Hanbit eBook

Realtime 18

JSP 바이블

STEP 01

JSP 시작과 개발환경 구축

조효은 지음

한빛미디어 Hanbit Media, Inc.

JSP 바이블

STEP 01: JSP 시작과 개발환경 구축

지은이_ **조효은**

알고리즘/최적화를 이용한 게임 엔진, 증권/주식 분석 엔진 , SOA 서비스 구현 경험을 바탕으로 프로그램 언어, 아키텍처, CBD, SOA, 웹 프로그래밍, 안드로이드 등 실무 위주의 강의를 진행하고 있다.

(현) 프리랜서
(전) 한경닷컴 수석 강사/차장
(전) 한국정보기술연구원 OOP/CBD 선임 연구원
(전) ORACLE SOA 강사
(전) 썬마이크로 시스템즈 공인 강사

JSP 바이블 STEP 01: JSP 시작과 개발환경 구축

초판발행 2013년 2월 28일

지은이 조효은 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-601-2 15560 / **비매품**

책임편집 배용석 / **기획·편집** 스마트미디어팀
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
마케팅 박상용, 박주훈, 정민하

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanb.co.kr / **이메일** ask@hanb.co.kr

대상 독자 및 사용 툴

초급 | 초중급 | 중급 | 중고급 | 고급

- JDK 7.X : http://www.oracle.com/technetwork/java/index.html
- Apache Tomcat : http://tomcat.apache.org/
- 이클립스 : http://www.eclipse.org

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook 입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내시는 선배, 전문가, 고수분에게는 보다 쉽게 집필하실 기회가 되리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 준비 중이며, 조만간 선보일 예정입니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정한 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나, 저자(역자)와 독자가 소통하면서 보완되고 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여, DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT기기에서 자유롭게 활용하실 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해, 독자 여러분이 언제 어디서 어떤 기기를 사용하시더라도 편리하게 전자책을 보실 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

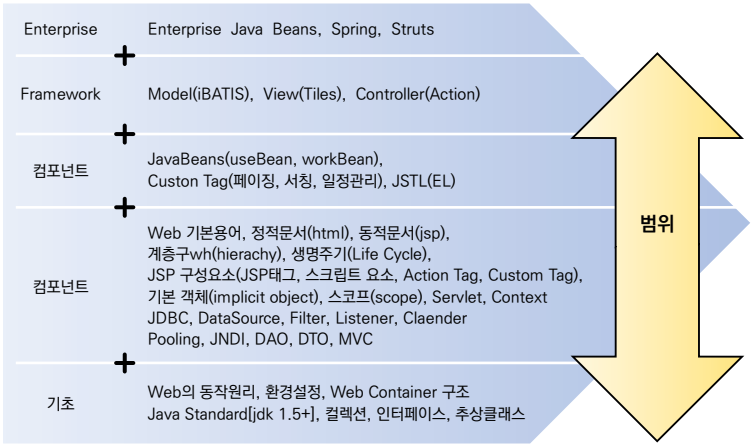
차례

1 | JSP 학습 범위와 개념 정리

1.1 이 책에서 다룰 JSP 범위

JSP는 기초부터 JSP 기본, 컴포넌트, 프레임워크, 그리고 엔터프라이즈 수준까지 폭 넓게 활용할 수 있지만 이 책에서는 기초, JSP 기본, 컴포넌트, 프레임워크(iBATIS, Tiles, Log4j)까지 다룬다. JSP 역시 기초가 중요하므로 웹의 동작원리, 환경설정, 컨테이너 구조, 반드시 알아야 할 웹 용어부터 시작한다. 그리고 이를 기반으로 JSP 구성요소와 JDBC를 익혀서 자주 사용하는 부분을 웹 컴포넌트화하여 개발 속도를 높이는 방법도 익힌다. 또한 기존 프레임워크를 이용하거나 프레임워크를 만들어서 개발 방법을 통일화하고 의도한 대로 개발하는 방법도 배워보자.

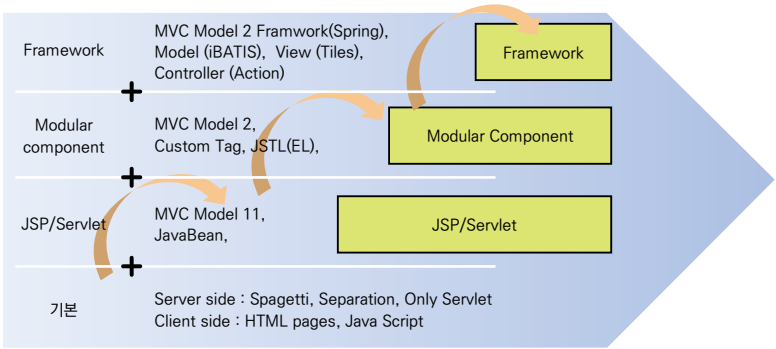
그림 1-1 JSP 범위



1.2 이 책에서 다루는 JSP 수준

JSP 사용범위가 넓어질수록 복잡도도 높아진다. 처음에는 HTML 소스와 JAVA 코드가 섞여있는 스파게티 소스로 만들어볼 수 있다. 그 다음에는 세퍼레이션 소스(자바빈 또는 유즈빈)를 사용(MVC Model 1)한 예제소스와 프로젝트도 만나게 된다. 여기에서 그치지 않고 MVC Model 2에서 JSP 중심(센트릭)으로 작성하는 방법, 서블릿 중심으로 작성하는 방법, 멀티 서블릿, 액션 서블릿으로 발전시켜 MVC Model 2 기반 프레임워크(스피링, 스트러츠) 원리까지도 자연스럽게 알 수 있다. 모듈화(화면 분할) 방법은 프레임셋Frameset), 액션 인클루드Action Include, 타일즈(Tiles 프레임워크)도 다룬다. 이 책으로 학습하면 JDBC를 직접 사용하는 스파게티부터 DAO까지 사용할 수 있을 뿐 아니라 iBATIS 프레임워크도 활용할 수 있다. 일반적인 JSP 책에서는 잘 다루지 않는 서비스 객체도 익힐 수 있으므로 기초부터 고급 기술로 발전하면서 다양한 기술을 터득할 수 있다.

그림 1-2 JSP 수준



1.3 이 책에서 다루는 JSP 내용

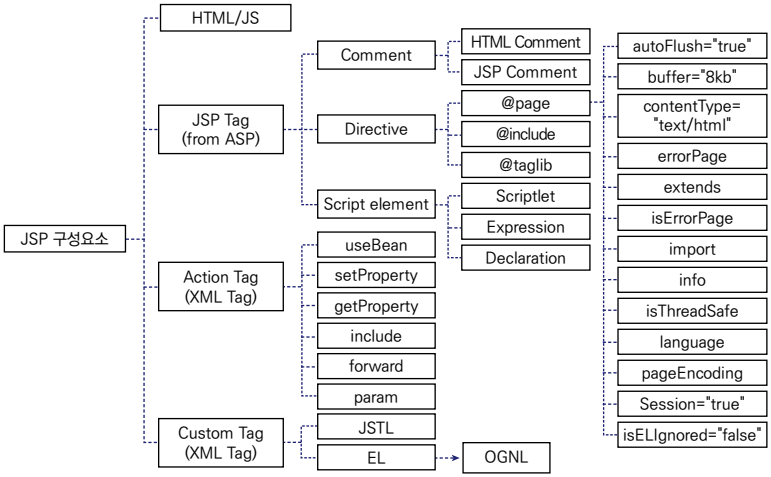
이 책에서 다루는 JSP 내용은 다음과 같다.

