기존의 TLD 파일을 복사하여 이용하자.

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

다음 function 태그는 JSP 파일에서 함수를 이용할 수 있도록 등록하는 태그로 그 내부에는 <description>, <name>, <function-class>, <function-signature> 태그 등이 필요하다. 이 태그는 각각 가음을 의미한다.

태크	기능	예
<description>	태그로 저장하여 이용할 함수의 설명	<description>Date 객체를 (yyyy-MM-dd(E) HH:mm:ss) 형태로 출력</description>
<name>	표현언어에서 직접 호출에 이용할 함수의 이름	<name>format</name>
<function-class>	함수가 소속된 클래스 이름	<function-class>form.ELDateFormat</function-class>
<function-signature>	함수의 반환유형, 이름, 인자유형인 시그네처	<function-signature>java.lang.String toFormat(java.util.Date)</function-signature>

태그 <function>은 지정된 클래스의 함수를 태그로 정의하는 태그로 그 내부에 <name>, <function-class>, <function-signature>태그를 사용한다.

```
<function>
    <description>Date 객체를 (yyyy-MM-dd(E) HH:mm:ss) 형태로 출력</description>
    ...
</function>
```

다음과 같이 태그 <name>으로 함수이름 format을 등록하며, 태그 <function-class>로 함수가 정의된 클래스 이름 form.ELDateFormat을 기술하고, 태그<function-signature>를 이용하여 함수이름 format으로 사용할 함수의 시그네처인 java.lang.String toFormat(java.util.Date)을 각각 기술한다

- ELfunction.tld

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
    <description>EL에서 함수실행</description>
    <tlib-version>1.0</tlib-version>
    <short-name>ELfunctions</short-name>
    <uri>/ELfunctions</uri>
    <function>
        <description>Date 객체를 (yyyy-MM-dd(E) HH:mm:ss) 형태로 출력</description>
        <name>format</name>
        <function-class>
            form.ELDateFormat
        </function-class>
        <function-signature>
            java.lang.String toFormat( java.util.Date )
        </function-signature>
    </function>
</taglib>
```

19.5.4 JSP 파일 작성

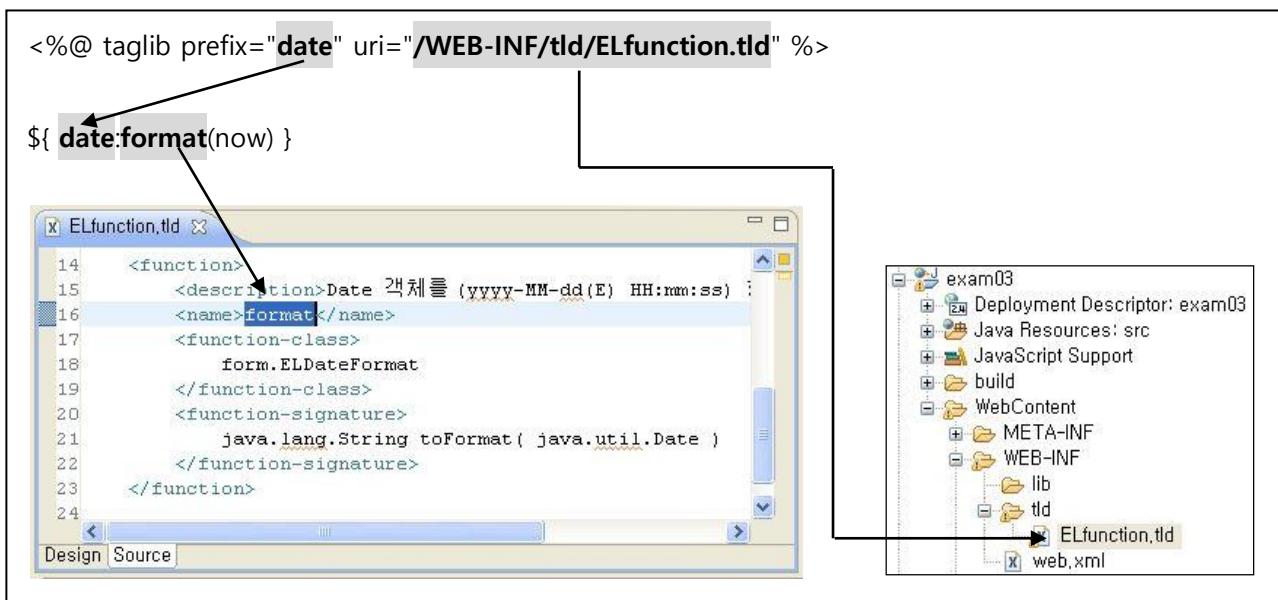
- 표현언어에서 함수호출

표현언어에서 등록한 태그의 함수를 호출하려면 가장 먼저 <taglib>태그를 이용하여 사용할 태그 접두어와 이용할 함수가 정의되어 있는 TLD 파일을 지정해야 한다. 이를 위하여 다음 구문을 이용한다.

```
<%@ taglib prefix="prefixname" uri="/WEB-INF/tld/ELfunction.tld" %>

${ prefixname:functionname() }
```

태그 <taglib>를 살펴보면 속성 prefix와 uri가 필요한데, 속성 **prefix**는 표현언어에서 기술할 태그명의 접두어를 나타내며, 속성 **uri**는 표현언어를 이용할 함수가 정의되어 있는 TLD 파일을 나타낸다.



표현언어에서 함수 호출시, 이용되는 함수이름은 클래스에 정의되어 있는 함수이름이 아니라, TLD 파일의 태그 <name>에 정의되어 있는 함수이름이다. TLD 파일 [ELfunction.tld]의 <function> 태그, 내부 태그 <name>에 정의된 함수이름이 format이므로 표현언어 \${ date:format(now) }으로 format함수를 호출한다. 즉 표현언어에서의 함수호출 문장은 다음과 같다.

\${ taglib의 속성prefix에서 지정한 태그명:TLD파일에서 지정한 함수이름(인자1, 인자2 ...) }

표현언어의 함수호출은 \${ date:format(now) }에서 사용되는 인자는 매개변수 now와 표현언어 내부 객체 sessionScope의 now이다. 물론 이 now들은 함수를 호출하기 이전에 request와 세션 속성으로 등록한 이름이다.

```
java.util.Date today = new java.util.Date();
request.setAttribute("now", today);
if(session.isNew()) session.setAttribute("now", today);
...
[Refresh]하면 현재 시간 : ${ date:format(now) }
처음 접속한 시간 : ${ date:format(sessionScope.now) }
```

표현언어에서 클래스에 지정한 함수를 호출하는 예제 function.jsp의 전체 소스와 결과이다. 예제 function.jsp를 실행하려면 이클립스에서 현재 있는 서버를 삭제하고 새로운 서버로 다시 실행하기 바란다.

- function.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>EL 함수</title>
```

```

</head>
<body>
<%@ taglib prefix="date" uri="/WEB-INF/tld/ELfunction.tld" %>
<%
    java.util.Date today = new java.util.Date();
    request.setAttribute("now", today);
    if ( session.isNew() ) session.setAttribute("now", today);
%>
<h2> EL 함수 예제 </h2>
[Refresh]하면 현재 시간 : ${ date:format(now) } <p>
처음 접속한 시간 : ${ date:format(sessionScope.now) } <p>
</body>
</html>

```

[결과]



19.6 표현언어 비활성화

19.6.1 표현언어 비활성화 방법

- 페이지 지시자 속성 isELIgnored

표현언어는 JSP 페이지 규약 2.0에 추가된 기능으로서 만일 JSP 규약 2.0 이전 버전에서 개발된 JSP 프로그램을 JSP 규약 2.0에서 실행한다면 \$로 시작하는 문자열을 표현언어로 인식하지 못하여 오류가 발생할 수 있다. 이러한 경우 대비해서 JSP는 JSP 페이지에서 표현언어를 사용하지 않겠다는 표현언어 비활성화 지시를 내릴 수 있다.

표현언어의 비활성화는 페이지 단위나 응용 프로그램 단위 또는 서버 단위로 가능하며 각각 다음과 같은 작업이 필요하다.

표현언어 비활성화 단위	수정 내용	수정파일
페이지 단위	페이지 지시자 속성 isELIgnored 추가	각 JSP 페이지
응용프로그램 단위	태그 <el-ignored>추가	[WEB-INF]/web.xml
서버 단위	태그 <el-ignored>추가	[conf]/web.xml

19.6.2 페이지에서 표현언어 비활성화

- 페이지 지시자 속성 isELIgnored

isELIgnored가 페이지의 표현언어 비활성화 관련 속성으로 "true"로 지정하면 표현언어가 비활성화되고, "false"로 지정하면 표현언어가 활성화된다. 속성 isELIgnored은 지정하지 않으면 기본값이 "false"로 표현언어가 활성화된다.

```
<%@ page isELIgnored="true" %>
```

즉 다음 JSP 페이지는 표현언어가 비활성화되어, \$로 기술된 부분이 그대로 \${1+1}으로 출력된다.

```
<%@ page isELIgnored="true" %>
${1 + 1}
```

반면 JSP 페이지에서는 표현언어가 활성화되어, \$로 기술된 부분이 표현언어로 인식되어 \${1+1}의 결과인 2가 출력된다.

```
<%@ page isELIgnored="false" %>
${1 + 1}
```

- inactiveEL.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>표현언어 비활성화</title>
</head>
<body>
    <%@ page isELIgnored="true" %>
    ${1 + 1}
</body>
</html>
```

[결과]



19.6.3 서버 또는 응용 프로그램에서 표현언어 비활성화

- 설정파일 web.xml에서 표현언어를 비활성화시키는 방법은 다음과 같이 태그 <JSP-config>의 내부 태그 <JSP-property-group>의 내부에서 <el-ignored> 태그를 삽입하여 그 값으로 true를 지정하는 방법이다.

```

<display-name>표현언어 비활성화</display-name>
<jsp-config>
  <JSP-property-group>
    <url-pattern>*.jsp</url-pattern>
    <el-ignored>true</el-ignored>
  </JSP-property-group>
</jsp-config>

```

서버 차원에서 표현언어를 비활성화시키려면 서버 설치 폴더의 하부 폴더 [conf]의 파일 web.xml을 수정하며 응용프로그램에서 표현언어를 비활성화시키려면 응용프로그램 폴더 [WEB-INF]하부의 web.xml을 수정하도록 한다.

설정파일 web.xml의 수정을 통하여 표현언어의 활성화와 비활성화를 테스트할 경우, web.xml의 수정뿐만 아니라 테스트하는 JSP 파일도 수정하여 다시 JSP 서블릿이 생성된 후 실행되도록 해야 web.xml에 수정된 표현언어의 활성화 설정이 JSP 페이지에 정확히 반영된다.

다음은 응용 프로그램에서 표현언어를 비활성화시키기 위해, 이클립스에서 응용프로그램폴더 [WEB-INF] 하부의 web.xml을 수정한다.

만일 web.xml과 JSP 페이지의 설정이 다르다면 JSP페이지 설정이 우선한다. 만일 서버 설정이 위와 같이 표현언어가 비활성화이고, JSP 페이지는 다음과 같이 표현언어가 활성화라면 다음 페이지의 표현언어는 활성화 된다.

```

<%@ page isELIgnored="false" %>
${ 1 + 1 }

```

만일 서버 설정이 위와 같이 표현언어가 비활성화이고, JSP 페이지에는 isELIgnored의 속성이 없다면 다음 페이지의 표현언어는 서버 설정에 따라 표현언어가 비활성화된다.

```

${ 1 + 1 }

```

다음표는 JSP 페이지 지시자의 isELIgnored 속성과 설정파일 web.xml의 태그 <el-ignored>설정에 따른 표현언어의 활성화와 비활성화의 결과를 나타낸 표이다.

페이지 지시자 속성 isELIgnored	web.xml의 태그 <el-ignored>	표현언어 사용
false	관계 없이	표현언어 활성화(0)
true	관계 없이	표현언어 비활성화(X)
-	false	표현언어 활성화(0)
-	true	표현언어 비활성화(X)
-	-	표현언어 활성화(0)

20. JSTL 개요

20.1.1 표준 커스텀 태그

- 액션 태그와 커스텀 태그

액션태그는 JSP에서 XML유형의 태그를 사용하며, 특별한 동작 기능을 수행하는 태그이다. 태그 <jsp:useBean ...>, <jsp:include ...>와 같이 이미 정해진 액션태그 외에, JSP에서는 프로그래머가 직접 필요한 태그를 만들어 사용할 수 있는 커스텀 태그(Custom Tag)를 제공한다.

액션태그와 커스텀 태그 모두 XML 태그이므로 다음과 같이 시작태그와 종료태그가 반드시 있어야 하며, 속성과 값을 지정하고, 값에는 반드시 "value", 'value'와 같이 큰따옴표 또는 작은 따옴표를 붙여야 한다. 또한 같은 태그 이름의 충돌을 피하기 위해 prefix인 접두어를 사용한다. 태그를 <prefix:tagName>로 시작했다면 몸체가 없는 태그 />이 종료 태그이며, 몸체가 있는 태그는 </prefix:tagName>이 종료 태그이다.

몸체(body)가 없는 태그 방식

```
<prefix:tagName var1="value" ... />
```

몸체(body)가 있는 태그 방식

```
<prefix:tagName var1="value" ... >  
    태그몸체(tag body)  
</prefix:tagName>
```

- JSTL(필요한 기능들을 모아놓은 커스텀 태그)

자바에서 커스텀 태그 기능을 이용하여 활용 빈도가 높은 태그를 개발, 발표한 것이 자바 표준 라이브러리(JSTL: Java Standard Tag Library)이다. 즉 JSTL은 표준 커스텀 태그라 할 수 있다.

JSTL은 여러 태그를 5가지 부류로 나누어 제공한다. 즉 JSTL은 변수지원, 반복, 조건과 같은 제어흐름, 주소인 URL 관리, 출력, 예외처리 등을 처리하는 Core 태그와 함께 XML, 국제화(Internationalization), SQL, 일반함수(functions)로 분류한 표준 태그 라이브러리의 집합이다. 다음은 JSTL이 제공하는 5가지 부류에 대한 태그의 세부영역과 태그에서 사용되는 접두어, 그리고 URI이다.

분류	세부영역	접두어(PREFIX)	URI
Core	변수지원	c	http://java.sun.com/jsp/jstl/core
	제어흐름		
	URL 관리		
	출력, 예외처리		
XML	코아	x	http://java.sun.com/jsp/jstl/xml
	흐름제어		
	변환		
Internationalization	지역화(Locale)	fmt	http://java.sun.com/jsp/jstl/fmt
	메시지포맷		
	수와 날짜 포맷		

Database	SQL	sql	http://java.sun.com/jsp/jstl/sql
Functions	집합체 길이	fn	http://java.sun.com/jsp/jstl/functions
	문자열 처리		

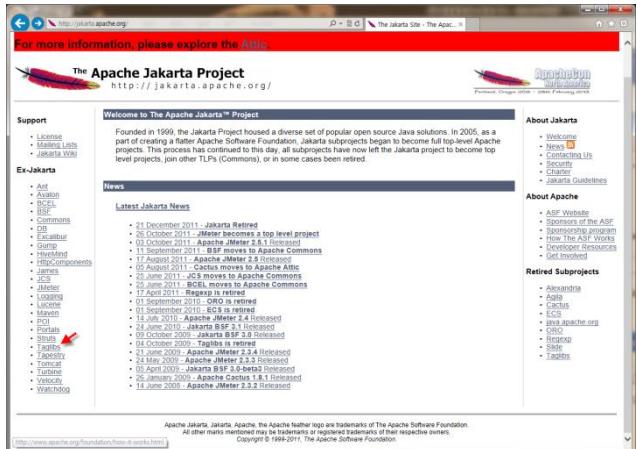
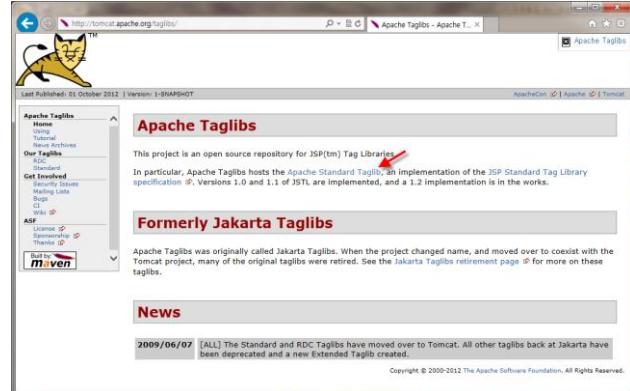
20.1.2 JSTL 사용을 위한 환경 설정

- JSTL

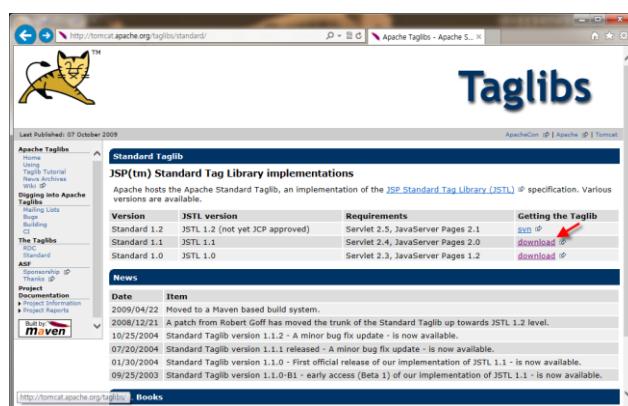
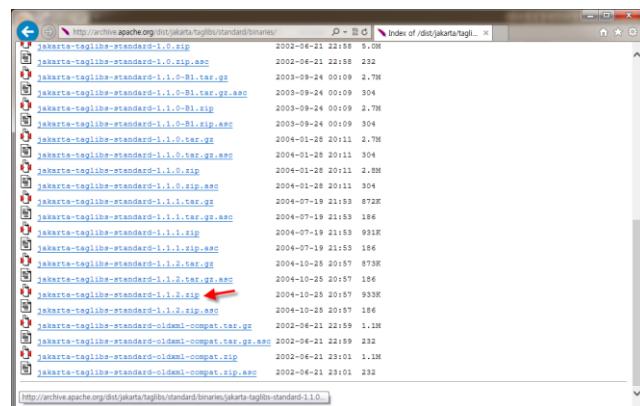
JSTL은 JSP 2.0의 표준 규약으로 제정되었으나, 톰캣 서버는 아직 JSTL의 표준 라이브러리를 제공하지 않으므로, 톰캣에서 JSTL을 사용하려면 JSTL 표준 라이브러리 파일인 2개의 파일을 톰캣 서버에 설치해야 한다.

- jstl.jar
- standard.jar

JSTL 표준 라이브러리를 내려 받기 위해, 먼저 자카르타 홈페이지 주소[Jakarta.apache.org]에 접속한다.

The left screenshot shows the main Apache Jakarta Project homepage with a sidebar containing links like Support, News, and Downloads. The right screenshot shows the Apache Taglibs homepage with a prominent 'Standard Taglib' section featuring a yellow cat logo and a brief description of the project's history and current status.

The left screenshot shows the 'Downloads' page for the Standard Taglib, listing versions 1.2 and 1.1.1 with download links. The right screenshot shows the 'Binaries' page, listing numerous files for versions 1.0 and 1.1, with a red arrow pointing to a file named 'jakarta-taglibs-standard-1.1.2.tag.gz'. Both screenshots show the Apache Jakarta Project header and footer.

접속한 자카르타 홈페이지 좌측 메뉴에서 [Downloads]로 접속한 후, 다시 중앙에 위치한 링크 [Taglibs]로 접속한다. 접속한 [Taglibs Downloads]화면에서 다시 링크 [Standard 1.1 Taglib]를 누르면 JSTL 라이브러리를 내려 받을 수 있는 페이지에 접속된다. 접속한 페이지인 [Standard 1.1 Taglib Downloads]는 다음 주소로 바로 접속할 수 있다.

<http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/>

Apache/2.3.15-dev (Unix) mod_ssl/2.3.15-dev OpenSSL/1.0.0c Server at archive.apache.org Port 80

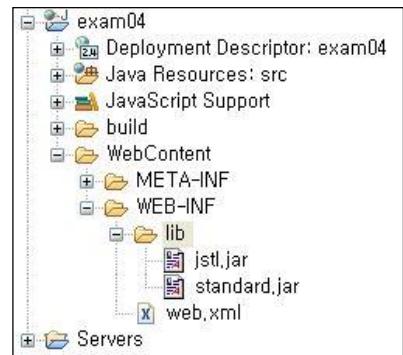
[jakarta-taglibs-standard-1.1.2.zip]를 적당한 폴더로 내려 받는다. JSTL 1.1에 관련된 소스와 자료를 보려면 링크 [Source]의 [1.1.2-src.zip]을 내려 받아 사용할 수 있다.

내려 받은 다음 파일에서 압축을 풀어, 하부 폴더 lib에 있는 2개의 파일 [jstl.jar]와 [standard.jar]를 준비한다.

JSTL을 위한 2개의 파일 [jstl.jar]와 [standard.jar]은 각각 JSTL API classes, JSTL implementation classes을 위한 파일이다.

이름	설명	파일이름
JSTL API classes	JSTL API 클래스	jstl.jar
JSTL implementation classes	Standard Taglib JSTL 구현 클래스	standard.jar

이 파일들을 톰캣의 라이브러리 폴더와 사용할 이클립스 프로젝트의 라이브러리 폴더에 넣어주어야 한다. 이렇게 두파일을 복사하면 JSTL을 사용할 준비가 끝난 것이다. 톰캣 폴더의 [common]/[lib]폴더에 복사하여야 한다. 톰캣을 사용하는 현재 이클립스 프로젝트에서 JSTL 태그를 사용하려면 JSTL 라이브러리 파일 [jstl.jar]와 [standard.jar]를 현재 프로젝트 폴더 하부 [WEB-INF]/[lib]에 복사한다.



20.1.3 JSTL 태그의 사용

- taglib 지시자

JSP 페이지에서 커스텀 태그의 한 종류인 JSTL 태그를 사용하려면 taglib 지시자를 사용해야 한다. 즉 **taglib 지시자를 통해 사용할 커스텀 태그 라이브러리의 식별자를 속성 uri(태그 라이브러리가 존재하는 위치를 의미)에 지정하며, 사용할 태그의 접두어를 속성 prefix에 지정한다.**

속성	필수	자료유형	기능
prefix	o	String	태그에서 사용될 접두어
uri	o	String	사용할 커스텀 태그 라이브러리의 식별자

예를 들어, Core 기능으로 분류된 태그 out을 이용하려면 다음과 같은 taglib 지시자에서 태그 식별자인 uri를 [http://java.sun.com/jsp/JSTL/core]로 지정해야 한다.

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

위에서 prefix를 c로 지정했으므로 태그 out은 <prefix:tagName ...> 형태인 <c:out ...>으로 사용한다. 속성 prefix는 다른 것으로 지정할 수 있으나 표준 태그이므로 가능하면 추천하는 접두어 c를 이용하자.

```
<c:out value="Hello JSTL!!!" />
```

태그 out은 속성 value에 지정한 값을 브라우저에 출력하는 간단한 태그이다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL 첫 예제 : firstJSTL.jsp</title>
</head>
<body>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:out value="Hello JSTL!!!!" />
</body>
</html>
```

[결과]



JSTL의 기능 분류에 따른 taglib의 지시자와 추천 접두어인 prefix를 정리하면 다음과 같다.

기능분류	TAGLIB 지시자	PREFIX	태그예
Core	<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>	c	<c:tagname ...>
XML	<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>	x	<x:tagname ...>
Internationalization	<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>	fmt	<fmt:tagname ...>
Database	<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>	sql	<sql:tagname ...>
Functions	<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>	fn	\${fn:functionName(...)}

20.2 코어 태그 라이브러리

20.2.1 코어 태그 개요(JSTL에서 기본적인 기능들을 구현해 놓은 라이브러리)

- 기본 기능 태그

코어 태그는 JSTL에서 가장 기본이 되며 가장 많이 사용되는 태그로, 변수 지원, 제어흐름, URL 관리, 예외처리, 출력처리를 위한 태그로 구성된다.

분류	태그	기능
변수지원	remove	이미 설정한 변수를 삭제
	set	범위에서 사용될 변수를 지정
제어흐름	choose	태그 <when>과 <otherwise>로 구성되어 있는 여러 개의 조건 중에 하나만 선정하여 처리
	when	<choose>태그의 서브태그로 조건이 true이면 몸체를 실행
	otherwise	<choose>태그의 서브태그로 이전에 있는 태그 <when>에서 조건이 모두 false이면 태그 <otherwise> 몸체를 실행
	forEach	다양한 콜렉션 유형에서 반복을 처리
	forTokens	문자열을 구분자로 구분하여 토큰으로 나누며 반복 실행
	if	조건이 true이면 몸체를 실행
URL 관리	import	다른 페이지를 현재 위치, 또는 변수 또는 읽기 객체에 저장
	param	태그 <import>, <redirect>, <url>의 서브태그로, 매개변수 전송 처리
	redirect	새로운 URL로 이동 처리
	url	질의 매개변수를 이용하여 URL을 생성
예외처리, 출력	catch	예외처리
	out	출력처리

코어 태그를 이용하기 위해서는 사용하기 이전에 다음의 taglib 지시자를 사용한다.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="foo" value="set Tag 테스트입니다." />
```

20.2.2 변수 지원 태그 set, remove

- <c:set ...>

태그 set은 지정된 범위로 평가 값을 변수에 저장한다. 태그 set의 구문은 다음과 같으며 몸체가 있는 태그를 사용할 경우에 눈 몸체의 값이 속성 value에 지정되는 값이 된다.

```
<c:set var="변수이름" value="저장할 값" scope="4개 중의 하나" />
<c:set var="변수이름" scope="4개 중의 하나">
저장할 값
</c:set>
- 변수 값 설정
<c:set var="변수이름" scope="영역객체" value="저장할 값" />
<c:set var="변수이름" scope="영역객체">저장할 값</c:set>
-특정 EL 변수의 프로퍼티값 설정
```

```
<c:set target="대상" property="프로퍼티이름" value="값" />
<c:set target="대상" property="프로퍼티이름">저장할 값</c:set>
```

다음은 태그 set의 속성에 대한 설명이다. 속성 var와 target은 동시에 사용될 수 없으며, 저장할 객체인 target이 사용되면 속성 property도 사용되어 target 객체의 property 이름을 지정한다.

속성	필수	기본값	자료유형	기능
var	X		String	값이 저장되는 변수이름
target	X		String	값이 저장되는 자바빈즈 객체이거나 또는 Map 객체, 자바 빈즈 객체인 경우, setter인 property에 의해 값이 지정.
value	X		String	변수 또는 객체에 저장할 값
property	X		String	target 객체의 property 이름
scope	X	page	String	변수가 효력을 발휘하는 영역으로 page, request, session, application 중의 하나를 지정

다음 태그는 변수 foo에 문자열 "set Tag 테스트입니다."를 저장하며, 변수 foo는 표현언어에서 변수로 사용할 수 있다.

```
<c:set var="foo" value="set Tag 테스트입니다." />
${foo} = ${foo}
```

다음 set 태그는 변수 n의 몸체에 지정된 정수24를 저장한다.

```
<c:set var="n">
24
</c:set>
```

변수 d에 실수 31.54를 저장하면, 위에서 지정한 변수 n과 함께 표현언어에서 산술연산이 가능하다.

```
<c:set var="d">
31.54
</c:set>
${d} = ${d}
${n+d} = ${n+d}
```

다음과 같이 변수 b에 논리 값도 저장이 가능하다.

```
<c:set var="b" value="true" />
${!b} = ${!b}
```

태그 set에서 변수의 범위를 page, request, session, application 중 하나를 지정할 수 있으며, 지정하지 않으면 기본값은 page이다. 만일 scope를 session으로 지정하면 표현언어 내장 객체 sessionScope로도 참조가 가능하다.

```

<c:set var="str" value="Hello set Tag!!!" scope="session" />
${str} = ${str}
${sessionScope.str} = ${sessionScope.str}

```

- setvar.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: set</title>
</head>
<body>
<h2>JSTL Core Tag: set</h2>
<c:set var="foo" value="set Tag 테스트입니다." />
${foo} = ${foo} <br>
<c:set var="n" >
24
</c:set>
${n} = ${n} <br>
<c:set var="d" >
31.54
</c:set>
${d} = ${d} <br>
${n + d} = ${n + d} <br>
<c:set var="b" value="true" />
${!b} = ${!b} <br>
<c:set var="str" value="Hello set Tag!!!" scope="session" />
${str} = ${str} <br>
${sessionScope.str} = ${sessionScope.str} <br>
</body>
</html>

```

[결과]



속성 target을 이용하는 set 태그를 살펴보면, Map 또는 자바빈즈 객체에 속성 property로 value에 지정된 값을 저장한다. 다음은 클래스 java.util.HashMap()으로 선언된 변수 book에 <키, 값>으로 각각 <"java","자바로 배우는 프로그래밍 기초">, <"c","c로 배우는 프로그래밍 기초">, <"jsp","jsp로 배우는 웹 프로그래밍 기초">를 저장하는 태그이다. (target은 값을 설정할 프로퍼티에 대한 객체를 의미. property는 값을 설정할 객체의 프로퍼티를 의미)

```
<c:set var="book" value="<% new java.util.HashMap() %>" />
<c:set target="${book}" property="java" value="자바로 배우는 프로그래밍 기초" />
<c:set target="${book}" property="c" value="C로 배우는 프로그래밍 기초" />
<c:set target="${book}" property="jsp" value="JSP로 배우는 웹프로그래밍 기초" />
```

위의 변수 book에서 키 값 java에 대응하는 값을 참조하려면 \${book.java}를 사용한다.

```
W${book.java} = ${book.java}
```

다음은 자바빈즈 객체 oneday에서 setter인 year를 2009으로 지정하는 set태그이다.

다음과 같이 태그 set에서 target이 자바빈즈 객체 oneday라면, 속성 property를 "year"로 지정하고 value를 "2009"으로 지정하는 것이 oneday.setYear(2009)를 호출하는 것을 의미한다.

```
<jsp:useBean id="oneday" class="java.util.Date" />
<c:set target="${oneday}" property="year" value="2009" />
```

```
<% Member member = new Member();%>
<%-- member.setName("홍길동"); --%>
<c:set target="<%member%>" property="name" value="홍길동" />
<%=member.getName()%>
<c:set var="m" value="<%member%>" />
<%-- member.setName("김영희"); --%>
<c:set target="${m}" property="name" value="김영희" />
${m.name}
<% Map<String, String> book = new HashMap<String, String>(); %>
<%-- subject.put("java","자바 프로그래밍")--%>
<c:set target="<%book%>" property="java" value="자바프로그래밍" />
<%=book.get("java")%>
```

※ target 대상이 EL 변수인 경우 target 속성의 값을 \${ }와 같이 EL을 이용해서 설정해 주어야 한다.

- settarget.jsp

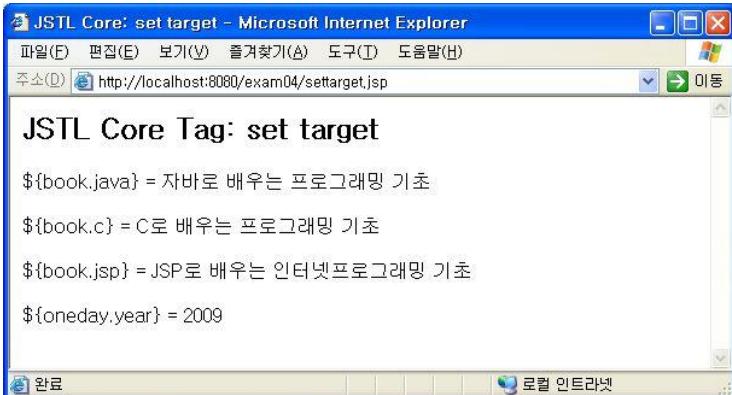
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: set target</title>
</head>
```

```

<body>
<h2>JSTL Core Tag: set target</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="book" value="<% new java.util.HashMap<String, String>() %>" />
<c:set target="${book}" property="java" value="자바로 배우는 프로그래밍 기초" />
<c:set target="${book}" property="c" value="C로 배우는 프로그래밍 기초" />
<c:set target="${book}" property="jsp" value="JSP로 배우는 인터넷프로그래밍 기초" />
${book.java} = ${book.java} <p>
${book.c} = ${book.c} <p>
${book.jsp} = ${book.jsp} <p>
<jsp:useBean id="oneday" class="java.util.Date" />
<c:set target="${oneday}" property="year" value="2009" />
${oneday.year} = ${oneday.year} <p>
</body>
</html>

```

[결과]



- <c:remove ...>

태그 remove는 이미 사용하는 변수를 삭제하는 태그이다. 태그 remove는 다음과 같이 몸체가 없는 태그만을 사용한다.

```
<c:remove var="삭제할 변수이름" scope="삭제할 변수의 scope" />
```

다음은 태그 remove의 속성에 대한 설명으로, 속성 var는 삭제할 변수이름으로 반드시 사용되어야 한다. 속성 scope를 기술한 경우, 삭제할 변수의 scope와 다르면 삭제할 수 없다. 그러나 scope를 아예 지정하지 않으면 scope에 관계없이 지정된 변수가 삭제된다.

속성	필수	기본값	자료유형	기능
var	O		String	삭제할 변수 이름
scope	X		String	삭제할 변수의 영역으로 page, request, session, application 중의 하나를 지정. 지정하지 않으면 scope에 관계없이 변수를 삭제.

다음은 속성 scope가 session으로 지정된 변수 str를 삭제하는 태그로, 삭제 이후 출력하면 아무것도 표시되지 않는다.

```
<c:remove var="str" scope="session" />  
₩${str} = ${str}
```

다음과 같이 속성 scope를 삭제할 변수의 scope와 다르게 지정하면, 변수 str를 삭제할 수 없다.

```
<c:remove var="str" scope="page" />
```

태그 remove에서 scope를 아예 지정하지 않으면 scope에 관계없이 지정된 변수가 삭제된다.

```
<c:set var="app" value="응용프로그램변수" scope="application" />  
<c:remove var="app" />
```

- remove.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head><meta charset="UTF-8">  
<title>JSTL Core: remove</title>  
</head>  
<body>  
<h2>JSTL Core Tag: remove</h2>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<c:set var="str" value="Hello set Tag!!!" scope="session" />  
₩${str} = ${str} <br>  
₩${sessionScope.str} = ${sessionScope.str} <br>  
<c:remove var="str" scope="page" />  
₩${str} = ${str} <br>  
<c:remove var="str" scope="session" />  
₩${str} = ${str} <br>  
<c:set var="app" value="응용프로그램변수" scope="application" />  
₩${app} = ${app} <br>  
<c:remove var="app" />  
₩${app} = ${app}  
</body>  
</html>
```

[결과]



20.2.3 제어흐름 태그

- <c:if ...>

태그 if는 속성 test를 평가하여 그 결과가 true이면 몸체를 실행하는 태그로, 자바의 if문과 같이 조건을 처리하는 태그이다. 속성 test에 지정하는 구문은 평가값이 논리값이여야 하며, 일반적으로 표현언어의 조건식을 기술하고, 속성 var에는 평가 결과인 true 또는 false가 저장된다. 태그 if는 일반 언어의 if-else구문은 지원하지 않는다.

```
<c:if test="${num mod 2 ==0 }" var="bool">  
 짹수  
</c:if>
```

다음은 태그 if의 속성에 대한 설명으로 속성 scope는 변수 var의 범위 지정에 사용된다.

속성	필수	기본값	자료유형	기능
test	O		boolean	참 거짓을 판별하여 참이면 몸체를 실행
var	X		String	test의 결과를 저장
scope	X		String	변수 var의 범위로 page, request, session, application 중의 하나를 지정.

- if.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head><meta charset="UTF-8">  
<title>JSTL Core: if</title>  
</head>  
<body>  
<h2>JSTL Core Tag: if</h2>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
안녕하세요.  
<c:set var="today" value="<% new java.util.Date()%>" />  
<c:if test="${today.hours > 17 }" var="bool">  
저녁 식사는 하셨는지요?<br>  
</c:if>  
<p>  
<hr>  
₩${today.hours} = ${today.hours} <br>  
₩${bool} = ${bool}  
</body>  
</html>
```

[결과]



- <c:choose>

자바의 switch문장과 같이 태그 choose는 여러 개의 내부 태그 when의 속성 test를 만족하면 그 몸체를 처리하는 태그이다. 태그 choose는 순서대로 태그 when을 검사하여 속성 test가 true인 첫 태그 when의 몸체를 실행하고 종료하는데, 태그 when의 조건이 모두 false라 하나도 처리하지 않으면 태그 otherwise의 몸체를 실행한다. 태그 choose에서 내부 태그 otherwise는 없을 수도 있다.

```
<c:choose>
  <c:when test="${today.hours < 12 }" >
    Good morning!
  </c:when>
  <c:when test="${today.hours < 18 }" >
    Good afternoon!
  </c:when>
  <c:otherwise>
    Good evening!
  </c:otherwise>
</c:choose>
```

다음은 태그 when의 속성에 대한 설명으로 태그 choose, when, otherwise에서 태그 when만이 단지 1개의 속성 test를 갖는다.

속성	필수	기본값	자료유형	기능
test	O		boolean	태그 when의 몸체를 실행할 논리값을 검사, true면 몸체를 실행

- choose.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: choose</title>
</head>
```

```

<body>
<h2>JSTL Core Tag: choose</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="today" value="<%= new java.util.Date()%>" />
<c:choose>
  <c:when test="${today.hours < 12 }" >
    Good morning!
  </c:when>
  <c:when test="${today.hours < 18 }" >
    Good afternoon!
  </c:when>
  <c:otherwise>
    Good evening!
  </c:otherwise>
</c:choose>
<hr>학점 :
<c:set var="jumsu" value="95" />
<c:choose>
  <c:when test="${jumsu >=90 }" >
    A
  </c:when>
  <c:when test="${jumsu >=80 }" >
    B
  </c:when>
  <c:when test="${jumsu >=70 }" >
    C
  </c:when>
  <c:when test="${jumsu >=60 }" >
    D
  </c:when>
  <c:otherwise>
    F
  </c:otherwise>
</c:choose>
</body>
</html>

```

[결과]



- <c:forEach ...>

태그 forEach는 배열, Collection, Map에 저장된 원소를 순차적으로 처리하거나 지정하는 횟수만큼 반복을 처리하는 태그이다. 태그 forEach는 다음과 같이 몸체가 있는 태그를 사용하며, items에 지정된 집합체에서 순서대로 한 원소를 속성 var에 저장하며 몸체에서 반복 업무를 수행한다.

```
<c:forEach var="한 원소를 저장하는 변수" items="배열 또는 Map등의 집합체" >
    body
</c:forEach>
```

자바 for문과 같이, 태그 forEach의 속성 begin, end를 이용하여 반복의 시작과 끝을 지정할 수 있다. 또한 속성 step으로 다음 반복의 증加分을 지정할 수 있다. 물론 step이 사용되지 않으면 기본으로 1씩 증가한다. 그러므로 다음은 i가 3부터 시작하여 6,9,... 99까지 반복하는 태그이다.

```
<c:forEach var="i" begin="3" end="100" step="3" >
    body
</c:forEach>
```

태그 forEach에서 다음과 같이 Map 유형의 객체를 속성 items에 저장하면, Map의 한 원소가 지정된 변수 i를 이용하여 i.key와 i.value로 <키, 값>을 참조할 수 있다.

```
<c:forEach var="i" items="${sessionScope}" >
    ${i.key} = ${i.value}
</c:forEach>
```

다음은 태그 forEach의 속성에 대한 설명으로, 속성 items가 배열인 경우, 첨자를 속성 begin, end, step으로 사용할 수 있다. 속성 items가 Iterator, Enumeration, Map 등과 같은 집합체라면, 순서를 알 수 없으므로 begin, end, step은 사용하지 않는 것이 일반적이다.

속성	필수	기본값	자료유형	기능
var	X		String	반복에서 집합체의 현재 원소의 값
items	X		Object	반복을 실행하는 집합체
begin	X	O	int	속성 items가 있으면 반복을 실행하는 항목의 첨자이며 속성 items가 없으면 기술된 첨자로 반복

				을 시작
end	X	집합체 지정값	int	속성 items가 있으면 반복을 종료하는 항목의 첨자이며, 속성 items가 없으면 기술된 첨자로 반복을 종료
step	X	1	int	속성 items가 있으면 반복을 실행하는 항목의 첨자이며 속성 items가 없으면 기술된 참자로 반복을 시작
varStatus	X		String	반복 상태값

다음은 배열 scope의 각각의 원소값을 출력하고 변수 sum에 모두 더하는 모듈이다.

```
<c:set var="score" value="<% new int[] {95, 88, 77, 45, 99} %>" />
<c:forEach var="point" items="${score}" >
    ${point} = ${point} <br>
    <c:set var="sum" value="${sum + point}" />
</c:forEach>
```

다음은 변수 sum에 1부터 100까지의 합을 저장하는 모듈이다.

```
<c:set var="sum" value="0" />
<c:forEach var="i" begin="1" end="100" >
    <c:set var="sum" value="${sum + i}" />
</c:forEach>
```

- foreach.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: forEach</title>
</head>
<body>
<h2>JSTL Core Tag: forEach</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<h3>배열 처리</h3>
<c:set var="score" value="<% new int[] {95, 88, 77, 45, 99} %>" />
<c:forEach var="point" items="${score}" >
    ${point} = ${point} <br>
    <c:set var="sum" value="${sum + point}" />
</c:forEach>
합 = ${sum} <br>
```

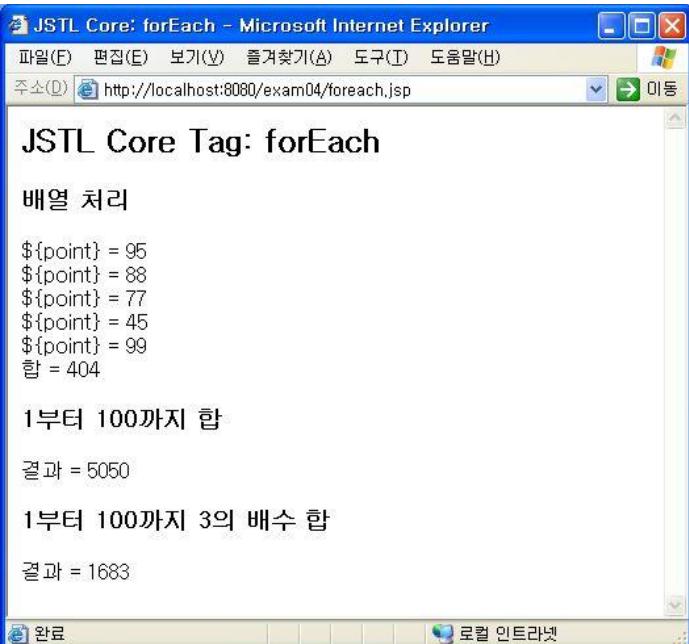
```

<h3>1부터 100까지 합</h3>
<c:set var="sum" value="0" />
<c:forEach var="i" begin="1" end="100" >
    <c:set var="sum" value="${sum + i}" />
</c:forEach>
결과 = ${sum}

<h3>1부터 100까지 3의 배수 합</h3>
<c:set var="sum" value="0" />
<c:forEach var="i" begin="3" end="100" step="3" >
    <c:set var="sum" value="${sum + i}" />
</c:forEach>
결과 = ${sum}
</body>
</html>

```

[결과]



- <c:forTokens ...>

태그 forTokens는 속성 delims에 지정된 구분자를 사용하여, 속성 items에 지정된 배열을 토큰을 사용해 반복적으로 나누는 작업을 처리하는 태그이다. 그러므로 태그 forEach에서 속성 items와 delims는 반드시 있어야 한다. 태그 forTokens는 자바 클래스 java.util.StringTokenizer와 같은 기능을 수행한다.

```

<c:forTokens var="구분자로 잘라진 token이 저장" delims="각문자가 구분자로 이용" items="토큰으로
나눌 문자열">
    body
    ${var}
</c:forTokens>

```

속성	필수	기본값	자료유형	기능
var	X		String	토큰을 추출하기 위한 문자열에서 현재 추출된 토큰
items	O		Object	토큰을 추출하기 위한 문자열
delims	O		String	문자열을 토큰으로 구분할 문자로 구성된 구분자의 집합
begin	X	O	int	지정한 첨자로 분리된 토큰으로 반복을 시작, 시작은 0
end	X	집합체 지정값	int	지정한 첨자로 분리된 토큰으로 반복을 종료
step	X	1	int	반복을 지정한 값만큼 증가시켜 반복
varStatus	X		String	반복 상태 값

다음은 주어진 str 문장에서 토큰을 추출해 출력하는 모듈이다.

```
<c:set var="str" value="JSTL은 표준태그로서 코어, XML, 국제화, SQL, 함수 관련 태그로 구성된다."/>

<c:forTokens var="token" delims=".," items="${str}">
${token}
</c:forTokens>
```

구분자는 ".,"로 각각의 문자인 [], [.,], [.,[은]], [로],[서],[된],[다] 8개가 토큰을 나누는 구분자로 이용된다. 그러므로 주어진 문장에서 밑줄 친 부분이 구분자이고, 토큰은 왼쪽부터 나머지 단어 순서로 추출된다.

JSTL은 표준태그로서 코어, XML, 국제화, SQL, 함수 관련 태그로 구성된다.

- fortokens.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: forTokens</title>
</head>
<body>
<h2>JSTL Core Tag: forTokens</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="str" value="JSTL은 표준태그로서 코어, XML, 국제화, SQL, 함수 관련 태그로 구성된다."/>
${str} <p><hr>
위 문장은 forTokens의 속성 delims=".," 지정으로 다음 단어로 나뉘어진다.<hr><p>
<c:forTokens var="token" delims=".," items="${str}">
<font color="blue"> ${token}</font>
</c:forTokens>
```

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

[결과]



20.2.4 URL 관리 태그

URL 관련 태그는 외부/내부 페이지를 현재 위치에 삽입(<c:import>태그), URL생성(<c:url>태그), 지정한 페이지로 이동(<c:redirect>태그) 세 가지 기능을 제공한다. 이 태그를 이용하면 스크립트릿이나 표현식을 사용하지 않고 간결한 코드로 URL 관련 기능을 수행할 수 있다.

- <c:import ...>(지정된 URL을 태그가 사용된 JSP 페이지에 출력시키는 기능)

<c:import>태그는 특정 URL의 결과를 읽어와 현재 위치에 삽입하거나 EL 변수에 저장할 때 사용된다.

<jsp:include ...>는 동일한 웹 어플리케이션 내에 위치한 자원을 포함해 주는 기능이라면, <c:import>태그는 동일한 웹 어플리케이션뿐만 아니라 외부의 다른 자원을 읽어와 포함시킬 수 있도록 해준다.

다음과 같이 변수 var(리소스가 저장될 변수명)를 사용하지 않으면 몸체의 존재 유무와 상관없이 바로 URL의 페이지가 삽입된다.

```
<c:import url="내부 또는 외부 URL 모두 지원" />  
  
<c:import url="내부 또는 외부 URL 모두 지원" >  
</c:import>
```

url 속성에는 다음과 같이 절대 URL과 상대 URL의 두 가지 타입의 값을 입력할 수 있다

- 절대 URL: http://javastudy.com/java와 같은 완전한 URL
- 상대 URL

웹 어플리케이션 내에서의 절대 경로: 슬래스로 시작. 예)/view/list.jsp

현재 JSP에 대한 상대 경로: 슬래스로 시작하지 않음. 예)..../view/list.jsp

절대 URL일 경우 <c:import>는 java.net.URL과 java.netURLConnection을 이용해서 데이터를 읽어온다.

상대 URL을 입력한 경우 <c:import>태그는 <jsp:include ...>액션 태그와 동일한 방식으로 동작한다. 즉 동일한 웹 어플리케이션에 속한 자원의 실행결과를 읽어온다. 차이점이 있다면 <jsp:include ...> 액션 태그는 현재 위치에 무조건 결과를 출력하는 반면, <c:import>태그는 EL 변수에 보관한 뒤 필요에 따라

알맞은 처리를 할 수 있다는 점이다.

다음과 같이 속성 var를 사용하면 지정한 변수에 URL의 내용이 저장되므로, 변수를 출력하면 URL의 페이지가 삽입된다.

```
<c:import url="내부 또는 외부 URL 모두 지원" var="변수명" />

<c:import url="choose.jsp" var="choose" />
${choose}
```

다음은 태그 import의 속성에 대한 설명으로, 속성 context를 이용하여 다른 응용 프로그램 페이지도 삽입할 수 있으며, 속성 var를 사용하면 포함하고자 하는 URL 자원 내용을 문자열 형태로 저장하고, 속성 varReader를 사용하면 URL 자원 내용을 Reader 객체로 포함한다. 그러므로 속성 var와 varReader는 함께 명시되지 않는다.

속성	필수	기본값	자료유형	기능
url	O		String	포함시킬 URL
context	X	현재응용 프로그램	String	현재 응용프로그램 컨텍스트 이름
charEncoding	X	ISO- 8859-1	String	현재 페이지 내에 포함시킬 인코딩 방법
var	X		String	포함될 페이지의 내용이 저장되는 변수
scope	X	page	String	변수 var의 범위로 page, request, session, application 중의 하나를 지정
varReader	X		String	자원 내용을 읽기 위한 변수로 java.io.Reader 객체로 읽어올 때 사용한다.

요청 파라미터를 추가하는 방식은 다음의 두 가지 방식이 있다.

URL에 직접 입력

<c:param> 태그를 이용해서 입력

태그 import에서 내부 태그 param을 사용하여 URL 페이지에 매개변수를 전달할 수 있다. 다음은 URL 페이지 paramhandle.jsp에 매개변수 user로 kang을 전송하여 그 결과가 화면에 보이는 기능을 수행하는 태그이다.

```
<c:import url="paramhandle.jsp" >
    <c:param name="user" value="king" />
</c:import>
```

태그 param은 태그 import와 redirect, URL의 서브 태그로 사용되는 태그로서 속성 name과 value를 사용한다.

속성	필수	기본값	자료유형	기능
name	O		String	매개변수의 이름
value	O		String	매개변수 이름에 대한 값

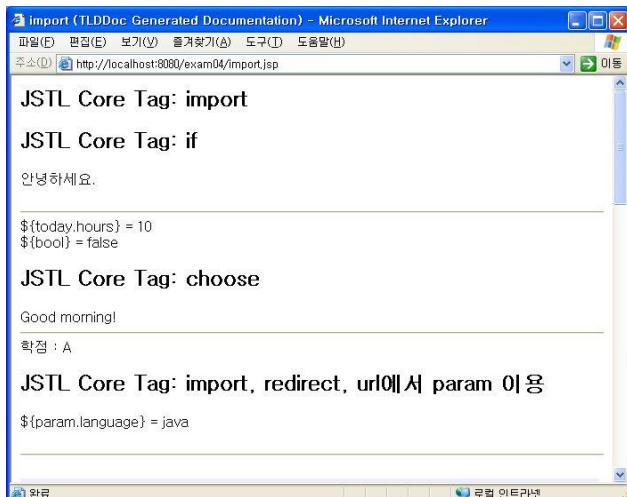
다음은 태그 import로 [JSTL 라이브러리] 설명 페이지를 접속하여 화면에 표시하는 구문이다.

```
<c:import url="http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/c/import.html" var="java" />
${java}
```

- import.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: import</title>
</head>
<body>
<h2>JSTL Core Tag: import</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="/if.jsp" />
<c:import url="choose.jsp" var="choose" />
${choose}
<c:import url="paramhandle.jsp" >
    <c:param name="language" value="java" />
</c:import>
<p><hr><p>
<c:import url="http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/c/import.html" var="java" />
${java}
</body>
</html>
```

[결과]



- paramhandle.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: param</title>
</head>
<body>
<h2>JSTL Core Tag: import, redirect, url에서 param 이용</h2>
<${param.language} = ${param.language} <p>
</body>
</html>
```

- <c:redirect ...>

태그 redirect는 속성 URL에 지정된 페이지로 이동시키는 태그이다. 그러므로 태그 redirect는 내장 객체를 이용한 response.sendRedirect()와 액션태그 <jsp:forward ...> 기능과 유사하나, 실제로 태그 redirect를 이용하면 브라우저의 주소 줄에 import의 URL에 기술한 페이지가 표시되므로 <jsp:forward ...> 액션태그와 같은 기능을 수행한다.

```
<c:redirect url="paramhandle.jsp" />
```

내부 태그 param을 이용하여 매개변수를 전송할 경우, 몸체가 있는 태그를 이용한다.

```
<c:redirect url="paramhandle.jsp">
    <c:param name="language" value="oracle" />
</c:redirect>
```

다음은 태그 redirect의 속성에 대한 설명으로, 속성 context를 이용하여 다른 응용 프로그램으로 이동이 가능하다.

속성	필수	기본값	자료유형	기능
url	X		String	기본 URL
context	X	현재응용 프로그램	String	웹 응용 프로그램의 컨텍스트 이름

- redirect.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: redirect</title>
</head>
```

```

<body>
<h2>JSTL Core Tag: redirect</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:redirect url="paramhandle.jsp">
    <c:param name="language" value="oracle" />
</c:redirect>
</body>
</html>

```

[결과]



- <c:url ...>

태그 URL은 속성 value에 지정한 페이지를 속성 var에 저장하는 태그이다.

```
<c:url var="ph1" value="paramhandle.jsp" />
```

내부 태그 param을 이용하여 매개변수를 지정하면, 변수에 매개변수가 포함된 URL 값이 저장된다.

```
<c:url var="ph2" value="paramhandle.jsp" >
    <c:param name="language" value="jsp" />
</c:url>
```

다음은 태그 URL의 속성에 대한 설명으로, 속성 context를 이용하여 다른 응용 프로그램 페이지로 이동이 가능하다. 다음 소스로 변수 site에는 /exam03/userEL.jsp가 저장된다.

```
<c:url var="site" value="/userEL.jsp" context="/exam03" />
```

속성	필수	기본값	자료유형	기능
value	O		String	URL 페이지
context	X	현재응용 프로그램	String	웹 응용 프로그램의 컨텍스트 이름
var	X		String	value의 값을 가지는 변수
scope	X		String	변수 var의 범위로 page, request, session, application 중의 하나를 지정

- url.jsp

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

```

<html>
<head><meta charset="UTF-8">
<title>JSTL Core: url</title>
</head>
<body>
<h2>JSTL Core Tag: url</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url var="ph1" value="paramhandle.jsp" />
<c:url var="ph2" value="paramhandle.jsp" >
    <c:param name="language" value="jsp" />
</c:url>
${ph1} = ${ph1}<p>
<a href="${ph2}">${ph2}</a>
<p>
<c:url var="site" value="/userEL.jsp" context="/exam03" />
<a href="${site}">${site}</a>
</body>
</html>

```

[결과]



20.2.5 출력과 예외처리 태그

- <c:catch ...>

태그 catch는 몸체 부분에 예외가 발생할 가능성이 있는 코드를 배치하여, 예외가 발생하면 지정한 속성 var 변수에 예외 메시지를 저장하는 태그이다.

```

<c:catch var="errMessage">
    예외가 발생할 수 있는 코드
</c:catch>

```

속성	필수	기본값	자료유형	기능
var	O		String	예외 메시지를 저장할 변수

- catch.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: catch</title>
</head>
<body>
<h2>JSTL Core Tag: catch</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:catch var="errMessage">
    <%= 2/0 %>
</c:catch>
<p>
<c:if test="${!empty errMessage}">
예외가 발생하였습니다. 예외 메시지 : <hr>
${errMessage}
</c:if>
</body>
</html>
```

[결과]



- <c:out ...>

태그 out은 속성 value에 지정된 문자열 또는 변수의 내용을 출력하는 태그이다.

```
<c:out value="출력 할 내용" />
```

태그 out에서 속성 value에 지정된 값이 없을 때는 몸체에 있는 값이 기본값이 되어 출력된다.

```
<c:out value="${param.name}">
    이 부분은 value에 값이 null일 때 출력되는 기본 출력 값이다.
</c:out>
```

태그 out에서 속성 default도 위와 같이, 속성 value에 지정된 값이 없을 경우, 기본 값을 지정하는 속성

이다.

```
<c:out value="${param.name}" default="이 부분은 value에 값이 null일 때 출력되는 기본 출력 값이다." />
```

다음은 태그 out의 속성에 대한 설명으로, 속성 escapeXML을 이용하여 JSP 파일에서 HTML 또는 XML의 코드 모양을 그대로 화면에 출력할 것인지, 아니면 문법적인 의미로 사용할 것인지 결정한다. 즉 문자열을 구성하는 <, >, &, ', " 5개의 문자는 속성 escapeXML을 true 또는 지정하지 않으면, 각각 <, >, &, ', "의 형태로 변환되어 문자 모양 그대로 출력되고, false로 지정하면 문자 그대로 해석되어 태그 역할을 수행한다.

escapeXML속성이 true일 경우 변환되는 문자

문자	변환된 형태
<	<
>	>
&	&
'	'
"	"

속성	필수	기본값	자료유형	기능
value	O		String	출력될 내용 또는 표현식
default	X		String	value에 값이 없는 경우 이용되며, 기본값으로 지정된 값이나 또는 태그 몸체에 있는 내용
escapeXML	X	true	String	true이면 <, >, &, ', " 5개의 문자를 문자 코드로 변환하여 브라우저에 태그 모양이 표시되고 false이면 그대로 출력되어 태그 의미로 반영.

- out.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL Core: out</title>
</head>
<body>
<h2>JSTL Core Tag: out</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:out value="<hr>" />
<c:out value="<hr>" escapeXml="false" />
<p>
```

```

<c:out value="${param.userid}">
    패러미터가 없습니다.
</c:out>
<p>
<c:out value="${param.userid}" default="패러미터가 없습니다." />
<p>${param.userid} = ${param.userid}
</body>
</html>

```

[결과]



20.3 SQL 태그 라이브러리

20.3.1 SQL 태그

- 데이터베이스 참조를 위한 SQL 태그

SQL 태그는 데이터베이스 연결과 질의를 위한 태그로 데이터베이스 연결 자원 지정을 위한 setDataSource태그와 질의를 위한 query, update, transaction 태그를 지원한다. 그리고 태그 query와 update의 내부 태그로, 질의 문장을 구성하는 매개변수 [?] 부분에는 값을 지정하는 param태그와 dateParam 태그를 제공한다.

분류	필수	기본값	기능
리소스 지정	setDataSource	<sql:setDataSource ...>	데이터소스 지정
질의	query	<sql:query ...>	조회 관련 SQL 문장 실행
	dateParam	<sql:dateParam ...>	날짜 형태로 SQL의 매개변수 값 지정
	param	<sql:param ...>	SQL의 매개변수 값 지정
	update	<sql:update ...>	수정 관련 SQL문장 실행
	dateParam	<sql:dateParam ...>	날짜 형태로 SQL의 매개변수 값 지정
	param	<sql:param ...>	SQL의 매개변수 값 지정
	tansaction	<sql:transaction ...>	트랜잭션 처리

SQL 태그를 사용하려면 다음과 같은 태그 taglib가 필요하며, 접두어는 [SQL]로, uri는[http://java.sun.com/jsp/jstl/sql]로 지정한다.

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

20.3.2 태그 sql을 이용한 테이블 조회

- <sql:setDataSource ...>

태그 setDataSource는 데이터베이스 연결을 위한 자원을 지정하는 태그이다.

다른 태그에서 이 자원을 사용하기 위해 자원을 저장하는 변수

```
<sql:setDataSource var="stuDS" dataSource="jdbc/oracleDB" />
```

연결할 데이터베이스 접속 정보가 있는 자원을 지정

```
<sql:setDataSource var="ds"
    driver="oracle.jdbc.OracleDriver"
    user="scott" password="tiger"
    url="jdbc:oracle:thin:@localhost:1521:orcl"/>

<sql:query dataSource="${ ds }" var="rs">
    select * from member
</sql:query>
```

다음은 태그 setDataSource의 속성에 대한 설명으로, 직접 데이터베이스 접속 정보인 속성 driver, URL, user, password를 입력하여 자원을 생성할 수 있다.

속성	필수	기본값	자료유형	기능
var	X		String	데이터소스를 위한 변수
dataSource	X		java.sql.DataSource 또는 String	데이터소스
driver	X		String	JDBC 드라이버 클래스
url	X		String	JDBC URL
user	X		String	데이터베이스 사용자 계정
password	X		String	데이터베이스 사용자 계정의 암호
scope	X	page	String	변수가 효력을 발휘하는 영역으로 page, request, session, application 중의 하나를 지정

- <sql:query ... >

태그 query는 데이터베이스 연결 자원을 사용하여 SQL문자를 실행하며, 지정하는 변수에 그 결과를 저장하는 태그이다.

결과가 저장되는데 사용

실행할 SQL문장을 지정

```
<sql:query var="studentsRS" dataSource="${stuDS}" sql="select * from student" />
```

데이터베이스 연결 자원을 지정

- 속성 var는 자료유형 ResultSet과 비슷한 결과의 집합체이다

태그 query는 위와 같이 몸체가 없이 속성 SQL에 SQL문장을 기술하거나 다음과 같이 속성 SQL 대신에 몸체에 SQL문장을 사용할 수 있다.

```
<sql:query var="studentsRS" dataSource="${stuDS}">
    select * from student
</sql:query>
```

다음은 태그 query의 속성에 대한 설명으로, 질의 결과를 저장하는 속성 var가 반드시 필요하다.

속성	필수	기본값	자료유형	기능
var	O		String	질의 결과를 저장하는 변수
dataSource	X		java.sql.DataSource 또는 String	데이터소스
sql	X		String	SQL 문장
startRows	X	0	int	질의 결과로 가져올 시작 행
maxRows	X	질의 결과의 최대행수	int	질의 결과로 가져올 최대행. -1로 지정해도 질의 결과의 최대 행수
scope	X	page	String	변수가 효력을 발휘하는 영역으로 page, request, session, application 중의 하나를 지정

- <sql:param ...> / <sql:dateParam ...>

태그 param은 실행할 SQL 문장 매개변수 [?]부분에 값을 지정하는 태그이다. 질의 문장을 구성하는 매개변수 [?]부분에, 순서대로 태그 param을 사용하여 지정한 후 SQL문을 실행한다.

```
<sql:query var="comRS" dataSource="${stuDS}">
    select * from subject where s_num = ?
    <sql:param value="01" />
</sql:query>
```

태그 param의 속성값을 지정하는 value 하나뿐이다.

속성	필수	기본값	자료유형	기능
var	O		String	SQL 문장의 매개변수 부분을 지정하는 값

JSTL은 태그 param과 비슷한 태그로 태그 dateParam을 제공한다. 태그 dateParam은 매개변수가 있는 SQL문에서 date, time, timestamp 유형으로 매개변수 값을 지정할 때 사용한다.

속성	필수	기본값	자료유형	기능
value	O		String	SQL 문장의 매개변수 부분을 지정하는 값
type	X		String	필드의 유형으로 date, time, timestamp 중의 하나를 지정

태그 setDataSource에서, 속성 dataSource가 리소스에서 설정한 이름인 "jdbc/oracleDB"로 반드시 지정되어야 한다.

```
<sql:setDataSource var="stuDS" dataSource="jdbc/oracleDB" />
```

다음 소스의 태그 query에서 질의한 결과는 속성 var인 studentsRS에 저장된다. foreach 태그에서 item에 \${studentsRS.rows}를 지정하며, 질의 결과의 한행씩 var에 지정된 studentRow 변수에 저장되어 반복을 수행한다. 태그 foreach의 몸체에서 변수 studentRow.sd_num과 같이 테이블 필드 이름을 한 행의 필드 값을 참조할 수 있다. 이 경우 반드시 필드 이름을 이용해야 한다.

```
<sql:query var="studentsRS" dataSource="${stuDS}" sql="select * from student" />
<c:forEach var="studentRow" begin="0" items="${studentsRS.rows}">
${studentRow.sd_name}, ${studentRow.sd_id}, ${studentRow.s_num}, ${studentRow.sd_hpone} <br>
</c:forEach>
```

태그 foreach에서 태그 query의 속성 var에 지정된 변수로 rowByIndex를 참조하면, 이름이 아닌 문자값으로 현재 행의 필드를 참조할 수 있다.

```
<c:forEach var="studentRow" begin="0" items="${comRS.rowsByIndex}">
${studentRow[1]}, ${studentRow[2]}, ${studentRow[4]}, ${studentRow[8]} <br>
</c:forEach>
```

- query.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL SQL Tag</title>
</head>
<body>
<h2>JSTL SQL Tag: setDataSource, query, param</h2>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<hr><h2>전체 학생 조회</h2>
<sql:setDataSource var="stuDS" dataSource="jdbc/oracleDB" />
<sql:query var="studentsRS" dataSource="${stuDS}">
    select sd_name,sd_id,s_num,sd_hpone from student
</sql:query>
<table border="1" width="450">
    <tbody align="center">
        <tr bgcolor="#FFCC00">
            <th>이름</th>
            <th>아이디</th>
            <th>학과번호</th>
```

```

<th>연락처</th>
</tr>
<c:forEach var="studentRow" begin="0" items="${studentsRS.rows}">
<tr>
    <td>${studentRow.sd_name}</td>
    <td>${studentRow.sd_id}</td>
    <td>${studentRow.s_num}</td>
    <td>${studentRow.sd_hpone}</td>
</tr>
</c:forEach>
</tbody>
</table>
<hr><h2>학생 조회</h2>
<sql:query var="comRS" dataSource="${stuDS}" >
    select * from student where s_num = ?
    <sql:param value="01" />
</sql:query>
<table border="1" width="450">
    <tbody align="center">
        <tr bgcolor="#CCFF00">
            <th>이름</th>
            <th>아이디</th>
            <th>연락처</th>
            <th>이메일주소</th>
        </tr>
        <c:forEach var="studentRow" begin="0" items="${comRS.rows}">
        <tr>
            <td>${studentRow.sd_name}</td>
            <td>${studentRow.sd_id}</td>
            <td>${studentRow.sd_hpone}</td>
            <td>${studentRow.sd_email}</td>
        </tr>
        </c:forEach>
    </tbody>
</table>
<hr><h2>위와 같은 결과</h2>
<table border="1" width="450">
    <tbody align="center">
        <tr bgcolor="#FF99CC">
            <th>학번</th>

```

```

<th>이름</th>
<th>연락처</th>
<th>이메일주소</th>
</tr>
<c:forEach var="studentRow" begin="0" items="${comRS.rowsByIndex}">
<tr>
    <td>${studentRow[1]}</td>
    <td>${studentRow[2]}</td>
    <td>${studentRow[4]}</td>
    <td>${studentRow[8]}</td>
</tr>
</c:forEach>
</tbody>
</table>
</body>
</html>

```

[결과]

이름	아이디	학과번호	연락처
김정수	javajsp	01	01012345678
김수현	jdbcmania	01	0113452468
공지영	gonji	04	01612657455
조수영	water	05	0107681499
최경란	novel	04	0119872455
안익태	korea	02	0168452345
전미정	dbcp	03	0111294567

이름	아이디	연락처	이메일주소
김정수	javajsp	01012345678	java12@naver.com
김수현	jdbcmania	0113452468	jdbcmania@naver.com

20.3.3 태그 sql을 이용한 테이블 수정

- <sql:update ...>

태그 update는 데이터베이스 연결 자원을 사용하여 DDL 관련 문장 또는 insert, update, delete 질의 문장을 실행하여 그 결과를 저장하는 태그이다. 태그 update의 속성은 태그 query와 비슷하며, 속성 dataSource에 데이터베이스 연결 자원을 지정하고, SQL에는 실행할 SQL문장을 지정하며, 속성 var에는 수정한 결과인 수정이 발생한 레코드 수가 저장된다.

```
<sql:update var="ucount" dataSource="${stuDS}" sql="SQL 문장" />
```

SQL문장에 여러 개의 매개변수를 사용하려면 질의 문장에 있는 여러 개의 [?] 부분을 순서대로 태그 param을 이용하여 값을 지정한 후 SQL을 실행한다. 필드가 날짜 형태인 경우는 태그 dateParam을 사용한다. 태그 dateParam의 속성 type은 date, time, timestamp 중의 하나로 날짜와 시간을 동시에 지정하려면 timestamp를 지정한다.

```
<sql:update var="n" dataSource="${ds}">
```

```

update board set title=?, email=?, regdate=?, content=? where no=?
<sql:param value="Connection Pool" />
<sql:param value="java1234@naver.com" />
<sql:dateParam value="<% new java.util.Date() %>" type="timestamp" />
<sql:param value="커넥션 풀이란 미리 여러 개의 데이터베이스 커넥션을 만들어 확보해 놓고 클라
이언트의 요청이 있는 경우, 커넥션을 서비스해 주는 커넥션 관리 기법이다." />
<sql:param value="1" />
</sql:update>

```

다음은 태그 update의 속성에 대한 설명으로, 질의 결과를 저장할 속성 var가 반드시 필요하다.

속성	필수	기본값	자료유형	기능
var	O		String	질의 결과인 값이 수정된 레코드 수 를 저장하는 변수
dataSource	X		java.sql.DataSource 또는 String	데이터소스
sql	X		String	SQL 문장
scope	X	page	String	변수가 효력을 발휘하는 영역으로 page, request, session, application 중 의 하나를 지정

- update.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL SQL Tag</title>
</head>
<body>
<h2>JSTL SQL Tag: update, dateParam</h2>
<hr><h3>전체 게시판 조회</h3>
<sql:setDataSource var="ds" dataSource="jdbc/oracleDB" />
<sql:query var="rs" dataSource="${ds}" sql="select * from board" />
<table border="1" width="550">
<tbody align="center">
<tr bgcolor="#CCFF00">
<th>번호</th>
<th>이름</th>

```

```

<th>제목</th>
<th>이메일주소</th>
<th>등록일</th>
</tr>
<c:forEach var="brdRow" begin="0" items="${rs.rows}">
<tr>
    <td>${brdRow.no}</td>
    <td>${brdRow.name}</td>
    <td>${brdRow.title}</td>
    <td>${brdRow.email}</td>
    <td>${brdRow.regdate}</td>
</tr>
</c:forEach>
</tbody>
</table>
<hr><h3>번호 1번 레코드에서 제목, 이메일, 등록일, 내용 수정 후,</h3>
<sql:update var="n" dataSource="${ds}">
    update board set title=?, email=?, regdate=?, content=? where no=?
    <sql:param value="Connection Pool" />
    <sql:param value="java1234@naver.com" />
    <sql:dateParam value="<% new java.util.Date() %>" type="timestamp" />
    <sql:param value="커넥션 풀이란 미리 여러 개의 데이터베이스 커넥션을 만들어 확보해 놓고 클라이언트의 요청이 있는 경우, 커넥션을 서비스해 주는 커넥션 관리 기법이다." />
    <sql:param value="1" />
</sql:update>
<hr><h3>다시 게시판 전체 조회</h3>
<sql:query var="rs" dataSource="${ds}" sql="select * from board" />
<table border="1" width="550">
    <tbody align="center">
        <tr bgcolor="#CCFF00">
            <th>번호</th>
            <th>이름</th>
            <th>제목</th>
            <th>이메일주소</th>
            <th>등록일</th>
        </tr>
        <c:forEach var="brdRow" begin="0" items="${rs.rows}">
        <tr>
            <td>${brdRow.no}</td>
            <td>${brdRow.name}</td>

```

```

<td>${brdRow.title}</td>
<td>${brdRow.email}</td>
<td>${brdRow.regdate}</td>
</tr>
</c:forEach>
</tbody>
</table>
</body>
</html>

```

[결과]

번호	이름	제목	이메일주소	등록일
1	홍길동	게시판의 기능	java77@naver.com	2009-07-14
2	김희진	게시판의 프로그램 구성	javarva@naver.com	2009-07-14

번호	이름	제목	이메일주소	등록일
1	홍길동	Connection Pool	java1234@naver.com	2009-07-19
2	김희진	게시판의 프로그램 구성	javarva@naver.com	2009-07-14

20.4 함수 라이브러리

20.4.1 길이와 문자열 처리 함수

- \${ fn:functionName() }

JSTL 함수 라이브러리는 length()를 제외하고는 모두 문자열 처리에 관련된 함수이다. 함수 라이브러리 대부분은 자바의 String이 제공하는 메소드와 비슷하다. 함수 length()는 인자로 문자열뿐 아니라 일반 콜렉션 객체를 사용할 수 있으며, 인자인 문자열 또는 콜렉션의 길이를 반환한다.

함수 태그를 이용하기 위해서는 다음의 taglib 지시자를 지정해야 한다.

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

반환유형	함수와 예문	설명
boolean	contains(String name, String search) <c:if test="\${fn:contains(name, search)}">	search에 name이 있는지 검사

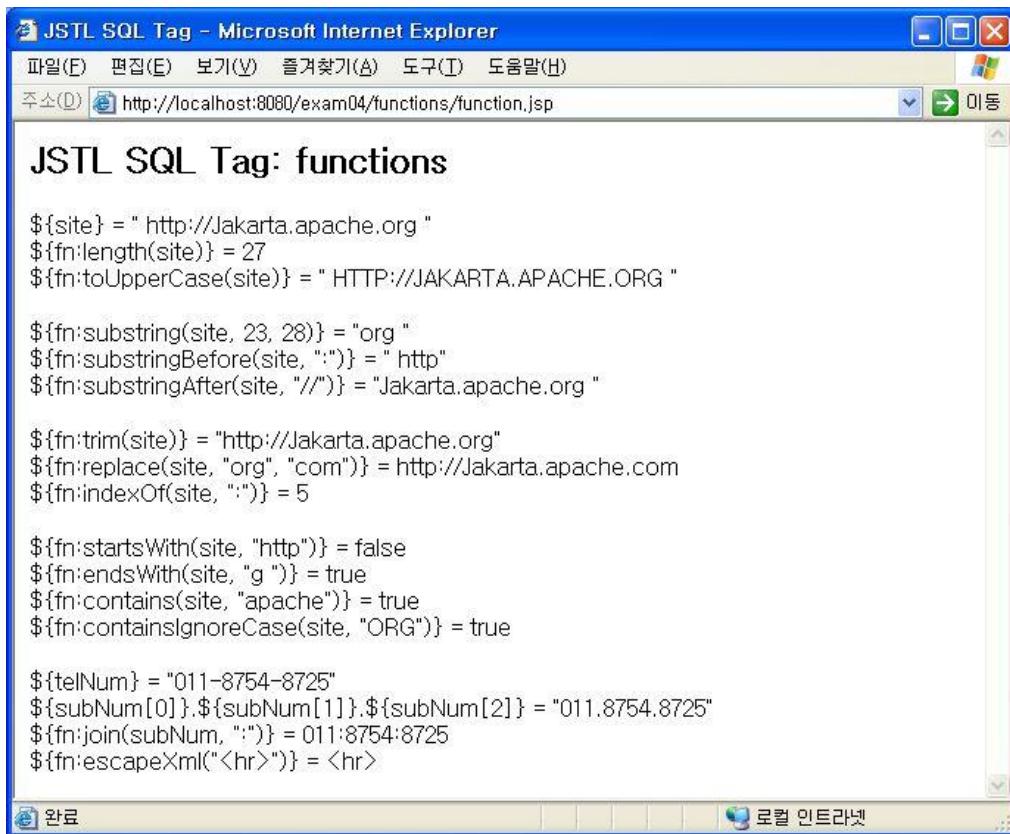
boolean	containsIgnoreCase(String name, String search)	대소문자를 구별하지 않고 search에 name이 있는지 검사
	<c:if test="\${fn:containsIgnoreCase(name, search)}">	
boolean	endsWith(String src, String name)	src가 name으로 종료되는지 검사
	<c:if test="\${fn:endsWith(String src, String name)}">	
String	escapeXml(String str)	str에서 마크업 문자를 브라우저에 그대로 표시되는 문자열로 반환
	\${fn:escapeXml(String str)}	
int	indexOf(String src, String search)	src에서 search 문자열이 처음 나타내는 첫 자를 반환
	\${fn:indexOf(String src, String search)}	
String	join(String[] array, String ins)	배열 array의 원소와 원소 사이에 ins를 삽입한 문자열을 반환
	\${fn:join(array, ":")}	
int	length(Object obj)	집합체 obj의 길이를 반환
	\${fn:length(shoppingCart.products)}	
String	replace(String str, String before, String after)	src에서 before문자열을 모두 after 문자열로 변환하여 반환
	\${fn: replace(text, "-", "•") }	
String[]	split(String src, String search)	src에서 search로 계속 구분해서 나머지를 문자열 배열로 반환
	\${fn:split(customerNames, ":")}	
boolean	startsWith(String src, String search)	src가 search 문자열로 시작하는지 검사
	<c:if test="\${fn: startsWith(product.no, "10")}">	
String	substring(String src, int start, int end)	src의 첫자 start에서 end-1까지의 부분 문자열을 반환. End가 음수이면 끝까지
	\${fn:substring(jumin, 8, -1)}	
String	substringAfter(String src, String search)	src에서 search이후의 부분 문자열을 반환
	\${fn:substringAfter(zip, "-")}	
String	substringBefore(String src, String search)	src에서 search이전의 부분 문자열을 반환
	\${fn:substringBefore(zip, "-")}	
String	toLowerCase(String src)	src의 모든 대문자를 소문자로 변환하여 반환
	\${fn: toLowerCase(product.name)}	
String	toUpperCase(String src)	src의 모든 소문자를 대문자로 변환하여 반환
	\${fn:toUpperCase(product.name)}	
String	trim(String src)	src의 앞•뒤에 있는 빈 공간을 제거하여 반환

	\${fn: trim(name)}
--	--------------------

- function.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>JSTL SQL Tag</title>
</head>
<body>
<h2>JSTL SQL Tag: functions</h2>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<c:set var="site" value=" http://Jakarta.apache.org " />
${site} = "${site}" <br>
${fn:length(site)} = ${fn:length(site)} <br>
${fn:toUpperCase(site)} = "${fn:toUpperCase(site)}" <p>
${fn:substring(site, 23, 28)} = "${fn:substring(site, 23, 28)}" <br>
${fn:substringBefore(site, ".")}= "${fn:substringBefore(site, ".")}" <br>
${fn:substringAfter(site, "//")}= "${fn:substringAfter(site, "//")}" <p>
${fn:trim(site)} = "${fn:trim(site)}" <br>
${fn:replace(site, "org", "com")}= ${fn:replace(site, "org", "com")}<br>
${fn:indexOf(site, ":")}= ${fn:indexOf(site, ":" )}<p>
${fn:startsWith(site, "http")}= ${fn:startsWith(site, "http")}<br>
${fn:endsWith(site, "g ") }= ${fn:endsWith(site, "g ") }<br>
${fn:contains(site, "apache") }= ${fn:contains(site, "apache")}<br>
${fn:containsIgnoreCase(site, "ORG") }= ${fn:containsIgnoreCase(site, "ORG")}<p>
<c:set var="telNum" value="011-8754-8725" />
<c:set var="subNum" value="${fn:split(telNum, '-')}" />
${telNum} = "${telNum}" <br>
${subNum[0]}.${subNum[1]}.${subNum[2]} = "${subNum[0]}.${subNum[1]}.${subNum[2]}" <br>
${fn:join(subNum, ":") }= ${fn:join(subNum, ':')}<br>
${fn:escapeXml("&lt;hr&gt;") }= ${fn:escapeXml("<hr>")}<br>
</body>
</html>
```

[결과]



JSTL SQL Tag – Microsoft Internet Explorer

파일(E) 편집(E) 보기(V) 즐겨찾기(A) 도구(I) 도움말(H)

주소(D) http://localhost:8080/exam04/functions/function.jsp

JSTL SQL Tag: functions

```
 ${site} = " http://Jakarta.apache.org "
 ${fn:length(site)} = 27
 ${fn:toUpperCase(site)} = " HTTP://JAKARTA.APACHE.ORG "

 ${fn:substring(site, 23, 28)} = "org"
 ${fn:substringBefore(site, ":")}= " http"
 ${fn:substringAfter(site, "//")}= "Jakarta.apache.org"

 ${fn:trim(site)} = "http://Jakarta.apache.org"
 ${fn:replace(site, "org", "com")}= http://Jakarta.apache.com
 ${fn:indexOf(site, ":")}= 5

 ${fn:startsWith(site, "http")}= false
 ${fn:endsWith(site, "g ")}= true
 ${fn:contains(site, "apache")}= true
 ${fn:containsIgnoreCase(site, "ORG")}= true

 ${telNum}= "011-8754-8725"
 ${subNum[0]}.${subNum[1]}.${subNum[2]}= "011.8754.8725"
 ${fn:join(subNum, ":")}= 011:8754:8725
 ${fn:escapeXml("<hr>")}= <hr>
```

완료 로컬 인트라넷

20.5 JSTL의 국제화/형식화 액션

JSTL fmt란 국제화/형식화의 기능을 제공해주는 JSTL 라이브러리이다. 이 내용을 구체적으로 설명하면 **국제화는 다국어 내용을 처리, 형식화는 날짜와 숫자 형식 등을 처리하는 것을 말한다.** JSTL fmt 라이브러리를 사용할 때는 core 라이브러리를 사용할 때처럼 사용할 JSP 페이지에 태그 라이브러리로 등록해 주어야 한다.

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

```
<fmt:requestEncoding value="인코딩값">
```

<fmt:requestEncoding>은 request 객체로부터 전달받은 값을 인코딩할 때 사용한다. 보통 한글 값이 넘어올 경우 <%request.setCharacterEncoding("UTF-8");%> 코드로 request 객체로 받는 값을 한글에 맞게 인코딩해 준다.

```
<fmt:setLocale value="값" variant="" scope="범위">
```

<fmt:setLocale>은 다국어 페이지를 사용할 때 언어를 지정하는 태그이다. 보통 세계적으로 알려져 있는 사이트의 경우는 여러 나라에 해당하는 언어를 지원해준다. 이 태그는 여러 나라에 해당하는 언어를 지원하도록 설계되어 있을 때 어떤 언어를 사용하여 페이지를 표시할지 설정하는 태그이다. value속성은 어떤 언어를 사용할지 지정하는 속성이며, 언어 코드를 입력할 수 있다. variant 속성은 브라우저의 스펙을 지정할 때 사용한다.

```
<fmt:TimeZone value="값">
```

<fmt:timeZone>은 지정한 지역값으로 시간대를 맞출 때 사용된다.

이와 비슷한 태그로는 <fmt:setTimeZone>이 있는데, <fmt:setTimeZone>는 특정 페이지 전체에 적용이 되지만, <fmt:timeZone> 태그는 첫 태그와 끝 태그 사이의 body 부분의 내용에만 적용된다.

```
<fmt:setTimeZone value="값" var="변수명" scope="범위">
```

<fmt:setTimeZone>은 <fmt:timeZone>과 유사한 태그로 지정한 지역값으로 시간대를 맞추는 기능은 같지만, 적용되는 영역이 다르다. <fmt:timeZone>의 경우는 첫 태그와 끝 태그 사이의 body 영역만 적용되지만, <fmt:setTimeZone>은 페이지 전체에 영향을 주게 된다.

var속성의 변수에 지정한 시간대 값이 저장되며, scope로 범위를 지정할 수 있다.

```
<fmt:bundle basename="basename" prefix="prefix">
```

<fmt:bundle>은 properties 확장명을 가진 파일의 리소스를 불러올 때 사용한다. 보통 페이지의 다국어 처리를 위해 많이 사용된다. basename에는 properties 확장명을 가진 파일을 지정할 수 있다. 패키지까지 포함한 경로를 지정하여야 한다는 것에 주의하도록 한다.

prefix는 properties내의 키 값에 쉽게 접근할 수 있도록 간략한 접근어를 사용할 수 있게 해준다. 그리고 불러온 리소스는 이 태그의 body 내에서만 사용할 수 있다.

```
<fmt:setBundle basename="basename" var="변수명" scope="범위">
```

이 태그는 <fmt:bundle> 태그와 같은 기능을 하는 태그이다. 차이점은 사용 영역이다. <fmt:bundle> 태그는 첫 태그와 끝 태그 사이의 body 영역에서만 불러온 리소스를 사용할 수 있지만, <fmt:setBundle> 태그는 페이지 전체에 적용시킬 수 있다. 그 외 나머지 기능은 <fmt:bundle>과 같다.

var에는 불러온 리소스 내용을 가지고 있을 변수명을 입력하면 되며, scope로 범위를 지정할 수 있다.

```
<fmt:message key="키값" bundle="bundle변수" var="변수명" scope="범위">
```

<fmt:message>는 불러온 properties 파일의 리소스 내용을 읽을 때 사용한다. properties 파일에는 여러 키 값들로 내용이 구분되어 있으며, <fmt:message>태그의 key 속성으로 키 값으로 구분된 내용을 가져올 수 있다. bundle 속성에는 <fmt:setBundle> 태그에서 var 속성에 지정했던 변수를 입력할 수 있다. 즉 bundle 속성을 사용하여 properties 파일에 접근할 수 있다는 것이다. <fmt:bundle>을 썼을 때는 물론 bundle 속성이 필요가 없다.

```
<fmt:param value="파라미터값">
```

<fmt:param> 태그는 <fmt:message>태그로 읽어온 리소스 내용에 파라미터를 전달하는 태그이다. 리소스 내용이 예를 들어 '{0}님 안녕하세요'일 경우 <fmt:param>태그로 {0}대신 원하는 파라미터를 지정할 수 있다.

```
<fmt:formatNumber value="값" type="타입" pattern="패턴" currencyCode="값"
currencySymbol="값" groupingUsed="true or false" maxIntegerDigits="값" minIntegerDigits="값"
maxFractionDigits="값" minFractionDigits="값" var="변수명" scope="범위">
```

<fmt:formatNumber>태그는 숫자 형식의 패턴을 설정할 때 사용한다. value속성에는 패턴을 적용시킬 원래의 값을 입력하며, type는 숫자의 타입을 의미한다. 타입은 숫자, 통화, 퍼센트 중 원하는 타입으로

설정할 수 있다. Pattern 속성은 지정한 값을 어떤 패턴으로 변화시킬지를 정할 수 있다. currencyCode는 통화코드를 의미하며, 숫자 타입이 통화일 경우만 유효하다. currencySymbol도 숫자 타입이 통화일 경우만 유효하며, 통화 기호를 지정할 수 있다. groupingUsed는 그룹 기호의 포함 여부를 나타낸다. maxIntegerDigits는 최대 정수 길이, minIntegerDigits는 최소 정수 길이, maxFractionDigits은 최대 소수점 자릿수, minFractionDigits은 최소 소수점 자릿수를 의미한다.

```
<fmt:parseNumber value="값" type="타입" pattern="패턴" parseLocale="값"  
integerOnly="true or false" var="변수명" scope="범위">
```

<fmt:parseNumber>는 문자열을 숫자, 통화 또는 퍼센트의 형태로 변환할 때 사용하는 태그이다. value속성에 문자열 값을 입력하고, type에는 숫자, 통화, 퍼센트 중 변환할 타입을 설정한다. pattern 속성은 value 속성의 값을 설정된 타입으로 변환하면서 어떤 패턴을 갖게 할 것인지를 지정할 수 있다. parseLocale은 파싱할 기본 패턴을 지정하며, integerOnly는 지정된 값 중 정수 부분만 해석할지의 여부를 지정하는 속성이다.

```
<fmt:formatDate value="값" type="타입" dataStyle="값" timeStyle="값" pattern="패턴"  
timeZone="값" var="변수명" scope="범위">
```

<fmt:formatDate>는 날짜 형식의 패턴을 설정할 때 사용되는 태그이다. value 속성에는 날짜 또는 시간을 입력할 수 있고, type 속성으로 날짜인지 시간인지 또는 날짜와 시간 둘 다 포함한 타입인지를 지정할 수 있다. dataStyle은 날짜의 스타일을 지정할 수 있으며 이것은 type 속성이 date 또는 both일 때만 적용된다. timeStyle은 시간의 스타일을 지정할 수 있다. type속성이 time 또는 both 일 때만 적용된다. timeZone 속성은 날짜와 시간이 표시될 시간대를 지정 할 수 있다.

```
<fmt:parseDate value="값" type="타입" pattern="패턴" parseLocale="값" dataStyle="값"  
timeStyle="값" timeZone="값" var="변수명" scope="범위">
```

<fmt:parseDate>는 문자열을 날짜와 시간의 형태로 변환하는 태그이다. value 속성에 입력된 문자열 값을 type 속성에 지정된 타입으로 날짜와 시간의 형태로 변환한다. 나머지 속성은 <fmt:formatNumber>의 속성과 기능이 같다. 단지 <fmt:parseDate>는 문자열 값을 파싱하여 날짜, 시간 형태로 변환해 주는 것뿐이다.