- 웹 기본 용어
- HTTP/HTML/JS
- 웹 서버Web Server/웹 컨테이너
- 요청/응답
- 기본 객체/구성요소/스크립트요소
- 주석
- 디렉티브
- 파라미터parameter와 파라미터 처리
- 서블릿(Servlet)/생명주기(라이프사이클life cycle)
- 서블릿 리스너/서블릿 필터
- XML
- Custom Tag/JSTL/EL
- 유즈빈useBean(자바빈JavaBean)/workBean
- 흐름제어/모듈
- DAOdata access object/DTOdata transfer object/iBATIS
- MVC(Model, View, Controller), MVC Model 1, Model 2
- 팩토리 패턴/싱글톤 패턴/헬퍼뷰 패턴
- 액션 서블릿/멀티 서블릿Multi Servlet
- JSP 웹 역사(Only Servlet, Spagetti, Separation, JSP Centric, Servlet Centric, Multi Servlet, Action Servlet, Framework)
- 일반게시판, 답변형 게시판, 페이징, 서칭, 일정관리, 입사지원관리, 인사관리

1.3.1 JSP 구성요소

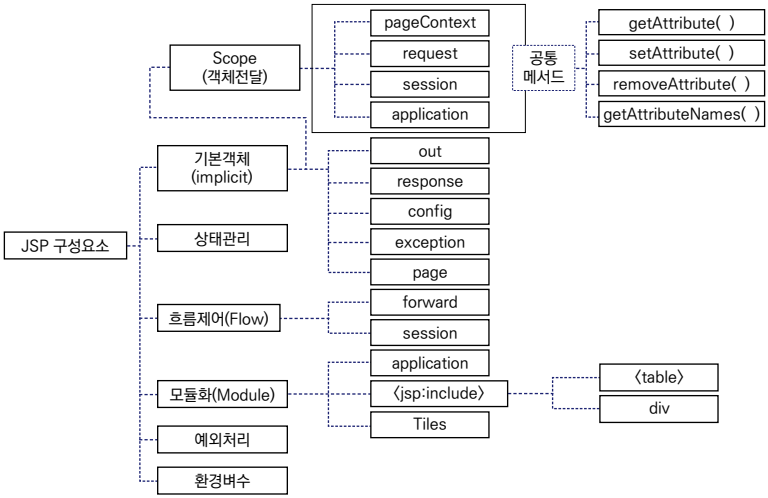
JSP 작성 시 사용하는 태그와 클래스 등을 JSP 구성요소라고 한다. 한 페이지의 특징을 설정하는 디렉티브directive, HTML사이에 자바코드를 넣어서 원하는 화면을 만드는 스크립트script요소, 스크립트요소를 줄이고 자바코드를 숨기면서 웹 프로그래밍 기능을 하는 액션action 태그 등이 JSP 기본 구성요소다.

그림 1-3 JSP 기본 구성요소



원래 한 페이지에서 다른 페이지로 객체를 전달할 수 없지만 전달하게 만드는 스코프scope 객체, 상태가 없는stateless JSP에서 상태를 만들어statefull 사용하는 세션 session, 여러 화면을 보여주기 위한 모듈화, 객체를 만들지 않고도 이미 준비된 객체를 사용하는 기본 객체implicit object, 한 화면에서 다른 화면으로 이동하는 흐름제어flow control, 예외 발생 시 처리 방법, 저장되어 있는 환경변수를 사용하는 방법 모두가 중요한 JSP 구성요소다.

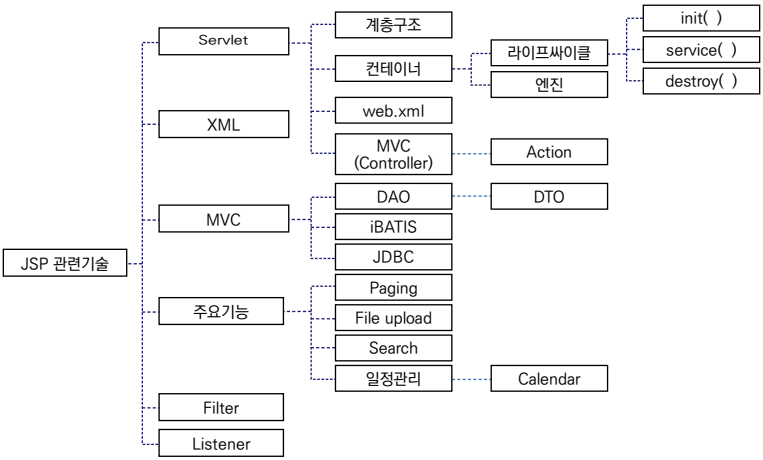
그림 1-4 JSP 중요 구성요소



1.3.2 JSP 관련 기술

JSP는 실행될 때 서블릿Servlet으로 변환되므로 서블릿을 아는 만큼 JSP를 알 수 있다. 서블릿은 웹 서버의 컨테이너에서 실행되기 때문에 컨테이너, 엔진, 생명주기life cycle도 파악해야 한다. 또한 환경변수와 서블릿을 매핑시키는 web.xml도 필요하다. 세션을 추적하고 관리하는 세션 리스너session listener, 한 페이지에서 다른 페이지로 이동할 때 가로챌 수 있는 서블릿 필터servlet filter 등도 살펴본다. 한 페이지에 화면, DB관련 작업, 제어 로직이 섞인 스파게티를 각 용도에 맞게 MVCModel, View, Controller로 분할하는 방법과 Model의 발전 순서인 JDBC, DAO, iBATIS, 서비스 객체도 배운다.

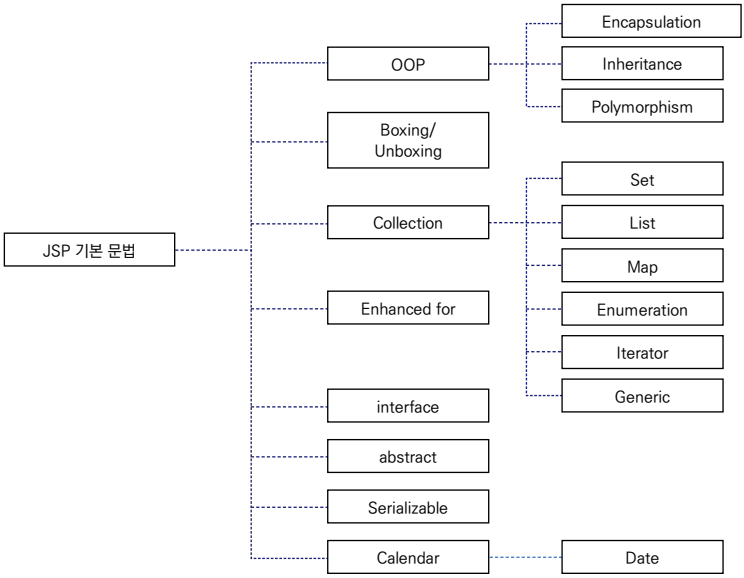
그림 1-5 JSP 관련 기술



1.3.3 JSP에서 사용되는 기본 문법

JSP는 적절하게 작동하도록 잘 만들어야 하는데, Java 기본 문법을 알면 많은 도움이 된다. JSP/Servlet은 인터페이스와 추상클래스로 이루어지므로 인터페이스와 추상클래스를 반드시 이해해야 한다. DAO의 반환타입은 주로 DTO와 List〈DTO〉 타입이므로 컬렉션을 사용할 수 있어야 한다. 이 책에서는 JSP에서 사용하는 기본 문법도 함께 설명한다.