- test.properties

```
name=Hongkildong  
say=Hello. I'm Hongkildong.  
say2=My friend is {0}.
```

- test_ko.properties

```
name=%ud64d%uae38%ub3d9  
say=%uc548%ub155%ud558%uc138%uc694 %ud64d%uae38%ub3d9%uc785%ub2c8%ub2e4.
```

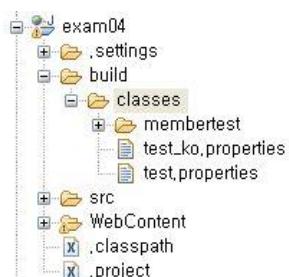
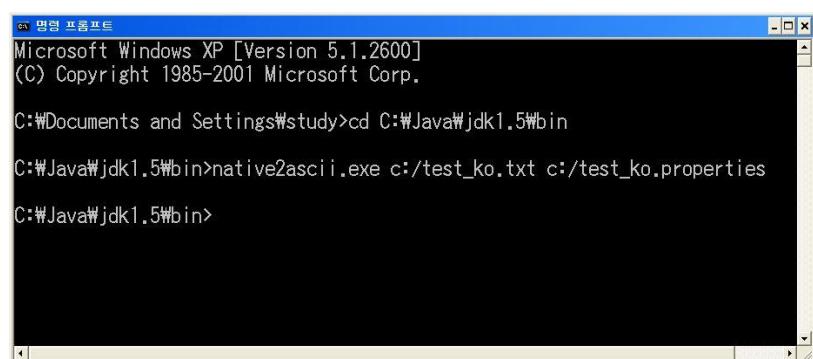
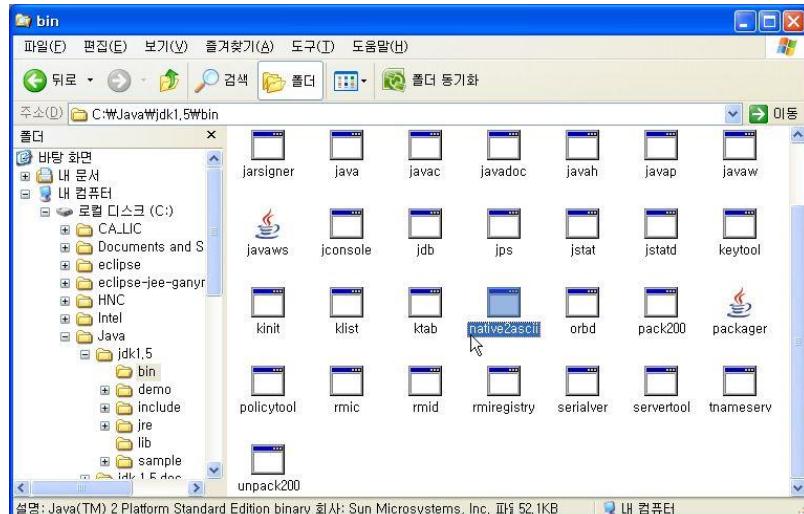
```
say2=%ub0b4 %uce5c%uad6c%ub294 {0} %uc785%ub2c8%ub2e4.
```

properties 파일은 주로 국제화를 구현할 때 사용된다. test.properties파일은 영문 페이지를 구현할 때 사용하고, test_ko.properties 파일은 한글 페이지를 구현할 때 사용한다. 그런데 test_ko.properties 파일 내용을 보면 한글은 보이지 않는다. 이것은 한글로 된 문서를 아스키화시킨 것이다. 먼저 원본 한글로된 properties 파일을 가지고 위의 예제 파일처럼 아스키화하도록 할 것이다.

- test_ko.txt

```
name=홍길동  
say=안녕하세요. 홍길동입니다.  
say2=내 친구는 {0}입니다.
```

먼저 test_ko.txt파일로 위의 내용을 입력하고 저장한다. (c:\) 저장했다면 위의 파일을 아스키화하기 위해 필요한 native2ascii.exe를 복사할 것이다. 탐색기를 띄우고 JDK 폴더의 bin 폴더로 이동한다.



native2ascii.exe c:/test_ko.txt c:/test_ko.properties라고 입력하였다. 이것은 c:/에 있는 test_ko.txt 파일의 내용을 아스키화하며, 아스키화된 파일을 test_ko.properties로 저장한다는 의미를 갖는다.

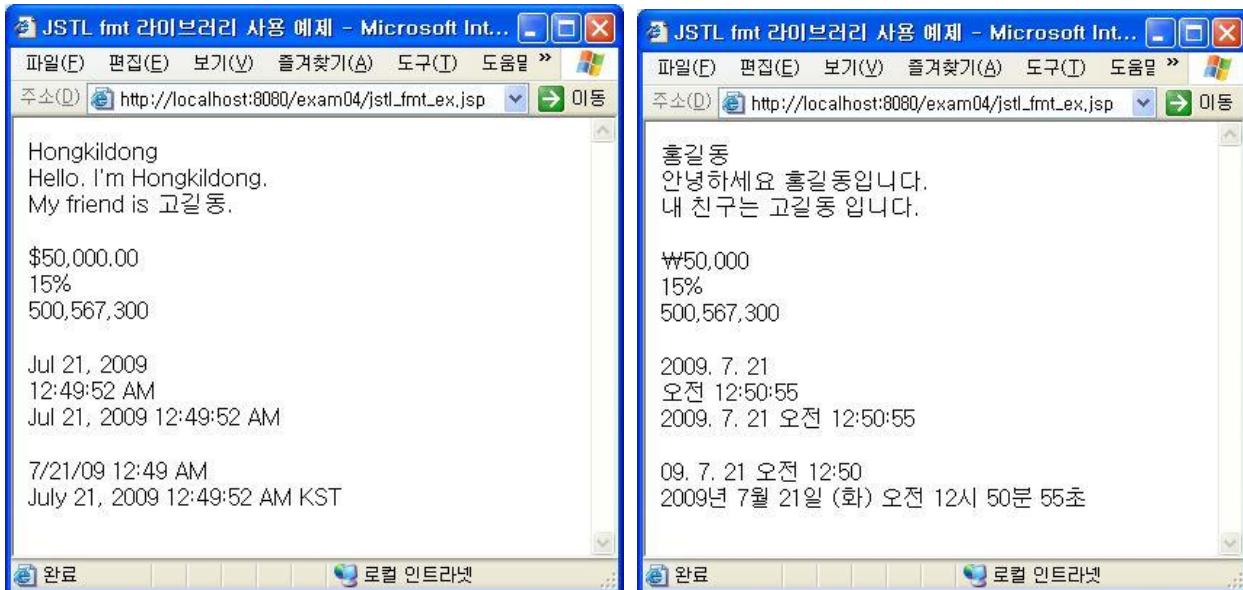
test.properties 파일과 test_ko.properties 파일을 사용하기 위해서는 이클립스 프로젝트 폴더의 [build]/[classes] 폴더에 복사하여야 한다. 이 폴더는 톰캣에서 동작할 때 [WEB-INF]/[classes]와 같은 폴더이다. 이와 같이 이클립스 프로젝트 폴더의 [build]/[classes]에 properties파일들을 복사해 주어야 국제

화 관련 태그를 사용할 때 이 파일을 읽을 수 있게 된다.

- jstl_fmt_ex.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- JSTL의 fmt라이브러리를 사용하기 위해 uri를 fmt 라이브러리가 존재하는 곳으로 설정하고 접두사 속성인 prefix 값은 fmt로 설정한다.--&gt;
&lt;jsp:useBean id="date" class="java.util.Date"/&gt;
&lt;!DOCTYPE html&gt;
&lt;html&gt;
&lt;head&gt;&lt;meta charset="UTF-8"&gt;
&lt;title&gt;JSTL fmt 라이브러리 사용 예제&lt;/title&gt;
&lt;/head&gt;
&lt;body&gt;
    ko_KR로 지정
&lt;b&gt;&lt;fmt:setLocale value="en_US"/&gt;&lt;/b&gt;
    <!--value 값은 언어코드와 국가코드로 조합되며, en_US는 영문을 의미한다. 앞에서 만든 test.properties 파일이 영문 페이지에 해당하는 파일이다. 언어를 한글로 지정하려면 value 값을 ko_KR로 지정하면 test_ko.properties 파일을 사용하기 때문에 언어가 한글로 표시된다.--&gt;
&lt;fmt:bundle basename="test"&gt;&lt;!--test 이름을 가진 properties 파일을 읽는다 --&gt;
    &lt;fmt:message key="name"/&gt;&lt;br&gt;&lt;!--key에 해당하는 값을 출력한다--&gt;
    &lt;fmt:message key="say"/&gt;&lt;br&gt;
    &lt;fmt:message key="say2"&gt;&lt;!--{0}에 고길동을 지정된다 --&gt;
        &lt;fmt:param value="고길동"/&gt;
    &lt;/fmt:message&gt;
&lt;/fmt:bundle&gt;
&lt;p&gt;
    &lt;fmt:formatNumber value="50000" type="currency"/&gt;&lt;br&gt;
    &lt;fmt:formatNumber value="0.15" type="percent"/&gt;&lt;br&gt;
    &lt;fmt:formatNumber value="500567300" pattern="###,###,###/}&gt;&lt;p&gt;
    &lt;fmt:formatDate value="${date}" type="date"/&gt;&lt;br&gt;
    &lt;fmt:formatDate value="${date}" type="time"/&gt;&lt;br&gt;
    &lt;fmt:formatDate value="${date}" type="both"/&gt;&lt;p&gt;
    &lt;fmt:formatDate value="${date}" type="both" timeStyle="short" dateStyle="short"/&gt;&lt;br&gt;
    &lt;fmt:formatDate value="${date}" type="both" timeStyle="long" dateStyle="long"/&gt;&lt;br&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

[결과]



21. 커스텀 태그 개요

21.1.1 커스텀 태그 정의와 유형

- 커스텀 태그 정의

JSP 프로그램에서 자바빈즈가 비즈니스 로직을 담당하면서 커스텀 태그는 JSP 프로그램에서 반복적으로 수행되는 모듈 부분을 담당한다고 할 수 있다. 즉 JSP 프로그램에서 반복적으로 수행되는 모듈 부분은 주로 표현부분과 제어흐름의 스크립트릿으로 구성되는 경우가 대부분이다.

즉 커스텀 태그는 반복적으로 사용되는 조건, 반복 등의 제어흐름과 다양한 태그의 표현 부분을 하나의 새로운 태그로 정의하여 사용할 수 있는 XML 유형의 사용자 정의 태그이다.

커스텀 태그는 복잡한 표현 또는 기능들을 하나의 태그로 정의함으로써 개발자에게 좀 더 편리하게 작업을 할 수 있도록 해주고 궁극적으로 디자인과 개발을 분리하여 개발 효율성과 생산성을 높이는 데에 그 목적이 있다.

- 커스텀 태그 유형

커스텀 태그는 XML 태그이므로 다음과 같이 시작 태그와 종료 태그가 반드시 있어야 하며, 속성="값"과 같이 값에는 반드시 큰 따옴표 또는 작은 따옴표를 붙여야 한다. 또한 같은 태그 이름의 충돌을 피하기 위해 prefix인 접두어를 사용한다. 태그를 <prefix:tagName>로 시작한다면 몸체가 없는 태그로 />이 종료태그이며, 몸체가 있는 태그는 </prefix:tagName>이 종료 태그이다.

21.1.2 커스텀 태그 생성

- 커스텀 태그 만드는 방법

JSP 1.1 스펙에서 소개된 커스텀 태그는 JSP 1.2 지원 커스텀 태그 처리 방식과 JSP 2.0 지원 커스텀 태그 처리 방식으로 나눌 수 있다. 그러므로 커스텀 태그를 만드는 방법은 JSP 버전 1.2와 2.0으로 크게 나뉜다. JSP 2.0에서는 태그 처리기(Tag Handler) 방식과 태그(Tag File) 방식, 2가지를 제공한다. 특히 JSP 2.0에서 지원하는 태그 파일 방식은 태그 처리기 방식보다 쉽게 커스텀 태그를 만들 수 있다.

버전	이름	특징
JSP 1.2	JSP 1.2 태그 처리기	자바 프로그래머에게 적합하고, 상대적으로 다소 복잡하며, JSP 2.0을 사용할 수 없는 경우에 사용
JSP 2.0	JSP 2.0 태그 처리기	자바 프로그래머에게 적합하며, JSP 1.2 태그 처리기에 비해 한결 간편해짐
	태그 파일	JSP 프로그램과 유사하며 표현언어와 JSTL에 익숙한 프로그래머에게 적합

태그 처리기 방식은 일반적으로 태그 처리기인 자바 클래스 작성, 태그 라이브러리 디스크립터인 TLD(Tag Library Descriptor) 작성이 필요하다. 그러나 태그 방식은 태그 파일만 필요하다.

태그 처리기 방식은 일반 자바 프로그램으로 태그 처리기를 작성하며, 태그 파일 방식은 JSP 프로그램과 같이 스크립트 요소와 HTML 그리고 다른 커스텀 태그를 사용하여 태그 처리를 작성한다.

이름	구현 인터페이스 또는 상속 클래스	구현 파일	구현방법
JSP 1.2 태그 처리기	java.servlet.jsp.tagext.Tag javax.servlet.jsp.tagext.TagSupport	자바파일	Tag 또는 TagSupport를 상속받은 자바 클래스 구현

JSP 2.0 태그 처리기	java.servlet.jsp.tagext.SimpleTag javax.servlet.jsp.tagext.SimpleTagSupport	자바파일	SimpleTag 또는 SimpleTagSupport를 상속받은 자바 클래스 구현
태그 파일	확장자가 tag인 태그 파일	태그파일	JSP 프로그램과 같은 태그 파일 구현

JSP 1.2 태그 처리기는 일반적으로 클래스 javax.servlet.jsp.tagext.TagSupport를 상속받아 구현하며, JSP 2.0 태그 처리기는 javax.servlet.jsp.tagext.SimpleTagSupport를 상속받아 구현한다. 상속받는 클래스 이름 TagSupport에서 보듯이, JSP 2.0 태그 처리기 방식이 JSP 1.2 태그 처리기보다 간편하고 쉽다. 태그 처리기 방식은 태그 처리기뿐만 아니라 태그 처리기를 태그로 사용할 수 있도록 TLD 파일의 작성이 필요하다.

태그 파일 방식은 JSP 프로그램과 같이 <%@ tag ... %>와 같이 특별히 추가된 지시자와 스크립트 요소, HTML 그리고 다른 커스텀 태그를 사용하여 태그 처리를 작성하므로 태그 처리기 방식보다 훨씬 쉽다. 또한 태그 처리기 방식이 TLD 파일이 필요한 것에 비하면 태그 방식은 TLD 파일이 필요 없어 더욱 간편하다.

이클립스에서 개발된 커스텀 태그에 관련된 파일을 살펴보면, JSP 1.2와 JSP 2.0의 태그 처리기 방식은 [Java Resources: src] 하부에 태그 처리기인 자바 프로그램과 [WEB-INF]/[tlds] 하부에 TLD 파일이 필요하다. 반면에 JSP 2.0 태그 방식 파일 방식은 [WEB-INF]/[tags] 하부에 tag 파일만이 필요하다.

21.2. JSP 2.0 태그 처리기의 커스텀 태그

21.2.1 JSP 2.0 커스텀 태그 개요

- 커스텀 태그 작성 절차

커스텀 태그를 만들기 위해서는 태그 처리 클래스 작성, TLD 파일 작성이 필요하다. 물론 만들어 놓은 커스텀 태그를 활용해 보기 위해서는 태그를 사용하는 JSP 파일이 필요할 것이다.

순서	이름	장소	파일확장자	내용
1	태그 처리기 (Tag Handler)	[Java Resources: src]	*.java	태그를 처리하는 자바파일로 클래스 SimpleTagSupport를 상속(확장)하여 작성
2	태그 라이브러리 기술(TDL)	[WEB-INF]/[tlds]	*.tld	태그 처리기에서 만든 태그를 JSP 페이지 에서 사용할 수 있도록 태그 이름을 등록 하는 절차
3	태그 활성 JSP 프로그램	[WEBContent]	*.jsp	태그 라이브러리 기술에서 등록한 태그이 름을 taglib 지시자를 사용하여 이용

태그 클래스를 작성하기 위해서는 다음 클래스 SimpleTagSupport를 상속하여 메소드 doTag()를 구현해야 한다.

```
javax.servlet.jsp.tagext.SimpleTagSupport
```

그러므로 태그 처리기 클래스 HelloCustomTag를 작성한다면 자바 소스는 다음과 같다.

```
public class HelloCustomTag extends SimpleTagSupport {
```

```

public void doTag() throws IOException {
    //구현
}
}

```

- 클래스 SimpleTagSupport

새로이 만드는 커스텀 태그 처리기 클래스는 SimpleTagSupport를 상속받아 메소드 doTag()를 재정의해야 한다. 클래스 SimpleTagSupport는 인터페이스 SimpleTag를 구현한 클래스로 다음과 같은 주요 메소드를 제공한다.

반환 유형	메소드	설명
void	doTag()	태그가 수행해야 할 일을 처리하는 메소드로, 태그 처리 클래스에서 오버라이딩(overriding)해서 구현.
JspFragment	getJspBody()	태그의 몸체 부분을 반환
JspContext	getJspContext()	페이지 context를 반환하며, 주로 getJspContext.getOut()을 통해 출력에 사용할 JspWriter 객체를 얻음

21.2.2 첫 커스텀 태그 만들기

- 문자열 출력 커스텀 태그 작성 절차

간단한 문자열을 출력하는 커스텀 태그를 작성하자. 즉 태그 hello는 몸체는 없으며 문자열 "Hello Custom Tag!"를 출력하는 태그이다. 태그 hello에서 접두어인 prefix를 myfirsttag로 지정하면 다음과 같이 사용할 수 있다.

```
<myfirsttag:hello />
```

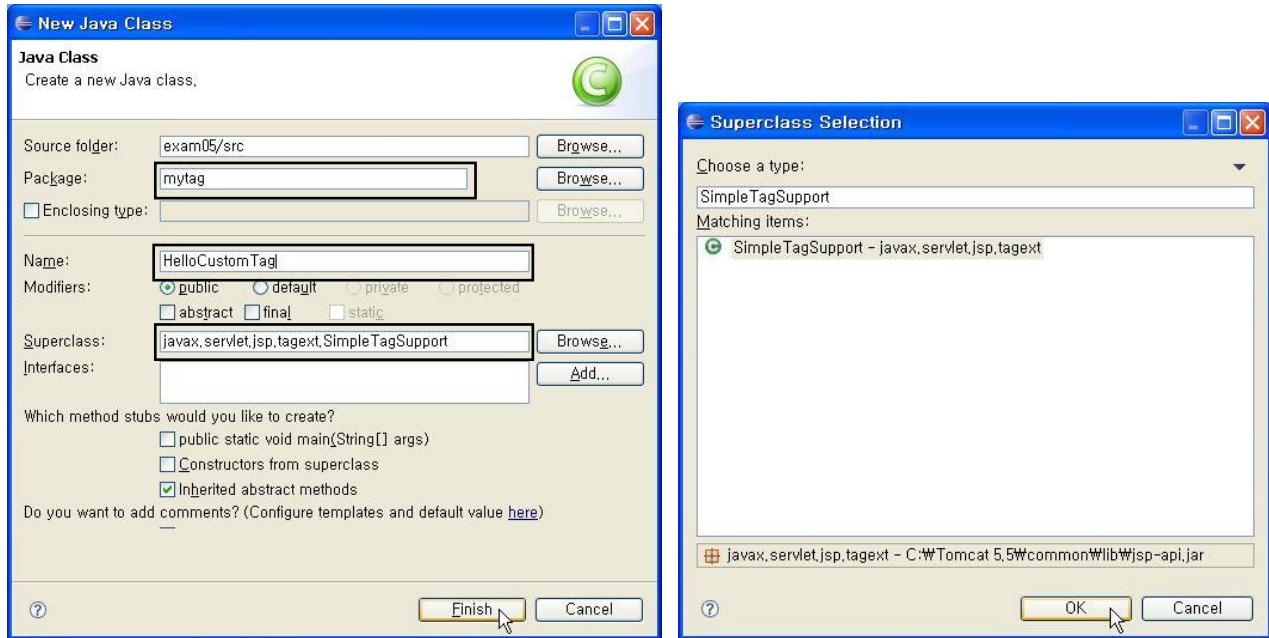
태그 hello를 만들기 위해서는 HelloCustomTag.java와 HelloCustomTag.tld가 필요하며, 태그 hello를 테스트하기 위해서는 HelloCustomTag.jsp가 필요하다.

순서	이름	장소	파일 이름
1	태그 클래스 작성	[Java Resources: src]	HelloCustomTag.java
2	TLD 작성	[WEB-INF]/[tlds]	HelloCustomTag.tld
3	태그 활용 JSP 작성	[WEContent]	HelloCustomTag.jsp



- 커스텀 태그를 위한 자바 파일 작성

커스텀 태그 hello를 처리하는 태그 처리기 자바 프로그램 HelloCustomTag.java을 작성하기 위해 이클립스에서 [New Java Class] 대화상을 실행한다. 커스텀 태그 처리 프로그램의 클래스는 HelloCustomTag로, 패키지는 mytag로, 상위클래스인 Superclass는 [javax.servlet.jsp.tagext.SimpleTagSupport]로 각각 지정한다.



태그 처리는 메소드 doTag()에서 구현한다. 메소드 getJspContext().getOut();을 통해 출력에 사용할 JspWriter 객체를 out에 저장하여 원하는 문자열을 출력한다.

- HelloCustomTag.java

```
package mytag;  
import java.io.IOException;  
import javax.servlet.jsp.JspWriter;  
import javax.servlet.jsp.tagext.SimpleTagSupport;  
  
public class HelloCustomTag extends SimpleTagSupport {  
    public void doTag() throws IOException {  
        JspWriter out = getJspContext().getOut();  
        out.println("<font color=blue>");  
        out.println("Hello My Custom Tag!!!");  
        out.println("</font>");  
    }  
}
```

- TLD 파일 작성

태그 라이브러리 기술인 TLD(Tag Library Descriptors)는 이미 만든 태그 처리기를 태그로 사용하기 위해 태그를 선언하는 파일이다. TLD 파일은 확장자 tld이며 [WEB-INF] 하부폴더 [tlds]를 만들어 저장한다.

TLD <?xml ...?> 태그로 시작하며, <taglib> 태그에서 속성 xmlns, xsi:schemaLocation, version 등으로 문서의 규칙을 담고 있는 스키마 파일과 요구되는 JSP 버전을 지정한다.

TLD 파일에서 태그 <taglib>...</taglib>는 HTML 파일의 시작과 끝인 <HTML>...</HTML>과 같다고 할 수 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
         version="2.0">
...
</taglib>
```

태그 <taglib> 내부에 사용하는 주요한 태그를 살펴보면 다음과 같다.

태그	설명
description	태그 라이브러리 사용 설명
display-name	개발 도구에서 표시되는 이름
icon	개발 도구에서 표시되는 아이콘
tlib-version	태그 라이브러리 버전
short-name	현재 태그 라이브러리의 간단한 이름
uri	태그 라이브러리를 구별하는 URI
tag	해당 태그 라이브러리에 포함되어 있는 태그를 기술

실제 JSP 파일에서 사용할 태그는 <taglib> 내부에 <tag>로 선언한다. <tag>를 구성하는 내부태그를 살펴보면 다음과 같다.

태그	설명
description	태그 라이브러리 사용 설명
display-name	개발 도구에서 표시되는 이름
icon	개발 도구에서 표시되는 아이콘
name	사용자 정의 태그 이름으로, TLD 내에서 유일한 이름이어야 함.
tag-class	해당 태그를 정의한 태그 처리기 클래스를 지정하며, 패키지 이름을 포함한 이름으로 입력
tei-class	TEI 파일의 클래스 이름을 지정하며, 일반적으로 javax.servlet.jsp.tagext.TagExtranInfo를 상속받아서 구현
body-content	태그의 몸체인 시작 태그와 끝 태그 사이에 들어갈 내용의 형태를 지정하며, 값은 empty, scriptless, tagdependent 중 하나이고, 몸체가 없으면 empty로 하며, scriptless는 스크립트 요소는 올 수 없고, 일반 문자열, 표현언어, 커스텀 태그를 허용하며, tagdependent는 태그 처리기에 의해 처리됨.
variable	스크립팅 변수 정보를 표현
attribute	현재 태그에 대한 속성 정의

<tag>에서 <name>은 가장 중요한 태그이름을 지정하며, <tag-class>는 이미 구현한 태그 처리기의 클래스 이름을 지정하고, <body-content>는 태그 몸체에 대한 특성에 따라 empty, tagdependent, scriptless 중에 하나를 지정해야 한다. 만일 <body-content> 값이 empty라면 태그 사이에 어떠한 값도 있어서는 안된다. 즉 커스텀 태그의 형태가 <prefix:tagname .../>와 같이 작성해 주어야 한다. 속성 값 scriptless는 JSP 스크립트 요소를 사용할 수 없고, 일반 문자열, 표현언어, 다른 커스텀 태그를 하용하며, 마지막으로 tagdependent는 사용자 정의 태그의 태그 처리기에서 직접 처리하는 것을 의미한다.

지금 작성하는 태그 이름을 hello로 하려면 <name>을 hello, 이미 만든 태그 처리기가 mytag.HelloCustomTag이므로 <tag-class>는 mytag.HelloCustomTag로, <body-content>는 몸체가 없으므로 empty로 지정한다.

다음은 태그 처리기 HelloCustomTag를 태그 hello로 선언하는 TLD 파일 HelloCustomTag.tld이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
         version="2.0">
    <description>처음 만드는 커스텀 태그</description>
    <display-name>HelloCustomTag</display-name>
    <tlib-version>1.2</tlib-version>
    <jsp-version>2.0</jsp-version>
    <short-name>HelloCustomTag</short-name>
    <tag>
        <description>문자열 Hello My Custom Tag!!! 출력 예제</description>
        <display-name>문자열 출력</display-name>
        <name>hello</name>
        <tag-class>mytag.HelloCustomTag</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

- 첫 커스텀 태그 hello를 사용하는 JSP 파일 작성

이제 처음으로 만든 커스텀 태그 hello를 사용하는 JSP 프로그램을 작성하자. 커스텀 태그 hello를 사용하기 위해서는 지시자 taglib가 필요하다. 지시자 taglib에서 속성 uri에 TLD 파일을 지정하고, prefix에 원하는 태그의 접두어를 지정한다.

```
<%@ taglib prefix="myfirsttag" uri="/WEB-INF/tlds>HelloCustomTag.tld" %>
```

위와 같이 prefix="myfirsttag"로 하면, <myfirsttag:hello />로 커스텀 태그 hello를 사용할 수 있다.

- HelloCustomTag.jsp

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

```

<head><meta charset="UTF-8">
<title>커스텀 태그</title>
</head>
<body>
<%@ taglib uri="/WEB-INF/tlds/HelloCustomTag.tld" prefix="myfirsttag" %>
    <h2 align="center">첫 커스텀 태그 예제 </H2>
    <center><HR>
        <b><myfirsttag:hello /></b>
    </center>
</body>
</html>

```

[결과]



21.3. 커스텀 태그 속성 처리

21.3.1 속성이 있는 커스텀 태그 만들기

- 테이블 출력 커스텀 태그 작성 절차

selectstudentCPbean.java 프로그램에서 학생 테이블 student를 조회하여 출력하는 부분을 커스텀 태그로 작성해 보자.

즉 지금 만드는 커스텀 태그 print는 몸체가 있으며, 속성으로 border와 bgcolor 그리고 list를 가지고 테이블 형태로 list에 저장된 학생 정보를 출력하는 태그이다. 즉 자바빈즈와 자바 스크립트릿 코드 그리고 테이블 마크업 언어가 혼재되어 있는 부분을 다음과 같이 커스텀 태그 print로 대체하고자 한다. 커스텀 태그 print는 다음과 같이 속성 border, bgcolor, list가 있으며 제목이 몸체로 표현된다.

```

<%@ taglib uri="/WEB-INF/tld>SelectStudentTag.tld" prefix="mytag" %>
<jsp:useBean id="stdtdb" class="studentmgr.StudentDatabaseCP" scope="page" />

<mytag:print border="1" bgcolor="skyblue" list="${stdtdb.studentList}" >
    학생정보조회
</mytag:print>

```

태그 print를 만들기 위한 태그 처리기 SelectStudentTag.java, TDL 파일인 SelectStudentTag.tld, 그리고 태그 print를 테스트하기 위한 SelectStudentTag.jsp을 작성한다.

순서	이름	장소	파일 이름	비고
1	태그 클래스 작성	[Java Resources: src]	SelectStudentTag.java	속성에 대한

				setter / getter
2	TLD 작성	[WEB-INF]/[tlds]	SelectStudentTag.tld	속성 처리
3	태그 활용 JSP 작성	[WEBContent]	SelectStudentTag.jsp	

- 태그 처리기 자바 파일 작성

태그 처리기 자바 프로그램 SelectStudentTag.java는 패키지를 mytag로, 상위클래스 SimpleTagSupport를 상속받아 구현한다. 태그 속성을 처리하기 위해 속성이름인 border, bgcolor, list를 멤버변수로 정의하며, 이 변수들의 setter와 getter를 구현해야 한다.

태그 속성인 border, bgcolor는 각각 테이블 태그에서 두께와 첫 행 제목의 배경색으로 이용되며, 기술하지 않을 수 있는 선택 속성의 기본값은 2와 white이다. 그러므로 멤버 변수의 초기값으로 기본값을 저장한다.

```
public class SelectStudentTag extends SimpleTagSupport {
    // 커스텀 태그의 속성을 위한 변수
    private int border = 2;
    private String bgcolor = "white";
    private Vector<StudentEntity> list;
    public int getBorder() {
        return border;
    }
    public void setBorder(int border) {
        this.border = border;
    }
    ...
}
```

태그 처리를 위해 메소드 doTag()에서 getJspContext().getOut()을 통해 출력에 사용할 JspWriter 객체를 out에 저장한다. 또한 태그 몸체 처리를 위해 메소드 getJspBody()로 JspFragment 객체를 body에 저장한다. 몸체를 출력하려면 변수 body를 이용하여 메소드 invoke()를 호출한다. 몸체 처리에 인자가 필요하면 사용할 수 있으며, 여기서는 단순한 문자열인 제목을 출력하므로 null을 인자로 호출한다.

```
public void doTag() throws IOException, JspException {
    JspWriter out = getJspContext().getOut();
    JspFragment body = getJspBody();
    if(body != null){
        out.println("<center><h2>");
        body.invoke(null);
        out.println("</h2></center>");
    }
    ...
}
```

커스텀 태그 print의 속성 border와 bgcolor는 table 태그의 속성 border와 테이블 첫행의 bgcolor를 지정하기 위한 속성이다. 메소드 getBorder()와 getBcolor()를 사용하여 테이블 태그의 속성을 구성한 후 출력한다.

```
out.println("<table width=100% cellpadding=1 " +
            "border=" + getBorder() + " >" +
            "<tr align=center " + " bgcolor=" + getBcolor() + " >" +
```

태그 처리를 위한 메소드 doTag() 내부에서 for문을 이용하여 속성 list에 원소로 저장된 StudentEntity를 하나씩 꺼내 테이블 태그를 이용하여 각각의 저장값을 출력한다.

```
for( StudentEntity stdt : list ) {
    out.println("<tr align=center>" +
                "<td>" + stdt.getId() + "</td>" +
                ...
}
```

다음은 커스텀 태그 print를 위한 태그 처리기 SelectStudentTag.java 전 소스이다.

```
package mytag;
import java.io.IOException;
import java.util.ArrayList;
import studentmgr.StudentEntity; // studentmgr/StudentDatabaseCPjava StudentEntity.java 필요
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class SelectStudentTag extends SimpleTagSupport {
    // 커스텀 태그의 속성을 위한 변수
    private int border = 2;
    private String bgcolor = "white";
    private ArrayList<StudentEntity> list;
    public void doTag() throws IOException, JspException {
        JspWriter out = getJspContext().getOut();
        JspFragment body = getJspBody();
        if(body != null){
            out.println("<center><h2>");
            body.invoke(null);
            out.println("</h2></center>");
        }
        // 태그 바디 처리후 student 테이블 출력을 위한 테이블 구성
        int counter = list.size();
        if (counter > 0) {
```

```

        out.println("</center>");
        out.println("<table width='100%' cellpadding='1' " +
            "border=" + getBorder() + " >" +
            "<tr align=center " + " bgcolor=" + getBcolor() + " >" +
            "<th>학번</th>" +
            "<th>이름</th>" +
            "<th>아이디</th>" +
            "<th>비밀번호</th>" +
            "<th>학과번호</th>" +
            "<th>주민등록번호</th>" +
            "<th>휴대폰1</th>" +
            "<th>주소</th>" +
            "<th>이메일</th>" +
            "<th>등록일자</th>" +
            "</tr>");
        for( StudentEntity stdt : list ) {
            out.println("<tr align='center'>" +
                "<td>" + stdt.getSd_num() + "</td>" +
                "<td>" + stdt.getSd_name() + "</td>" +
                "<td>" + stdt.getSd_id() + "</td>" +
                "<td>" + stdt.getSd_passwd() + "</td>" +
                "<td>" + stdt.getS_num() + "</td>" +
                "<td>" + stdt.getSd_jumin() + "</td>" +
                "<td>" + stdt.getSd_hpone() + "</td>" +
                "<td>" + stdt.getSd_address() + "</td>" +
                "<td>" + stdt.getSd_email() + "</td>" +
                "<td>" + stdt.getSd_date() + "</td>" + "</tr>");
        }
        out.println("</table>");
        out.println("</center>");
    }
    out.println("<p><hr>조회된 학생 수가 " + list.size() + "명 입니다.");
}

public int getBorder() {
    return border;
}

public void setBorder(int border) {
    this.border = border;
}

public String getBcolor() {

```

```

        return bgcolor;
    }

    public void setBgcolor(String bgcolor) {
        this.bgcolor = bgcolor;
    }

    public ArrayList<StudentEntity> getList() {
        return list;
    }

    public void setList(ArrayList<StudentEntity> list) {
        this.list = list;
    }
}

```

- TLD 파일 작성

위에서 만든 커스텀 태그 처리기 SelectStudentTag를 태그 print로 사용하기 위해 태그 처리기와 태그 이름을 연결하는 TLD 파일 SelectStudentTag.tld를 작성하자.

태그 이름을 지정하는 태그 <name>을 print로, 태그 처리기를 지정하는 <tag-class>는 mytag. SelectStudentTag로, <body-content>는 몸체가 단순한 문자열이므로 scriptless로 지정한다.

```

<description>학생 테이블 출력 태그</description>
<display-name>학생 테이블 출력</display-name>
<name>print</name>
<tag-class>mytag.SelectStudentTag</tag-class>
<body-content>scriptless</body-content>

```

태그의 속성을 선언하려면 <attribute>태그가 필요하다. 속성 list를 위한 <attribute> 태그는 다음과 같으며, <name>에는 이름, <type>에는 속성의 자료유형을 기술한다. 또한 속성 list는 필수 속성으로 지정해야 하므로 태그 <required>를 true로 지정하고, 표현언어도 지원하려면 <rtextvalue>를 true로 지정한다.

```

<attribute>
    <name>list</name>
    <type>java.util.Vector</type>
    <required>true</required>
    <rtextvalue>true</rtextvalue>
</attribute>

```

<attribute> 태그에는 내부에 다음과 같은 서브태그를 사용하며 그 의미는 다음과 같다.

서브태그	설명
description	태그 라이브러리 사용 설명
name	태그로 사용될 태그 이름으로 반드시 기술해야 하는 필수 속성

required	필수 속성이면 true로 지정. 선택 속성이면 false로 지정. 기술하지 않으면 기본값은 false
rtexprvalue	실행시간에 표현언어의 표현식을 사용하면 true, 아니면 false로 지정. 기술하지 않으면 기본값은 false
type	속성 값의 자료유형으로, 기술하지 않으면 기본값은 java.lang.String

속성 border을 반드시 지정해야 하는 필수 속성으로 만들려면 태그 <required>를 true로 지정하고, 표현언어도 지원하려면 <rtexprvalue>를 true로 지정한다.

```
<attribute>
    <name>border</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
</attribute>
```

속성 border는 태그 <required>를 false로 지정하여 선택(옵션) 속성으로 만들고, 표현언어는 지원하지 않게 <rtexprvalue>를 false로 지정한다.