그림 1-6 JSP 기본 문법



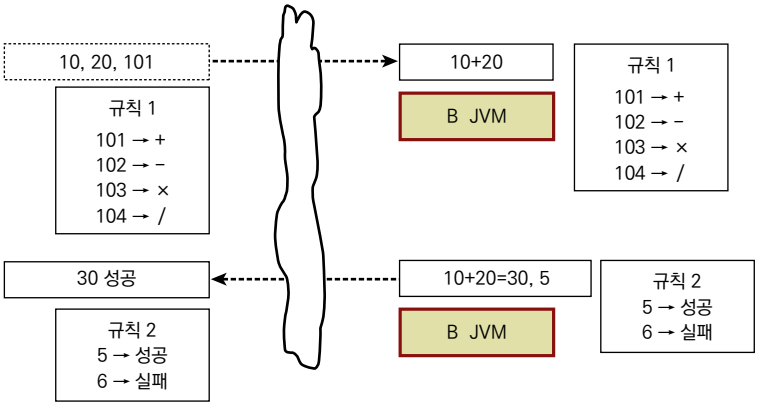
1.4 웹 용어

■ **서버Server와 클라이언트(Client)**

내 컴퓨터를 이용하지 않고 다른 컴퓨터(B JVM)를 이용하여 문제를 해결한다고 가정해보자. 그리고 '10+20'을 구한다고 할 때 숫자와 ','만 사용하는 규칙을 만들었다고 해보자. 10+20을 10, 20, 101로 만들어 다른 컴퓨터(B JVM)에 보내면, B에서는 규칙1을 보고 10+20의 값을 구하게 된다. 10+20=30을 해결한 후 규칙 2를 이용하여 30, 5를 A에게 돌려보내면 10+20은 30이라는 값이 도출되어 문제가 해결되었다는 사실을 알 수 있다.

문제를 해결해 달라고 요청하는 쪽이 클라이언트, 문제를 해결해서 돌려보내는 쪽이 서버다.

그림 1-7 서버와 클라이언트 구성



■ 요청(Request)/응답(Response)

클라이언트가 서버에게 문제를 해결해달라고 요구하는 행위는 요청Request이라고 한다. 반대로 서버가 문제를 해결해서 클라이언트에게 보여주는 행위는 응답Response이다. 따라서 웹 브라우저가 클라이언트이고, 웹 서버는 서버라 할 수 있다. 웹 브라우저를 이용하여 서버에게 한빛미디어 홈페이지를 요청하면 서버는 웹 브라우저에게 한빛미디어 홈페이지를 보여주도록 응답한다.

■ 프로토콜(Protocol, 규약, 규칙, 약속)

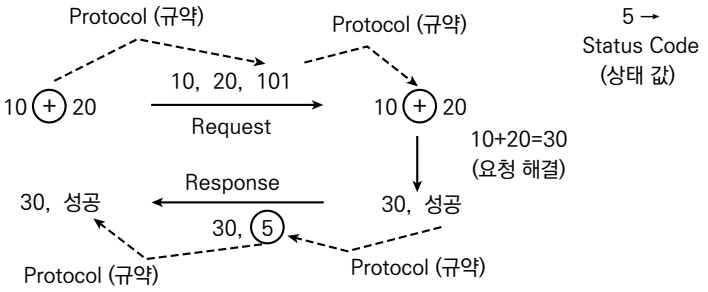
10+20을 10, 20, 101로 바꿔서 보내는 것과 10, 20, 101을 10+20으로 다시 바꾼 후 계산해 30을 얻는 과정은 전송 시의 문제를 제거할 수 있다. 같은 방법으로 10+20의 결과를 돌려보낼 때 30, 5를 사용하면 클라이언트 쪽에서는 규약을 지킨 연산결과가 30이고, 제대로 도출되었다는 점을 알 수 있다. 이처럼 클라이언트와 서버 간 전송을 위한 규약이 프로토콜이다. 또한 30, 5의 5처럼 서버 쪽에서 문제를 제대로 해결했는지 실패했는지에 따라 약속된 값을 클라이언트로 보낼 수 있는데 이를 응답 상태코드status code라고 한다.

■ 웹 서버(Web Server)

웹 서버는 서버쪽 컴퓨터에 있는 소프트웨어다. 클라이언트의 요청을 받아서 웹 페이지(HTML, 그림파일, CSS, 자바스크립트 등으로 구성된 문서)를 클라이언트인 웹 브라우저에 응답하는 역할을 한다.

웹Web은 World Wide Web의 줄임말로 문서들이 인터넷으로 연결된 컴퓨터 세계를 말한다. 하이퍼텍스트hypertext는 글자뿐 아니라 그림처럼 보여줄 수 있는 내용물을 컴퓨터 사용자가 마우스나 키보드입력으로 요청할 때 바로 접근하고 사용할 수 있는 문서다. 웹 페이지는 웹에서 인터넷으로 연결되어 다른 웹 페이지로 자유롭게 이동navigate하고 연결되어야 하는 문서이므로 하이퍼텍스트로 만들어진다.

그림 1-8 자유로운 이동이 가능한 웹 페이지

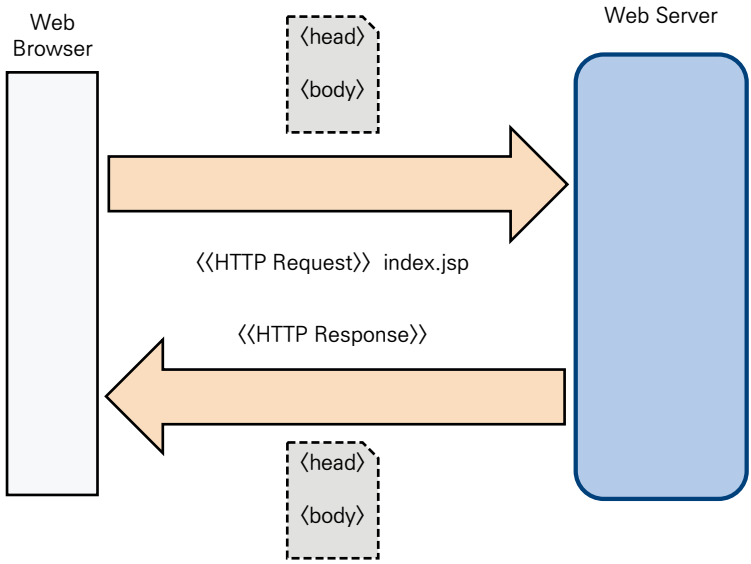


■ HTTP(Hypertext Transfer Protocol)