```
<attribute>
    <name>bgcolor</name>
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
</attribute>
```

다음은 태그 처리기 SelectStudentTag를 태그 print로 선언하는 TLD 파일 SelectStudentTag.tld이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
    <description>데이터베이스 처리 태그</description>
    <tlib-version>1.2</tlib-version>
    <jsp-version>2.0</jsp-version>
    <short-name>SelectStudentTag</short-name>
    <tag>
        <description>학생 테이블 출력 태그</description>
        <display-name>학생 테이블 출력</display-name>
        <name>print</name>
        <tag-class>mytag.SelectStudentTag</tag-class>
        <body-content>scriptless</body-content>
        <attribute>
            <name>list</name>
```

```

<type>java.util.Vector</type>
<required>true</required>
<rtpvalue>true</rtpvalue>
</attribute>
<attribute>
    <name>border</name>
    <required>true</required>
    <rtpvalue>true</rtpvalue>
</attribute>
<attribute>
    <name>bgcolor</name>
    <required>false</required>
    <rtpvalue>false</rtpvalue>
</attribute>
</tag>
</taglib>

```

- 커스텀 태그 print를 사용하는 JSP파일 작성

이제 만든 커스텀 태그 print를 사용하는 JSP 프로그램을 작성하자. 커스텀 태그 print를 사용하기 위해서는 지시자 taglib에서 속성 uri에 TLD 파일을 지정하고, prefix에 원하는 태그의 접두어를 지정한다.

```
<%@ taglib uri="/WEB-INF/tlds>SelectStudentTag.tld" prefix="mytag" %>
```

데이터베이스 연결과 질의를 위한 자바빈즈인 studentmgr.StudentDatabaseCP를 stdtdb로 선언한다.

```
<jsp:useBean id="stdtdb" class="studentmgr.StudentDatabaseCP" scope="page" />
```

커스텀 태그 print는 <mytag:print ...>로 사용하며 몸체에는 출력하려는 제목을 입력하고, 속성 border와 bgcolor에는 각각 테이블 두께와 테이블 첫 행의 제목 부분 배경색을 지정한다. 또한 속성 list는 출력하려는 학생 정보가 원소로 저장된 Vector를 지정해야 하므로 표현언어 \${stdtdb.studentList}로 지정한다. 즉 표현언어 \${stdtdb.studentList}는 자바빈즈 객체 stdtdb로부터 stdtdb.getStudentList()을 호출하여 그 결과를 속성 list에 저장한다.

```
<mytag:print border="1" bgcolor="skyblue" list="${stdtdb.studentList}" >
    학생정보조회
</mytag:print>
```

- SelectStudentTag.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>커스텀 태그</title>
```

```

</head>
<body>
<h2> 커스텀 태그를 이용한 student 조회 프로그램 </h2>
<hr>
<%@ taglib uri="/WEB-INF/tlds>SelectStudentTag.tld" prefix="mytag" %>
<jsp:useBean id="stdtdb" class="studentmgr.StudentDatabaseCP" scope="page" />
<mytag:print border="1" bgcolor="#99CC00" list="${stdtdb.studentList}" >
    학생정보조회
</mytag:print>
</body>
</html>

```

[결과]

학번	이름	아이디	비밀번호	학과 번호	주민등록번호	휴대폰1	주소	이메일	등록일자
06010001	김정수	javajsp	java1234	01	8710101653872	01012345678	서울시 서대문구 창전동	java12@naver.com	2009-07-07 22:51:28.0
95010002	김수현	jdbcmania	mania12	01	7505052129875	0113452468	서울시 서초구 양재동	jdbcmania@naver.com	2009-07-07 22:51:28.0
98040001	공지영	gonji	mania12	04	7812242250875	01612657455	부산광역시 해운대구 반송동	gonji@nate.com	2009-07-07 22:51:28.0
02050001	조수영	water	java1234	05	8303161204369	0107681499	대전광역시 중구 은행동	water@korea.com	2009-07-07 22:51:28.0
94040002	최경란	novel	novel2468	04	7410091893562	0119872455	경기도 수원시 장안구 이복동	novel@naver.com	2009-07-07 22:51:28.0
08020001	안익태	korea	korea99	02	8908151734120	0168452345	서울시 마포구 대흥동	korea@nate.com	2009-07-07 22:51:29.0
90030001	전미정	dbcp	dbcp1234	03	7106192345623	0111294567	서울시 영등포구 당산동	dbcp@dreamwiz.com	2009-07-08 11:40:11.0

조회된 학생 수가 7명입니다.

21.4. JSP 2.0 태그 파일의 커스텀 태그

21.4.1 태그 파일 개요

- 태그 파일의 장점

커스텀 태그를 사용하면 JSP 소스가 한결 간단해지며 재사용성이 높다. 그러나 커스텀 태그를 잘 만들어 사용하려면 태그 처리기 개발을 위한 자바 프로그래밍 기술이 요구된다. 또한 태그 처리기 자바 프로그램에서 표현 부분이 많아 HTML 코드 출현 빈도가 많으면, 태그 처리기의 메소드 doTag() 구현 부분이 더욱 복잡해지는 문제점을 안고 있다.

JSP 2.0에서 지원하는 태그 파일을 이용한 커스텀 태그의 구현은 자바에 익숙하지 않은 비개발자도 재사용이 가능한 커스텀 태그를 작성할 수 있게 할 뿐만 아니라 프로그래머도 더 쉽게 작업할 수 있도록 도와준다. 즉 태그 파일은 일반 JSP 프로그램과 같이 JSP 스크립트 요소, HTML 코드, JSTL, 그리고 표현언어를 사용할 수 있으므로 자바 코드를 작성할 필요없이 손쉽게 커스텀 태그를 만들어 사용할 수 있도록 한다.

태그 파일은 태그 처리기 자바 프로그램보다 제약이 있을 수 있으나 HTML 코드와 같은 표현 부분이 많은 모듈을 태그로 만든다면 태그 처리기보다 적합하다. 반면 태그처리기 클래스를 이용한 커스텀 태그는 JSP 페이지에서 애플리케이션 로직을 재사용할 때 더 효과적이다.

- 태그 파일로 커스텀 태그 작성 절차

태그 파일을 이용하여 커스텀 태그를 만드는 과정은 TLD 작성 과정이 필요 없으며, 태그 처리기 대신에 태그 파일을 작성하는 것으로 완료된다. 즉 태그 파일로 커스텀 태그를 만드는 과정은 태그 파일 작성 그리고 만들어 놓은 커스텀 태그를 활용해 보기 위한 JSP 파일 작성 과정만이 필요하다.

순서	이름	장소	파일 확장자	설명
1	태그 파일(Tag File)	[WEB-INF]/[tags]	*.tag	태그를 처리하는 태그 파일로 JSP 파일과 비슷해 작성이 쉽고, 간단함.
2	태그 활용 JSP 작성	[WEBContent]	*.jsp	위에서 만든 태그 파일 이름을 태그로 사용하며, taglib 지시자를 사용하여 이용

태그 파일을 만드는 경우, 사용할 커스텀 태그 이름으로 태그 파일 이름 tagname을 지정하고 확장자는 tag로 한다. 폴더 [WEB-INF] 하부에 폴더 [tags]를 만든 후, 이 [tags] 폴더에 태그 파일 tagname.tag를 저장한다.

- [WEB-INF]/[tags]/[tagname.tag]

21.4.2. 태그 파일로 커스텀 태그 만들기

- 문자열 출력 커스텀 태그 작성 절차

앞에서 구현한 간단한 문자열을 출력하는 커스텀 태그 hello를 태그 파일로 작성하자. 즉 태그 hello는 몸체가 없으며 문자열 "Hello Custom Tag using Tag File!!!~~~"을 출력하는 태그이다

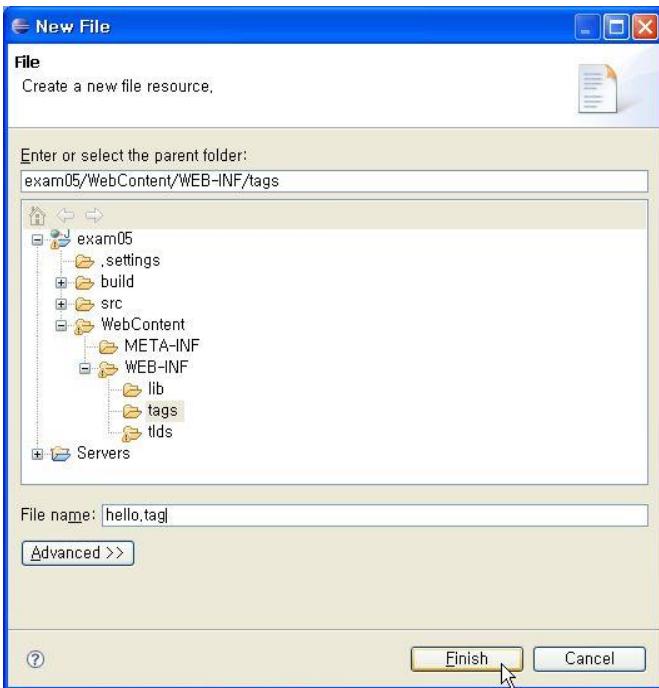
```
<mytag:hello />
```

순서	이름	장소	파일 이름
1	태그 파일	[WEB-INF]/[tags]	hello.tag
2	태그 활용 JSP 작성	[WEBContent]	HelloCustomTagFile.jsp

- 커스텀 태그를 위한 태그 파일 작성

커스텀 태그 hello를 처리하는 태그 프로그램 hello.tag를 작성하자. 가장 먼저 태그 파일을 저장할 폴더 tags를 폴더 [WEB-INF] 하부에 [tags]로 만든다. 태그 파일 hello.tag는 이클립스에서 일반 파일로 만들어, [WEB-INF]/[tags] 폴더에 저장한다.

이클립스에는 태그 파일을 만드는 특별한 방법을 제공하지 않으므로, 폴더 [WEB-INF]/[tags]에서 메뉴 [New]/[File]로 확장자를 포함한 파일 이름 hello.tag를 모두 입력하여 생성한다.



속성	설명
display-name	태그에 대한 표시 이름을 기술하며, 일반적으로 확장자 tag를 제외한 파일 이름을 지정
body-content	몸체의 형태에 따라 empty, tagdependent, 또는 scriptless를 기술하며, 기본값은 scriptless
dynamic-attribute	동적 속성을 지정하는 부분으로 <이름, 값>의 쌍의 Map 형식으로 관리
small-icon	개발도구에서 태그를 대표하는 작은 아이콘으로 지정
large-icon	개발도구에서 태그를 대표하는 큰 아이콘으로 지정
description	태그에 대한 설명
example	태그에 대한 예
language	JSP page 지시어와 같은 지원 언어
import	JSP page 지시어와 같은 해당 클래스 import
pageEncoding	JSP page 지시어와 같은 페이지 인코딩 방법
isELIgnored	JSP page 지시어와 같은 표현언어 사용 여부

태그 파일 hello.tag에서 태그 처리 프로그램은 매우 간단하게 출력하려는 문자열을 그대로 기술한다. 즉 태그 파일은 JSP 소스와 같이 문자열이 그대로 출력된다.

```
<%@ tag body-content="empty" pageEncoding="UTF-8"%>
<font color="blue"><b>
Hello Custom Tag using Tag File !!!~~
</b></font>
```

- 커스텀 태그 hello를 사용하는 JSP 파일 작성

이제 태그 파일로 만든 커스텀 태그 hello를 사용하는 JSP 프로그램을 작성하자. 자바 클래스로 만든 태그 처리기 커스텀 태그의 사용과 같이, 태그 파일로 만든 커스텀 태그 hello를 사용하기 위해서는 지시자 taglib가 필요하다. 지시자 taglib에서 속성 uri 대신에 tagdir이 필요한데, tagdir에는 태그 파일 hello.tag가 저장된 폴더를 지정하며, prefix에 원하는 태그의 접두어를 지정한다. 여기서 주의할 점은 태그 처리기 클래스를 지정하는 것과 다르게 속성은 uri가 아니라 tagdir이며, 지정할 값은 파일 이름이 아니라 폴더 "/WEB-INF/tags"이라는 것이다.

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

위와 같이 속성 tagdir를 "/WEB-INF/tags"로 지정하고, prefix를 "mytag"로 지정하면, 태그 파일로 만든 hello 태그를 <mytag:hello />로 사용할 수 있다.

HelloCustomTagFile.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>커스텀 태그</title>
</head>
<body>
    <%@ taglib tagdir="/WEB-INF/tags" prefix="mytag" %>
    <h2 align="center">태그 파일을 만든 커스텀 태그 예제 </h2>
    <center><hr>
        <mytag:hello />
    </center>
</body>
</html>
```

[결과]



21.4.3. 태그 파일 커스텀 태그의 속성 처리

- 태그 파일로 만드는 테이블 출력 커스텀 태그 작성 절차

SelectStudentTag.jsp 프로그램에서 사용하는 커스텀 태그 print를 태그 파일 print로 작성하여 사용하도록 프로그램을 수정해 보자. 즉 태그 print를 만들기 위한 태그 파일 print.tag, 그리고 태그 print를 테스트하기 위한 JSP 프로그램 HelloCustomTagFile.jsp를 작성하자.

순서	이름	장소	파일 이름
1	태그 파일	[WEB-INF]/[tags]	print.tag
2	태그 활용 JSP 작성	[WEBContent]	SelectStudentTagFile.jsp

- 태그 파일 작성

태그 파일 print.tag는 다음 tag 지사자로 시작한다. 태그 print의 몸체는 제목이므로 속성 body-content를 "scriptless"로 지정하고 description에는 설명을 쓰며, java.util.Vector를 사용해야 하므로 import에 지정한다.

태그 hello에서 사용하는 속성 처리를 위해 속성이름인 border, bgcolor, list를 지시자 attribute로 지정한다. 태그의 속성을 필수 속성으로 지정하려면 required를 true로 지정한다.

```
<%@ tag body-content="scriptless" pageEncoding="UTF-8"
   description="테이블 student 레코드 출력태그"
   import="java.util.Vector" %>
<%@ attribute name="bgcolor" required="true" %>
<%@ attribute name="border" required="true" %>
<%@ attribute name="list" required="true" type="java.util.ArrayList"%>
```

태그 파일에서 사용하는 지시자 attribute는 다음과 같은 속성을 제공한다.

속성	설명
description	속성에 대한 설명
name	속성의 이름으로 라이브러리에서 유일해야 함
required	속성이 필수이면 true, 옵션이면 false이며, 기본값은 false
rteprvalue	실행 시간에 표현언어의 표현식을 사용하면 true 아니면 false로 지정, 기술하지 않으면 기본값은 true
type	속성값의 자료유형으로 기본값은 String

태그 파일에서 표현언어를 사용하려면 JSP 파일과 같이 taglib 지시자를 사용해야 한다. JSTL set 태그로 list의 크기를 변수 count에 저장하자

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

<c:set var="count" value="<% = list.size() %>" />
```

if 태그를 사용하여 테이블 student를 조회한 레코드 수인 count가 0보다 크면 테이블 형태로 테이블 조회 결과를 출력한다.

```
<c:if test="${count > 0}" var="bool" >
...
</c:if>
```

이제 HTML과 표현언어를 함께 사용하여 태그 처리기 SelectStudentTag.jsp의 메소드 doTag()에서 구현했던 부분을 기술한다. 가장 먼저 태그 doBody를 사용해 몸체를 구성하는 제목을 출력한다.

```
<h2><jsp:doBody /></h2>
```

커스텀 태그 print의 속성 border와 bgcolor는 표현언어 \${border}와 \${bgcolor}를 사용하여 테이블 속성을 지정한다.

```
<table width=100% border="${border}" cellpadding=1>
  <tr bgcolor="${bgcolor}">
    <th>학번</th>
    <th>이름</th>
    ...
  </tr>
```

속성 list에 원소로 저장된 StudentEntity를 하나씩 빼고 테이블 태그를 이용하여 각각의 저장값을 출력하자. 이 모듈은 다음과 같이 JSTL 태그 forEach와 표현언어를 사용하여 구현하면 매우 쉽고 간단하게 처리할 수 있다.

```
<c:forEach var="stdt" items="${list}" >
  <tr>
    <td align=center>${stdt.sd_num}</td>
    <td align=center>${stdt.sd_name}</td>
    ...
  </tr>
</c:forEach>
```

태그 table을 이용하는 표현 부분이 많은 print태그는 태그 파일로 구현하는 것이 이전의 태그 처리기 자바 프로그램으로 구현하는 것보다 훨씬 쉽고 간결한 것을 알 수 있다.

print.tag

```
<%@ tag body-content="scriptless" pageEncoding="UTF-8"
       description="테이블 student 레코드 출력태그"
       import="java.util.ArrayList" %>
<%@ attribute name="bgcolor" required="true" %>
<%@ attribute name="border" required="true" %>
<%@ attribute name="list" required="true" type="java.util.ArrayList" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="count" value="<% = list.size() %>" />
<c:if test="${count > 0}" var="bool" >
  <center>
    <H2><jsp:doBody /></H2>
    <table width=100% border="${border}" cellpadding=1>
      <tr bgcolor="${bgcolor}">
```

```

<th>학번</th>
<th>이름</th>
<th>아이디</th>
<th>비밀번호</th>
<th>학과번호</th>
<th>주민등록번호</th>
<th>휴대폰</th>
<th>주소</th>
<th>이메일</th>
<th>등록일자</th>
</tr>
<c:forEach var="stdt" items="${list}" >
<tr>
<td align=center>${stdt.sd_num}</td>
<td align=center>${stdt.sd_name}</td>
<td align=center>${stdt.sd_id}</td>
<td align=center>${stdt.sd_passwd}</td>
<td align=center>${stdt.s_num}</td>
<td align=center>${stdt.sd_jumin}</td>
<td align=center>${stdt.sd_hpone}</td>
<td align=center>${stdt.sd_address}</td>
<td align=center>${stdt.sd_email}</td>
<td align=center>${stdt.sd_date}</td>
</tr>
</c:forEach>
</table>
</center>
</c:if>
<p><hr>조회된 학생 수가 ${count}명 입니다.

```

- 커스텀 태그 print를 사용하는 JSP 파일 작성

이제 태그 파일 hello.tag로 만든 커스텀 태그 print를 사용하는 JSP 프로그램을 작성하자. 커스텀 태그 print를 사용하기 위해서는 지시자 taglib에서 속성 taglib에 태그 파일이 저장된 폴더를 지정하고, prefix에 원하는 태그의 접두어 이름을 지정한다.

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

데이터베이스 연결과 질의를 위한 자바빈즈인 studentmgr.StudentDatabaseCP를 stdtdb로 선언한다.

```
<jsp:useBean id="stdtdb" class="studentmgr.StudentDatabaseCP" scope="page" />
```

파일 SelectStudentTag.jsp에서 작성한 것과 같이 커스텀 태그 print는 <mytag:print ...>로 사용하며 몸체

에는 출력하려는 제목을 입력하고, 속성 border, bgcolor 그리고 list를 지정한다.

```
<mytag:print border="1" bgcolor="skyblue" list="${stdtdb.studentList}" >
    학생정보조회
</mytag:print>
```

SelectStudentTagFile.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>커스템 태그</title>
</head>
<body>
<h2> 태그 파일을 이용한 커스텀 태그 print로 student 조회 프로그램 </h2>
<hr>
    <%@ taglib tagdir="/WEB-INF/tags" prefix="mytag" %>
    <jsp:useBean id="stdtdb" class="studentmgr.StudentDatabaseCP" scope="page" />
    <mytag:print border="1" bgcolor="skyblue" list="${stdtdb.studentList}" >
        학생정보조회
    </mytag:print>
</body>
</html>
```

[결과]

The screenshot shows a Microsoft Internet Explorer window with the title "커스텀 태그 - Microsoft Internet Explorer". The address bar displays the URL "http://localhost:8080/exam05>SelectStudentTagFile.jsp". The main content area is titled "학생정보조회" and contains a table with 10 columns and 8 rows of student data. The table includes columns for 학번 (Student ID), 이름 (Name), 아이디 (ID), 비밀번호 (Password), 학과 번호 (Major Number), 주민등록번호 (Resident Registration Number), 휴대폰 (Mobile Phone), 주소 (Address), 이메일 (Email), and 등록일자 (Registration Date). The data is as follows:

학번	이름	아이디	비밀번호	학과 번호	주민등록번호	휴대폰	주소	이메일	등록일자
06010001	김정수	javajsp	java1234	01	8710101653872	01012345678	서울시 서대문구 창전동	java12@naver.com	2009-07-07 22:51:28.0
95010002	김수현	jdbcmania	mania12	01	7505052129875	0113452468	서울시 서초구 양재동	jdbcmania@naver.com	2009-07-07 22:51:28.0
98040001	공지영	gonji	mania12	04	7812242250875	01612657455	부산광역시 해운대구 반송동	gonji@nate.com	2009-07-07 22:51:28.0
02050001	조수영	water	java1234	05	8303161204369	0107681499	대전광역시 중구 은행동	water@korea.com	2009-07-07 22:51:28.0
94040002	최경란	novel	novel2468	04	7410091893562	0119872455	경기도 수원시 장안구 이복동	novel@naver.com	2009-07-07 22:51:28.0
08020001	안익태	korea	korea99	02	8908151734120	0168452345	서울시 마포구 대흥동	korea@nate.com	2009-07-07 22:51:29.0
90030001	전미정	dbcp	dbcp1234	03	7106192345623	0111294567	서울시 영등포구 당산동	dbcp@dreamwiz.com	2009-07-08 11:40:11.0

At the bottom of the page, a message says "조회된 학생 수가 7명입니다." (The number of students queried is 7).

21.4.4. 태그 파일로 만드는 구구단 커스텀 태그

- 구구단 커스텀 태그 작성 절차

구구단을 출력하는 커스텀 태그 multiplication을 만들어 보자. 태그 multiplication의 태그 파일 multiplication.tag, 이 구구단 태그를 사용하는 JSP 프로그램 mutiplicationtable.jsp를 작성하자.

순서	이름	장소	파일 이름
1	태그 파일	[WEB-INF]/[tags]	multiplication.tag
2	태그 활용 JSP 작성	[WEBContent]	mutiplicationtable.jsp

- 태그 파일 작성

구구단 태그의 이름은 mutiplicationtable이고, 속성은 begin, end, bgcolor로 하며, 모두 옵션으로 다음과 같이 구성한다.

속성이름	필수 또는 옵션	기본값	설명
begin	옵션	2	구구단의 시작수
end	옵션	9	구구단의 끝 수
bgcolor	옵션	white	구구단 테이블의 배경색

태그 파일 multiplication.tag는 다음 tag 지시로 시작한다. 태그 print의 몸체는 제목이므로 속성 body-content를 "scriptless"로 지정하고 description에는 설명을 기술한다.

```
<%@ tag body-content="scriptless" pageEncoding="UTF-8"
   description="구구단(multiplication table) 출력태그"%>
```

태그 multiplication에서 사용하는 속성 처리를 위해 속성이름인 begin, end, bgcolor를 지시자 attribute로 지정한다. 모두 옵션 속성이므로 속성 required는 기술하지 않아도 된다.

```
<%@ attribute name="begin" %>
<%@ attribute name="end" %>
<%@ attribute name="bgcolor" %>
```

태그 속성 begin을 지정하지 않으면 구구단의 시작 기본값 2를 지정한다. 속성 end와 bgcolor도 같은 방법으로 구구단의 종료 기본 값인 9와 구구단 테이블 배경색 white를 지정한다.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:if test="${empty(begin)}" var="bool">
    <c:set var="begin" value="2" />
</c:if>
```

이제 HTML과 표현언어를 사용하여 태그 내용을 기술한다. 가장 먼저 태그 doBody를 사용해 몸체를 구성하는 제목을 출력한다.

```
<center>
<H2><jsp:doBody /></H2>
```

커스텀 태그 multiplication의 속성 bgcolor는 표현언어 \${bgcolor}를 사용하여 테이블 속성을 지정하며, 속성 begin, end도 표현언어를 이용하여 forEach 태그의 begin과 end로 지정한다.

```
<table width=100% border=1 cellpadding=1 bgcolor="${bgcolor}" >
    <c:forEach var="i" begin="${begin}" end="${end}" >
        <tr align="center" >
            <c:forEach var="j" begin="1" end="9" >
                <td>${i} * ${j} = ${i * j}</td>
            </c:forEach>
        </tr>
    </c:forEach>
</table>
```

multiplication.tag

```
<%@ tag body-content="scriptless" pageEncoding="UTF-8"
    description="구구단(multiplication table) 출력태그"%>
<%@ attribute name="begin" %>
<%@ attribute name="end" %>
<%@ attribute name="bgcolor" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <c:if test="${empty(begin)}" var="bool">
        <c:set var="begin" value="2" />
    </c:if>
    <c:if test="${empty(end)}" var="bool">
        <c:set var="end" value="9" />
    </c:if>
    <c:if test="${empty(begin)}" var="bool">
        <c:set var="bgcolor" value="white" />
    </c:if>
    <center>
        <H2><jsp:doBody /></H2>
        <table width=100% border=1 cellpadding=1 bgcolor="${bgcolor}" >
            <c:forEach var="i" begin="${begin}" end="${end}" >
                <tr align="center" >
                    <c:forEach var="j" begin="1" end="9" >
                        <td>${i} * ${j} = ${i * j}</td>
                    </c:forEach>
                </tr>
            </c:forEach>
        </table>
    </center>
```

```
<p><hr>
```

- 커스텀 태그 multiplication 이용

이제 태그 파일 multiplication.tag로 만든 커스텀 태그를 사용하는 JSP 프로그램을 작성하자.

가장 먼저 커스텀 태그 multiplication을 사용하기 위한 지시자 taglib를 기술한다.

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

커스텀 태그 multiplication은 <mytag:multiplication ...>으로 사용하며 몸체에는 출력하려는 제목을 입력하고, 필요하면 속성 begin, end, bgcolor를 지정한다. 다음과 같이 속성을 지정하지 않으면 기본 값으로 구구단이 출력되므로, 2단에서 9단까지 배경색 white로 출력된다.

```
<mytag:multiplication>
    구구단(2단에서 9단까지)
</mytag:multiplication>
```

multiplicationtable.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>커스텀 태그</title>
</head>
<body>
<h2> 태그 파일을 이용한 커스텀 태그 : multiplication </h2>
<hr>
    <%@ taglib tagdir="/WEB-INF/tags" prefix="mytag" %>
    <mytag:multiplication>
        구구단(2단에서 9단까지)
    </mytag:multiplication>
    <mytag:multiplication end="5" bgcolor="linen">
        구구단(2단에서 5단까지)
    </mytag:multiplication>
    <mytag:multiplication begin="3" end="7" bgcolor="yellow">
        구구단(3단에서 7단까지)
    </mytag:multiplication>
</body>
</html>
```

[결과]

커스텀 태그 - Microsoft Internet Explorer

파일(E) 편집(E) 보기(V) 즐겨찾기(A) 도구(I) 도움말(H)

주소(D) http://localhost:8080/exam05/multiplicationtable.jsp 미동

태그 파일을 이용한 커스텀 태그 : multiplication

구구단(2단에서 9단까지)

2 * 1 = 2	2 * 2 = 4	2 * 3 = 6	2 * 4 = 8	2 * 5 = 10	2 * 6 = 12	2 * 7 = 14	2 * 8 = 16	2 * 9 = 18
3 * 1 = 3	3 * 2 = 6	3 * 3 = 9	3 * 4 = 12	3 * 5 = 15	3 * 6 = 18	3 * 7 = 21	3 * 8 = 24	3 * 9 = 27
4 * 1 = 4	4 * 2 = 8	4 * 3 = 12	4 * 4 = 16	4 * 5 = 20	4 * 6 = 24	4 * 7 = 28	4 * 8 = 32	4 * 9 = 36
5 * 1 = 5	5 * 2 = 10	5 * 3 = 15	5 * 4 = 20	5 * 5 = 25	5 * 6 = 30	5 * 7 = 35	5 * 8 = 40	5 * 9 = 45
6 * 1 = 6	6 * 2 = 12	6 * 3 = 18	6 * 4 = 24	6 * 5 = 30	6 * 6 = 36	6 * 7 = 42	6 * 8 = 48	6 * 9 = 54
7 * 1 = 7	7 * 2 = 14	7 * 3 = 21	7 * 4 = 28	7 * 5 = 35	7 * 6 = 42	7 * 7 = 49	7 * 8 = 56	7 * 9 = 63
8 * 1 = 8	8 * 2 = 16	8 * 3 = 24	8 * 4 = 32	8 * 5 = 40	8 * 6 = 48	8 * 7 = 56	8 * 8 = 64	8 * 9 = 72
9 * 1 = 9	9 * 2 = 18	9 * 3 = 27	9 * 4 = 36	9 * 5 = 45	9 * 6 = 54	9 * 7 = 63	9 * 8 = 72	9 * 9 = 81

완료 로컬 인트라넷

커스텀 태그 - Microsoft Internet Explorer

파일(E) 편집(E) 보기(V) 즐겨찾기(A) 도구(I) 도움말(H)

주소(D) http://localhost:8080/exam05/multiplicationtable.jsp 미동

구구단(2단에서 5단까지)

2 * 1 = 2	2 * 2 = 4	2 * 3 = 6	2 * 4 = 8	2 * 5 = 10	2 * 6 = 12	2 * 7 = 14	2 * 8 = 16	2 * 9 = 18
3 * 1 = 3	3 * 2 = 6	3 * 3 = 9	3 * 4 = 12	3 * 5 = 15	3 * 6 = 18	3 * 7 = 21	3 * 8 = 24	3 * 9 = 27
4 * 1 = 4	4 * 2 = 8	4 * 3 = 12	4 * 4 = 16	4 * 5 = 20	4 * 6 = 24	4 * 7 = 28	4 * 8 = 32	4 * 9 = 36
5 * 1 = 5	5 * 2 = 10	5 * 3 = 15	5 * 4 = 20	5 * 5 = 25	5 * 6 = 30	5 * 7 = 35	5 * 8 = 40	5 * 9 = 45

구구단(3단에서 7단까지)

3 * 1 = 3	3 * 2 = 6	3 * 3 = 9	3 * 4 = 12	3 * 5 = 15	3 * 6 = 18	3 * 7 = 21	3 * 8 = 24	3 * 9 = 27
4 * 1 = 4	4 * 2 = 8	4 * 3 = 12	4 * 4 = 16	4 * 5 = 20	4 * 6 = 24	4 * 7 = 28	4 * 8 = 32	4 * 9 = 36
5 * 1 = 5	5 * 2 = 10	5 * 3 = 15	5 * 4 = 20	5 * 5 = 25	5 * 6 = 30	5 * 7 = 35	5 * 8 = 40	5 * 9 = 45
6 * 1 = 6	6 * 2 = 12	6 * 3 = 18	6 * 4 = 24	6 * 5 = 30	6 * 6 = 36	6 * 7 = 42	6 * 8 = 48	6 * 9 = 54
7 * 1 = 7	7 * 2 = 14	7 * 3 = 21	7 * 4 = 28	7 * 5 = 35	7 * 6 = 42	7 * 7 = 49	7 * 8 = 56	7 * 9 = 63

완료 로컬 인트라넷

22. MVC 모델 구현

22.1 프로젝트 개요

(1) 로그인 처리

로그인 처리 과정을 MVC 모델로 구현한다. 로그인 프로젝트는 뷰로 index.html, login.jsp가 있으며 모델은 자바빈즈인 UserBean.java, 컨트롤러는 서블릿 프로그램인 UserLogin.java로 구성된다.

MVC 요소	구현 프로그램	소스	기능
Model	자바빈즈	UserBean.java	컨트롤러인 UserLogin에서 사용하며 뷰로 전달 받은 사용자가 ID와 암호를 이용하여 로그인 인증 결과를 반환한다.
View	HTML	index.html	로그인을 위한 폼을 구성하여 사용자 ID와 암호를 컨트롤러인 UserLogin에 전달한다.
View	JSP	login.jsp	로그인 결과에 따라 성공하면 메시지를 출력하고 실패하면 다시 로그인 화면을 출력한다.
Controller	서블릿	UserLogin.java	뷰인 index.html에서 사용자 ID와 암호를 전달 받아 사용자 인증 결과를 얻어 다시 뷰인 login.jsp로 인증 결과 전송과 함께 제어가 이동된다.

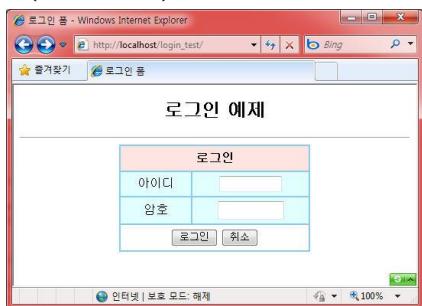
(2) 실행과정

로그인 프로젝트는 login_test로 정의하고 로그인 사용자 입력 폼을 구성하는 index.html을 작성한다.

로그인 처리를 MVC 모델로 구성한 경우 처리되는 과정이다.

- ① 뷰인 index.html은 사용자 ID와 암호를 입력받아 컨트롤러인 UserLogin에 전달한다.
- ② 컨트롤러인 UserLogin은 모델인 UserBean에게 index.html에서 전송받은 사용자 ID와 암호를 전송하여 사용자 인증 결과를 기다란다.
- ③ 모델 UserBean은 전송 받은 사용자 ID와 암호를 사용하여 이미 프로그램에서 정해진 사용자와 비교하여 사용자 인증 결과를 컨트롤러인 UserLogin에게 전달한다.
- ④ 컨트롤러인 UserLogin은 모델 UserBean에게 전달 받은 사용자 인증 결과를 다시 뷰인 login.jsp에 전송하고, 제어도 함께 이동한다.
- ⑤ 뷰인 login.jsp는 컨트롤러 UserLogin으로부터 전달 받은 사용자 인증 결과에 따라 적절한 출력을 처리한다.

뷰(index.html)



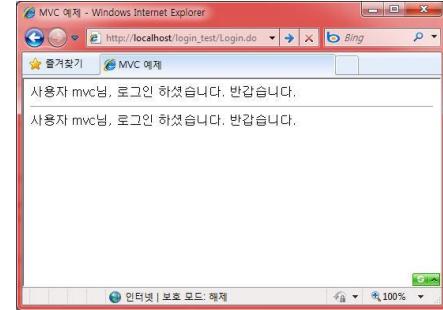
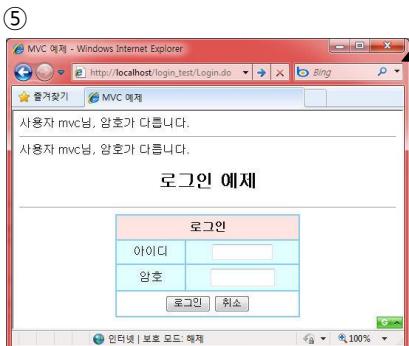
모델(UserBean.java)

```
1 package mvc.model;
2
3 public class UserBean {
4
5     private String userid;
6     private String passwd;
7
8     private String db_userid;
9     private String db_passwd;
10
11    // 생성자
```

컨트롤러(UserLogin.java)

```
1 package mvc.controller;
2
3 import mvc.model.UserBean;
4
5 /**
6  * Servlet implementation class UserLogin
7  */
8 public class UserLogin extends HttpServlet {
9     private static final long serialVersionUID = 1L;
```

뷰(Login.jsp)



①

②
③

④

⑤

22.2 로그인 처리 구현

(1)로그인 폼 작성

사용자 로그인을 위한 폼을 구성하는 태그 form은 아래와 같으며 로그인 버튼을 누르면 컨트롤러인 서블릿이 실행된다. 이 때 폼 태그의 속성인 action="Login.do"가 실행된다.
실행되는 컨트롤러인 서블릿 프로그램은 서블릿의 URL 맵핑 이름을 Login.do로 지정해야 한다.

```
<form method="post" action="Login.do" name="form1">
<table width="250" border="1" align="center" ... >
...
<input type="submit" name="submit" value="로그인">
...
</table>
</form>
```

index.html

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>로그인 폼</title>
</head>
<body>
<center>
<h2>로그인 예제</h2>
<hr>
<form method="post" action="Login.do" name="form1">
<table width="250" border="1" align="center" bordercolor="skyblue" cellspacing="0"
cellpadding="5">
<tr bgcolor="mistyrose">
<td colspan="2" height="22" align="center">
<b><font size="3">로그인</font></b>
</td>
</tr>
<tr bgcolor="lightcyan">
<td>아이디</td>
<td><input type="text" name="userid" size=10></td>
</tr>
<tr bgcolor="lightcyan">
<td>암호</td>
<td><input type="password" name="passwd" size=10></td>
</tr>
```

```

<tr>
    <td colspan="2" align="center">
        <input type="submit" name="submit" value="로그인">
        <input type="reset" name="reset" value="취소">
    </td>
</tr>
</table>
</form>
</center>
</body>
</html>

```

웹 브라우저 주소줄에 도메인 또는 폴더만 입력했을 때 자동으로 실행되는 파일을 환영 파일 (welcome file)이라 하는데 web.xml 파일에 파일이름을 지정할 수 있다.

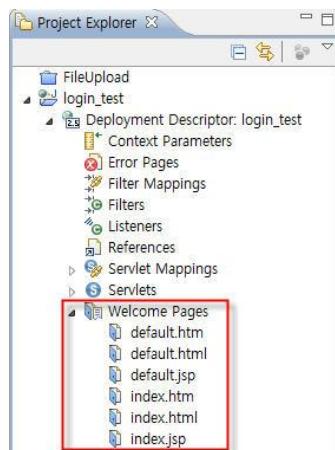
http://localhost/login_test/index.html 또는 http://localhost/login_test/로 접속한다.

web.xml

```

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

```



(2) 모델 작성

모델인 자바빈즈 UserBean은 사용자의 ID와 암호를 확인하여 그 결과를 반환하는 기능을 수행한다. 실제 프로젝트라면 로그인 폼에 입력한 사용자 ID로 데이터베이스에 접속해 암호를 조회하여 그 인증 결과를 반환하겠지만 여기서는 간단히 정해진 사용자 ID와 암호로 인증을 처리한다. 모델인 자바빈즈 UserBean은 패키지를 mvc.model로 하며 필드로는 로그인 폼에서 입력된 사용자 정보를 저장하는 userid, passwd를 사용하고 가상으로 조회된 결과가 저장되는 dbUserId, dbPasswd를 사용한다.

로그인을 성공할 수 있는 사용자 ID와 암호를 "mvc"와 "model"로 한다.

```

package mvc.model;
public class UserBean {

    private String userid;
    private String passwd;
}

```

```

private String dbUserId;
private String dbPasswd;
// 생성자
public UserBean() {
    // 인증에 사용할 기본값 설정,
    // 현재 저장하는 사용자와 암호인 경우 로그인 성공
    dbUserId = "mvc";
    dbPasswd = "model";
}
...
}

```

getCheckUser() 메소드는 로그인 폼에 입력된 사용자 암호를 이용하여 로그인을 성공할 수 있는 ID와 암호를 비교하여 그 결과를 반환한다.

```

// 아이디와 비밀번호가 맞는지 체크하는 메서드
public boolean getCheckUser() {
    if(userid.equals(dbUserId) && passwd.equals(dbPasswd))
        return true;
    else
        return false;
}

```

UserBean.java

```

package mvc.model;
public class UserBean {
    private String userid;
    private String passwd;
    private String dbUserId;
    private String dbPasswd;
    // 생성자
    public UserBean() {
        // 인증에 사용할 기본값 설정,
        // 현재 저장하는 사용자와 암호인 경우 로그인 성공
        dbUserId = "mvc";
        dbPasswd = "model";
    }
    // 아이디와 비밀번호가 맞는지 체크하는 메서드
    public boolean getCheckUser() {
        if(userid.equals(dbUserId) && passwd.equals(dbPasswd))
            return true;
    }
}

```

```

        else
            return false;
    }

    //getter, setter
    public void setUserid(String userid) {
        this.userid = userid;
    }

    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }

    public String getUserid() {
        return userid;
    }

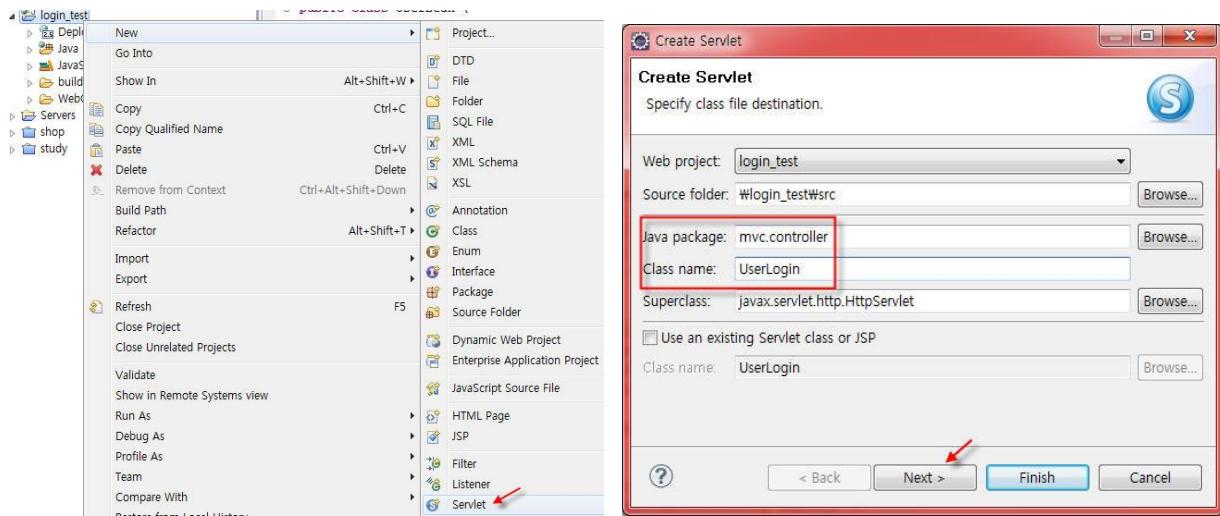
    public String getPasswd() {
        return passwd;
    }
}

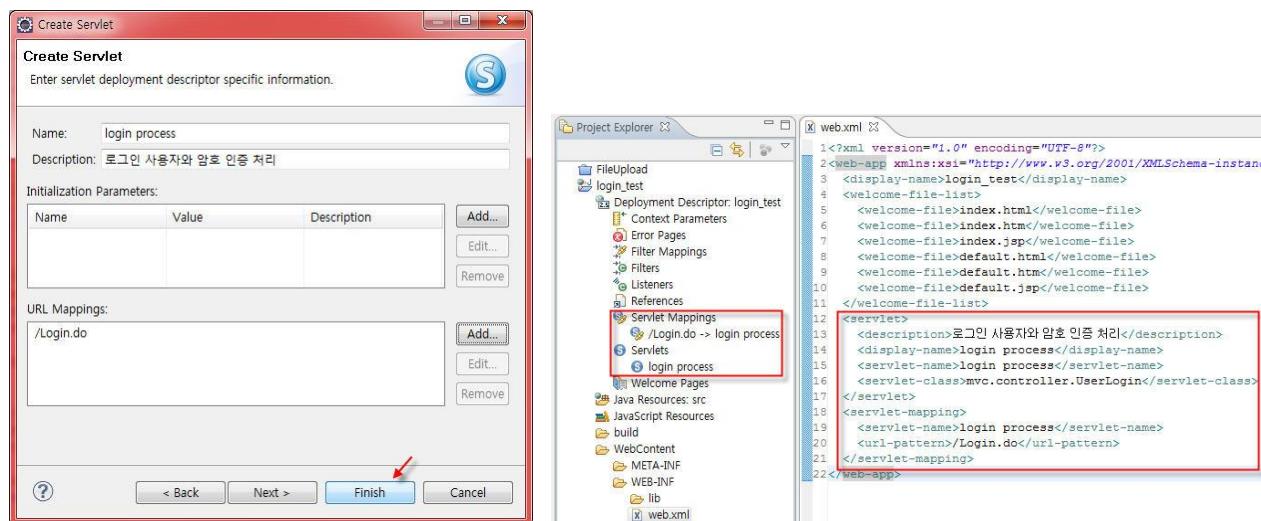
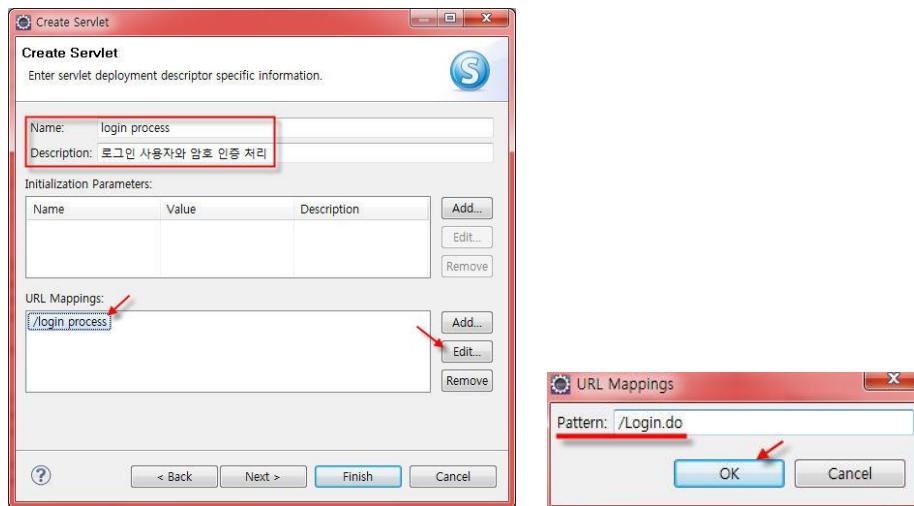
```

(3) 컨트롤러 작성(애노테이션 @WebServlet()를 사용하지 않을 경우)

뷰와 모델을 연결해주는 컨트롤러 UserLogin을 작성한다. 컨트롤러 UserLogin은 서블릿으로서 패키지는 mvc.controller로 한다.

로그인 폼이 구현된 뷰인 index.html에서 컨트롤러 UserLogin을 Login.do로 실행하려면 URL 매핑을 /Login.do로 해야한다.





web.xml 을 살펴보면 서블릿 매핑 정보가 자동으로 입력된 것을 볼 수 있다.()

URL 매핑을 위해 <servlet> 태그와 <servlet-mapping> 태그가 필요하다.

<servlet> 태그에서 웹 응용프로그램 내부에서 사용될 서블릿 이름을 <servlet-name> 태그에 지정하며, 실제 클래스 이름은 <servlet-class> 태그에 지정한다. <servlet-mapping> 태그의 <servlet-name> 태그에서 <servlet>의 <servlet-name>에 지정한 이름과 같은 서블릿 이름을 지정하고 <url-pattern>에 지정한 이름은 웹 브라우저에서 사용되는 서블릿의 URL이다.

```

<servlet>
    <description>로그인 사용자와 암호 인증 처리</description>
    <display-name>login process</display-name>
    <servlet-name>login process</servlet-name>
    <servlet-class>mvc.controller.UserLogin</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>login process</servlet-name>
    <url-pattern>/Login.do</url-pattern>
</servlet-mapping>

```

- 컨트롤러가 하는 작업

- ① 클라이언트의 요청에서 매개 변수를 받는다.
- ② 필요하면 클라이언트가 요청한 명령을 파악한다.
- ③ 작업 흐름에 따라 비즈니스 로직을 처리하는 모델을 생성한다.
- ④ 모델로부터 필요한 필드 값을 저장하여 결과를 얻는다.
- ⑤ 모델로부터 얻은 결과를 request, session 또는 application의 속성으로 저장한다.
- ⑥ 처리 결과와 제어를 전달할 뷰를 선정하여 RequestDispatcher 객체를 생성한다.
- ⑦ 웹 브라우저에서 처리 결과를 보여주기 위한 뷰로 이동(forward)한다.

컨트롤러인 서블릿 UserLogin의 doGet() 메소드 내부 구현을 위해 가장 먼저 사용자 인증을 위한 요청 매개 변수를 얻어와 저장한다.

```
String userid = request.getParameter("userid");
String passwd = request.getParameter("passwd");
```

사용할 자바 빈즈 UserBean의 객체를 생성하여 사용자가 입력한 ID와 암호를 저장한다. 사용자 인증 결과를 getCheckUser() 메소드로 얻어와 요청 속성 resultlogin 으로 저장한다.