웹 서버에서 서버-클라이언트 사이에 대화(Request/Response)를 할 수 있도록 만든 규약이다. HTTP는 헤더header와 바디body로 구성된다. HTTP의 요청과 응답에는 메시지가 포함되어 요청과 응답에 대한 상태를 알 수 있게 한다. 서버는 요청 헤더 메시지를 읽고 클라이언트의 요청 사항을 파악한 다음 클라이언트에게 응답을 보낸다. 응답헤더에는 요청이 제대로 처리되었는지, 응답해주는 서버의 간단한 정보, 응답 내용의 타입 및 인코딩, 응답 크기 등이 포함된다. 응답헤더가 브라우저

에게 전달되면 응답바디 내용이 브라우저에 출력된다. 헤드와 바디 사이에는 한 줄이 비어있으므로 쉽게 구분할 수 있다.

그림 1-9 HTTP를 통한 통신



요청 메시지의 첫 줄에 요청라인request line이 있고(GET /pagecentric01_comp01/index.jsp HTTP/1.1), 2번째 줄부터 다음과 같은 헤더내용이 있다.

```
Accept:image/jpeg, */*
Accept-Language: ko-KR
```

헤더가 끝나면 한 줄이 비어있다. 요청의 GET 방식은 바디가 없어서 한 줄이 비어있는 상태로 끝나지만, 포스트POST방식일 때 파라미터가 있다면 빈줄 밑에 파라미터가 들어간다. 응답 메시지는 'HTTP/버전 응답상태코드 상태'를 반환하고 다음 줄은 서버, 컨텐트타입, 응답크기, 응답시간 등을 보여준다.

HTTP/1.1 200 OK

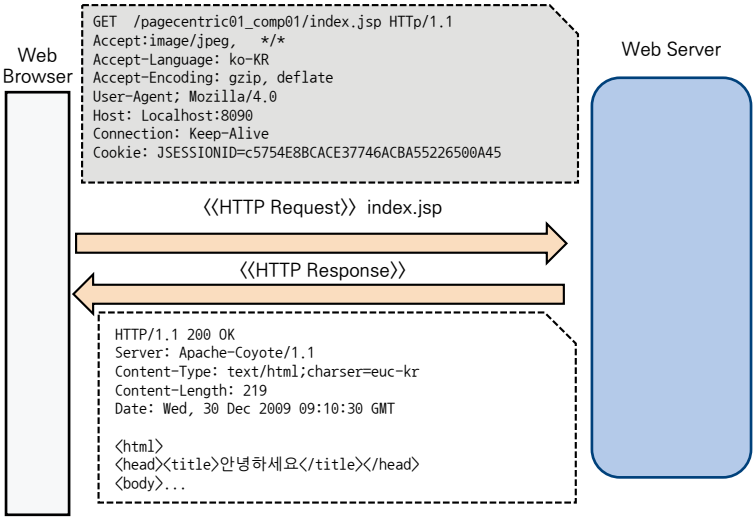
Server: Apache-Coyote/1.1

Content-Type: text/html;charset=euc-kr

Content-Length: 219

Date: Wed, 30 Dec 2009 09:10:30 GMT

그림 1-10 응답 메시지



■ HTML(HyperText Markup Language)

HTML은 웹 페이지를 구성하는 핵심요소며, 마크업 언어Markup Language는 문자를 이용하여 다른 문자의 특성을 표시하는 방법이다.

〈b〉안녕하세요〈/b〉

〈b〉〈/b〉를 태그tag라 한다. 앞의 〈b〉는 시작태그, 뒤의 〈/b〉는 '끝태그'라고 부르며, 태그사이의 '안녕하세요'는 컨텐트다. 〈b〉안녕하세요〈/b〉와 같은 '시작태그-

컨텐트-끝태그'를 엘리먼트element라고 한다. 〈b〉 태그는 '안녕하세요'를 볼드체boldface로 표현하라는 의미다. 다시 말하면 〈b〉 태그는 컨텐트의 특성을 표시한다.

안녕하세요

웹 브라우저에서는 위와 같이 〈b〉 태그가 없어지고 볼드체 '안녕하세요'만 보여준다. 웹 브라우저에서 컨텐트를 표현(출력)하려고 태그를 이용하는 방법이 바로 HTML이다.

■ 상태코드(Status Code)[01]

클라이언트가 서버에게 요청하면, 서버는 요청을 처리한 다음 그 결과를 3자리 숫자로 된 상태코드와 함께 클라이언트에게 보내준다. 성공 200이외에는 서버 쪽에서 오류가 발생했다고 볼 수 있어 오류코드Error Code라고도 한다. '200 성공', '404 경로가 잘못됨', '500 서버쪽에서 문법적으로 예외발생'은 꼭 기억하자.

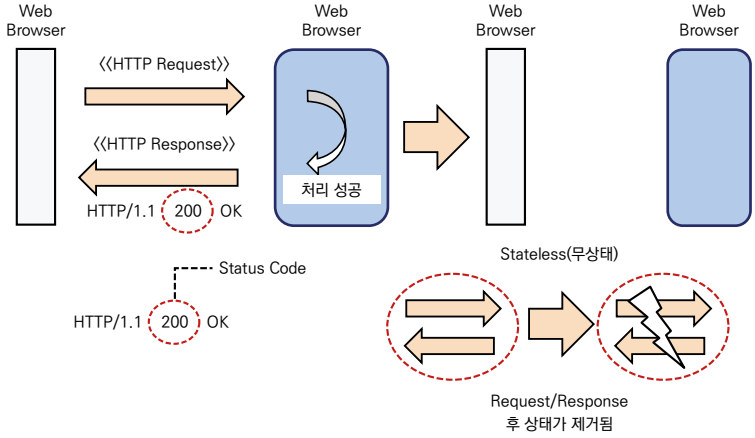
상태 코드	상태 메시지	상태 내용	발생 예
100	Continue	클라이언트로부터 일부분만 받았으니 나머지 요청정보를 요청	
200	OK	오류 없이 클라이언트로 전송 성공	성공
300	MultipleChoices	최근에 옮겨진 데이터를 요청	
404	Not Found	서버가 요청한 파일이나 스크립트를 찾지 못함	경로가 잘못되거나 없는 jsp/servlet을 호출
405	Method Not Allowed	메서드 허용 안됨	doGet()이 없는데 GET 방식으로 호출
500	Internal Server Error	서버 내부 오류	문법오류 등 서버코드에서 문제 발생

01 상태코드 관련 웹 사이트: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

■ 상태(State) – 무상태/유상태(Stateless/Statefull)

HTTP를 통한 클라이언트와 서버 사이의 대화는 방금 전 대화를 기억하지 못하는 '무상태stateless'다. 서버는 웹 브라우저를 통해 받은 요청에 응답한 다음 연결을 끊는다. 이 상태에서 클라이언트가 서버에게 요청을 하려면 서버에 다시 연결해야 한다. 반면 DB 서버는 클라이언트와 나눈 대화를 기억(유상태, statefull)한다. 대화가 끝난 후 다시 연결하지 않아도 요청을 할 수 있다. DB 서버가 대화를 끝내려면 명시적(close 호출)으로 끝내야 한다. HTTP Session을 무상태로 한 이유는 많은 클라이언트들의 요청에 의한 웹 서버의 과부하를 방지하기 위한 것이다. 그러나 DB 서버는 사용자가 제한적이므로 연결을 유지하는 편이 좋다.

그림 1-11 상태 연결



■ HTTP 요청 메서드

웹 서버나 웹 애플리케이션 서버는 서버-클라이언트 사이의 요청/응답용 프로토콜로 HTTP를 지원한다. HTTP는 헤드와 바디로 나뉘며, 전송데이터를 보내는 방법에 따라 요청 메서드 종류가 달라진다. JSP/Servlet은 GET, POST를 주로 사용하고 HEAD는 가끔 사용한다.

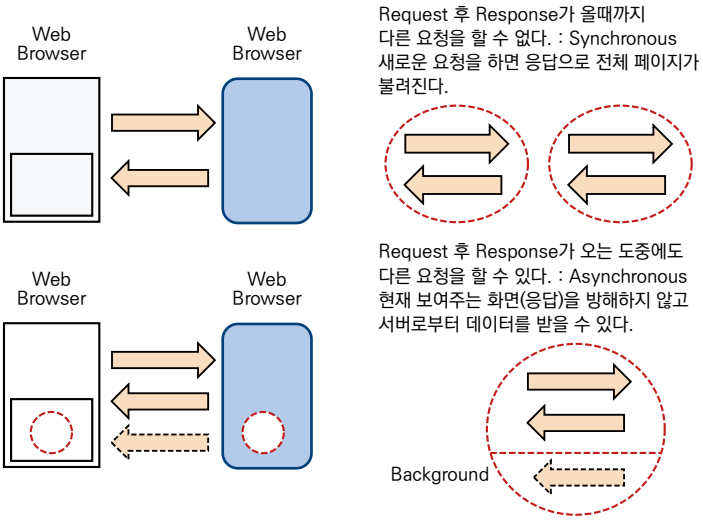
표 1-1 HTTP 주요 요청 메서드

요청 메서드	설 명
GET	서버에 요청 메서드를 보낼 때 헤드에만 내용이 있고 바디에는 전송데이터가 없다. 요청을 간단하게 하기 위한 것으로 헤드에 포함되는 내용의 크기도 제한적이다. 또한 URL에 전송데이터가 노출될 수 있다. HTTP Server는 GET과 HEAD를 반드시 지원해야 한다.
POST	서버에 보낼 전송데이터를 바디에 넣어서 요청한다. 〈FORM〉을 이용하여 전송데이터를 서버로 보낼 때 사용한다. 전송데이터 크기에 제약이 없다.
HEAD	응답용으로 사용되며 헤드에만 내용이 있다. 바디 없이 헤드정보, 캐시정보를 클라이언트에 보낼 때 사용한다.

■ 동기 / 비동기(Synchronous / Asynchronous) 전송

요청 후 응답이 올 때까지 다른 요청을 할 수 없는 전송을 동기라고 한다. 새로운 요청을 하면 한 페이지 전체를 다시 불러온다. 응답할 때마다 페이지 전체를 다시 불러오고 프로그레스바가 작동한다. 요청한 다음 응답을 받아 화면을 보여주면서 백그라운드에서 서버로부터 데이터를 받는 등 다른 작업을 할 수 있는 전송상태를 비동기라고 한다. 비동기 상태에서 서버는 필요한 데이터만 전송되도록 응답한다. 1991년부터 비동기를 사용하기 시작했고, 1995년 이후 AJAX(Asynchronous Javascript And XML)라는 이름으로 많이 사용하고 있다.

그림 1-12 새로운 요청에 대한 응답

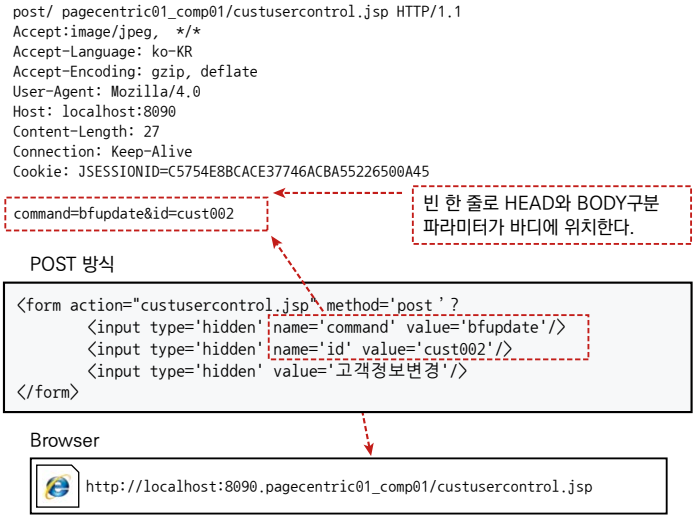


■ 파라미터(parameter, 쿼리스트링)

클라이언트에서 서버로 요청을 할 때 요청경로 이외에도 데이터를 보낼 수 있다. 서버로 보내는 데이터는 '키=밸류'형태로 전송된다. 예를 들어 'command=detail&id=cust090'과 같이 서버로 보내는 데이터를 파라미터라고 한다. GET 방식은 헤드에 파라미터가 포함되어 있다. 헤드부분은 웹경로에 노출되므로 웹 브라우저의 경로 부분에서도 볼 수 있다. 파라미터는 '?'로 시작을 표시하고 파라미터가 여러 개면 '&'를 이용하여 연결한다. 헤드 길이는 제한적이기 때문에 파라미터 크기에도 제한이 있다. POST 방식은 바디에 파라미터가 포함되므로 경로에 파라미터가 노출되지 않는다. 파라미터의 크기에도 제한이 없다.

그림 1-13 GET방식과 파라미터

그림 1-14 POST방식과 파라미터



■ 정적문서/동적문서

서버에서 클라이언트로 응답할 때 바디 부분은 HTML로 구성된다. HTML은 클라이언트인 웹 브라우저에 전달되어 화면에 출력된다. 서버에서는 변환이나 실행이 되지 않으며, 그대로 웹 브라우저에 전달된다. HTML과 같은 문서를 정적문서static document라고 한다. 서버에서는 변환이나 실행을 하지 않다가 웹 브라우저에서 정상문자 입력 검증(validate-null인지 아닌지 등)이나, 변환 혹은 실행을 하는 스크립트(자바스크립트)를 클라이언트 동적문서dynamic document라 한다. HTML이나 스크립트 언어는 클라이언트 문서다. 스크립트는 브라우저에서 한 줄씩 인터프리터로 전달되어 실행된다. JSP/Servlet은 서버 문서이고, 컴파일되어 실행된 결과를 HTML로 만든다. JSP/Servlet는 서버에서 실행되며 클라이언트에게 응답하는 문서를 만들므로 동적문서다. 웹 프로그래밍은 클라이언트의 요청을 받아 웹 서버에서 JSP/Servlet을 실행하고 결과를 웹 브라우저로 응답하는 것이다.