```
UserBean user = new UserBean();
user.setUserId(userid);
user.setPassword(passwd);
request.setAttribute("resultlogin", user.getCheckUser());
```

컨트롤러의 역할은 뷰로부터 받은 요청을 모델을 사용하여 처리한 후 다시 뷰로 결과를 전달하는 것이다. 이 결과를 뷰인 login.jsp로 전달한다.

이러한 역할을 담당하는 인터페이스가 javax.servlet.RequestDispatcher 이며, 이 객체는 매개 변수 request의 메소드 getRequestDispatcher("login.jsp")로 얻을 수 있다.

RequestDispatcher의 객체 view를 이용하여 서블릿 매개변수 request와 response를 매개 변수로 forward() 메소드를 호출한다. forward() 메소드는 처리 결과를 뷰로 전송하며 동시에 프로그램 제어권을 이동시킨다.

```
RequestDispatcher view = request.getRequestDispatcher("login.jsp");
view.forward(request, response);
```

UserLogin.java

```
package mvc.controller;
import mvc.model.UserBean;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import java.io.*;
```

```

@WebServlet(name = "login process", urlPatterns = { "/Login.do" })
public class UserLogin extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public UserLogin() {
        super();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String userid = request.getParameter("userid");
        String passwd = request.getParameter("passwd");
        UserBean user = new UserBean();
        user.setUserid(userid);
        user.setPasswd(passwd);
        request.setAttribute("resultlogin", user.getCheckUser());
        //처리 결과가 저장된 request를 전송하며 동시에 뷰인 login.jsp로 제어 이동
        RequestDispatcher view = request.getRequestDispatcher("login.jsp");
        view.forward(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }
}

```

(4) 뷰 작성

컨트롤러인 서블릿으로부터 전달받은 로그인 사용자 인증 결과를 출력하는 뷰인 login.jsp를 작성한다.
내장 객체 request를 이용해 속성 resultlogin과 매개 변수 UserID를 얻어온다.

```

Boolean res = (Boolean)request.getAttribute("resultlogin");
String userid = request.getParameter("userid");
if ( res.booleanValue() ) {
    out.println("사용자 " + userid + "님, 로그인 하셨습니다. 반갑습니다.");
} else {
    out.println("사용자 " + userid + "님, 암호가 다릅니다.");
}

```

JSTL을 사용해서 사용자 인증 결과가 false이면 다시 로그인을 할 수 있도록 한다.

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="result" value="${requestScope.resultlogin}" />
<c:choose>
    <c:when test="${result}" >

```

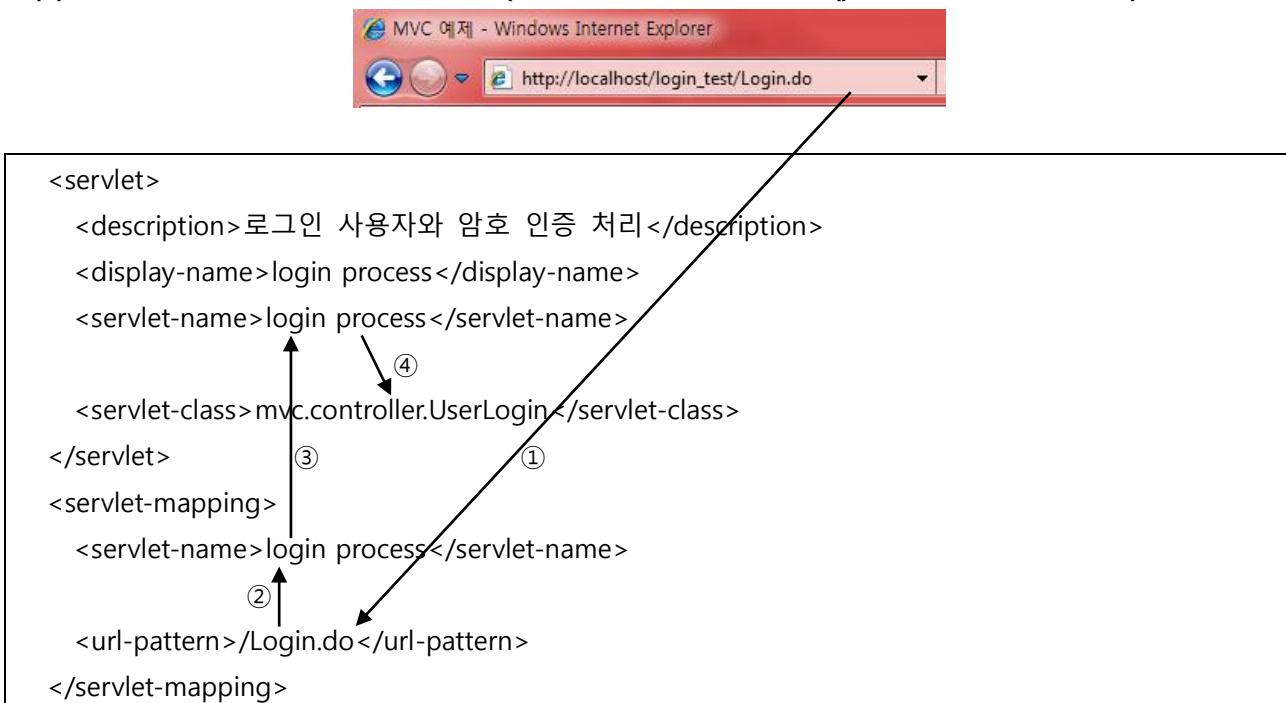
```
사용자 ${param.userid}님, 로그인 하셨습니다. 반갑습니다.
```

```
</c:when>
<c:otherwise>
    사용자 ${param.userid}님, 암호가 다릅니다.
    <jsp:include page="index.html" />
</c:otherwise>
</c:choose>
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
<title>MVC 예제</title>
</head>
<body>
<%
    Boolean res = (Boolean)request.getAttribute("resultlogin");
    String userid = request.getParameter("userid");
    if ( res.booleanValue() ) {
        out.println("사용자 " + userid + "님, 로그인 하셨습니다. 반갑습니다.");
    } else {
        out.println("사용자 " + userid + "님, 암호가 다릅니다.");
    }
%>
<hr>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="result" value="${requestScope.resultlogin}" />
<c:choose>
    <c:when test="${result}" >
        사용자 ${param.userid}님, 로그인 하셨습니다. 반갑습니다.
    </c:when>
    <c:otherwise>
        사용자 ${param.userid}님, 암호가 다릅니다.
        <jsp:include page="index.html" />
    </c:otherwise>
</c:choose>
</body>
</html>
```

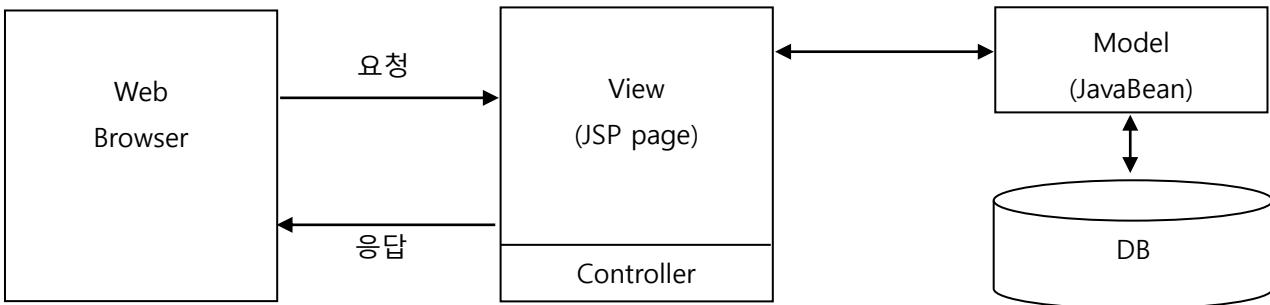
(5) URL 매팅에 의한 서블릿 실행 과정(애노테이션 @WebServlet()를 사용하지 않을 경우)



23. MVC 모델2

23.1 모델1과 모델2

모델1 구조에서는 웹 브라우저의 요청(request)을 받아들이고, 웹 브라우저에 응답(response)해 주는 처리에 대해 JSP 페이지 단독으로 처리하는 구조이다



MVC의 구조를 적용하면 뷰(View)와 컨트롤러(Controller)가 같은 JSP 페이지 안에서 실행된다. 즉 JSP 페이지가 뷰(View)와 컨트롤러(Controller)의 역할을 같이 하므로 모든 사용자 요청의 진입점이 요청되는 JSP 페이지가 된다. 모델1 구조는 단지 간단한 웹 애플리케이션을 구축할 때 적당하며, 중대형 프로젝트에서는 비즈니스 로직(Business logic)과 뷰(View) 사이의 구분의 미비로 인한 개발자와 디자이너의 작업의 분리가 어려운 문제가 종종 발생한다.

웹 어플리케이션을 개발할 때 JSP만을 사용하여 개발하는 설계 방법

클라이언트의 요청 처리, DB연동, 세션 관리, 응답 처리 등과 같은 작업을 JSP만으로 구현하는 방법이다.

Model 1 아키텍처의 구조



하나의 JSP 코드 내에 HTML로 만든 화면 구성 코드인 Presentation Logic과 자바 코드로 작성한 Business Logic 코드가 같이 포함되어있기 때문에 개발 속도는 빠르다. 서로 모듈화가 안되어 있기 때문에 유지보수는 어려워진다. 개발 작업 속도가 빨라야 되는 소규모의 웹 어플리케이션 개발에 적합하다.

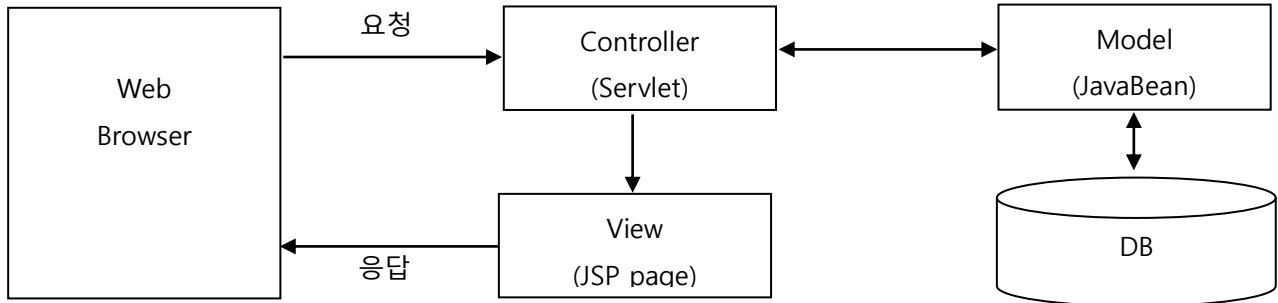
※ 장점

- 단순한 페이지 흐름으로 인해 개발 기간이 단축된다.
- MVC구조에 대한 추가적인 교육이 필요없고 개발팀의 팀원의 수준이 높지 않아도 된다.
- 중소형 프로젝트에 적합하다.

※ 단점

- 웹 애플리케이션이 복잡해질수록 유지 보수가 힘들다.
- 디자이너와 개발자 간의 원활한 의사소통이 필요하다.

모델2 구조에서는 요청(request) 처리, 데이터 접근(data access), 비즈니스 로직(Business logic)을 포함하고 있는 컨트롤러 컴포넌트와 뷰 컴포넌트가 엄격히 구분 되어져 있다. 뷰는 어떠한 처리 로직도 포함하고 있지 않다.



사용자의 요청의 진입점은 컨트롤러의 역할을 하는 서블릿이 담당하고 모든 흐름을 통제한다. 이러한 구조는 개발자와 디자이너와 역할과 책임에 대한 명확한 구분을 해 줄 수가 있다.

웹 어플리케이션을 개발할 때 Presentation Logic과 Business Logic 처리를 명확하게 분리하여 설계하는 방법

MVC 설계 패턴에 기반을 둔 개발 방법

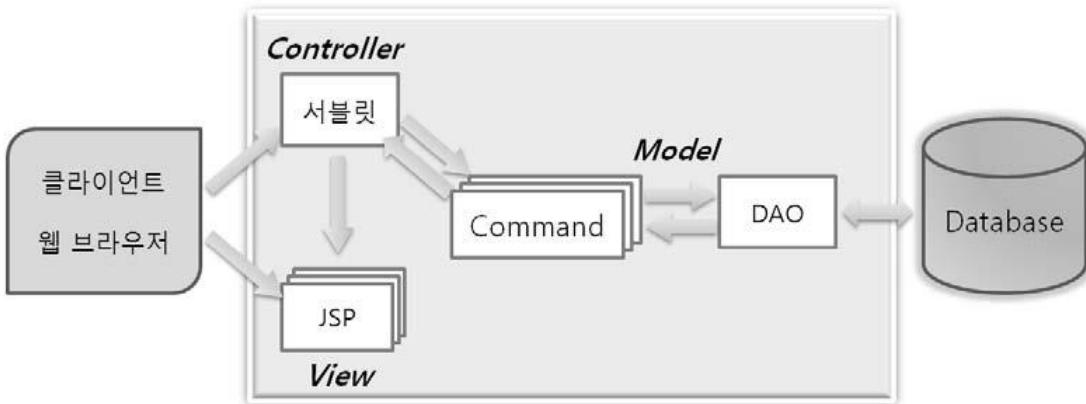
MVC 패턴을 구현하는 웹 어플리케이션은 Model, View, Controller라는 3개의 모듈로 구성된다.

Model : Business Logic을 의미하고 자바빈(JavaBeans) 또는 일반 클래스를 사용하여 구현, 대표적으로 DAO, DTO, Command 클래스들에 해당된다.

View : Presentation Logic을 의미하고 JSP를 사용하여 구현된다.

Controller : Model과 View를 적절하게 관리하는 로직을 의미하고 서블릿을 사용하여 구현된다.

Model 2 아키텍처의 구조



※ 장점

- 비즈니스 로직과 뷰의 분리로 인해 애플리케이션이 명료해지며 유지 보수와 확장이 용이하다.
- 개발자와 디자이너의 작업이 분리되어 역할과 책임 구분이 명확하다.

※ 단점

- 개발 초기에 구조 설계를 위한 시간이 많이 소요되므로 개발 기간이 증가한다.

- MVC 구조에 대한 개발자들의 이해가 필요해서 개발팀의 팀원의 높은 수준이 요구된다.
- 웹 애플리케이션을 개발할 때 모델1 구조와 모델2 구조 중 어떤 것을 선택해야 하는가의 문제는 개발하려는 애플리케이션의 복잡도(규모), 유지 보수의 빈도, 애플리케이션 컴포넌트의 재사용성 그리고 팀원의 수와 수준에 따라 결정해야 한다.

23.2 MVC 패턴(Model-View-Controller Pattern)

MVC(Model-View-Controller) 구조는 전통적인 GUI(Graphic User Interface)기반의 애플리케이션을 구현하기 위한 디자인 패턴이다. MVC구조는 사용자의 입력을 받아서, 그 입력 혹은 이벤트에 대한 처리를 하고 그 결과를 다시 사용자에게 표시하기 위한 최적화된 설계를 제시한다.

MVC 패턴은 다음과 같은 구조를 가지고 있다.

뷰(View)는 화면에 내용을 표출하는 역할을 담당하는 것으로 데이터가 어떻게 생성되고, 어디서 왔는지에 전혀 관여하지 않는다. 단지 정보를 보여주는 역할만을 담당한다. JSP 기반의 웹 애플리케이션에서는 JSP 페이지가 뷰에 해당한다.

모델(Model)은 로직을 가지는 부분으로 DB와의 연동을 통해서 데이터를 가져와 어떤 작업을 처리하거나 처리한 작업의 결과를 데이터로서 DB에 저장하는 일을 처리한다. 모델은 애플리케이션의 수행에 필요한 데이터를 모델링하고 비즈니스 로직을 처리한다. 즉 데이터를 생성하고 저장하고 처리하는 역할만을 담당한다. JSP 기반의 웹 애플리케이션에서는 자바빈(JavaBean)이 모델에 해당한다.

컨트롤러(Controller)는 애플리케이션의 흐름을 제어하는 것으로 뷰와 모델 사이에서 이들의 흐름을 제어한다. 컨트롤러는 사용자의 요청을 받아서 모델에 넘겨주고, 모델이 처리한 작업의 결과를 뷰에 보내주는 역할을 한다. JSP 기반의 웹 애플리케이션에서는 보통 서블릿을 컨트롤러로 사용한다.

보통 컨트롤러에 서블릿을 많이 사용하는데 이 서블리시 사용자의 요청을 받아서 비즈니스 로직을 처리하는 모델을 통해 수행 결과를 받아와 해당 JSP 페이지에 결과를 보내서 뷰가 사용자에 응답을 보내는 구조를 가지고 있다. JSP 기반의 웹 애플리케이션에서 컨트롤러(Controller), 모델(Model), 뷰(View)에 포함되어야 할 수행 단계는 다음과 같다.

(1) 컨트롤러(Controller) : 서블릿(Servlet)

- ① 웹 브라우저의 요청을 받는다. 서블릿의 서비스 메소드인 doGet() 혹은 doPost()메소드가 사용자의 요청을 받는다.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
public void doPost(HttpServletRequest request, HttpServletResponse response)
```

- ② 웹 브라우저가 요구하는 작업을 분석한다.

```
String type = request.getParameter("type");
```

- ③ 요청한 작업을 처리하기 위해서 비즈니스 로직을 처리하는 모델을 사용한다.

- ④ 처리 결과를 request 또는 session의 속성에 저장한다.

```
request.setAttribute("result", result);
```

- ⑤ 적당한 뷰를 선택 후 해당 뷰로 포워딩한다.

```
RequestDispatcher dispatcher = request.getRequestDispatcher("testjsp.jsp")
dispatcher.forward(request, response);
```

RequestDispatcher 클래스는 javax.servlet 패키지에 있으며 클라이언트로부터 요청을 받고 그것을 서버 상의 어떤 리소스(Servlet, HTML, JSP page)로 보내는 작업을 할 때 사용된다. RequestDispatcher 클래스의 forward(request, response) 메소드는 서블릿에서 다른 리소스(Servlet, HTML, JSP page)로 요청을 보낸다. 이때 다른 리소스와 request, response 객체를 공유한다.

- 컨트롤러가 하는 작업

- ① 클라이언트의 요청에서 매개 변수를 받는다.
- ② 필요하면 클라이언트가 요청한 명령을 파악한다.
- ③ 작업 흐름에 따라 비즈니스 로직을 처리하는 모델을 생성한다.
- ④ 모델로부터 필요한 필드 값을 저장하여 결과를 얻는다.
- ⑤ 모델로부터 얻은 결과를 request, session 또는 application의 속성으로 저장한다.
- ⑥ 처리 결과와 제어를 전달할 뷰를 선정하여 RequestDispatcher 객체를 생성한다.
- ⑦ 웹 브라우저에서 처리 결과를 보여주기 위한 뷰로 이동(forward)한다.

(2) 뷰(View) : JSP 페이지

비즈니스 로직을 가지고 있지 않은 점을 제외하고는 일반적인 JSP 페이지와 다를 바 없다. 다만 서블릿에서 dispatcher.forward(request, response)로 해당 JSP 페이지와 request, response 객체를 공유한 경우 해당 JSP 페이지에서 request.setAttribute("result", result)와 같이 사용해서 객체 결과를 화면에 표출한다. 이때 JSP 페이지의 request는 컨트롤러인 서블릿과 같은 객체로 공유되어진다.

(3) 모델(Model) : 자바빈

JSP 페이지에서 비즈니스 로직의 처리를 요청받아서 처리했던 것과 달라지는 것이 없다. 다만 요청을 하는 주체가 JSP 페이지에서 컨트롤러인 서블릿으로 바뀐 것 뿐이다.

- ① 컨트롤러(Controller)의 요청을 받는다.
- ② 비즈니스 로직을 처리한다.
- ③ 처리한 비즈니스 로직의 결과를 컨트롤러(Controller)로 반환한다.

23.3 DAO 패턴 및 DTO 패턴

데이터베이스를 연동하는 프로그램을 개발할 때는 두 가지 패턴이 있다.

(1) DAO(Data Access Object) 패턴

일반적으로 어플리케이션 개발은 GUI 화면을 가지며 화면에 보여줄 수 있는 데이터 관리를 위해서 데이터베이스를 사용해서 개발한다.

웹 브라우저에서 보여지는 것처럼 GUI 화면을 구성하는 코드를 presentation logic이라고 하며 GUI 화면에 데이터를 보여주기 위해서 데이터베이스를 검색하는 코드 및 GUI 화면에서 새로 발생된 데이터(예를 들면 회원가입)를 데이터베이스에 저장하는 코드와 같은 실제적인 작업을 처리하는 코드를 business logic이라고 한다.

presentation logic과 business logic을 하나의 클래스로 모두 구현 할 수도 있고 여러 클래스로 모듈화 시켜서 구현이 가능하지만 하나의 클래스로 구현하면 유지보수가 어려워진다. 따라서 모듈화시켜 개발

하게 되며 모듈화한 클래스들 중에서 데이터베이스 처리하는 코드만을 관리하는 클래스를 DAO(Data Access Object) 클래스라고 한다. DAO 클래스를 사용해서 개발하는 것이 보편화되었기 때문에 DAO 패턴이라고 한다.

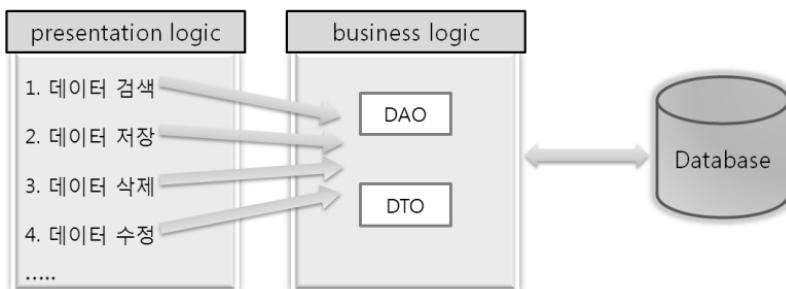
일반적으로 DAO 클래스는 테이블 당 한 개씩 생성해서 사용한다. DAO 클래스 안에는 특정 테이블에서 수행할 작업을 메서드로 정의해서 구현하며 presentation logic에서는 DAO 클래스의 메서드를 호출하면서 원하는 작업 구현한다.

(2) DTO(Data Transfer Object) 패턴

presentation logic과 business logic을 여러 클래스로 분리해서 작업은 하지만 서로 간에 긴밀한 관계가 유지되면서 작업이 이루어진다. presentation logic에서 보여줄 데이터를 얻기 위해서 business logic에게 요청을 하면 business logic은 필요한 데이터를 데이터베이스에서 검색해서 presentation logic에게 반환하는 작업 등을 수행한다. 이렇게 데이터를 다른 logic에게 전송 및 반환할 때 효율적으로 데이터를 사용할 수 있게 클래스를 작성한다. 이 클래스를 DTO(Data Transfer Object) 클래스라고 한다.

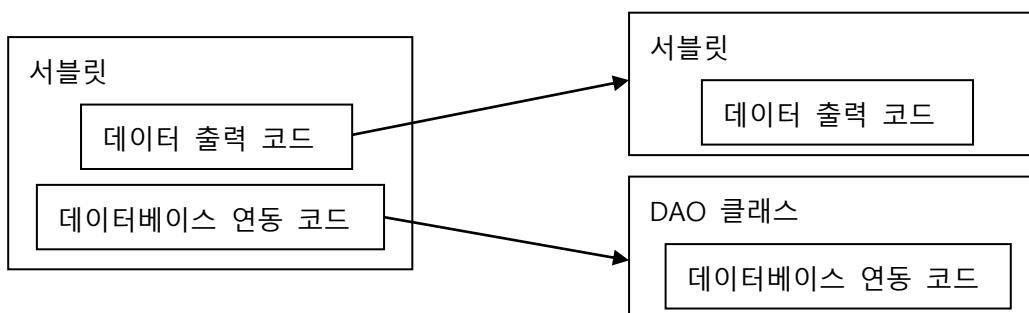
DTO 클래스는 이름 그대로 데이터를 전송할 때 사용되는 클래스이며 데이터를 전송할 때와 전송된 데이터를 얻어서 사용할 때 효율적으로 사용할 수 있는 장점이 있다. 일반적으로 도메인 객체(Domain Object), VO(Value Object)라고도 한다.

DAO와 DTO 패턴 사용 구조

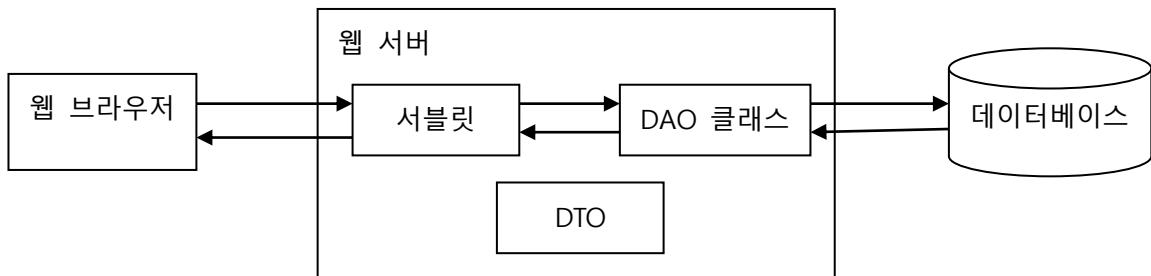


[예제] DAO와 DTO 패턴을 적용하여 데이터를 출력

데이터베이스 연동 코드와 웹 브라우저에 데이터를 출력하는 코드가 모두 서블릿 처리가 되었으나 DAO 패턴을 적용하여 서블릿으로부터 데이터베이스 연동 코드를 모듈화 시킨다.



전체 아키텍처



오라클의 교육 테이블을 이용한다.

EmpDTO.java

```
package com.emp;
public class EmpDTO {
    String emp_id;
    String ename;
    int salary;
    String depart;
    public EmpDTO() {}
    public EmpDTO(String emp_id, String ename, int salary, String depart) {
        this.emp_id = emp_id;
        this.ename = ename;
        this.salary = salary;
        this.depart = depart;
    }
    public String getEmp_id() {
        return emp_id;
    }
    public void setEmp_id(String emp_id) {
        this.emp_id = emp_id;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
}
```

```

    }
    public String getDepart() {
        return depart;
    }
    public void setDepart(String depart) {
        this.depart = depart;
    }
}

```

EmpDAO.java

```

package com.emp;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class EmpDAO {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:orcl";
    String userid = "scott";
    String passwd = "tiger";
    public EmpDAO(){
        try {
            Class.forName(driver);
        } catch (ClassNotFoundException e){
            e.printStackTrace();
        }
    }
    // 데이터 검색
    public ArrayList<EmpDTO> select(){
        ArrayList<EmpDTO> list = new ArrayList<EmpDTO>();
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try{
            con = DriverManager.getConnection(url, userid, passwd);
            String sql = "select empno, ename, sal, deptno from emp";
            pstmt = con.prepareStatement(sql);

```

```

        rs = pstmt.executeQuery();
        while(rs.next()){
            String emp_id = rs.getString("empno");
            String ename = rs.getString("ename");
            int salary = rs.getInt("sal");
            String depart = rs.getString("deptno");
            EmpDTO dto = new EmpDTO(emp_id, ename, salary, depart);
            list.add(dto);
        }
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        try {
            if(rs!=null) rs.close();
            if(pstmt!=null) pstmt.close();
            if(con!=null) con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return list;
}//end select
}//end class

```

EmpSelectDAOServlet.java

```

package com.emp;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/EmpSelectDAO")
public class EmpSelectDAOServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    doPost(request, response);
}

```

```

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        EmpDAO dao = new EmpDAO();
        ArrayList<EmpDTO> list = dao.select();
        for (EmpDTO dto : list) {
            String emp_id = dto.getEmp_id();
            String ename = dto.getEname();
            int salary = dto.getSalary();
            String depart = dto.getDepart();
            out.print(emp_id + " " + ename + " " + salary + " " + depart + "<br>");
        }
        out.print("</body></html>");
    }
}

```

23.4 커넥션 풀

context.xml 에 다음 코드 추가

```

<Resource name="jdbc/Oracle11g" auth="Container" type="javax.sql.DataSource"
maxActive="100" maxWait="10000"
username="scott" password="tiger" driverClassName="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />

```

(1) InitialContext 클래스를 사용하는 방법

이전 버전의 서블릿 스펙에서 사용하던 방법으로 InitialContext 클래스를 사용하여 DataSource 객체를 얻는다.(위의 예제에서 EmpDAO.java 수정)

EmpDAO.java

```

package com.emp;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

```

```

public class EmpDAO {
    DataSource dataFactory;
    public EmpDAO(){
        try {
            Context ctx = new InitialContext();
            dataFactory = (DataSource)ctx.lookup("java:comp/env/jdbc/Oracle11g");
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
    //데이터 검색
    public ArrayList<EmpDTO> select(){
        ArrayList<EmpDTO> list = new ArrayList<EmpDTO>();
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try{
            con = dataFactory.getConnection(); //컨넥션 객체 얻기
            String sql = "select empno, ename, sal, deptno from emp";
            pstmt = con.prepareStatement(sql);
            rs = pstmt.executeQuery();
            while(rs.next()){
                String emp_id = rs.getString("empno");
                String ename = rs.getString("ename");
                int salary = rs.getInt("sal");
                String depart = rs.getString("deptno");
                EmpDTO dto = new EmpDTO(emp_id, ename, salary, depart);
                list.add(dto);
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try {
                if(rs!=null) rs.close();
                if(pstmt!=null) pstmt.close();
                if(con!=null) con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

    }
    return list;
}//end select
}//end class

```

(2) @Resource 어노테이션을 사용하는 방법

서블릿에서 @Resource 어노테이션을 사용하여 DataSource 객체를 이용하는 방법이다.

DAO 클래스와 같은 일반 클래스에서는 사용하지 못하고 서블릿에서만 사용 가능하다.

EmpSelectPoolAnnoServlet.java

```

package com.emp;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.annotation.Resource;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/EmpSelectPoolAnno")
public class EmpSelectPoolAnnoServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        ArrayList<EmpDTO> list = select();
        for (EmpDTO dto : list) {
            String emp_id = dto.getEmp_id();
            String ename = dto.getEname();

```

```

        int salary = dto.getSalary();
        String depart = dto.getDepart();
        out.print(emp_id + " " + ename + " " + salary + " " + depart + "<br>");
    }
    out.print("</body> </html>");

}//end Post

@Resource (name="jdbc/Oracle11g" )javax.sql.DataSource dataFactory;

public ArrayList<EmpDTO> select(){
    ArrayList<EmpDTO> list = new ArrayList<EmpDTO>();
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try{
        con = dataFactory.getConnection();
        String sql = "select empno, ename, sal, deptno from emp";
        pstmt = con.prepareStatement(sql);
        rs = pstmt.executeQuery();
        while(rs.next()){
            String emp_id = rs.getString("empno");
            String ename = rs.getString("ename");
            int salary = rs.getInt("sal");
            String depart = rs.getString("deptno");
            EmpDTO dto = new EmpDTO(emp_id, ename, salary, depart);
            list.add(dto);
        }
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        try {
            if(rs!=null) rs.close();
            if(pstmt!=null) pstmt.close();
            if(con!=null) con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return list;
}//end select
}

```

23.5 FrontController 패턴

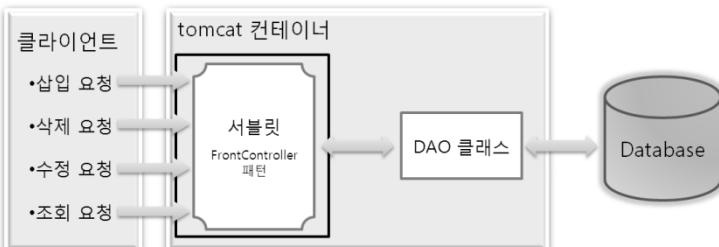
웹 어플리케이션 개발 시 사용자의 요청을 처리하기 위한 최초 진입점(Initial Point)을 정의하고 사용하는 패턴을 의미한다. 모든 사용자의 요청을 집중화시키면 요청을 분산시켜 발생되는 중복된 코드를 제거할 수 있고, 사용자의 요청을 일관된 방법으로 관리할 수 있는 장점이 있다.

FrontController 패턴을 적용하지 않은 경우의 아키텍처



클라이언트의 개별적인 요청을 서로 다른 서블릿이 처리하기 때문에 중복 코드가 발생될 수 있고, 다수의 서블릿으로 인한 유지보수가 어려워질 수 있다.

FrontController 패턴을 적용한 전체적인 아키텍처



사용자의 모든 요청을 단 하나의 서블릿이 처리하는 집중화 형태이기 때문에 중복코드가 제거되고 유지보수가 쉬워진다. FrontController 패턴을 적용한 서블릿에서 고려해야 되는 사항은 사용자가 어떤 동작을 요청했는지를 식별할 수 있어야 된다. 따라서 사용자가 서블릿에 요청할 때, 다음과 같은 메커니즘으로 서블릿이 사용자의 요청을 식별할 수 있도록 지원한다.

`http://서버IP번호:포트번호/context명/식별값`

사용자는 명시적으로 URL 값에 '식별값'을 추가하여 요청하고, 서블릿에서는 '식별값'을 비교하여 어떤 요청인지를 구별할 수 있다. '식별값'은 임의의 문자열 값으로서 일반적으로 웹 프레임워크(Spring, Struts2)에서 사용하는 방식으로 지정한다.(예: xxx.do, xxx.nhn).

다음과 같이 데이터를 저장하는 요청인 경우에는 insert.do로 지정하고 조회를 하는 요청인 경우에는 select.do 형식으로 지정할 수 있다.



요청 받은 서블릿에서는 다음과 같이 두 가지 작업을 통하여 원하는 '식별값'을 얻는다.

- ① 서블릿의 맵핑명을 다음과 같이 *.do 값의 확장자 패턴 형식으로 지정한다. 따라서 반드시 사용자는 'xxx.do' 형식으로 요청해야 한다.

```
@WebServlet("*.do")
public class 서블릿명 extends HttpServlet { .... }
```

- ② 서블릿에서 다음 코드를 사용하여 '식별값'을 비교 처리할 수 있다.



```
String requestURI = request.getRequestURI();
String contextPath = request.getContextPath();
String command = requestURI.substring(contextPath.length());
if( command.equals( "/insert.do" )){
    //저장
} else if( command.equals( "/delete.do" ) ){
    //삭제
}
```

[예제]

request.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>FrontController 패턴 실습</title>
</head>
<body>
    <h1>FrontController 패턴 실습</h1>
    <a href="insert.do">저장하기</a><br>
    <a href="/frontController/delete.do">삭제하기</a><br>
    <a href="http://localhost/frontController/update.do">수정하기</a><br>
    <a href="select.do">조회하기</a><br>
</body>
</html>
```

FrontControllerServlet.java

```
package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
```

```

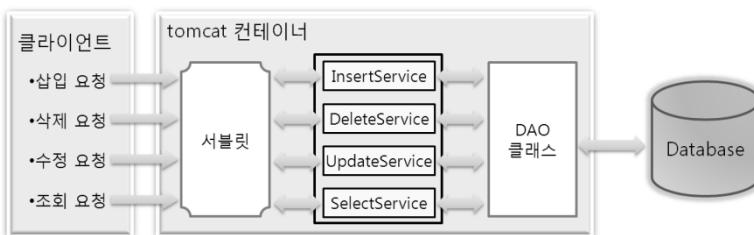
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("*.do")
public class FrontControllerServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String requestURI = request.getRequestURI();
        String contextPath = request.getContextPath();
        String command = requestURI.substring(contextPath.length());
        if(command.equals("/insert.do")){
            System.out.println("insert 요청");
        }else if(command.equals("/delete.do")){
            System.out.println("delete 요청");
        }else if(command.equals("/update.do")){
            System.out.println("update 요청");
        }else{
            System.out.println("select 요청");
        }
    }//end doPost
}//end class

```

23.6 Command 패턴

사용자의 요청을 객체인 클래스로 처리하는 것을 의미한다. 객체 형태로 사용하면 서로 다른 사용자의 요청 값을 필요에 의해서 저장하거나 또는 취소가 가능하고 요청을 처리할 작업을 일반화시켜 요청의 종류와 무관하게 프로그램 작성이 가능하게 구현한다. 구현 방법은 Command 패턴을 적용한 Service 이름의 클래스를 추가한다.



Service 클래스는 작업 수행을 요청하는 객체인 서블릿과 실제 작업을 수행하는 객체로 분리시켜 주기

때문에 시스템간의 결합도를 낮출 수 있다. 일반적으로 의존성을 낮추기 위해서 인터페이스를 사용하여 구현하고 서블릿과 DAO간의 의존성도 감소시키는 역할을 담당한다.

[예제]

request.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Command 패턴 실습</title>
</head>
<body>
    <h1>Command 패턴 실습</h1>
    <a href="select.do">조회하기</a><br>
</body>
</html>
```

com/domain/EmpDTO.java(코드는 앞의 것과 동일)

EmpDAO.java

```
package com.dao; //수정

// import 내용 동일
import com.domain.EmpDTO; // 추가

public class EmpDAO {
    //내용 동일
}
```

SelectService.java

```
package com.service;

import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.EmpDAO;
import com.domain.EmpDTO;

public class SelectService {
    public ArrayList<EmpDTO> execute(HttpServletRequest request, HttpServletResponse response){
        EmpDAO dao = new EmpDAO();
```

```

        return dao.select();
    }
}

```

FrontControllerServlet.java

```

package com.controller;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
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 com.domain.EmpDTO;
import com.service.SelectService;

@WebServlet("*.do")
public class FrontControllerServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String requestURI = request.getRequestURI();
        String contextPath = request.getContextPath();
        String command = requestURI.substring(contextPath.length());
        if(command.equals("/insert.do")){
        }else if(command.equals("/delete.do")){
        }else if(command.equals("/update.do")){
        }else{
            response.setContentType("text/html; charset=UTF-8");
            PrintWriter out = response.getWriter();
            out.print("<html><body> ");
            SelectService service = new SelectService();
            ArrayList<EmpDTO> list = service.execute(request, response);
            for (EmpDTO dto : list) {
                String emp_id = dto.getEmp_id();
                String ename = dto.getEname();
            }
        }
    }
}

```

```

        int salary = dto.getSalary();
        String depart = dto.getDepart();
        out.print(emp_id + "wt" + ename + "wt" + salary + "wt" + depart + "<br>");
    }
    out.print("</body></html>");
}
//end doPost
}//end class

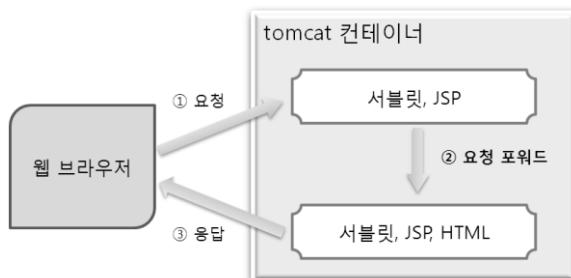
```

23.7 요청 포워딩(Request Forwarding)

사용자의 요청을 받은 서블릿 또는 JSP에서 다른 컴포넌트(서블릿, JSP, html)로 요청을 위임할 수 있는 방법이다.

포워드(forward)하는 이유는 처리 작업을 모듈화하기 위해서이다. 직접 요청받은 서블릿 또는 JSP에서 모든 작업을 처리하지 않고 모듈화 시킨 다른 컴포넌트로 요청을 위임하여 처리할 수 있다.

재사용성도 높아지고 유지보수가 쉬워진다. 일반적으로 MVC 패턴에서 서블릿이 JSP를 포워딩할 때 주로 사용한다. 사용자의 요청을 받는 웹 컴포넌트와 사용자에게 응답을 처리하는 웹 컴포넌트를 모듈화하여 구현한다.



일반적으로 요청을 처리하는 웹 컴포넌트는 FrontController 패턴을 적용한 서블릿으로 구현하고 응답을 처리하기 위한 웹 컴포넌트는 JSP로 구현할 수 있으며 MVC 패턴 또는 Model 2 Architecture라고 한다.

요청 포워딩을 구현하는 방법은 두 가지 방법이 제공

- ① RequestDispatcher 클래스를 이용한 forward 방법
- ② HttpServletResponse 클래스를 이용한 redirect 방법

(1) RequestDispatcher 클래스를 이용한 forward 방법

웹 브라우저를 통하여 사용자가 서블릿에 요청을 하면 HttpServletRequest 객체가 자동으로 생성된다. 생성된 request 객체를 사용하여 응답 처리하는 웹 컴포넌트(서블릿, JSP, html)로 재요청하는 방식을 의미한다. request 객체의 주된 용도는 사용자의 입력 파라미터 값을 얻어오거나 또는 한글 인코딩 처리 및 request scope에 해당되는 속성(Attribute) 설정에 사용 가능하다.

요청받은 서블릿에서 다른 컴포넌트로 포워드하는 방법

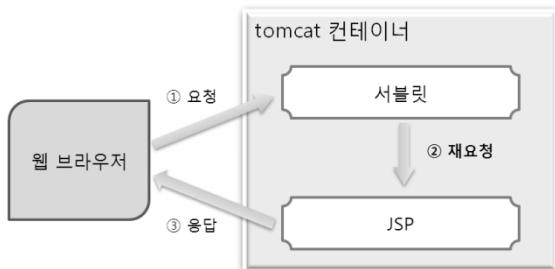
```

RequestDispatcher dis = request.getRequestDispatcher(target);
dis.forward( request, response );

```

지정된 target에는 서블릿 맵핑명 또는 JSP 파일 및 html 파일을 지정할 수 있다.

forward의 아키텍처



1번의 요청과 2번의 요청이 같은 HttpServletRequest를 사용한다. JSP로 응답이 되었기 때문에 웹 브라우저의 URL 값이 서블릿에서 JSP로 변경되어야 하지만 포워드 방법은 동일한 HttpServletRequest이기 때문에 URL 값이 서블릿으로 고정되어 있다.

1번의 요청을 받은 서블릿에서 request scope에 속성(Attribute) 설정을 하면, 2번의 요청을 받은 JSP에서 속성 값을 가져올 수 있다. 일반적으로 MVC 기반의 웹 어플리케이션에서 주로 사용된다.

[예제] 일반적으로 응답처리는 JSP로 하지만 예제에서는 서블릿으로 응답처리를 한다. 또한 request scope에 속성(attribute)을 설정하여 응답 서블릿에서 속성 값을 가져오는지 확인한다.

RequestServlet.java

```
package com.controller;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/Request")
public class RequestServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //속성 설정
        request.setAttribute("username", "홍길동");
        request.setAttribute("useraddress", "서울");
        //forward
        RequestDispatcher dis = request.getRequestDispatcher("Response");
        dis.forward(request, response);
    }
}
```

```
    }  
}
```

ResponseServlet.java

```
package com.controller;  
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;  
  