클라이언트사이드 문서의 종류

- 정적문서 : HTML, CSS
- 동적문서 : 스크립트

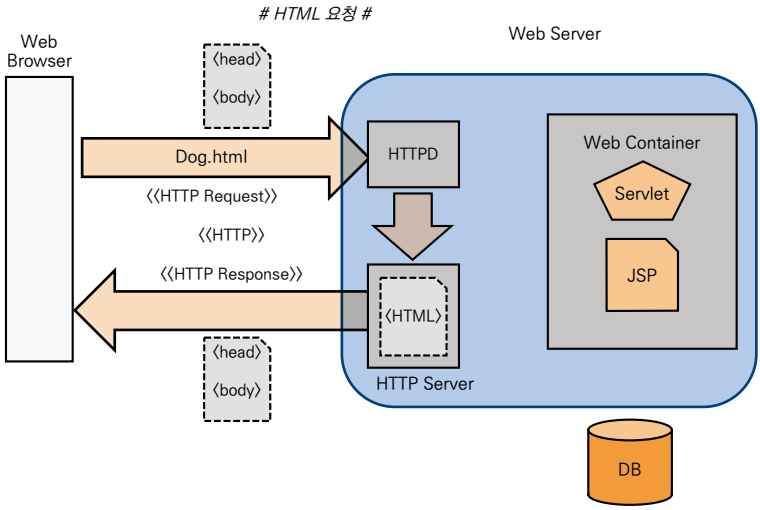
서버사이드 문서의 종류는

- 동적문서: JSP/Servlet

■ HTTP Server

HTML, CSS, 자바스크립트로 구성된 문서는 서버에서 변환되거나 실행되지 않는다. 이런 클라이언트사이드 문서를 요청한 웹 브라우저로 응답해주는 서버가 HTTP Server다. HTTP Server는 요청에 대한 응답만 한다.

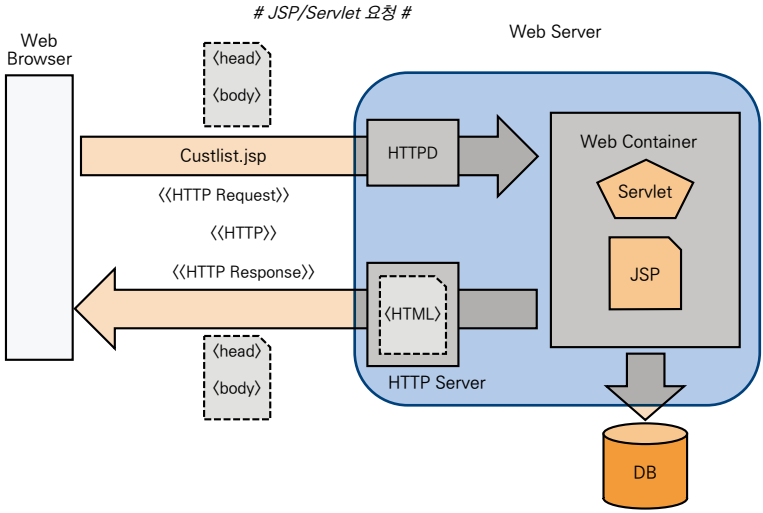
그림 1-15 HTTP Server



■ 웹 서버 구조

요청이 들어오면 확장자(.jsp)등을 확인하여 서버사이드 문서를 요청했는지 판단한다. HTTPD는 요청된 문서가 클라이언트 사이드인지 서버사이드인지 판단한다. 서버사이드라면 웹 컨테이너에서 JSP/Servlet을 실행시킨 다음, 요청 결과를 클라이언트 사이드 문서(HTML)로 만들어 HTTP Server로 보내면 HTTP Server가 웹 브라우저로 보낸다.

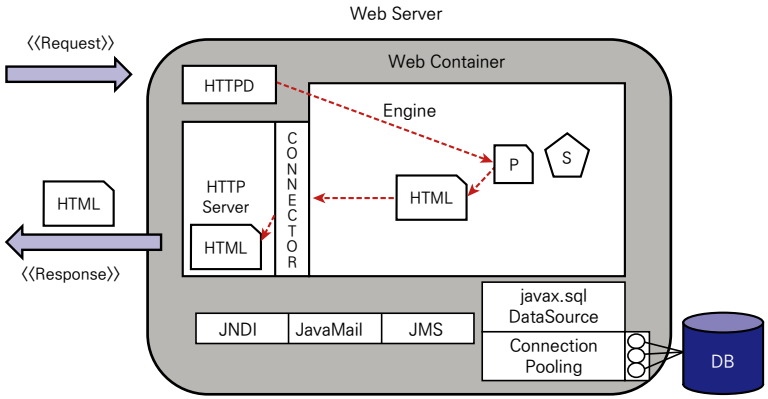
그림 1-16 웹 서버 구조



■ 웹 컨테이너(Web Container)

웹 서버에는 HTML을 클라이언트로 보내는 HTTP Server, JSP/Servlet을 실행하는 엔진, 엔진과 JSP/Servlet를 실행하는데 필요한 라이브러리 등을 포함한 웹 컨테이너가 있다. 엔진은 JSP/Servlet을 실행해 결과물인 HTML을 만든다. 커넥터connector는 HTML을 HTTP Server로 보내 웹 브라우저가 응답하게 한다. Tomcat 웹 서버Tomcat web server에는 카탈리나catalina 컨테이너, 자스퍼jasper 엔진, 컨테이너와 HTTP Server를 연결하는 코요테coyote 커넥터가 있다. 웹 프로그래밍에는 웹 컨테이너가 반드시 있어야 한다. HTTP Server는 자바로, Apache HTTP Server는 C로 만들어졌다는 점도 알아두자.

그림 1-17 웹 컨테이너



■ 파라미터 처리

클라이언트에서 서버로 요청할 때 요청경로와 데이터를 함께 보낼 수 있다. 이 데이터는 '키=값'형태로 전송되며 파라미터라고 한다. 웹 서버에서 파라미터를 받을 때는

String 값 = reuest.getParameter("키");

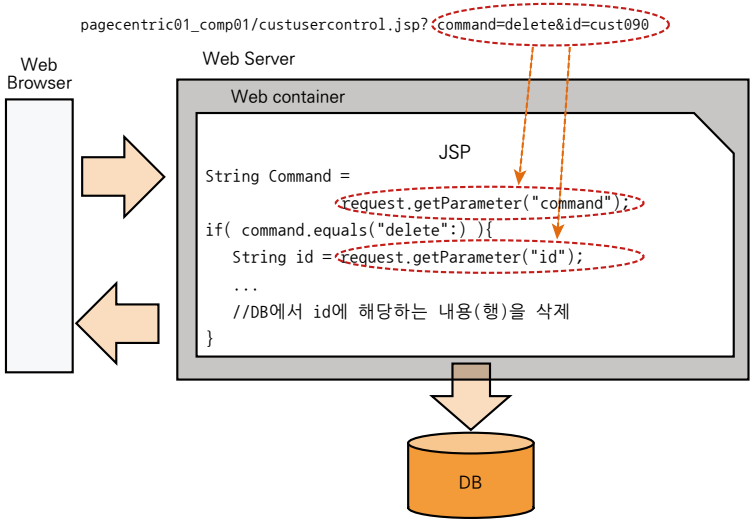
로 받는다. '키'의 이름으로 '값'을 받아 요청을 처리한다. 파라미터의 '키'와 '값'은 모두 String이다. 데이터베이스에서 id가 'cust003'인 행을 삭제하고 싶다면 다음처럼 요청에 파라미터를 함께 보낸다.

pagecentric01_comp01/custusercontrol.jsp?command=delete&id=cust003

서버는 id를 이용하여 요청한 행을 삭제한다. 요청 중 상세내용을 확인하기 위해 command를 받고, if문에서 command가 삭제, 삽입, 변경인지 확인하고 요청별로 처리한다.

```
String command = request.getParameter( " command " );
 if( command.equals( " delete " ) )
    String id = request.getParameter( " id " );
     …
    //DB에서 id에 해당하는 내용(행)을 삭제한다.
```

그림 1-18 파라미터 처리



■ 서블릿(Servlet)

서블릿은 "Servlet = Server + let"이며, 'let'는 컨테이너에서 실행되는 프로그램이다. 다시 말해 서블릿은 웹 서버(Tomcat)의 컨테이너(카탈리나)에서 실행되는 웹 프로그래밍이다. 요청 또는 요청과 동반하는 파라미터를 서블릿에서 받아 요청을 처리하고 결과를 HTML로 만든다. 웹 브라우저는 응답으로 보내진 HTML을 받아서 화면에 출력한다.

'let'가 붙은 다른 프로그램도 있다. 애플릿applet은 applet = application + let로 컴파일된 프로그래밍이 웹 브라우저의 컨테이너에서 실행되는 애플리케이션이며, 애플릿은 클라이언트사이드 프로그래밍이다. 미들릿middlet은 midMobile Information Device Profile+let를 말하며 모바일 정보기기 프로파일을 사용하는 애플리케이션으로 핸드폰 컨테이너에서 실행된다.

■ 라이프사이클(Life Cycle)

JSP/Servlet은 웹 프로그래밍이다. 웹 프로그래밍은 컨테이너가 반드시 필요하며, 프로그래밍을 엔진에서 실행하여 원하는 결과를 생성한다. 결과물은 HTML로 만들어서 웹 브라우저에 응답한다. 서블릿의 'let'는 서블릿이 컨테이너에서 실행되는 프로그래밍이라는 점을 알려준다. 'let'가 붙은 애플리케이션은 컨테이너의 명령에 따라 생성, 초기화, 실행, 소멸 등 정해진 행동을 한다. 이렇게 하는 것을 라이프사이클이라고 한다. 컨테이너는 요청에 따라 정해진 메서드를 호출한다. 요청을 처음으로 받으면 서블릿을 생성한다. 초기화(init()) 메서드를 호출하여 생성된 서블릿을 실행하는 데 필요한 데이터나 값을 얻는다. 실행(service()) 메서드를 호출하여 요청을 처리하고 결과를 응답해준다. 두 번째 요청부터는 실행(service()) 메서드만 호출한다. 더 이상 요청이 없다면 소멸(destroy()) 메서드를 호출하여 자원을 회수하고, 서블릿의 객체instnace를 제거한다. 컨테이너는 초기화와 소멸 메서드를 한 번만 호출한다.

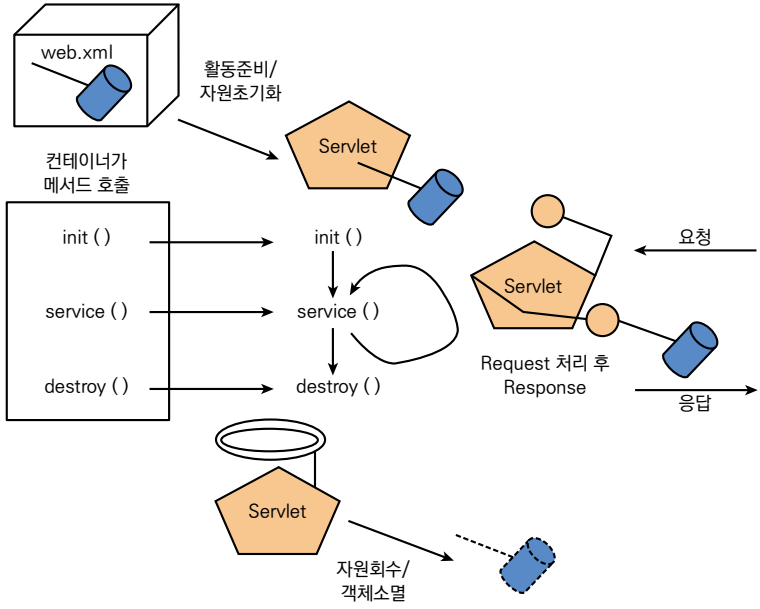
요청에 따라 필요한 로직을 실행하려면 아래와 같이 service() 메서드 바디( )에 프로그래밍하면 된다.

```
public void service(request, response)
 //실행로직
```

요청을 받으면 컨테이너는 서블릿의 service() 메서드를 호출하여 로직을 실행한

다.

그림 1-19 웹 라이프 사이클



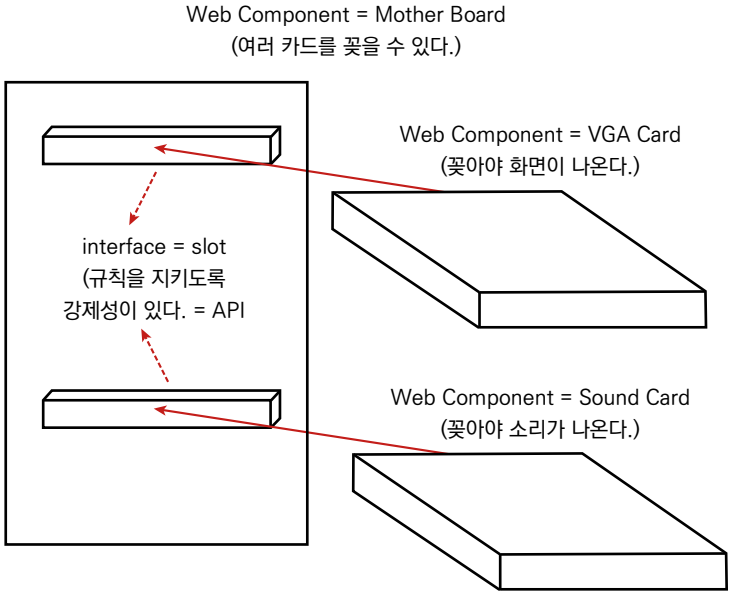
■ JSP(Java Server Page)

JSP 중 Server Page는 서버에서 실행되는 라이프사이클이 있는 웹 프로그래밍용 클래스를 말한다. Java는 개발 언어를 가리킨다. Tomcat 웹 서버에서 자스퍼jasper 엔진이 JSP를 서블릿으로 변환한다. 라이프사이클을 고려하지 않고 개발해도 서블릿으로 변환될 때 라이프사이클 메서드가 자동으로 만들어져 서블릿처럼 실행된다. hello.jsp를 만들면 org.apache.jsp.hello_jsp.java와 같이 서블릿으로 변환되고 org.apache.jsp.hello_jsp.class로 컴파일된 다음 실행된다. JSP도 서블릿이므로 서블릿에서 사용하는 객체나 메서드를 JSP에서도 사용할 수 있다.