@WebServlet("/Response")  
public class ResponseServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
    ServletException, IOException {  
        doPost(request, response);  
    }  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
    ServletException, IOException {  
        String username = (String) request.getAttribute("username");  
        String useraddress = (String) request.getAttribute("useraddress");  
        response.setContentType("text/html; charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        out.print("<html> <body>");  
        out.print("username 값:" + username + "<br>");  
        out.print("useraddress 값:" + useraddress + "<br>");  
        out.print("</body> </html>");  
    }  
}
```

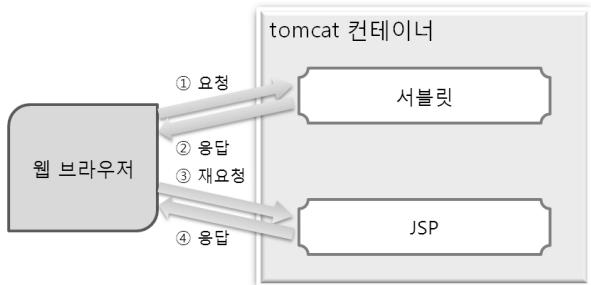
(2) **HttpServletResponse** 클래스를 이용한 redirect 방법

forward 방식과 마찬가지로 다른 웹 컴포넌트에게 재요청을 하는 방법이다. 차이점은 응답을 먼저 하고 재요청이 되기 때문에, 동일한 HttpServlet Request가 아닌 새로운 request 객체가 생성된다. 따라서 웹 브라우저의 URL 값이 변경되고 속성(Attribute)에 설정된 값을 가져오지 못한다.

요청받은 서블릿에서 다른 컴포넌트로 redirect 하는 방법

```
response.sendRedirect(target);
```

지정된 target에는 서블릿 맵핑명 또는 JSP 파일 및 html 파일을 지정할 수 있다.



1번 요청의 결과로 2번의 응답이 발생되고, 자동으로 `response.sendRedirect(target)` 메서드에 의해서 target으로 재요청이 발생된다. 따라서 1번의 요청과 3번의 요청은 서로 다른 요청이다.

서블릿에서 `reqeust scope`에 속성(Attribute) 설정을 하면, 3번의 요청을 받은 JSP에서 속성 값을 가져올 수 없다. 일반적으로 JSP를 이용한 Model 1 Architecture 웹 어플리케이션 개발 방법에 주로 사용된다.

[예제]

`RequestRedirectServlet.java`

```
package com.controller;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RequestRedirect")
public class RequestRedirectServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    doPost(request, response);
}
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //속성 설정
    request.setAttribute("username", "홍길동");
    request.setAttribute("useraddress", "서울");
    //redirect
    response.sendRedirect("ResponseRedirect");
}
}
```

`ResponseRedirectServlet.java`

```
package com.controller;
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;

@WebServlet("/ResponseRedirect")
public class ResponseRedirectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = (String) request.getAttribute("username");
        String useraddress = (String) request.getAttribute("useraddress");
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("username :" + username + "<br>");
        out.print("useraddress :" + useraddress + "<br>");
        out.print("</body></html>");
    }
}
```

23. MVC 모델2 게시판

23.1 MVC Model 2 아키텍처

웹 어플리케이션을 개발할 때 Presentation Logic과 Business Logic 처리를 명확하게 분리하여 설계하는 방법으로서 MVC 설계 패턴에 기반을 둔 개발 방법이다.

MVC 패턴을 구현하는 웹 어플리케이션은 Model, View, Controller라는 3개의 모듈로 구성된다.

Model : Business Logic을 의미하고 자바빈(JavaBeans) 또는 일반 클래스를 사용하여 구현, 대표적으로 DAO, DTO, Command 클래스들에 해당된다.

View : Presentation Logic을 의미하고 JSP를 사용하여 구현한다.

Controller : Model과 View를 적절하게 관리하는 로직을 의미하고 서블릿을 사용하여 구현한다.

Controller는 클라이언트의 요청에 대한 진입점 역할을 담당하며 요청 분석을 통해서 Business Logic을 구현한 적당한 Model를 선택하게 된다. 또한 처리 결과를 보여주기 위해서 Presentation Logic을 구현한 적당한 View를 선택하여 응답 처리한다. 이렇게 Model과 View를 관리하는 역할을 Controller인 서블릿이 담당한다.

Model은 클라이언트 요청과 관련된 실제 작업을 의미하는 Business Logic을 처리한다. Command 패턴을 이용하여 요청을 모듈화하고 데이터베이스 연동을 위한 DAO 클래스 및 DTO 클래스로 구현한다.

View는 Model의 실행 결과를 클라이언트에게 응답하는 Presentation Logic을 담당하며 JSP로 구현한다. 현재 사용되는 모든 웹 어플리케이션 프레임워크는 MVC 기반의 Model 2 아키텍처로 구현한다.

23.2 MVC 모델2 게시판 설계

(1) DB 설계

컬럼명	데이터형	설명
num	NUMBER(4)	글 번호이며 시퀀스 객체를 사용하여 자동으로 증가됨 기본 키(Primary key)로 설정
author	VARCHAR2(20)	글 작성자
title	VARCHAR2(50)	글 제목
content	VARCHAR2(300)	글 내용
writeday	DATE	글 작성일(DEFAULT로 지정)
readcnt	NUMBER(4)	글 조회수
repRoot	NUMBER(4)	답변글 작성 시 사용(원래글의 번호 참조)
repStep	NUMBER(4)	답변글 작성 시 사용(답변글의 순서 지정)
repIndent	NUMBER(4)	답변글 작성 시 사용(답변글의 들여쓰기 지정)

```
create table board(
    num NUMBER(4) PRIMARY KEY,
    author VARCHAR2(20),
    title  VARCHAR2(50),
    content VARCHAR2(300),
```

```

        writeday DATE DEFAULT SYSDATE,
        readCnt NUMBER(4) DEFAULT 0,
        repRoot NUMBER(4),
        repStep NUMBER(4),
        replndent NUMBER(4)
    );

drop sequence board_seq;
create sequence board_seq;

insert into board( num, author, title, content, repRoot, repStep, replndent )
values ( board_seq.NEXTVAL , '홍길동', '테스트' , '테스트입니다.' , board_seq.CURRVAL, 0 , 0 );
commit;

```

(2) Controller 설계

클라이언트의 요청에 대한 최초 진입점 역할 담당(FrontController 패턴)하고 Model과 View을 관리하는 서블릿을 설계한다.

BoardFrontController.java	MVC의 Controller 역할 담당하는 서블릿 맵핑명은 *.do로 지정한다.
---------------------------	---

(3) Model 설계

클라이언트의 요청에 대한 실제 작업을 처리하는 Business Logic 설계이다. Command 패턴 및 DAO 패턴, DTO 패턴을 적용하여 설계한다.

Command 패턴 클래스 목록

Command 패턴 클래스	설 명
BoardCommand	Command 패턴 인터페이스
BoardListCommand	게시판 목록 보기 비즈니스 로직 처리 클래스
BoardWriteCommand	게시판 글쓰기 비즈니스 로직 처리 클래스
BoardRetrieveCommand	게시판 글 자세히 보기 비즈니스 로직 처리 클래스
BoardpwdCheckCommand	본인 글 확인 비밀번호 입력 품 보기 비즈니스 로직 처리 클래스
BoardpwdCheckFormCommand	본인 글 확인 비밀번호 확인 비즈니스 로직 처리 클래스
BoardUpdateCommand	게시판 글 수정하기 비즈니스 로직 처리 클래스
BoardDeleteCommand	게시판 글 삭제하기 비즈니스 로직 처리 클래스
BoardReplyUICommand	답변글 입력 품 보기 비즈니스 로직 처리 클래스
BoardReplyCommand	답변글 쓰기 비즈니스 로직 처리 클래스
BoardSearchCommand	게시판 검색 기능 비즈니스 로직 처리 클래스
BoardPageCommand	게시판 페이지 처리 비즈니스 로직 클래스

DAO 패턴 및 DTO 패턴 클래스 목록

DAO 및 DTO 패턴	설명
BoardDTO	board 테이블의 레코드를 저장하기 위한 도메인 클래스
BoardDAO	데이터베이스 관리 클래스
PageTO	게시판 페이징 처리 관련 데이터 관리 클래스

(4) View 설계

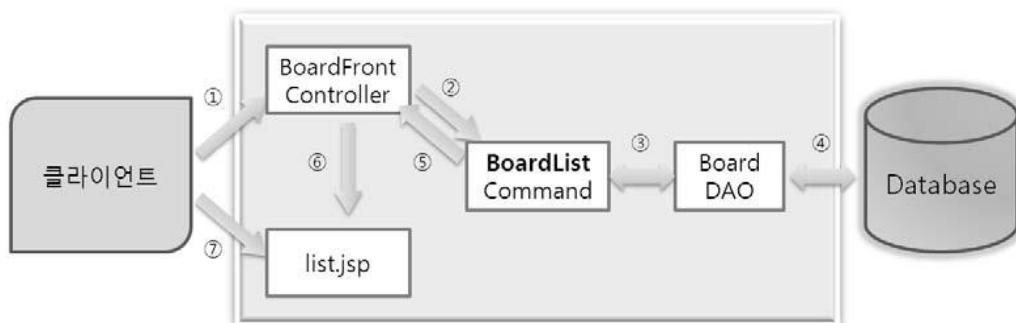
클라이언트 요청에 대한 응답 처리인 presentation logic 처리하는 설계이다. 최대한 간단하게 코드를 작성하기 때문에 기능 구현에 목적을 둔 자바 코드에 집중한다.

JSP 페이지 파일	설명
list.jsp	게시판의 목록 보기
write.jsp	게시판의 글쓰기 화면 폼
retrieve.jsp	글번호를 선택 시 자세히 보기
update.jsp	게시판의 글수정 화면 폼
passwdChk.jsp	본인이 작성한 글 여부를 확인하기 위한 비밀번호 입력 화면 폼
reply.jsp	답변글 입력 폼
page.jsp	페이징 처리 화면
listPage.jsp	페이징 처리 화면

23.3 MVC 모델2 게시판 구축

(1) 글 목록 보기

서블릿 명	com.controller.BoardFrontController.java
Command 인터페이스와 클래스	com.service.BoardCommand.java com.service.BoardListCommand.java
DTO 클래스	com.entity.BoardDTO.java
DAO 클래스	com.dao.BoardDAO.java
View 파일명	list.jsp



BoardDTO.java

```

package com.entity;
public class BoardDTO {
  
```

```
private int num;
private String author;
private String title;
private String content;
private int readcnt;
private String writeday;
private int repRoot;
private int repStep;
private int replndent;
private String passwd;

public BoardDTO() {}
public BoardDTO(int num, String author, String title, String content,
               int readcnt, String writeday, int repRoot, int repStep,
               int replndent) {
    this.num = num;
    this.author = author;
    this.title = title;
    this.content = content;
    this.readcnt = readcnt;
    this.writeday = writeday;
    this.repRoot = repRoot;
    this.repStep = repStep;
    this.replndent = replndent;
}
public int getNum() {
    return num;
}
public void setNum(int num) {
    this.num = num;
}
public String getAuthor() {
    return author;
}
public void setAuthor(String author) {
    this.author = author;
}
public String getTitle() {
    return title;
}
```

```
public void setTitle(String title) {
    this.title = title;
}
public String getContent() {
    return content;
}
public void setContent(String content) {
    this.content = content;
}
public int getReadcnt() {
    return readcnt;
}
public void setReadcnt(int readcnt) {
    this.readcnt = readcnt;
}
public String getWriteday() {
    return writeday;
}
public void setWriteday(String writeday) {
    this.writeday = writeday;
}
public int getRepRoot() {
    return repRoot;
}
public void setRepRoot(int repRoot) {
    this.repRoot = repRoot;
}
public int getRepStep() {
    return repStep;
}
public void setRepStep(int repStep) {
    this.repStep = repStep;
}
public int getReplndent() {
    return replndent;
}
public void setReplndent(int replndent) {
    this.replndent = replndent;
}
public String getPasswd()
```

```

        return passwd;
    }

    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }
}

```

BoardDAO.java

```

package com.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import com.entity.BoardDTO;

public class BoardDAO {

    DataSource dataFactory;
    public BoardDAO(){ //생성자
        //DataSource 얻기, 커넥션 풀 사용
        try{
            Context ctx = new InitialContext();
            dataFactory = (DataSource)ctx.lookup("java:comp/env/jdbc/Oracle11g");
        }catch(Exception e){ e.printStackTrace();}
    }//end 생성자

    //목록보기
    public ArrayList<BoardDTO> list(){
        ArrayList<BoardDTO> list = new ArrayList<BoardDTO>();
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try{
            con = dataFactory.getConnection();
            String query = "SELECT num , author, title, content , to_char( writeday , 'YYYY/MM/DD') writeday , readcnt , repRoot, repStep, repIndent FROM board order by repRoot desc , repStep asc";
            pstmt = con.prepareStatement(query);
            rs = pstmt.executeQuery();
        }

```

```

        while( rs.next()){
            BoardDTO data = new BoardDTO();
            data.setNum( rs.getInt( "num" ) );
            data.setAuthor( rs.getString( "author" ) );
            data.setTitle( rs.getString( "title" ) );
            data.setContent( rs.getString( "content" ) );
            data.setWriteday( rs.getString( "writeday" ) );
            data.setReadcnt( rs.getInt( "readcnt" ) );
            data.setRepRoot( rs.getInt( "repRoot" ) );
            data.setRepStep( rs.getInt( "repStep" ) );
            data.setRepIndent( rs.getInt( "repIndent" ) );

            list.add( data );
        } //end while
    } catch( Exception e ){
        e.printStackTrace();
    } finally{
        try {
            if( rs!= null) rs.close();
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e ) {
            e.printStackTrace();
        }
    }
    return list;
} //end select
} //end class

```

Command 패턴을 적용하기 위한 인터페이스 작성

BoardCommand.java

```

package com.service;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public interface BoardCommand {
    public void execute(HttpServletRequest request, HttpServletResponse response);
}

```

Commad 패턴을 적용하여 목록보기 기능을 구현한 클래스 작성

BoardListCommand.java

```
package com.service;
```

```

import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
import com.entity.BoardDTO;
public class BoardListCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        BoardDAO dao = new BoardDAO();
        ArrayList<BoardDTO> list = dao.list();
        request.setAttribute("list", list);
    }//end execute
}//end classs

```

Controller 서블릿 작성

BoardFrontController

```

package com.controller;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
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 com.service.BoardCommand;
import com.service.BoardListCommand;

@WebServlet("*.do")
public class BoardFrontController extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.setCharacterEncoding("EUC-KR");
        String requestURI = request.getRequestURI();
        String contextPath = request.getContextPath();
        String com = requestURI.substring(contextPath.length());
        BoardCommand command = null;
    }
}

```

```

String nextPage = null;
// 목록보기
if(com.equals("/list.do")){
    command = new BoardListCommand();
    command.execute(request, response);
    nextPage = "list.jsp";
}
RequestDispatcher dis = request.getRequestDispatcher(nextPage);
dis.forward(request, response);
}
}

```

presentation logic을 구현한 View 파일 작성(JSTL과 EL 표기법 사용)

list.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR" pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!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>목록보기</title>
</head>
<body>
    <h1>게시판 목록 보기</h1>
    <table border="1">
        <tr>
            <td colspan="5">
                <form action="search.do" >
                    <select name="searchName" size="1">
                        <option value="author">작성자</option>
                        <option value="title">글제목</option>
                    </select>
                    <input type="text" name="searchValue" >
                    <input type="submit" value="찾기" >
                </form>
            </td>
        </tr>
        <tr>
            <td>번호</td>

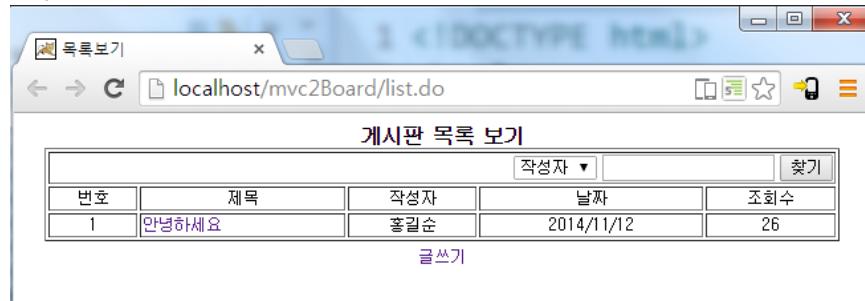
```

```

<td>제목</td>
<td>작성자</td>
<td>날짜</td>
<td>조회수</td>
</tr>
<c:forEach items="${list}" var="dto">
    <tr>
        <td>${dto.num}</td>
        <td>${dto.title}</td>
        <td>${dto.author}</td>
        <td>${dto.writeday}</td>
        <td>${dto.readcnt}</td>
    </tr>
</c:forEach>
</table>
<a href="writeui.do">글쓰기</a>
</body>
</html>

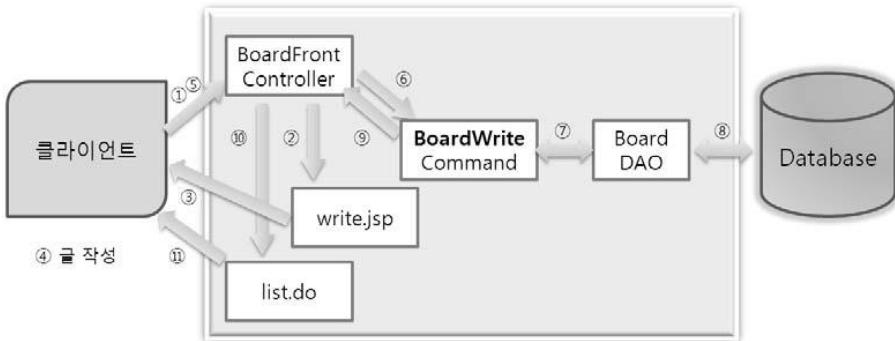
```

<http://localhost/mvc2Board/list.do>



(2) 글 쓰기 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardWriteCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	write.jsp



BoardDAO.java(코드 추가)

```

//목록보기
public ArrayList<BoardDTO> list(){      //생략...
}

//글쓰기
public void write(BoardDTO dto){
    Connection con = null;
    PreparedStatement pstmt = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("INSERT INTO board (num, title, author, content,");
        query.append(" repRoot, repStep, repIndent, passwd) values");
        query.append("(board_seq.nextval, ?, ?, ?, ?, board_seq.currval, 0, 0, ?)");
        pstmt = con.prepareStatement(query.toString());
        pstmt.setString(1, dto.getTitle());
        pstmt.setString(2, dto.getAuthor());
        pstmt.setString(3, dto.getContent());
        pstmt.setString(4, dto.getPasswd());
        int n = pstmt.executeUpdate();
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        try {
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }//end finally
}//end write

```

글쓰기 기능을 구현하는 클래스 작성

BoardWriteCommand.java

```
package com.service;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
public class BoardWriteCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        BoardDTO dto = new BoardDTO();
        dto.setTitle(request.getParameter("title"));
        dto.setAuthor(request.getParameter("author"));
        dto.setContent(request.getParameter("content"));
        dto.setPasswd(request.getParameter("passwd"));

        BoardDAO dao = new BoardDAO();
        dao.write(dto);
    }
}
```

BoardFrontController.java(코드 추가)

```
import com.service.BoardWriteCommand;
//중략
    // 글쓰기 폼
    if(com.equals("/writeui.do")){
        nextPage = "write.jsp";
    }
    // 글쓰기
    if(com.equals("/write.do")){
        command = new BoardWriteCommand();
        command.execute(request, response);
        nextPage = "list.do";
    }
}
```

게시판 글쓰기 파일 작성

write.jsp

```
<%@ 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>
```

```

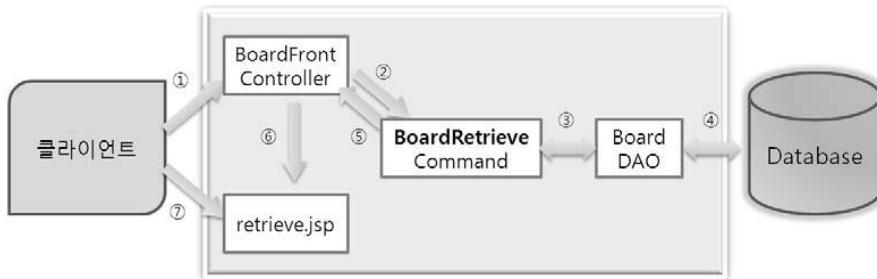
<head><meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>글쓰기</title>
</head>
<body>
    <h1>게시판 글쓰기 화면</h1>
    <form action="write.do" method="post">
        <table border="1" width="600" align="center">
            <colgroup>
                <col width="250"></col>
                <col width="350"></col>
            </colgroup>
            <tr>
                <td>제목</td>
                <td><input type="text" name="title"></td>
            </tr>
            <tr>
                <td>작성자</td>
                <td><input type="text" name="author"></td>
            </tr>
            <tr>
                <td>내용</td>
                <td><textarea name="content" cols="80" rows="5"></textarea>
            </td>
            </tr>
            <tr>
                <td>비밀번호</td>
                <td><input type="password" name="passwd"></td>
            </tr>
        </table>
        <div>
            <input type="submit" value="저장" />
            <input type="button" value="목록보기" onclick="location.href='list.do'" />
        </div>
    </form>
</body></html>

```

※ 게시판 목록에서 글쓰기 버튼을 클릭하면 아래와 같이 **게시판 글쓰기 화면**이 나타난다.

(3) 글 내용 보기 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardRetrieveCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	list.jsp(수정), retrieve.jsp



BoardDAO 클래스에 글 내용보기 추가 코드 작성과 조회수 값을 1 증가시키는 코드 추가

BoardDAO.java(코드 추가)

```
//조회수 1 증가
public void readCount( String _num ){
    Connection con = null;
    PreparedStatement pstmt = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("UPDATE board SET readcnt = readcnt + 1 ");
        query.append("WHERE num=?");
        pstmt = con.prepareStatement(query.toString());
        pstmt.setInt(1, Integer.parseInt(_num));
        pstmt.executeUpdate();
    }catch( Exception e){
        e.printStackTrace();
    }finally{
        try {
            if( pstmt!= null) pstmt.close();
        }
    }
}
```

```

                if( con!= null) con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        } //end finally
    } //end readCount

    //글 자세히 보기
    public BoardDTO retrieve( String _num ){
        // 조회수 증가
        readCount( _num );
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        BoardDTO data = new BoardDTO();
        try{
            con = dataFactory.getConnection();
            StringBuffer query = new StringBuffer();
            query.append("SELECT num, author, title, content, ");
            query.append("writeday, readcnt, repRoot, replndent, ");
            query.append("repStep FROM board WHERE num = ?");
            pstmt = con.prepareStatement( query.toString() );
            pstmt.setInt( 1, Integer.parseInt(_num));
            rs= pstmt.executeQuery();
            if(rs.next()){
                data.setNum(rs.getInt("num"));
                data.setTitle(rs.getString("title"));
                data.setAuthor(rs.getString("author"));
                data.setContent(rs.getString("content"));
                data.setWriteday(rs.getString("writeday"));
                data.setReadcnt(rs.getInt("readcnt"));
            }
        } //end if
        }catch( Exception e){
            e.printStackTrace();
        }finally{
            try {
                if( rs!= null) rs.close();
                if( pstmt!= null) pstmt.close();
                if( con!= null) con.close();
            } catch (SQLException e) {

```

```

        e.printStackTrace();
    }
}

return data;
}//end retrieve

```

글 내용 보기 구현하는 클래스 작성

BoardRetrieveCommand.java

```

package com.service;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
import com.entity.BoardDTO;
public class BoardRetrieveCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        String num = request.getParameter("num");
        BoardDAO dao = new BoardDAO();
        BoardDTO data = dao.retrieve( num );
        request.setAttribute( "retrieve" , data );
    }
}

```

BoardFrontController 서블릿에 추가

BoardFrontController.java(코드 추가)

```

import com.service.BoardRetrieveCommand;
//중략
// 글 자세히 보기
if(com.equals("/retrieve.do")){
    command = new BoardRetrieveCommand();
    command.execute(request, response);
    nextPage = "retrieve.jsp";
}

```

list.jsp를 다음과 같이 수정한다.

```

<td>${dto.title}</td>
↓
<td><a href="retrieve.do?num=${dto.num}">${dto.title}</a></td>

```

글 내용 보기를 구현하기 위한 View 파일 작성

retrieve.jsp

```
<%@ 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>글 자세히 보기</title>
    <script type="text/javascript">
        function pwdUrl(mode){
            document.frm.action="pwdCheckui.do";
            document.frm.method="post";
            document.frm.mode.value=mode;
            document.frm.submit();
        }
    </script>
</head>
<body>
    <h1>글 자세히 보기</h1>
    <form name="frm">
        <input type="hidden" name="num" value="${retrieve.num}">
        <input type="hidden" name="mode">
    </form>
    <table border="1" width="600">
        <colgroup>
            <col width="150"></col>
            <col width="150"></col>
            <col width="150"></col>
            <col width="150"></col>
        </colgroup>
        <tr>
            <td>글번호</td>
            <td>${retrieve.num} <span> (조회수: ${retrieve.readcnt}) </span></td>
            <td>작성일</td>
            <td>${retrieve.writeday}</td>
        </tr>
        <tr>
            <td>제목</td>
```

```

        <td colspan="3" ${retrieve.title}</td>
    </tr>
    <tr>
        <td>작성자</td>
        <td colspan="3" ${retrieve.author}</td>
    <tr>
        <td>내용</td>
        <td colspan="3" ${retrieve.content}</td>
    </tr>
</table>
<div>
<a href="list.do">[글목록]</a>&nbsp;&nbsp;
<a href="javascript:pwdUrl('update')">[글수정]</a>&nbsp;&nbsp;
<a href="javascript:pwdUrl('delete')">[글삭제]</a>&nbsp;&nbsp;
<a href="replyui.do?num=${retrieve.num}">[답변달기]</a>
</div>
</body>
</html>

```

The screenshot displays two windows from a web browser. The top window is titled '게시판 목록 보기' and shows a table of posts. The 41st post, which has the title '풀해도 얼마 남지 않았네요~!', is highlighted with a red border. A red arrow points to this post. The bottom window is titled '글 자세히 보기' and shows the detailed view of the post with ID 41. It includes fields for 글번호 (Post Number), 제목 (Title), 작성일 (Creation Date), 작성자 (Author), 날짜 (Date), 조회수 (View Count), and 내용 (Content). The content field contains the text '풀해도 얼마 남지 않았네요~! 조금만 더 힘을 내어 열심히 합시다.~~!'.

(4) 비밀번호 확인 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	BoardpwdCheckFormCommand(비밀번호 확인 폼) BoardpwdCheckCommand(비밀번호 확인)
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	passwdChk.jsp

BoardDAO 클래스에 글 수정 코드 추가

BoardDAO.java

```
// 비밀번호 체크
public Map<String, String> pwdCheck(String _num, String _mode, String _passwd){
    Map<String, String> map = new HashMap<String, String>();
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    String pwdChk = null;
    try{
        con = dataFactory.getConnection();
        String query = "SELECT passwd FROM board WHERE num = ?";
        pstmt = con.prepareStatement(query);
        pstmt.setInt(1, Integer.parseInt(_num));
        rs = pstmt.executeQuery();

        if( rs.next()){
            pwdChk=rs.getString("passwd");
        }

        if( pwdChk.equals(_passwd)){
            if(_mode.equals("update")){
                map.put("resultUrl", "updateui.do");
            }else if(_mode.equals("delete")){
                map.put("resultUrl", "delete.do");
            }
        } else{
            map.put("resultUrl", "pwdCheckui.do");
            map.put("resultMsg", "비밀번호가 일치하지 않습니다.");
        }
    }catch( Exception e){
        e.printStackTrace();
    }finally{
        try {
            if( rs!= null) rs.close();
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
        }  
        return map;  
    }  
}
```

비밀번호 확인 폼 및 처리를 구현하는 클래스 작성

BoardpwdCheckFormCommand.java(입력 폼 구현)

```
package com.service;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
public class BoardpwdCheckFormCommand implements BoardCommand {  
    @Override  
    public void execute(HttpServletRequest request, HttpServletResponse response) {  
        String num = request.getParameter("num");  
        String mode = request.getParameter("mode");  
  
        request.setAttribute("num", num);  
        request.setAttribute("mode", mode);  
    }  
}
```

BoardpwdCheckCommand.java(비밀번호 확인 구현)

```
package com.service;  
  
import java.util.Map;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import com.dao.BoardDAO;  
  
public class BoardpwdCheckCommand implements BoardCommand {  
    @Override  
    public void execute(HttpServletRequest request, HttpServletResponse response) {  
        String num = request.getParameter("num");  
        String mode = request.getParameter("mode");  
        String passwd = request.getParameter("passwd");  
  
        BoardDAO dao = new BoardDAO();  
        Map<String, String> map = dao.pwdCheck(num, mode, passwd);  
        request.setAttribute("num", num);  
    }  
}
```

```

        request.setAttribute("resultUrl", map.get("resultUrl"));
        request.setAttribute("resultMsg", map.get("resultMsg"));
    }
}

```

BoardFrontController 서블릿에 코드 추가

BoardFrontController.java

```

import com.service.BoardUpdateCommand;
//중략
// 비밀번호 입력 화면
if(com.equals("/pwdCheckui.do")){
    command = new BoardpwdCheckFormCommand();
    command.execute(request, response);
    nextPage = "passwdChk.jsp";
}
// 비밀번호 체크
if(com.equals("/pwdCheck.do")){
    command = new BoardpwdCheckCommand();
    command.execute(request, response);
    nextPage = (String)request.getAttribute("resultUrl");
}

```

비밀번호 입력 폼 구현하기 위한 View 파일 작성

passwdChk.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR" pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!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>비밀번호 확인</title>
    <script type="text/JavaScript">
        function chkSave(){
            if ( document.frm.passwd.value.replace(/\s/g,"")=="" ) {
                alert("암호를 입력해 주세요.");
                document.frm.passwd.focus();
                return false;
            }
        }
    </script>
</head>
<body>
    <form name="frm">
        <input type="password" name="passwd" value="" />
        <input type="button" value="확인" onclick="chkSave()" />
    </form>
</body>

```

```

function loadData(){
    var msg = "${resultMsg}";
    if(msg!=""){
        alert(msg);
    }
}

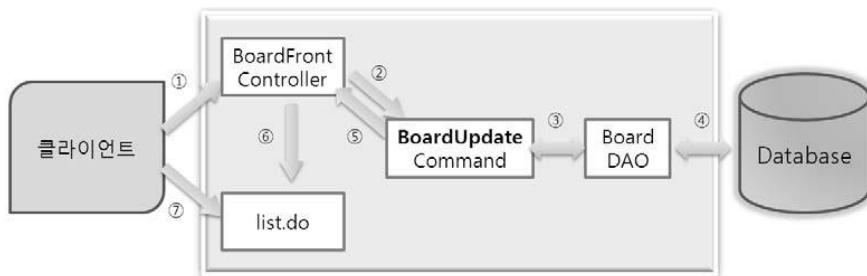
</script>
</head>
<body onload="loadData()">

    <h1>글수정 및 삭제 전 확인</h1>
    <form method="POST" name="frm" action="pwdCheck.do" onsubmit="return chkSave()"onclick="location.href='/mvc2Board/list.do'"

```

(5) 글 수정 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardUpdateCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	update.jsp



BoardDAO 클래스에 글 수정 코드 추가

BoardDAO.java

```

//글 수정하기
public void update(BoardDTO dto){
    Connection con = null;
    PreparedStatement pstmt = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("UPDATE board SET title = ?, author = ?, ");
        query.append("content = ?, passwd = ? WHERE num = ?");
        pstmt = con.prepareStatement(query.toString());
        pstmt.setString (1, dto.getTitle());
        pstmt.setString(2, dto.getAuthor());
        pstmt.setString(3, dto.getContent());
        pstmt.setString(4, dto.getPasswd());
        pstmt.setInt(5, dto.getNum());
        pstmt.executeUpdate();
    }catch( Exception e){
        e.printStackTrace();
    }finally{

```

```

        try {
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}//end update

```

글 수정을 구현하는 클래스 작성

BoardUpdateCommand.java

```

package com.service;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
import com.entity.BoardDTO;
public class BoardUpdateCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        BoardDTO dto = new BoardDTO();
        dto.setNum(Integer.parseInt(request.getParameter("num")));
        dto.setTitle(request.getParameter("title"));
        dto.setAuthor(request.getParameter("author"));
        dto.setContent(request.getParameter("content"));
        dto.setPasswd(request.getParameter("passwd"));

        BoardDAO dao = new BoardDAO();
        dao.update(dto);
    }
}

```

BoardFrontController 서블릿에 코드 추가

BoardFrontController.java

```

import com.service.BoardUpdateCommand;
//중략...
// 글 수정 화면 보기
if(com.equals("/updateui.do")){
    command = new BoardRetrieveCommand();
    command.execute(request, response);
}

```

```

        nextPage = "update.jsp";
    }
    //글 수정 하기
    if(com.equals("/update.do")){
        command = new BoardUpdateCommand();
        command.execute(request, response);
        nextPage = "list.do";
    }
}

```

글수정 폼 구현하기 위한 View 파일 작성

update.jsp

```

<%@ 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>글수정</title>
</head>
<body>
    <h1>게시판 글수정 화면</h1>
    <form method="post" action="update.do">
        <input type="hidden" name="num" value="${retrieve.num}">
        <table border="1" width="600" align="center">
            <colgroup>
                <col width="150"></col>
                <col width="150"></col>
                <col width="150"></col>
                <col width="150"></col>
            </colgroup>
            <tr>
                <td>글번호</td>
                <td>${retrieve.num} <span>(조회수: ${retrieve.readcnt})</span></td>
                <td>작성일</td>
                <td>${retrieve.writeday}</td>
            </tr>
            <tr>
                <td>제목</td>
                <td colspan="3">

```

```

<input type="text" name="title" value="${retrieve.title}" ></td>
</tr>
<tr>
    <td>작성자</td>
    <td colspan="3">
        <input type="text" name="author" value="${retrieve.author}" ></td>
    </tr>
    <td>내용</td>
    <td colspan="3">
        <textarea name="content" cols="80" rows="10" >${retrieve.content}</textarea>
    </td>
</tr>
<tr>
    <td>비밀번호</td>
    <td colspan="3"><input type="password" name="passwd"></td>
</tr>
</table>
<div><input type="submit" value="수정완료">
    <input type="button" value="목록보기" onclick="location.href='list.do'" />
</div>
</form>
</body>
</html>

```

게시판 글수정 화면

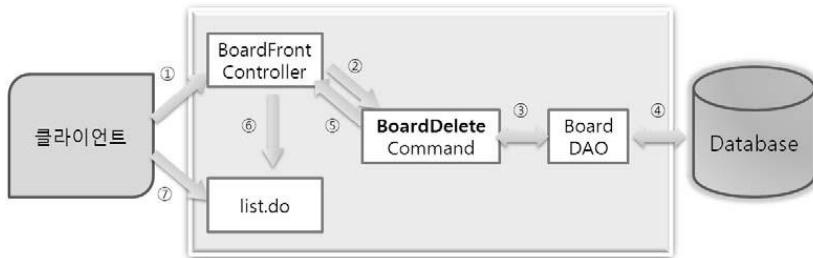
글번호	41 (조회수: 24)	작성일	2014-12-10 07:43:06.0
제목	얼마남지 않은 풀해 마무리 잘 합시다~!		
작성자	홍길동		
내용	<p>안녕하세요. 풀해도 얼마 남지 않았네요~~! 조금만 더 힘을 내어 열심히 합시다.~~!</p>		
비밀번호			

게시판 목록 보기

번호	제목	작성자	날짜	조회수
41	얼마남지 않은 풀해 마무리 잘 합시다~!	홍길동	2014/12/10	24
1	안녕하세요	홍길순	2014/11/12	27

(6) 글 삭제 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardDeleteCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)



BoardDAO 클래스에 글 삭제 작업 코드 추가

BoardDAO.java

```

//글 삭제하기
public void delete( String _num ){
    Connection con = null;
    PreparedStatement pstmt = null;
    try{
        con = dataFactory.getConnection();
        String query = "DELETE FROM board WHERE num = ?";
        pstmt = con.prepareStatement( query );
        pstmt.setInt(1, Integer.parseInt( _num ) );
        int n = pstmt.executeUpdate( );
    }catch( Exception e ){
        e.printStackTrace();
    }finally{
        try {
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
//end delete
  
```

글 삭제를 구현하는 클래스 작성

BoardDeleteCommand.java

```
package com.service;
```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
public class BoardDeleteCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        String num = request.getParameter("num");
        BoardDAO dao = new BoardDAO();
        dao.delete(num);
    }
}

```

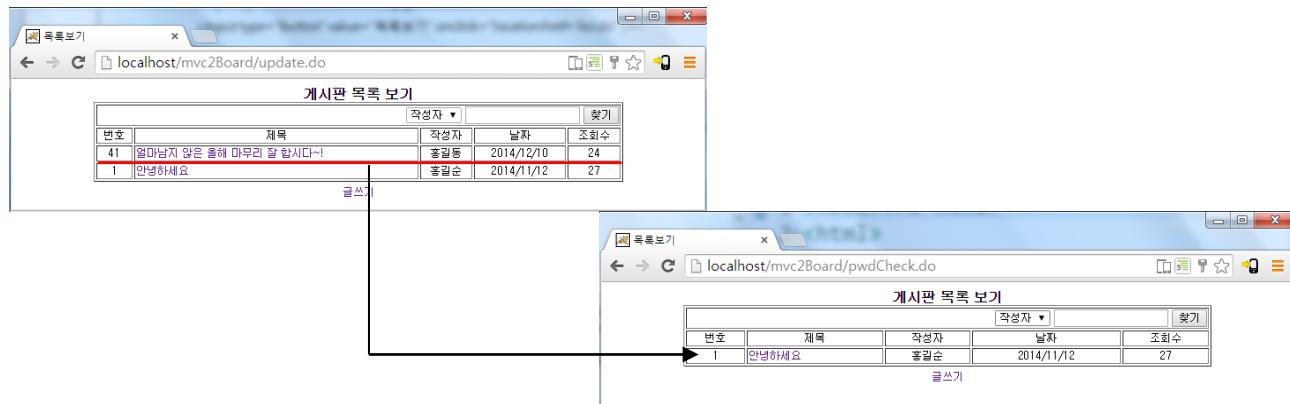
BoardFrontController 서블릿에 코드 추가

BoardFrontController.java

```

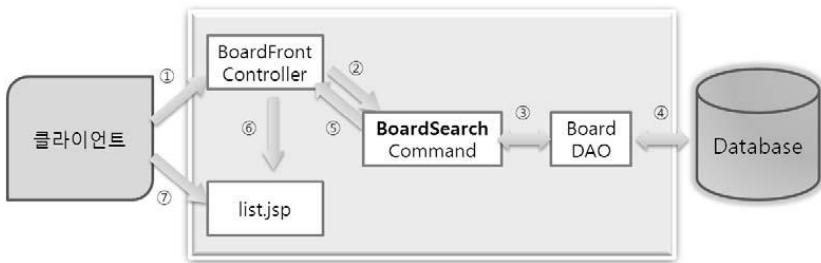
import com.service.BoardDeleteCommand;
//중략...
//글 삭제 하기
if(com.equals("/delete.do")){
    command = new BoardDeleteCommand();
    command.execute(request, response);
    nextPage = "list.do";
}

```



(7) 글 검색 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardSearchCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)



BoardDAO 클래스에 글 검색 코드 추가

BoardDAO.java

```

// 검색 하기
public ArrayList<BoardDTO> search( String _searchName, String _searchValue ) {
    ArrayList<BoardDTO> list = new ArrayList<BoardDTO>();
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try{
        con = dataFactory.getConnection();
        String query = "SELECT num, author, title, content,
to_char( writeday , 'YYYY/MM/DD') writeday, readcnt FROM board ";
        if( _searchName.equals( "title" )){
            query += " WHERE title LIKE ?";
        }else{
            query += " WHERE author LIKE ?";
        }
        pstmt = con.prepareStatement( query );
        pstmt.setString( 1, "%" + _searchValue + "%" );
        rs = pstmt.executeQuery( );
        while( rs.next() ){
            BoardDTO data = new BoardDTO();
            data.setNum(rs.getInt("num"));
            data.setAuthor(rs.getString("author"));
            data.setTitle(rs.getString("title"));
            data.setContent(rs.getString("content"));
            data.setWriteday(rs.getString("writeday"));
            data.setReadcnt(rs.getInt("readcnt"));

            list.add( data );
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}

```

```

        }finally{
            try {
                if( rs!= null) rs.close();
                if( pstmt!= null) pstmt.close();
                if( con!= null) con.close();
            }
            catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return list;
    }//end search
}

```

글 검색을 구현하는 클래스 작성

BoardSearchCommand.java

```

package com.service;

import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
import com.entity.BoardDTO;
public class BoardSearchCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        String searchName = request.getParameter( "searchName" );
        String searchValue = request.getParameter( "searchValue" );
        BoardDAO dao = new BoardDAO();
        ArrayList<BoardDTO>  list = dao.search( searchName ,  searchValue );
        request.setAttribute("list", list );
    }
}

```

BoardFrontController 서블릿에 코드 추가

BoardFrontController.java

```

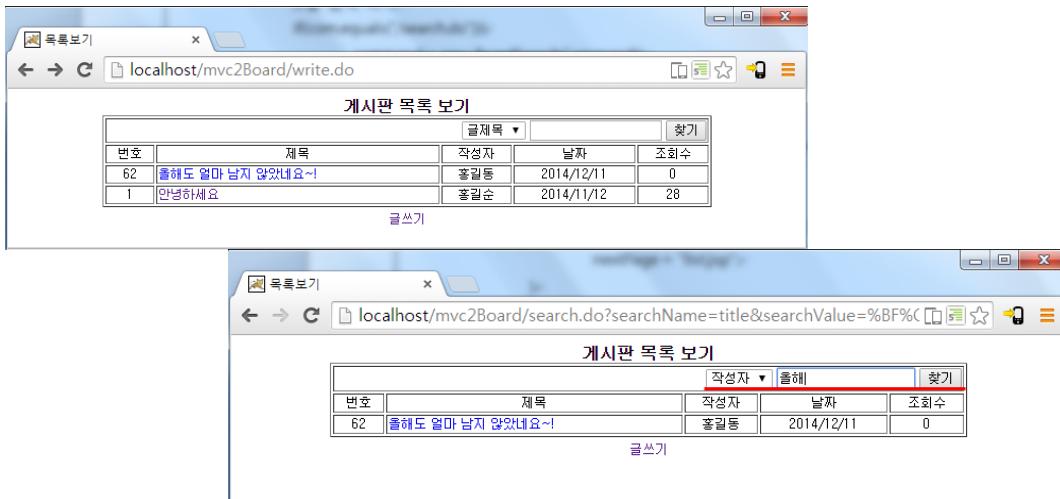
import com.service.BoardSearchCommand;
//중략
//글 검색 하기
if(com.equals("/search.do")){
    command = new BoardSearchCommand();
}

```

```

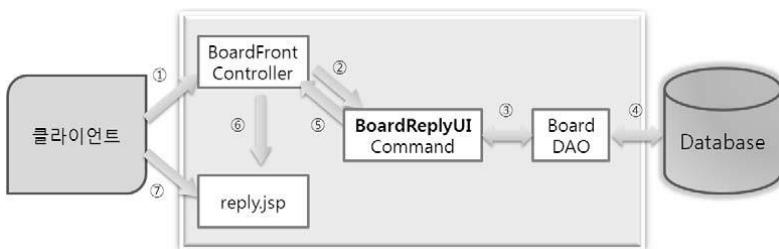
        command.execute(request, response);
        nextPage = "list.jsp";
    }
}

```



(8) 답변글 입력 폼 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardReplyUICommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	reply.jsp



BoardDAO 클래스에 답변글 입력 폼 코드 추가

BoardDAO.java

```

// 답변글 입력 폼 보기
public BoardDTO replyui( String _num ){
    BoardDTO data = new BoardDTO();
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();

```

```

        query.append("SELECT num, author, title, content, ");
        query.append("writeday, readcnt, repRoot, replndent, ");
        query.append("repStep FROM board WHERE num = ?");
        pstmt = con.prepareStatement(query.toString());
        pstmt.setInt( 1 , Integer.parseInt( _num ) );
        rs = pstmt.executeQuery();
        if( rs.next()){
            data.setNum( rs.getInt( "num" ) );
            data.setTitle( rs.getString( "title" ) );
            data.setAuthor( rs.getString( "author" ) );
            data.setContent( rs.getString( "content" ) );
            data.setWriteday( rs.getString( "writeday" ) );
            data.setReadcnt( rs.getInt( "readcnt" ) );
            data.setRepRoot( rs.getInt( "repRoot" ) );
            data.setRepStep( rs.getInt( "repStep" ) );
            data.setReplndent( rs.getInt( "replndent" ) );
        }//end if
    }catch( Exception e){
        e.printStackTrace();
    }finally{
        try {
            if( rs!= null) rs.close();
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return data;
}//end replyui

```

답변글 입력 폼 기능을 구현하는 클래스 작성

BoardReplyUICommand.java

```

package com.service;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
import com.entity.BoardDTO;
public class BoardReplyUICommand implements BoardCommand {

```

```

@Override
public void execute(HttpServletRequest request, HttpServletResponse response) {
    String num = request.getParameter("num");
    BoardDAO dao = new BoardDAO();
    BoardDTO data = dao.replyui( num );
    request.setAttribute( "replyui" , data );
}

```

BoardFrontController 서블릿에 코드 추가

BoardFrontController.java

```

import com.service.BoardReplyUICommand;
//중략
    //답변글 입력 폼 보기
    if(com.equals("/replyui.do")){
        command = new BoardReplyUICommand();
        command.execute(request, response);
nextPage = "reply.jsp";
    }

```

답변글 입력 폼인 View 파일 작성

reply.jsp

```

<%@ 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>답변글 쓰기</title>
</head>
<body>
    <h1>답변글 쓰기 화면</h1>
    <form action="reply.do" method="post">
        <!-- 답 글 필요 -->
        <input type="hidden" name="num" value="${replyui.num}">
        <input type="hidden" name="repRoot" value="${replyui.repRoot}">
        <input type="hidden" name="repStep" value="${replyui.repStep}">
        <input type="hidden" name="repIndent" value="${replyui.repIndent}">

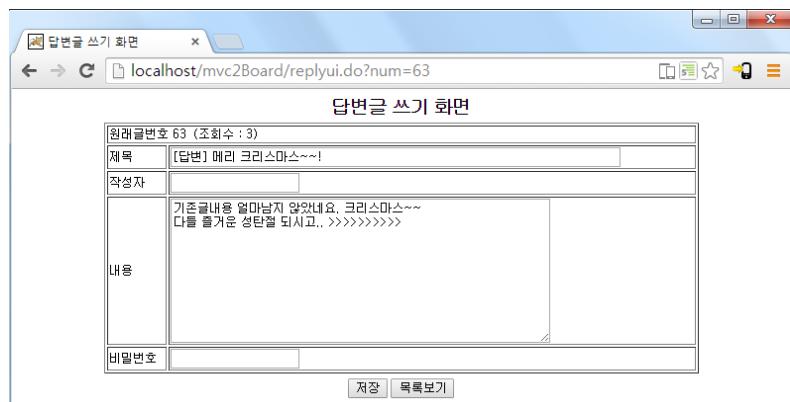
        <table border="1" width="600" align="center">

```

```

<tr>
    <td colspan="2">원래글번호 ${replyui.num}
        &nbsp;(조회수 : ${replyui.readcnt})</td>
</tr>
<tr>
    <td>제목</td>
    <td><input type="text" name="title" value="[답변] ${replyui.title}"></td>
</tr>
<tr>
    <td>작성자</td>
    <td><input type="text" name="author" size="20"></td>
</tr>
<tr>
    <td>내용</td>
    <td><textarea name="content" cols="60" rows="10" >
        기존글내용 ${replyui.content}>>>>>>></textarea></td>
</tr>
<tr>
    <td>비밀번호</td>
    <td><input type="password" name="passwd"></td>
</tr>
</table>
<div>
    <input type="submit" value="저장" />
    <input type="button" value="목록보기" onclick="location.href='list.do'" />
</div>
</form>
</body>
</html>

```



[참고]답변형 게시판 원리

※ 1. 답변형 게시판제작에 필요한 변수

답변형 게시판에 게시물들을 주제별로 관리하기 위해서는 원본 게시물과 답변을 구분하기 위한 참조형 변수와 답변에 대한 답변을 구분하는 계층변수, 동일 게시물들에 대한 출력순서를 지정하는 위치 변수가 있어야 한다.

1) 참조형 변수

참조형 변수 ref는 게시물과 답변을 구분해 주는 변수이다. 변수에 게시물의 번호를 구분하여 저장하고, 같은 그룹으로 분류하여 동일한 주제임을 구분한다.

2) 계층변수

계층변수 step은 게시물의 답변과 답변에 대한 답변을 구분하기 위해 사용되는 계층을 구분하기 위한 변수로 사용된다. 게시물이 0일 경우, 답변은 1로, 답변에 대한 답변은 2로 계층들을 구분한다.

3) 위치변수

위치변수 level은 동일한 게시물의 그룹에서 몇 번째 위치에 보여줄 것인가를 지정하는 변수이다. 게시물이 0일 때, 답변은 1이고, 답변에 대한 답변은 2로 구분한다.

※ 2. 답변형 게시판에서 변수 값 관리

1) 특정 게시물을 그룹으로 지정하기 위해 참조형 변수를 사용한다. 처음 게시된 특정 게시물의 참조형 변수(ref)를 1로 하고, 계층변수(step)와 위치변수(level)를 0으로 한다.

	ref	step	level	비고
게시물A	1	0	0	특정게시물
게시물B	2	0	0	"

2) 게시물A에 대한 [답변]-A1 게시물에 대해서 초기치를 ref=1, step=1, level=1로 한다.

	ref	step	level	비고
게시물A	1	0	0	
[답변]-A1	1	1	1	[답변]추가
게시물B	2	0	0	

3) 게시물A에 대한 [답변]-A2 게시물은 [답변]-A1 게시물과 동일하게 초기치를 ref=1, step=1, level=1로 하고[답변]-A1에 대한 게시물의 level을 1 증가한다.

	ref	step	level	비고
게시물A	1	0	0	
[답변]-A2	1	1	1	[답변]추가
[답변]-A1	1	1	2	level 증가
게시물B	2	0	0	

4) 게시물A에 대한 [답변]-A1에 대하여 [[답변]]-A11일 때, 초기치를 ref=1, step=2, level=2로 한다.

	ref	step	level	비고
게시물A	1	0	0	

[답변]-A2	1	1	1	[답변]추가
[답변]-A1	1	1	2	
[[답변]]-A11	1	2	3	[[답변]]추가
게시물B	2	0	0	

5) 게시물A에 대한 [답변]-A2에 대하여 [답변]-A21일 때, ref=1, step=2, level=2로 하고, [답변]-A1, [[답변]]-A11을 1씩 증가시킨다.

	ref	step	level	비고
게시물A	1	0	0	
[답변]-A2	1	1	1	
[[답변]]-A21	1	2	2	[[답변]]추가
[답변]-A1	1	1	3	
[[답변]]-A11	1	2	4	
게시물B	2	0	0	

답변형 게시판을 제작할 경우, ref, step, level 변수 값의 설정은

- 새 게시물의 경우 ref, step, level의 변수 값은 0으로 설정한다.
- [답변] 게시물은 자신의 상위 글의 ref와 동일한 값으로 한다.
- step은 상위 게시물보다 1을 증가시킨다.
- level은 상위 게시물보다 1을 증가시킨다.
- 추가한 [답변] 게시물보다 큰 step값을 가진 게시물이 존재할 경우 동일 그룹의 게시물에 대하여 1씩 증가 시킨다.

※ 3. 답변형 게시판 제작의 사용자 요구 사항

- 게시물에 대하여 저장, 유지, 검색할 수 있어야 한다.
- 누구나 새로운 게시물을 게시할 수 있어야 한다.
- 누구나 게시물을 읽을 수 있어야 한다.
- 게시물을 게시자가 수정할 수 있어야 한다.
- 게시물을 게시자가 삭제할 수 있어야 한다.
- 게시물에 대하여 [답변] 게시물을 게시할 수 있어야 한다.
- [답변] 게시물에 대한 [답변] 게시물을 게시할 수 있어야 한다.
- 게시물의 목록을 조회할 수 있어야 한다.
- 게시물의 조회 수를 출력해야 한다.

※ 4. 사용자의 게시판 기능별 요구 사항

게시판에서 일반 사용자는 게시물의 목록 조회, 게시물 올리기, 게시물 읽기, 게시물 수정, 게시물 답변, 게시물 삭제 기능만을 정의 한다.

기능	요구사항
게시물 목록조회	게시물번호, 아이디, 이름, 제목, 작성일, 조회수 등을 출력 화면 상단에 페이지 정보와 전체 게시물 수 출력 지정개수를 페이지 단위로 출력하고 페이지 단위로 이동 답변 게시물은 안쪽으로 표시
게시물 올리기	게시물 목록에서 [글쓰기]버튼을 누르면 게시물 등록화면 표시 등록화면에서 작성자, 비밀번호, email, 제목, 내용을 입력
게시물 읽기	게시물 목록 화면에서 게시물을 선택하면 내용을 출력 게시물을 읽으면 게시물 조회수 증가 게시물 읽기 화면에서 비밀번호를 확인하여 수정, 삭제 가능 [답변] 버튼을 누르면 현재 게시물에 답변 가능
게시물 수정	게시물 읽기 화면에서 비밀번호가 동일하게 입력될 때 수정 등록자, 작성일은 수정 불가능 수정한 후 해당 게시물이 있는 페이지로 이동
게시물 답변	게시물 읽기 화면에서 [답변] 버튼을 클릭하면 답변화면 출력 게시물과 답변을 구분
게시물 삭제	게시물 읽기 화면에서 [삭제] 버튼을 클릭하면 답변화면 출력 원본 게시물 삭제시 답변 그룹의 게시물도 동시에 삭제 게시물 삭제시 해당 게시물이 있는 페이지로 이동

※ 5. 게시물 제작을 위한 화면 설계

게시판 제작에 필요한 화면 설계는 웹 개발의 기초가 되며, 구현 전단계인 웹 디자인 부분이다. 화면 설계는 게시판 운영에 필요한 게시물 목록보기, 게시물 올리기, 게시물 읽기, 게시물 수정, 게시물 답변, 게시물 삭제를 위한 비밀번호 입력 화면을 설계한다. 화면 설계에서 중요한 점은 화면을 구성하는 항목들을 자세히 기술하는 것이다.

- 간단한 기능이라도 화면에 필요한 정보를 자세히 기술한다.
- 사용자 중심으로 디자인 한다.
- 화면의 흐름을 기술한다.
- 화면상에서 발생할 수 있는 이벤트를 모두 정의 한다.

1) 게시물 목록 조회 화면

게시물 목록은 게시물 번호, 제목, 글쓴이, 작성일, 조회수와 [글쓰기], [목록보기] 버튼으로 구성한다. 게시물 목록 출력은 페이지 네비게이션 기법을 적용하고, 번호순으로 출력되도록 한다.

게시판				
번호	제목	글쓴이	작성일	조회수
1	테스트입니다.	홍길동	2014.12.01	1

[이전] 1 2 3 4 5 [다음]

[글쓰기] [목록보기]

2) 게시물 쓰기 화면

게시물 쓰기 입력화면은 등록자, 비밀번호, E-mail, 제목, 내용과 [쓰기], [취소] 버튼으로 구성한다.

게시물 쓰기			
등록자	홍길동	비밀번호	123456
E-Mail	123@naver.com		
제목	테스트입니다.		
내용	게시판 글쓰기 테스트입니다. ^^		

[쓰기] [취소]

3) 게시물 읽기 화면

게시물 읽기 출력 화면은 등록자, e-mail 주소, 제목, 내용과 [답글쓰기], [수정], [삭제], [목록보기] 버튼으로 구성한다.

게시물 읽기			
등록자	홍길동	작성일	2014.12.01
E-Mail	123@naver.com		
제목	테스트입니다.		
내용	게시판 글쓰기 테스트입니다. ^^		

[답글쓰기] [수정] [삭제] [목록보기]

4) 게시물 수정화면

게시물 수정			
등록자	홍길동	작성일	2014.12.01
비밀번호	123456		
E-Mail	123@naver.com		
제목	테스트입니다.		
내용	게시판 글쓰기 테스트입니다. ^^		

[수정] [취소]

5) 게시물 답변 화면

게시물 답변			
등록자	홍길동	비밀번호	123456
E-Mail	123@naver.com		
제목	[답변] 테스트입니다.		
내용	게시판 글쓰기 테스트입니다. ^^		

[답변] [취소]

6) 게시물 삭제를 위한 비밀번호 입력화면

게시물의 비밀번호를 입력하세요	
12345	[확인]

※ 6. 게시판 작성을 위한 테이블 구성을

게시판 작성은 위해 설계한 입력화면과 출력화면을 기초로 게시물에 관련된 정보를 추출한다. 게시판 관리에 필요한 정보들은 게시물 구분번호, 글쓴이, 비밀번호, 이메일 주소, 제목, 내용, 작성일자, 조회수, IP주소, 동일 게시물 그룹번호, 동일그룹 계층번호, 동일 그룹 게시물 위치번호로 구성하며, 테이블명은 "board"로 하고, 게시물 구분번호를 기본키로 지정한다.

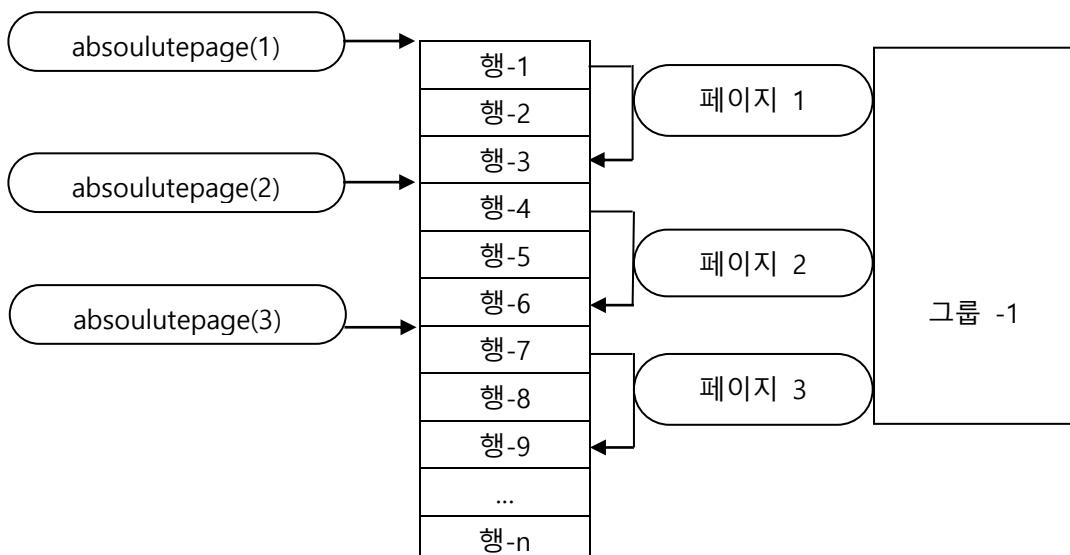
추가된 컬럼명

컬럼 설명	컬럼명	데이터형	크기	Not Null	Key
그룹번호	ref	숫자형	5		
계층번호	step	숫자형	5		
위치번호	level	숫자형	5		

1) 게시판 목록 조회

게시판 목록 조회는 게시판을 실행되는 초기 프로그램이다. 게시판 테이블(board)로부터 각각의 행들은 검색하여 순서대로 게시물 번호(b_id), 제목(b_title), 성명(b_name), 작성일(b_date), 조회수(b_hit)를 페이지 단위로 출력한다. 게시물의 목록을 조회할 때 페이지당 출력 목록을 제한하는 페이지 내비게이션을 사용한다. 페이지 내비게이션 구현 기법에 필요한 변수를 지정하고, 초기 값을 부여한다.

요청 페이지에 대한 커서의 포인터 위치



목록 조회를 위한 페이지 내비게이션에 필요한 변수 및 초기 값 표

변수	초기 값	설 명
pageSize	10	한 페이지에 출력되는 행의 수
limit	3	페이지를 관리할 그룹 수
pageNUM	1	요청 페이지
startPage	1	현재 페이지
dbcount	0	전체 행의 수
absolutepage	1	절대위치 페이지 번호
pageCount	1	전체 페이지 수

DB 서버 접속은 DBConnectionPool을 이용하고 SQL의 max() 함수를 이용하여 board 테이블로부터

전체 행의 수를 계산하고, 페이지 수는 (전체행의수/한페이지행의수)으로 계산한다.

※ 알고리즘

게시물의 개수 구하는 select 문

board 테이블로부터 count()함수를 사용하여 행의 수를 계산한다.

```
select count(b_id) from board;
```

총 페이지 수를 구하는 식

총 페이지 수 = (게시물의 개수) / (한 페이지의 게시물 수)

이 계산식의 나머지가 0 이 아니면 1 페이지를 증가한다.

페이지 단위로 출력

한 페이지 이상일 때, 페이지 크기에 따라 행을 페이지로 나눈다. 특정 페이지로 이동할 경우 absolute() 메서드를 사용한다. 이 메서드가 지원 되지 않으면 출력할 페이지 행을 계산하여 이동해야 한다. 출력할 행의 계산식

출력할 행의 수 = (출력페이지번호 -1) * 페이지당 행의 수 + 1

출력할 행으로 이동할 경우

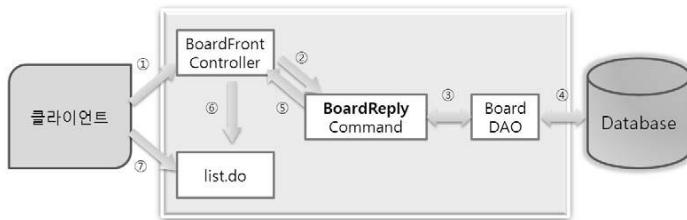
```
for(int k=1; k < absolutepage; k +1) rs.next();
```

출력할 게시물 내용을 구하는 SQL문

```
select b_id, b_name, b_title, b_content, b_date, b_hit , ref, step, level  
from board  
order by ref desc , level asc;
```

(9) 답변글 쓰기 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardReplyCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	list.jsp(코드 수정)



BoardDAO 클래스에 답변글 작업을 위한 코드 추가

BoardDAO.java

```

//답변글의 기준 repStep 1 증가
public void makeReply(int _root , int _step ){
    Connection con = null;
    PreparedStatement pstmt = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("UPDATE board SET repStep = repStep + 1 ");
        query.append("WHERE repRoot = ? AND repStep > ? ");
        pstmt = con.prepareStatement(query.toString());
        pstmt.setInt(1, _root);
        pstmt.setInt(2, _step);
        pstmt.executeUpdate();
    }catch( Exception e){
        e.printStackTrace();
    }finally{
        try {
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
  
```

```

//답변 달기
public void reply(BoardDTO dto){
    makeReply(dto.getRepRoot(), dto.getRepStep());
    Connection con = null;
    PreparedStatement pstmt = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("INSERT INTO board(num, title, author, ");
        query.append("content, repRoot, repStep, replIndent, passwd)");
        query.append("values( board_seq.nextVal, ?, ?, ?, ?, ?, ?, ?)");
        pstmt = con.prepareStatement(query.toString());
        pstmt.setString (1, dto.getTitle());
        pstmt.setString (2, dto.getAuthor());
        pstmt.setString (3, dto.getContent());
        pstmt.setInt (4, dto.getRepRoot());
        pstmt.setInt (5, dto.getRepStep() + 1);
        pstmt.setInt (6, dto.getReplIndent() + 1);
        pstmt.setString (7, dto.getPasswd());
        pstmt.executeUpdate();
    }catch( Exception e){
        e.printStackTrace();
    }finally{
        try {
            if( pstmt!= null) pstmt.close();
            if( con!= null) con.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}//end reply

```

답변글 쓰기 기능을 구현하는 클래스 작성

BoardReplyCommand.java

```

package com.service;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```

import com.dao.BoardDAO;
import com.entity.BoardDTO;
public class BoardReplyCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        if(request.getParameter("num")!=null){
            BoardDTO dto = new BoardDTO();
            dto.setNum(Integer.parseInt(request.getParameter("num")));
            dto.setTitle(request.getParameter("title"));
            dto.setAuthor(request.getParameter("author"));
            dto.setContent(request.getParameter("content"));
            dto.setRepRoot(Integer.parseInt(request.getParameter("repRoot")));
            dto.setRepIndent(Integer.parseInt(request.getParameter("repIndent")));
            dto.setRepStep(Integer.parseInt(request.getParameter("repStep")));
            dto.setPasswd(request.getParameter("passwd"));

            BoardDAO dao = new BoardDAO();
            dao.reply(dto);
        }
    }
}

```

BoardFrontController 서블릿에 코드 추가

BoardFrontController.java

```

import com.service.BoardReplyCommand;
//중략
    //답변글 쓰기
    if(com.equals("/reply.do")){
        command = new BoardReplyCommand();
        command.execute(request, response);
        nextPage = "list.do";
    }
}

```

답변글 들여쓰기 구현을 위해 list.jsp를 다음과 같이 수정한다.

<td> \${dto.title} </td>
--

↓

<td> <c:if test="\${dto.repIndent > 0}"> </c:if>

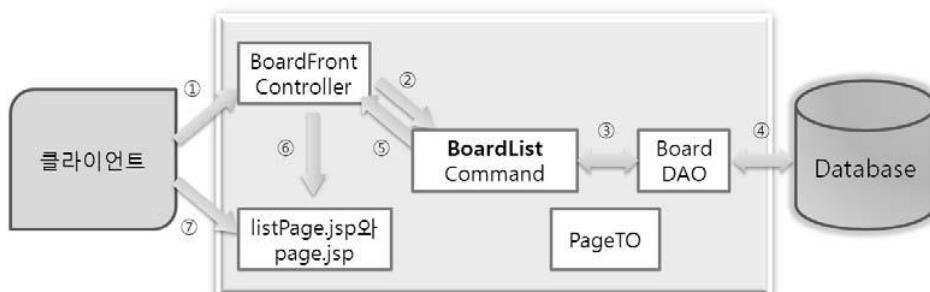
```
<a href="retrieve.do?num=${dto.num}">${dto.title}</a>
```

```
</td>
```

(10) 페이징 처리 구현

서블릿 명	com.controller.BoardFrontController.java(코드 추가)
Command 클래스	com.service.BoardPageCommand.java
DAO 클래스	com.dao.BoardDAO.java(코드 추가)
View 파일명	listPage.jsp
페이지 관련 클래스	PageTO.java

기존 list.jsp를 수정하지 않고 페이징 처리에 필요한 파일을 추가하는 방법으로 구현한다. 따라서 페이징 처리가 필요없는 경우와 필요한 경우를 분리할 수 있다.



페이징 처리시 필요한 데이터는 목록 리스트 데이터, 현재 페이지 번호, 전체 레코드 수, 페이지 당 보여줄 레코드 수 이다.

페이지 처리 시 필요한 데이터를 저장하는 클래스를 작성

PageTO

```
package com.entity;
import java.util.ArrayList;
public class PageTO {
    ArrayList<BoardDTO> list ; //목록 리스트 저장
    int curPage ; //현재 페이지 번호
    int perPage = 5; //페이지당 보여줄 레코드 수
    int totalCount; //전체 레코드 수
    public ArrayList<BoardDTO> getList() {
        return list;
    }
    public void setList(ArrayList<BoardDTO> list) {
        this.list = list;
    }
    public int getCurPage() {
        return curPage;
    }
    public void setCurPage(int curPage) {
        this.curPage = curPage;
    }
    public int getPerPage() {
        return perPage;
    }
    public void setPerPage(int perPage) {
        this.perPage = perPage;
    }
    public int getTotalCount() {
        return totalCount;
    }
    public void setTotalCount(int totalCount) {
        this.totalCount = totalCount;
    }
}
```

BoardDAO 클래스에 페이지 처리 메소드를 추가한다.

```
import com.entity.PageTO;
//중략
// 페이지 처리: 전체 레코드 갯수 구하기
public int totalCount(){
```

```

int count = 0;
Connection con = null;
PreparedStatement pstmt = null;
ResultSet rs = null;
try{
    con = dataFactory.getConnection();
    String query = "SELECT count(*) FROM board";
    pstmt = con.prepareStatement( query );
    rs = pstmt.executeQuery( );
    if( rs.next()){
        count = rs.getInt( 1 );
    }
}catch( Exception e){
    e.printStackTrace();
}finally{
    try {
        if( rs!= null) rs.close();
        if( pstmt!= null) pstmt.close();
        if( con!= null) con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
return count;
}//end totalCount

// 페이지 구현
public PageTO page( int curPage ){
    PageTO to = new PageTO();
    int totalCount = totalCount();
    ArrayList<BoardDTO> list = new ArrayList<BoardDTO>();
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try{
        con = dataFactory.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("SELECT num ,author, title, content, ");
        query.append("to_char(writeday, 'YYYY/MM/DD') writeday, ");
        query.append("readcnt , repRoot, repStep, repIndent ");

```

```

query.append("FROM board order by repRoot desc, repStep asc");
stmt = con.prepareStatement(query.toString(),
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY );
rs = stmt.executeQuery();

int perPage = to.getPerPage(); //5
int skip = (curPage - 1) * perPage;
if(skip > 0){
    rs.absolute( skip );
}
for(int i = 0 ; i < perPage && rs.next() ; i++){
    BoardDTO data = new BoardDTO();
    data.setNum( rs.getInt( "num" ) );
    data.setAuthor( rs.getString( "author" ) );
    data.setTitle( rs.getString( "title" ) );
    data.setContent( rs.getString( "content" ) );
    data.setWriteday( rs.getString( "writeday" ) );
    data.setReadcnt( rs.getInt( "readcnt" ) );
    data.setRepRoot( rs.getInt( "repRoot" ) );
    data.setRepStep( rs.getInt( "repStep" ) );
    data.setReplndent( rs.getInt( "replndent" ) );

    list.add( data );
}
//end for

to.setList( list ); // ArrayList 저장
to.setTotalCount( totalCount ); // 전체 레코드 갯수
to.setCurPage( curPage ); // 현재 페이지
}catch( Exception e){
    e.printStackTrace();
}finally{
    try {
        if( rs!= null) rs.close();
        if( stmt!= null) stmt.close();
        if( con!= null) con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

```
        return to;
    }//end page
```

페이지 처리를 구현할 클래스 작성

BoardPageCommand.java

```
package com.service;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.BoardDAO;
import com.entity.PageTO;
public class BoardPageCommand implements BoardCommand {
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) {
        String page="";
        int curPage = 1; //현재 페이지

        if( request.getParameter( "curPage" ) != null ) {
            curPage = Integer.parseInt(request.getParameter("curPage"));
        }
        BoardDAO dao = new BoardDAO();
        PageTO list = dao.page(curPage);

        int perPage = list.getPerPage();
        int totalCount = list.getTotalCount();

        int totalPage = totalCount / perPage; // 보여줄 페이지 번호개수
        if( totalCount % perPage != 0 ) totalPage++;

        for( int i = 1 ; i <= totalPage ; i++){
            if( curPage == i ){
                page+="<span id='pg'>" + i + "</span> ";
            }else{
                page+="<a href='list.do?curPage=" + i + "'>" + i + "</a>&ampnbsp";
            }
        }
        request.setAttribute("list", list.getList());
        request.setAttribute("page", list);
        request.setAttribute("pg", page);
    }
}
```

```
}
```

BoardFrontController 서블릿에 코드를 수정 추가

BoardFrontController.java

```
import com.service.BoardPageCommand;  
//중략  
/* if(com.equals("/list.do")){  
    command = new BoardListCommand();  
    command.execute(request, response);  
    nextPage = "list.jsp";  
} */  
// 페이지 처리  
if(com.equals("/list.do")){  
    command = new BoardPageCommand();  
    command.execute(request, response);  
    nextPage = "listPage.jsp";  
}
```

listPage.jsp

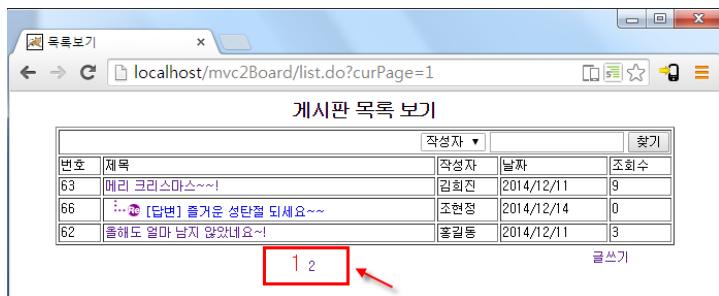
```
<%@ page language="java" contentType="text/html; charset=EUC-KR"  
pageEncoding="EUC-KR"%>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">  
<title>목록보기</title>  
<style type="text/css">  
    span#pg{color:red;font-size:22px;}  
    a{text-decoration:none;}  
</style>  
</head>  
<body>  
    <h1>게시판 목록 보기</h1>  
    <table border="1" width="600">  
        <tr>  
            <td colspan="5" align="right">  
                <form method="post" action="search.do">  
                    <select name="searchName" size="1">
```

```

        <option value="author">작성자</option>
        <option value="title">글제목</option>
    </select>
    <input type="text" name="searchValue">
    <input type="submit" value="찾기">
</form>
</td>
</tr>
<tr>
    <td>번호</td>
    <td>제목</td>
    <td>작성자</td>
    <td>날짜</td>
    <td>조회수</td>
</tr>
<c:if test="${page.totalCount<=0}">
    <tr>
        <td colspan="5" align="center">입력된 데이터가 존재하지 않습니다</td>
    </tr>
</c:if>
<c:if test="${page.totalCount>0}">
    <c:forEach items="${list}" var="dto">
        <tr>
            <td>${dto.num}</td>
            <td><c:if test="${dto.replIndent > 0}">
                
                
            </c:if>
            <a href="retrieve.do?num=${dto.num}">${dto.title}</a>
        </td>
            <td>${dto.author}</td>
            <td>${dto.writeday}</td>
            <td>${dto.readcnt}</td>
        </tr>
    </c:forEach>
</c:if>
</table>
<!-- page -->
<div id="pageView">${pg}</div>

```

```
<div id="butView"><a href="writeui.do">글쓰기</a></div>  
</body>  
</html>
```



24. Tiles를 이용한 레이아웃 템플릿 처리

1) 컴포지트 뷰 패턴(Composite View Pattern)

하나의 웹 사이트는 한 개 이상의 웹 페이지로 구성된다.

예를 들어, 까페 사이트의 경우 카페 홈 페이지와 카페 게시글 목록 페이지는 동일한 헤더, 좌측 메뉴, 푸터를 같는다.



위의 그림에서 좌측 화면과 우측 화면을 생성하는 JSP 코드는 아래 코드와 같이 제목과 내용 부분의 코드만 다르고 화면 레이아웃을 구성하는 나머지 코드는 동일할 것이다.

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
    <title>제목</title>
</head>
<body>
<table width="400" border="1" cellpadding="0" cellspacing="0">
<tr>
    <td colspan="2"><jsp:include page="/module/header.jsp" flush="false" /></td>
</tr>
<tr>
    <td valign="top" width="25%"><jsp:include page="/module/menu.jsp" flush="false" /></td>
    <td valign="top" width="75%">내용 부분 다름</td>
</tr>
<tr>
    <td colspan="2"><jsp:include page="/module/footer.jsp" flush="false" /></td>
</tr>
</table>
</body>
</html>
```

새로운 페이지를 추가할 때마다 페이지 구조와 관련된 코드를 작성해 주어야 하는데, 이 경우 각각의 페이지는 레이아웃을 위한 코드를 중복해서 갖게 된다. 이 경우 전체 페이지에 대해서 레이아웃 구조를 일부 변경해야 할 경우(메뉴 폭을 조금 늘릴 경우), 모든 페이지의 코드를 수정해 주어야 하는 불편함이

따른다.

웹 애플리케이션 개발에서 페이지 구조를 위한 코드를 매번 입력해야 하는 불편함을 줄이고 페이지의 레이아웃 구성을 위한 중복된 코드를 제거하기 위해 사용하는 일반적인 방식은 컴포지트 뷰(Composite View) 패턴을 적용하는 것이다. 컴포지트 뷰 패턴의 핵심은 레이아웃 구성 정보를 담고 있는 템플릿을 생성한다는 것이다.

웹 프로그래밍에서 템플릿은 다음과 같은 형식을 갖는다.

```
<table width="400" border="1" cellpadding="0" cellspacing="0">
<tr>
    <td colspan="2">[헤더 조각 삽입 코드]</td>
</tr>
<tr>
    <td valign="top">[메뉴 조각 삽입 코드]</td>
    <td valign="top">[내용 조각 삽입 코드]</td>
</tr>
<tr>
    <td colspan="2">[푸터 조각 삽입 코드]</td>
</tr>
</table>
```

레이아웃을 구성하는 각 영역에 알맞은 내용을 삽입해 주는 코드는 컴포지트 뷰 패턴을 구현한 라이브러리마다 다르다. 예를 들어 Tiles의 경우 템플릿 코드에서 아래과 같이 `<tiles:insertAttribute>` 커스텀 태그를 이용해서 코드 조각을 삽입할 수 있도록 한다.

```
<table width="100%" border="1" cellpadding="0" cellspacing="0">
<tr>
    <td colspan="2"><tiles:insertAttribute name="header" /></td>
</tr>
<tr>
    <td valign="top"><tiles:insertAttribute name="menu" /></td>
    <td valign="top"><tiles:insertAttribute name="body" /></td>
</tr>
<tr>
    <td colspan="2"><tiles:insertAttribute name="footer" /></td>
</tr>
</table>
```

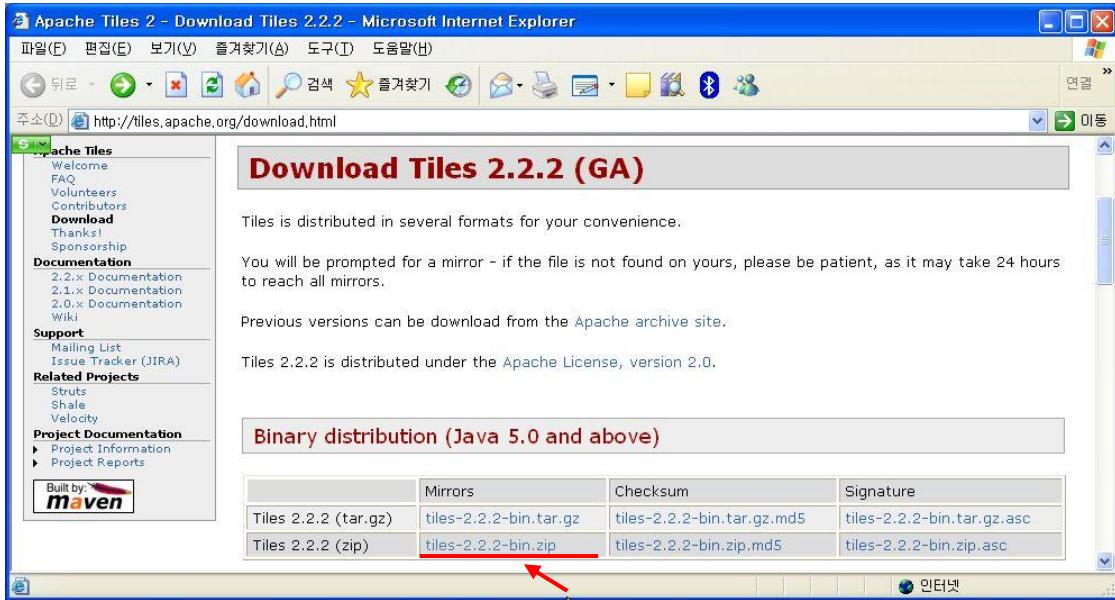
동일한 레이아웃 구성을 갖는 페이지는 위 코드와 같은 템플릿 코드를 이용해서 페이지 구성과 관련된 동일한 코드를 갖게 된다. 또한, 각 영역에 어떤 내용이 삽입될지에 대한 정보를 외부의 파일을 이용해서 설정함으로써 손쉽게 페이지 구성 요소를 변경할 수 있다.

2) Tiles 2를 이용한 컴포지트 뷰 구현

컴포지트 뷰 패턴을 구현한 라이브러리 중에서 현재 널리 사용되고 있는 라이브러리는 Tiles2로서 Tiles2를 이용해서 템플릿을 만들고 적용하는 방법을 알아 본다.

① Tiles 2 설치

Tiles2 는 <http://tiles.apache.org/download.html>에서 tiles-2.2.2-bin.zip 파일을 다운로드 받은 뒤 압축을 푼다.



추가로 jstl.jar/standard.jar 파일도 추가한다. 그리고 <http://commons.apache.org/>사이트에서 [The Commons Proper]에서 [Components] 중 Logging을 선택하여 commons-logging-1.1.1-bin.zip 파일을 다운로드 해서 commons-logging-1.1.1.jar 파일을 복사해서 WEB-INF/lib에 붙여 넣는다.
tiles-2.2.2 파일에서 아래의 jar 파일을 WEB-INF/lib 폴더에 복사한다.

tiles-2.2.2 디렉터리에 있는 jar 파일

- tiles-api-2.2.2.jar, tiles-core-2.2.2.jar, tiles-template-2.2.2.jar, tiles-jsp-2.2.2.jar, tiles-servlet-2.2.2.jar

tiles-2.2.2/lib 디렉터리에 있는 jar 파일

- commons-beanutils-1.8.0.jar
- commons-digester-1.8.1.jar
- slf4j-api-1.5.8.jar

tiles-2.2.2/lib/optional 디렉터리에 있는 jar 파일

- slf4j-jdk1.4-1.5.8.jar

② Tiles 2 시작

- 필요한 jar 파일을 WEB-INF/lib 디렉터리에 복사한다.
- web.xml에 초기화 코드 추가
- Tiles 2 설정 파일 작성

- 레이아웃 템플릿 JSP 파일 작성
- 템플릿을 사용하는 JSP 파일 작성

- web.xml에 Tiles 엔진 초기화 코드 추가

웹 애플리케이션에서 Tiles2를 사용하려면 Tiles 엔진을 초기화 해주어야 한다. Tiles 엔진을 초기화하는 첫 번

째 방법은 web.xml파일에 TilesServlet을 설정하는 것이다.

```
<web-app>
    <servlet>
        <servlet-name>tiles</servlet-name>
        <servlet-class>org.apache.tiles.web.startup.TilesServlet</servlet-class>
        <init-param>
            <param-name>
                org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG
            </param-name>
            <param-value>
                /WEB-INF/tiles-hello.xml,/WEB-INF/tiles-service.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
</web-app>
```

TilesServlet을 설정할 때 <load-on-startup>옵션을 주어 웹 애플리케이션이 초기화 될 때 TilesServlet이 실행되도록 해주어야 한다. 초기화 파라미터를 이용해서 org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG 초기화 파라미터의 값으로 Tiles 설정 파일의 목록을 입력 받으면 된다. 각 설정 파일의 경로는 콤마로 구분한다.

org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG 초기화 파라미터를 설정하지 않을 경우 기본값은 /WEB-INF/tiles.xml 이다.

주의

설정 파일 목록을 입력할 때 콤마 사이에 공백 문자가 들어가지 않도록 한다. 만약 아래와 같이 콤마 뒤에 공백 문자가 포함되어 있을 경우 해당 설정 파일을 올바르게 찾지 못하기 때문에 설정 퍼일에 포함된 Definition을 사용할 수 없게 된다.

```
<param-name>
    org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG
</param-name>
<param-value>
    /WEB-INF/tiles-hello.xml,/WEB-INF/tiles-service.xml
</param-value>
```

Tiles 엔진을 초기화하는 또 다른 설정 방법은 web.xml 파일에 TilesListener를 사용하는 것이다.

```
<web-app>
    <listener>
        <listener-class>org.apache.tiles.web.startup.TilesServlet</listener-class>
    </listener>
    <context-param>
        <param-name>
            org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG
        </param-name>
        <param-value>
            /WEB-INF/tiles-hello.xml
        </param-value>
    </context-param>
</web-app>
```

TilesListener를 사용하는 경우에는 웹 애플리케이션 콘텍스트 초기화 파라미터를 이용해서 설정 파일의 경로를 입력을 입력한다.

WEB-INF\web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
    <servlet>
        <servlet-name>tiles</servlet-name>
        <servlet-class>org.apache.tiles.web.startup.TilesServlet</servlet-class>
        <init-param>
            <param-name>
                org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG
            </param-name>
            <param-value>
                /WEB-INF/tiles-hello.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
</web-app>
```

- Tiles 2 설정 파일 작성

Tiles 설정 파일은 어떤 JSP를 템플릿으로 사용하고 템플릿의 각 영역을 어떤 내용으로 채울지에 대한 정보를 설정한다.

WEB-INF\.tiles-hello.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"
    "http://tiles.apache.org/dtds/tiles-config_2_1.dtd">
<tiles-definitions>
    <definition name="hello" template="/template/layout.jsp">
        <put-attribute name="title" value="헬로우 월드" />
        <put-attribute name="header" value="/template/header.jsp" />
        <put-attribute name="menu" value="/template/menu.jsp" />
        <put-attribute name="body" value="/hello_body.jsp" />
        <put-attribute name="footer" value="/template/footer.jsp" />
    </definition>

    <definition name="hello2" template="/template/layout.jsp">
        <put-attribute name="title" value="헬로우 월드2" />
        <put-attribute name="header" value="/template/header2.jsp" />
        <put-attribute name="menu" value="/template/menu.jsp" />
        <put-attribute name="body" value="/hello_body.jsp" />
        <put-attribute name="footer" value="/template/footer2.jsp" />
    </definition>
</tiles-definitions>
```

tiles-hello.xml은 Tiles 설정 파일로서 두 개의 Definition을 설정하고 있다. Definition은 템플릿의 각 내용을 어떻게 채울지에 대한 정보를 정의한다.

- 레이아웃 템플릿 JSP 파일 작성

"hello" Definition은 템플릿 파일로 layout.jsp를 사용하고 있는데, 템플릿 파일은 각 어트리뷰트의 값을 어느 위치에 어떻게 삽입할지를 결정한다.

프로젝트명\template\layout.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
    <title><tiles:getAsString name="title" /></title>
```

```

</head>
<body>
<table width="100%" border="1" cellpadding="0" cellspacing="0">
<tr>
    <td colspan="2">
        <b><tiles:insertAttribute name="header" /></b>
    </td>
</tr>
<tr>
    <td valign="top"><b><tiles:insertAttribute name="menu" /></b></td>
    <td valign="top"><b><tiles:insertAttribute name="body" /></b></td>
</tr>
<tr>
    <td colspan="2">
        <b><tiles:insertAttribute name="footer" /></b>
    </td>
</tr>
</table>
</body>
</html>

```

Tiles의 템플릿 파일은 동일한 페이지 구성을 갖는 화면을 위한 레이아웃 관련 코드를 포함하고 있으며, Tiles가 제공하는 태그를 이용해서 각 영역에 어떤 값을 삽입할지를 결정한다.

layout.jsp는 <table>태그를 이용해서 레이아웃 구조를 잡고 있으며, <tiles:getAsString>태그와 <tiles:insertAttribute>태그를 이용해서 각 영역에 삽입될 값을 지정한다.

<tiles:getAsString>태그는 해당 어트리뷰트의 값을 현재 위치에 문자열을 삽입한다.

name="title" 어트리뷰트의 값을 문자열로 삽입한다. tiles-hello.xml 설정 파일에서 "hello2" Definition을 사용할 경우 <tiles:getAsString name="title" /> 코드는 현재 위치에 "헬로우 월드2" 문자열을 삽입한다.

```

<definition name="hello2" template="/template/layout.jsp">
    <put-attribute name="title" value="헬로우 월드2" />
    <put-attribute name="header" value="/template/header2.jsp" />
    <put-attribute name="menu" value="/template/menu.jsp" />
    <put-attribute name="body" value="/hello_body.jsp" />
    <put-attribute name="footer" value="/template/footer2.jsp" />
</definition>

```

<tiles:insertAttribute>태그는 어트리뷰트의 값에 해당하는 JSP를 <jsp:include>와 동일한 방식으로 현재 위치에 삽입한다. <tiles:insertAttribute name="header" />태그는 Definition에서 정의한 "header" 어트리뷰트의 값인 "/template/header2.jsp"의 실행 결과를 현재 위치에 삽입한다.

- 템플릿의 각 구성 요소에 삽입될 JSP 파일 작성

앞에서 Tiles 설정 파일을 보면 <put-attribute>태그를 이용해서 각 영역에 삽입될 JSP파일을 값으로 설정했다.

Tiles 설정 파일에서는 header.jsp 파일을 비롯한 header2.jsp, footer.jsp, footer2.jsp, menu.jsp를 각각 "header",

"footer", "menu" 어트리뷰트 값으로 설정한다.

프로젝트명/template/header.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
첫 번째 헤더입니다.
```

- Definition을 사용하는 JSP 파일 작성

템플릿 파일과 템플릿의 구성 요소에 삽입될 JSP 파일을 작성했으므로 남은 작업은 실제로 Definition을 사용해서 필요한 화면을 삽입하는 것이다. Definition을 사용하려면 다음과 같이 Tiles가 제공하는 커스텀 태그 중

<tiles:insertDefinition>태그를 사용한다.

```
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>
...
<tiles:insertDefinition name="hello" />
```

<tiles:insertDefinition>태그는 name속성을 이용해서 사용될 Definition의 이름을 지정한다.

위의 코드에서 "hello" Definition을 사용하는데 "hello" Definition은 다음과 같다.

```
<definition name="hello" template="/template/layout.jsp">
    <put-attribute name="title" value="헬로우 월드" />
    <put-attribute name="header" value="/template/header.jsp" />
    <put-attribute name="menu" value="/template/menu.jsp" />
    <put-attribute name="body" value="/hello_body.jsp" />
    <put-attribute name="footer" value="/template/footer.jsp" />
</definition>
```

<tiles:insertDefinition name="hello" /> 코드는 현재 위치에 layout.jsp를 삽입하고 layout.jsp의 각 <tiles:insertAttribute>위치에는 header.jsp, menu.jsp, hello_body.jsp, footer.jsp가 삽입된다.

프로젝트명/hello.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>
<%
    request.setAttribute("greeting", "안녕하세요");
%>
<tiles:insertDefinition name="hello" />
```

"hello" Definition은 "body" 어트리뷰트 위치에 hello_body.jsp를 삽입하므로 hello_body.jsp를 작성해야 hello.jsp가 올바르게 실행된다. hello_body.jsp는 hello.jsp에서 전달한 값을 사용해서 알맞은 화면을 출력한다.

프로젝트명/hello_body.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<strong>${greeting}</strong> 고객님.  

<hr/>
사이트에 오신 걸 환영합니다.
```

필요한 파일들을 모두 작성했으므로 hello.jsp를 실행한다.

<http://localhost:8080/프로젝트명/hello.jsp>

첫 번째 헤더입니다.	
메뉴	안녕하세요 고객님.
사이트에 오신 걸 환영합니다.	
푸터	

hello.jsp에서 <tiles:insertDefinition name="hello" />코드에서 "hello" 대신에 "hello2"를 사용하도록 변경하여 실행해 본다.

두 번째 헤더입니다.	
메뉴	안녕하세요 고객님.
사이트에 오신 걸 환영합니다.	
푸터2	

25. 리스너와 필터

1. 웹 애플리케이션 초기화 매개변수 관리

1) 초기화 매개변수의 개요

웹 애플리케이션 초기화 매개변수는 웹 애플리케이션이 컨테이너에 의해 구동될 때 로딩되는 정보로서, 웹 애플리케이션 전반에 걸쳐 공통적으로 참조하는 값을 설정하는 용도로 사용한다.

일반적으로 프로그램을 동작시킬 때 필요한 기본 정보는 소스코드 내에 하드코딩하지 않고 별도의 환경 설정 파일을 통해 제공한다.

예를 들어, 데이터베이스 접속 주소나 환경설정 파일의 위치 같은 정보를 프로그램 내에 둘 경우, 해당 내용을 필요로 하는 소스코드마다 동일한 내용을 넣어야 한다.

따라서 파일이 분산될 경우, 변경에 따른 관리가 어렵고, 변경이 발생했을 경우 해당 클래스들을 모두 수정한 후 다시 컴파일해야 하는 등의 문제가 발생한다.

초기화 매개변수는 웹애플리케이션 전체 혹은 특정 서블릿에 대한 값을 설정 할 수 있다.

서블릿 초기화 매개변수는 ServletConfig를 통해 접근할 수 있다.

웹 애플리케이션 초기화 매개변수는 ServletContext를 통해 접근 할 수 있다.

초기 값의 설정은 web.xml에서 한다.

ServletConfig와 ServletContext의 비교

구분	적용 범위	web.xml 설정
ServletConfig	해당 서블릿에서만 사용할 수 있다.	<pre><servlet> <init-param> <param-name> </param-name> <param-value> </param-value> </init-param> </servlet></pre>
ServletContext	동일 웹 애플리케이션 내 모든 서블릿(혹은 JSP)에서 사용할 수 있다.	<pre><web-app> <context-param> <param-name> </param-name> <param-value> </param-value> </context-param> </web-app></pre>

① ServletConfig를 사용하는 경우

서블릿 단위로 설정하기 때문에 해당 서블릿에서만 참조할 수 있다. 서블릿 동작에 필요한 정보를 제공하기 위한 목적으로 사용한다.

서블릿 코드 내에서는 다음과 같이 값을 참조 할 수 있다.

```
getInitParameter("param-name에서의 설정 이름")
```

서블릿 3.0 이상에서는 web.xml 이외에도 서블릿 클래스에 애너테이션 기반으로 초기값을 설정 할 수 있다.

```
@WebServlet(name="test",
    urlPatterns={"/test"},
    initParams={@InitParam(name="n1", value="v1"), @InitParam(name="n2", value="v2")})
```

② ServletContext를 사용하는 경우

웹 애플리케이션 내 모든 JSP 혹은 서블릿에서의 참조가 가능하다.

서블릿 코드 내에서는 다음과 같이 값을 참조할 수 있다.

```
getServletContext().getInitParameter("param-name에서의 설정 이름")
```

웹 애플리케이션 초기화 매개변수는 web.xml에서 <context-param> 태그를 이용해 설정할 수 있다.

```
<web-app>
    <context-param>
        <param-name> </param-name>
        <param-value> </param-value>
    </context-param>
</web-app>
```

1.1 초기화 매개변수 관리

1) 실습 개요

이 실습은 web.xml 파일에 초기화 매개변수를 설정해두고 서블릿 및 JSP에서 이를 참조하는 방법을 살펴보는 것이다.

실습은 ServletConfig 및 ServletContext 초기화 매개변수를 설정하고, 서블릿과 JSP에서 해당 속성을 참조해 출력하도록 구성되어 있다

프로그램 소스 목록

경로	파일명	설명
com.test	PropertyServlet.java	서블릿 초기화 매개변수 실습할 서블릿 코드
WebContent/InitParam	property.jsp	서블릿 초기화 매개변수를 확인하기 위한 JSP 파일. 코드 수정을 통해 ServletContext 초기화 매개변수를 확인하는 데도 사용할 수 있다.

PropertyServlet.java

```
package com.test;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.*;
import javax.servlet.http.HttpServlet;
```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//서블릿 선언 애너테이션
@WebServlet(
    // 서블릿 호출 URL 지정
    urlPatterns = { "/PropertyServlet" },
    // 서블릿 초기화 파라미터 지정
    initParams = {
        @WebInitParam(name = "name1", value = "user1"),
        @WebInitParam(name = "name2", value = "user2")
    }
)
public class PropertyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public PropertyServlet() {
        super();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html; charset=euc-kr");
        out.println("<HTML><BODY><center>");
        out.println("<H2>ch14:PropertyServlet</H2>");
        out.println("<HR>");
        // 서블릿 초기화 파라미터로 전달된 값 출력
        out.println("name1 : "+getInitParameter("name1")+"<BR>");
        out.println("name2 : "+getInitParameter("name2"));
        out.println("<HR>");
        // 웹 어플리케이션 초기화 파라미터로 전달된 값 출력
        out.println("name3 : "+getServletContext().getInitParameter("name3"));
        out.println("</center></BODY></HTML>");
    }
}

```

애너테이션을 이용해 서블릿 초기화 매개변수를 설정한다.

애너테이션은 프로그램 소스에 들어가지만 클래스 파일 외부에서 접근이 가능하다.

서블릿 접근을 위한 urlPatterns와 초기화 매개변수인 initParams를 설정하다.

설정값은 name, value 쌍으로 구성된다.

```

@WebServlet(
    urlPatterns = { "/PropertyServlet" },
    initParams = { @WebInitParam(name = "name1", value = "user1"),
                   @WebInitParam(name = "name2", value = "user2")})

```

초기화 매개변수는 getInitParameter 메서드를 통해 가져올 수 있다.

```

out.println("name1 :" + getInitParameter("name1") + "<BR>");
out.println("name2 :" + getInitParameter("name2"));

```

① 서블릿 3.0에서 초기화 매개변수 설정하기

- ServletContext 초기화 매개변수 설정

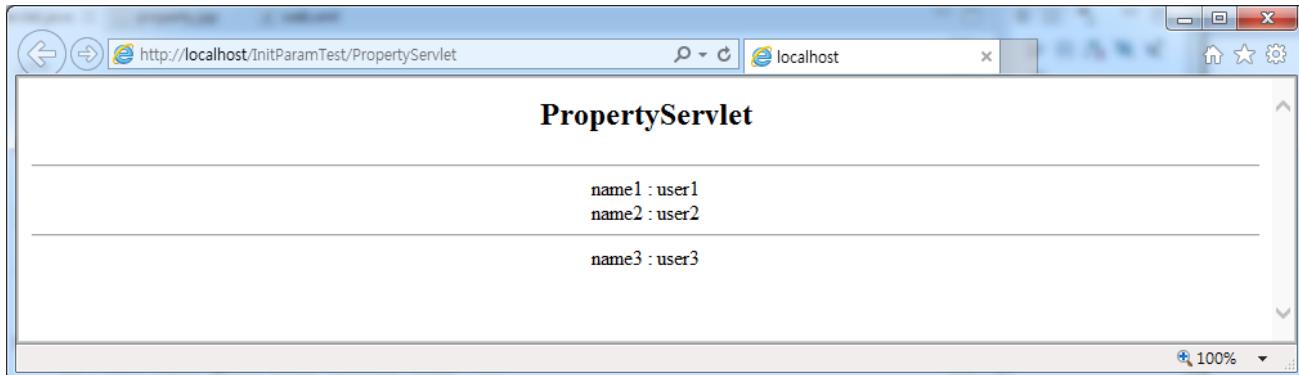
서블릿 3.0에서의 ServletContext 초기화 매개변수 설정

서블릿 3.0에서는 web.xml 파일에 서블릿 초기화 매개변수를 설정한다.

```

<context-param>
    <param-name>workspace</param-name>
    <param-value> C:\jsp2_exam</param-value>
</context-param>
<context-param>
    <param-name>name3</param-name>
    <param-value>user3</param-value>
</context-param>

```



property.jsp

```

<%@ 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>property.jsp</title>
</head>

```

```

<body>
    <div align="center">
        <H2>property.jsp</H2>
        <HR>
        ServletContext 설정값 출력
        <HR>
        name3 :
        <%=getServletContext().getInitParameter("name3")%> <BR>
        workspace :
        <%=getServletContext().getInitParameter("workspace")%>
    </div>
</body>
</html>

```



2. 리스너

2.1 리스너란

리스너는 컨테이너에서 발생하는 특정 이벤트 상황을 모니터링하다가 실행되는 특수한 형태의 서블릿이다. 웹 애플리케이션에 초기 데이터를 공급하거나 특정 상황에 따라 자동으로 동작하는 프로그램을 구현할 때 사용한다. 리스너는 일반적인 형태의 서블릿이 아니라, 특정 이벤트에 따라 동작하는 인터페이스를 구현한 클래스라고 이해하면 쉽다. 서블릿과 마찬가지로 애너테이션 기반으로 코드로 작성할 수 있다.

리스너 활용의 대표적인 유형은 다음과 같다.

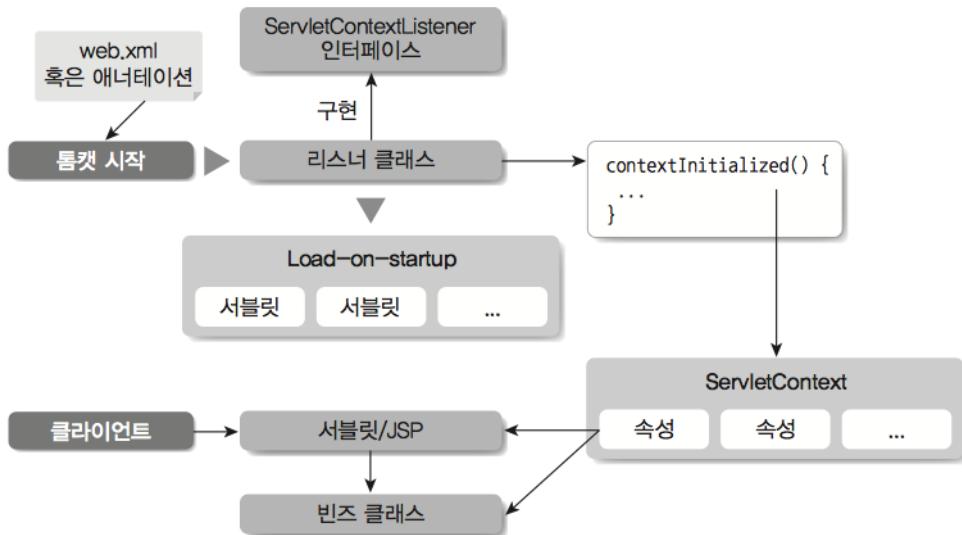
- ① 초기화 매개변수와 연동 : 톰캣이 시작될 때 초기화 매개변수를 읽어 그에 따라 특정 객체를 초기화한 후 서블릿이나 JSP에 제공하는 형태다.
- ② 예제 프로그램 등을 배포할 때 샘플 데이터 제공 : 프로그램을 실행할 때 데이터베이스가 필요 한 경우 데이터베이스의 연결, 테이블 생성 및 샘플 데이터 로딩 등의 작업을 사전에 수행해서 추가적인 작업 없이 프로그램을 실행할 수 있게 한다.
- ③ 복잡한 환경 설정 제공 : 프로그램 실행에 필요한 여러 정보를 파일로부터 읽어와 JSP 및 서블릿 등에 제공한다.
- ④ 특정 이벤트에 동작하는 기능 구현 : 웹 애플리케이션을 실행할 때 함께 동작해야 하는 외부

프로그램이나 서비스의 동작 유무를 확인하고 자동으로 실행하게 한다.

2.2 리스너 구조와 종류

리스너 동작 구조 : ServletContextListener 기준

리스너 개발 절차 및 동작 구조



톰캣 시작시 web.xml 혹은 애너테이션 정보를 참조해 ServletContextListener 인터페이스를 구현한 리스너 클래스를 시작한다. contextInitialized() 메서드가 호출되고 여기서 애플리케이션에서 공유할 객체들을 초기화 할 수 있다.

리스너는 크게 ServletContext, Session, Request 의 상태나 속성의 변화를 모니터링하고 동작한다.

대표적인 리스너 종류는 다음과 같다.

유형	설명	리스너 인터페이스
ServletContext 생명주기 변화	대표적인 리스너 이벤트로, ServletContext의 생성과 소멸 시점에 동작한다. 보통 톰캣의 시작과 종료와 일치한다.	javax.servlet.ServletContextListener
ServletContext 속성 변화	ServletContext 즉, JSP 내장객체에 있는 application scope의 속성이 추가되거나 변경되는 상황에 동작한다.	javax.servlet. ServletContextAttributeListener
Session 생명주기 변화	session의 생성과 소멸 등 변화에 따라 동작한다.	javax.servlet.http.HttpSessionListener
Session 속성 변화	JSP 내장객체에 있는 session scope의 속성이 추가되거나 변경되는 상황에 동작한다.	javax.servlet.http. HttpSessionAttributeListener
Request 생명주기 변화	HttpServletRequest 즉, request 내장객체의 생성과 소멸 등 변화에 따라 동작한다.	javax.servlet.HttpServletRequestListener
Request 속성 변화	HttpServletRequest 객체의 속성이 추가되거나 변경되는 상황에 동작한다.	javax.servlet. HttpServletRequestAttributeListener

1) 리스너 개발 : ServletContextListener 구현

ServletContextListener를 구현하고 톰캣이 시작될 때 Book 클래스의 인스턴스를 만든 뒤 ServletContext(application 내장객체) 범위에 속성으로 저장하고, 별도의 JSP에서 표현 언어를 이용해서 Book 클래스 객체를 참조하는 프로그램이다.

실습을 통한 리스너 개발과정과 동작원리를 이해한다.

소스 디렉터리 구조

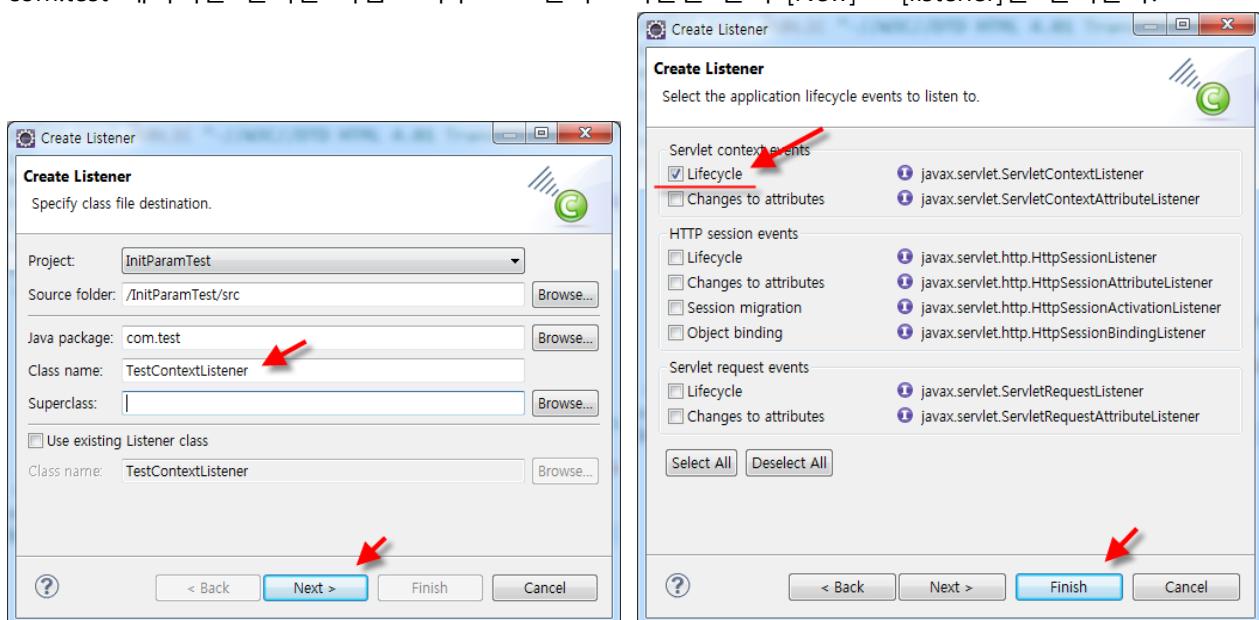
경로	파일이름
com.test	TestContextListener.java
WebContent/Listener	ListenerTest.jsp

2) 리스너 클래스 구현

리스너 클래스 구현은 이클립스에서 제공하는 템플릿을 통해 쉽게 생성 할 수 있다.

단계별로 필요한 정보를 입력하거나 선택해 서블릿 클래스를 생성한다.

com.test 패키지를 선택한 다음 <마우스 오른쪽> 버튼을 눌러 [New] → [listener]를 선택한다.



TestContextListener.java

```
package com.test;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class TestContextListener implements ServletContextListener {
    public TestContextListener() {
    }
    // 리스너 실행 메서드
}
```

```

public void contextInitialized(ServletContextEvent arg0) {
    ServletContext ctx = arg0.getServletContext();
    // Book 객체를 만들어 application scope 에 저장
    Book mybook = new Book("JSP 프로그래밍", "홍길동", 32000, "한국출판사");
    ctx.setAttribute("book", mybook);
    System.out.println("TestContextListener 시작됨..");
}

public void contextDestroyed(ServletContextEvent arg0) {
}

}

```

자동 생성된 코드를 기반으로 필요한 메서드의 내용을 구현하면 된다.

여기서는 웹 애플리케이션이 시작될 때 호출되는 contextInitialized() 메서드를 구현한다.

Book 객체를 생성하고 ServletContext 즉, JSP 관점에서는 application scope에 속성으로 저장한다.

```
public void contextInitialized(ServletContextEvent arg0) { }
```

Book.java

```

package com.test;
public class Book {
    // 멤버변수 선언
    private String title;
    private String author;
    private int price;
    private String publisher;
    // 기본 생성자, 파라미터로 데이터 초기화
    public Book(String title, String author, int price, String publisher) {
        this.title = title;
        this.author = author;
        this.price = price;
        this.publisher = publisher;
    }
    public String getAuthor() {
        return author;
    }
    public int getPrice() {
        return price;
    }
    public String getPublisher() {
        return publisher;
    }
    public String getTitle() {

```

```

        return title;
    }
}

```

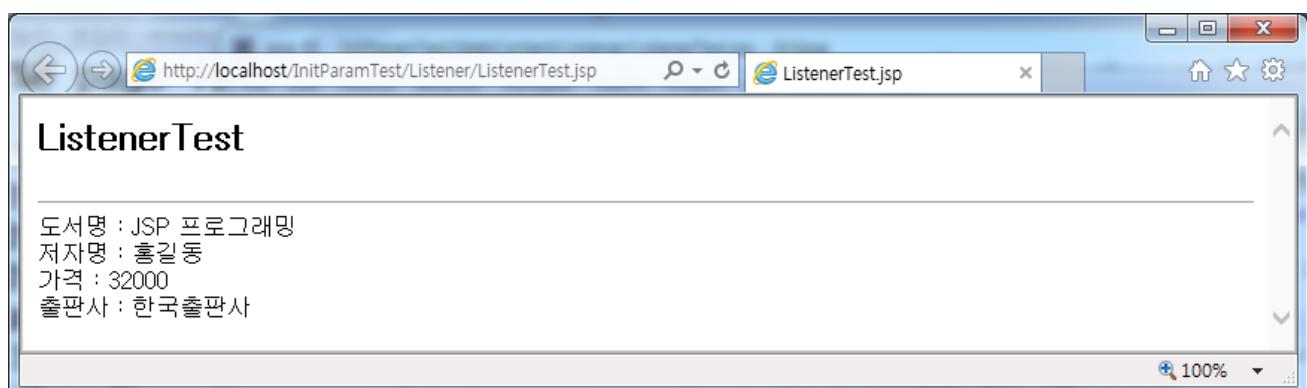
ListenerTest.jsp

```

<%@ 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>ListenerTest.jsp</title>
</head>
<body>
    <div align="center">
        <H2>ListenerTest</H2>
        <HR>
        도서명 : ${book.title} <BR> 저자명 : ${book.author} <BR>
        가격 : ${book.price} <BR> 출판사 : ${book.publisher} <BR>
    </div>
</body>
</html>

```

TestContextListener에서 application scope에 Book 객체를 넣어두었기 때문에 JSP에서는 별도의 객체 참조를 위한 선언 없이 표현언어를 사용해 출력할 수 있다.



3. 필터

3.1 필터란

필터란 특정 요청에 대해서만 동작하는 특수한 형태의 웹 프로그램을 말한다. 여러 개가 정해진 순서에 따라 배치될 수 있으며 사용자 요청 처리 이전에 먼저 실행된다.

리스너와 마찬가지로 단순히 기능만 구현하는 웹 프로그램의 경우에는 필터를 꼭 만들지 않아도 되지만, 애플리케이션 설계 관점에서 좀 더 유연하고 효과적인 애플리케이션 구현 및 운영이 필요하다면 필터에 대해 잘 알아둘 필요가 있다.

필터는 여러 분야에 활용할 수 있지만 널리 활용 되는 유형은 다음과 같다.

인증 (Authentication)

로깅 / 감사 (Logging and Auditing)

이미지 변환 (Image Conversion), 데이터 압축(Data Compression)

국제화 (Localization)

XML 변환(XSL/T Transformations of XML Content)

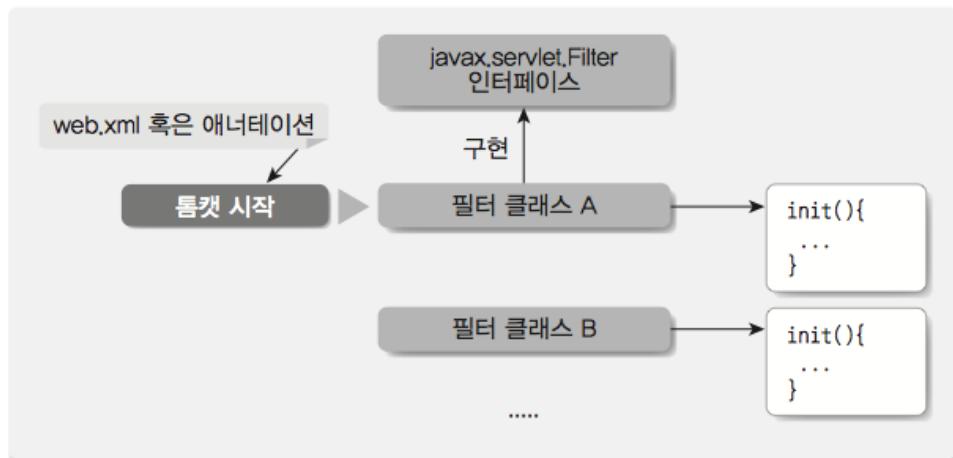
3.2 필터의 구조와 동작 과정

필터의 구조는 리스너와 유사하다. 다만 리스너는 특정 이벤트에 따라 실행되는 것이고 필터의 경우에는 사용자가 요청하는 리소스 패턴을 지정해 동작한다는 점이 차이가 있다.

필터는 여러 개가 존재할 수 있으며 각각의 필터는 init() 메서드를 통해 초기화 작업을 수행할 수 있다.

필터의 실행은 톰캣 시작시 web.xml 혹은 애너테이션 정보를 참조해 Filter 인터페이스를 구현한 클래스가 실행되는 구조이다.

필터의 구조

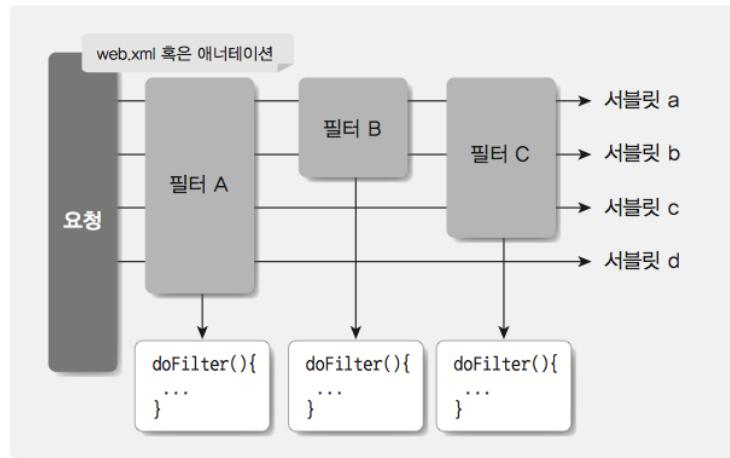


실제 필터의 기능 동작은 doFilter() 메서드에서 이루어진다.

만일 여러개의 필터가 있다면 하나의 필터가 수행된 다음 필터 체인에 의해 다음 필터가 실행된다.

다음은 여러 필터가 서로 다른 요청에 반응해 동작하는 과정이다.

필터의 동작 과정



3.3 필터 개발 : 한글 처리 필터 구현

HTML 폼을 통해 submit되는 항목의 한글이 깨지는 문제를 해결하기 위해 필터를 개발한다.

캐릭터셋 변환 필터는 스프링 프레임워크에는 기본적으로 포함되어 있을 정도로 웹 애플리케이션 구현에서 꼭 필요한 기능이다.

지금까지의 예제에서는 HTML form 데이터의 한글 처리를 위해 jsp에서 request.setCharacterEncoding("UTF-8") 또는 "EUC-KR"을 사용했지만 데이터 처리가 필요한 모든 jsp에서 해당 코드를 넣어줘야 하고 캐릭터셋 변경 시 유연하게 대응하기 어려운 문제가 있었다.

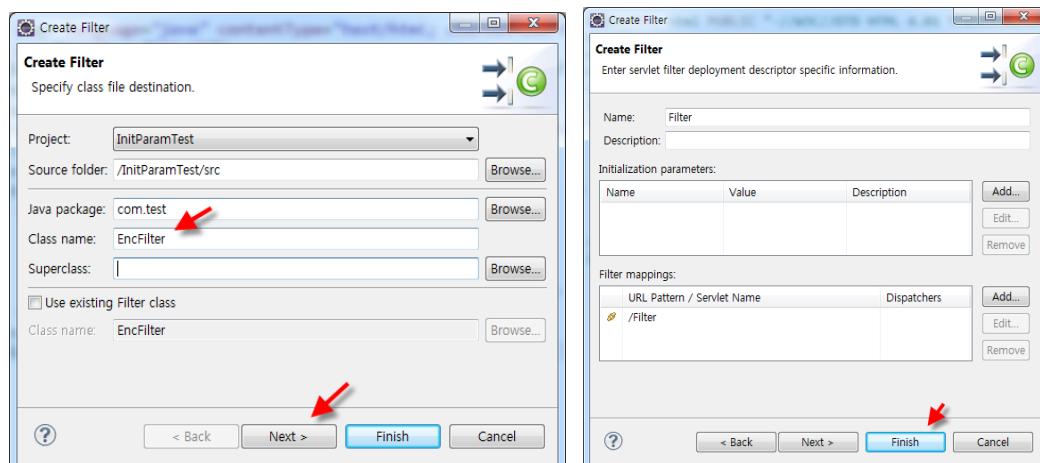
예제 프로그램 구조

경로	파일명	설명
com.test	EncFilter.java	캐릭터셋 변환을 수행할 필터 클래스
WebContent/Filter	EncForm.html	폼을 통해 입력된 한글이 제대로 처리되는지 확인하기 위한 파일
	result.jsp	

1) Encfilter 클래스 구현

com.test 패키지를 선택한 다음 <마우스 오른쪽> 버튼을 눌러 [New] → [filter]를 선택한다.

Class name에 EncFilter라고 입력한 다음 <Next> 버튼을 누른다. 다음 필터를 매핑할 URL을 "*jsp"로 수정하고 <Finish> 버튼을 누른다.



EncFilter.java

```
package com.test;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

@WebFilter("/Filter")
public class EncFilter implements Filter {
    private String encoding;

    public EncFilter() {
    }

    public void destroy() {
    }

    // 필터 처리 메서드
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
    IOException, ServletException {
        // 캐릭터 인코딩 설정이 없는 경우에만 캐릭터 인코딩 적용
        if(request.getCharacterEncoding() == null) {
            request.setCharacterEncoding(encoding);
            chain.doFilter(request, response);
        }
    }

    // 필터 초기화시 초기화 파라미터에서 인코딩 기본값 가져옴
    public void init(FilterConfig fConfig) throws ServletException {
        this.encoding = fConfig.getServletContext().getInitParameter("encoding");
    }
}
```

필터 역시 리스트와 마찬가지로 이클립스의 템플릿을 이용해 비교적 쉽게 기본 코드를 생성할 수 있다.
자동으로 생성된 코드에 init(), doFilter() 등의 메서드를 구현한다.

Init() 메서드에서는 초기화 매개변수를 읽어와 인코딩 캐릭터를 설정한다.(web.xml 별도 설정 필요)

```
<context-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
</context-param>
```

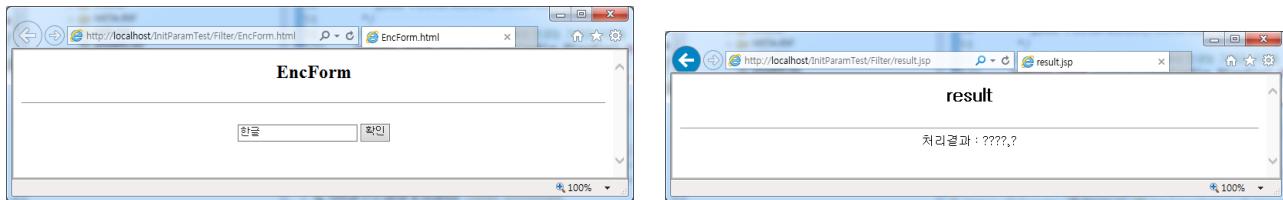
doFilter()에서는 캐릭터 인코딩 설정이 되어 있지 않은 경우 인코딩을 적용하도록 구현하였다.
필터 처리 후에는 chain.doFilter() 메서드를 통해 필터를 실행한다.

EncForm.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>EncForm.html</title>
</head>
<body>
    <div align="center">
        <H2>EncForm</H2>
        <HR>
        <form method=post action=result.jsp>
            <input type="text" name="title"> <input type="submit" value="확인">
        </form>
    </div>
</body>
</html>
```

result.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>result.jsp</title>
</head>
<body>
    <div align="center">
        <H2>result</H2>
        <HR>
        처리결과 : ${param.title}
    </div>
</body>
</html>
```



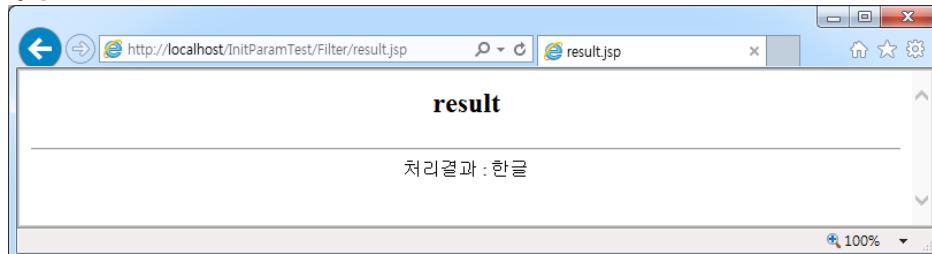
2) 필터 매핑 설정 및 실행

필터는 url 매핑에 따라 동작한다. 즉 사용자가 서버의 어떤 자원을 요청하는지 패턴을 정해 놓고 해당 형태의 요청이 들어왔을 때 해당 필터가 동작하는 구조이다.

필터 매핑은 필터 애너테이션 설정에서 변경할 수 있다.

여기서는 모든 jsp에 동작하도록 다음과 같이 애너테이션을 수정한다.

`@WebFilter("*.jsp")`



3.4 리스트 및 필터 : 애플리케이션 설정 관리 구현

이번에는 실제 프로젝트에서 더욱 유용하게 사용할 수 있는 고급 응용 예제를 구현해 볼 것이다.

앞서 초기화 매개변수를 통해 변경 가능한 정보들을 애플리케이션에 제공하는 예를 살펴보았는데, 실제 프로젝트에서는 더욱 많은 정보를 설정하고 참조한다.

이 경우에는 단순히 초기화 매개변수 이상의 기능이 요구된다.

프로그램 소스 목록

경로	파일명	경로	파일명
com.test	PropertyListener.java	WebContent/admin	PropAdTest.jsp
com.test	AdminFilter.java	WebContent/	PropTest1.jsp

- ① Property.java : ServletContextListener로 톰캣을 시작할 때 web.xml에서 초기화 매개변수를 읽어들이고 해당 경로의 매개변수 파일로부터 자바 매개변수 객체를 로딩해 application scope에 저장한다.
- ② AdminFilter.java : 특정 디렉터리의 jsp 요청이 있을 경우 동작하는 필터 클래스로서, 해당 요청에 대해서만 추가적인 정보를 제공한다.
- ③ PropTest : 각각의 동작을 테스트하는 jsp 파일이다.

1) 프로퍼티 파일 생성 및 초기화 매개변수 등록

자바 프로퍼티 파일 구조로 만들어진 설정 파일이다. key-value 쌍으로 원하는 정보를 등록 한다. 일반 텍스트 파일이므로 메모장 등을 이용해 작성 한다.

my.conf

```
version=1.0
url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
user=scott
passwd=tiger
```

한글은 처리가 안되므로 반드시 영문값만 사용하도록 한다. 주석은 #으로 처리한다.

만들어진 프로퍼티 파일은 컴퓨터에 적당한 디렉토리에 저장한다.

파일의 위치는 web.xml에 웹 애플리케이션 초기화 매개변수로 등록한다.

web.xml 추가

```
<context-param>
    <param-name>propfile</param-name>
    <param-value>c:/temp/my.conf</param-value>
</context-param>
```

2) PropertyListener 작성

프로퍼티 파일 정보를 읽어와 application scope에 저장하기 위한 리스너 클래스이다.

com.test 패키지를 선택한 다음 <마우스 오른쪽> 버튼을 눌러 [New] → [listener]를 선택한다.

PropertyListener.java

```
package com.test;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class PropertyListener implements ServletContextListener {
    public PropertyListener() {
    }
    // 리스너 실행 메서드
    public void contextInitialized(ServletContextEvent arg0) {
        ServletContext ctx = arg0.getServletContext();
        // 초기화 파라미터로 경로 정보를 가져옴
        String file = ctx.getInitParameter("propfile");
```

```

// 지정된 경로의 파일로 프로퍼티 객체를 생성
Properties p = new Properties();
try {
    p.load(new FileInputStream(file));
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
// application scope 에 prop 라는 이름으로 프로퍼티 객체 저장
ctx.setAttribute("prop", p);
}

public void contextDestroyed(ServletContextEvent arg0) {
}

}

```

3) PropTest.jsp 작성 및 실행

my.conf에 저장된 정보를 web.xml의 초기화 매개변수를 통해 리스너에서 저장한 프로퍼티 객체를 통해 표현어로 출력한다.

application scope에 저장된 객체이므로 별도의 선언 없이 표현언어에서 접근이 가능하다.

Property 객체의 get 메서드를 사용해 필요한 정보를 출력하면 된다.

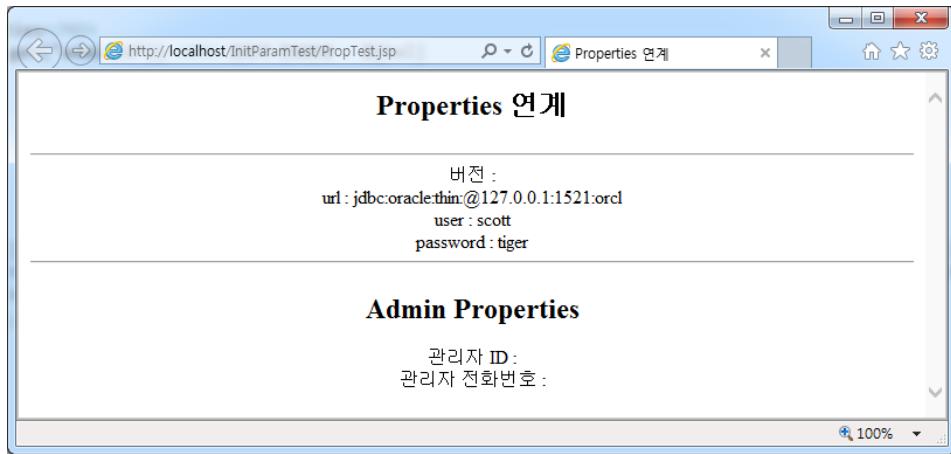
PropTest.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Properties 연계</title>
</head>
<body>
<div align="center">
    <H2>Properties 연계</H2>
    <HR>
    버전 : ${prop.get('version')} <BR> url : ${prop.get('url')} <BR>
    user : ${prop.get('user')} <BR> password : ${prop.get('passwd')}
    <HR>
    <H2>Admin Properties</H2>
    관리자 ID : ${prop.get("adminId")} <BR> 관리자 전화번호 : ${tel}
</div>
</body>

```

```
</html>
```



4) AdminFilter 작성 및 실행

my.conf에 설정된 정보 이외의 추가적인 정보를 제공하는 필터 클래스이다.

/admin 폴더 아래에 있는 자원에 접속하는 경우에만 제공한다. 즉 /admin 이외의 폴더에서는 필터가 실행되지 않으므로 추가적인 정보가 출력되지 않는다.

com.test 패키지를 선택한 다음 <마우스 오른쪽> 버튼을 눌러 [New] → [filter]를 선택한다.

AdminFilter.java

```
package com.test;

import java.io.IOException;
import java.util.Properties;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

@WebFilter("/admin/*")
public class AdminFilter implements Filter {
    Properties p;

    public AdminFilter() { }

    public void destroy() { }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        // 초기화 파라미터에서 prop로 지정된 프로퍼티 객체에 adminId로 "SuperUser" 추가
        p = (Properties) request.getServletContext().getAttribute("prop");
        p.put("adminId", "SuperUser");
    }
}
```

```

// request 에 tel 정보 추가
request.setAttribute("tel", "010-1234-1234");
// 필터처리 종료, 다음 필터 실행
chain.doFilter(request, response);
}

public void init(FilterConfig fConfig) throws ServletException { }

}

```

admin/PropAdTest.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Properties 연계</title>
</head>
<body>
<div align="center">
    <H2>Properties 연계 </H2>
    <HR>
    버전 : ${prop.get('version')} <BR> url : ${prop.get('url')} <BR>
    user : ${prop.get('user')} <BR> password : ${prop.get('passwd')}
    <HR>
    <H2>Admin Properties</H2>
    관리자 ID : ${prop.get("adminId")} <BR> 관리자 전화번호 : ${tel}
</div>
</body>
</html>

```