■ 웹 컴포넌트

컨테이너에서 실행될 수 있도록 필요한 규칙을 준수한 웹 프로그램 묶음을 웹 컴포넌트라고 한다. 예를 들어 컴퓨터의 마더보드는 여러 카드를 꽂을 수 있는 구멍(슬롯)이 있고, 각 카드는 정해진 목적이 있다. 목적을 이루기 위해 여러 부속품으로 여러 카드를 만든다. JSP나 서블릿은 부속품에, 웹 컴포넌트는 각 카드에 해당한다. 각 카드는 마더보드에 꽂혀야 실행된다. VGA 카드를 꽂아야 화면이 나오며, 사운드 카드를 꽂아야 소리가 나온다. 카드를 마더보드에 꽂을 때는 슬롯의 암수가 맞아야한다. 슬롯 설치 시 반드시 이 규칙을 지켜야 하는데, 웹 컴포넌트에서드 API라는 규칙을 반드시 지켜야한다. 따라서 웹 프로그래밍은 웹 컴포넌트를 만드는 방법, 웹 컴포넌트와 웹 컨테이너 사이의 규칙을 배우는 것이라 할 수 있다.

그림 1-20 웹 컴포넌트



■ 컨텍스트(Context)

배포단위, 실행단위의 디렉터리를 컨텍스트라고 한다. 개발 후 JSP나 서블릿, 그림, HTML, web.xml을 압축하여 묶어서 (아카이브) war를 만든다. 이 war를 웹 컨테이너에 놓으면 압축이 풀려 실행을 위한 디렉터리가 된다. 이 디렉터리가 바로 컨텍스트이며 모든 파일을 각 용도에 맞게 컨텍스트 안의 디렉터리에 놓아야 한다. 예를 들어 hello.war를 웹 컨테이너에 놓으면 hello.war가 풀어져 hello 디렉터리가 된다. 컨텍스트 바로 아래에는 HTML, CSS, 자바스크립트, 이미지, JSP, WEB-INF 디렉터리가 있다. WEB-INF 디렉터리 안에는 web.xml이 있고, 서블릿과 컴파일된 클래스는 classes 디렉터리에 있어야 하며, .jar로 압축된 라이브러리는 lib에 안에 있어야 한다. 컨테이너는 컨텍스트의 web.xml을 가장 먼저 한 번 읽어들인다.

그림 1-21 컨텍스트

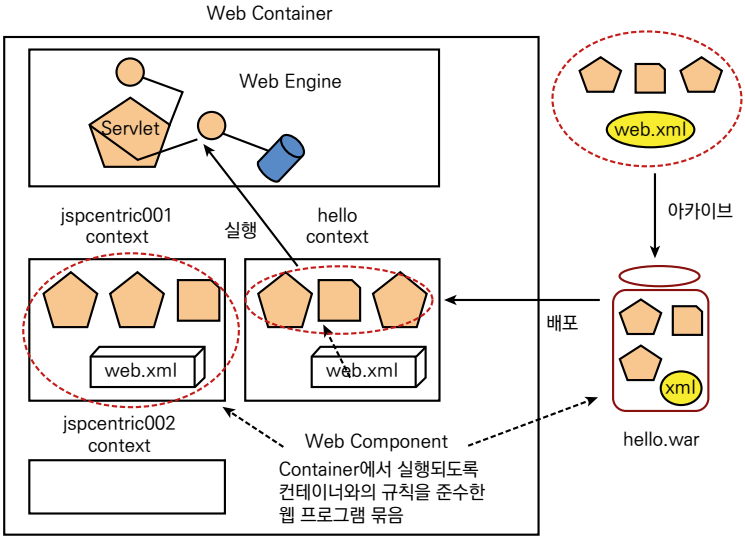
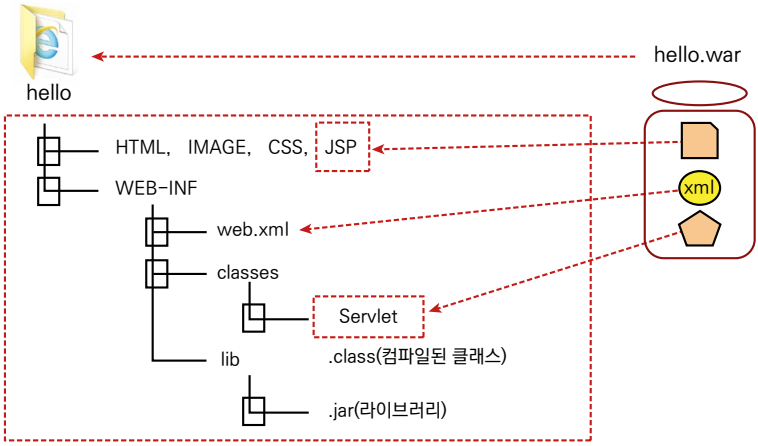


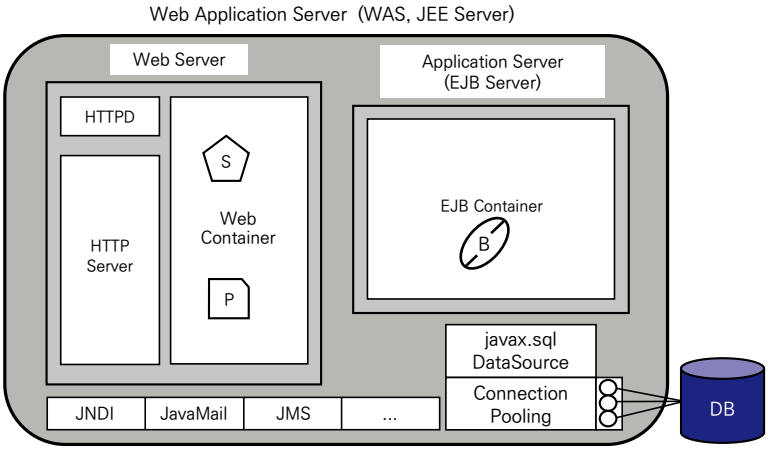
그림 1-22 컨텍스트



■ WAS(Web Application Server-Java Enterprise Edition Server)

WAS(JEE)는 "Web Server + Application Server + Service"의 의미다. 웹 서버는 JSP/Servlet를 위한 웹 컨테이너와 HTML을 서비스하는 HTTP Server가 있다. Application Server에서는 비즈니스 로직을 수행하는 EJB 컨테이너가 있다. 서비스는 JNDI, JMS, JTA, JavaMail을 제공하여 다른 기능의 서버와 연결하여 사용하게 하거나 서버 안에서 편리하게 사용할 수 있게 한다. Enterprise Edition의 필수요소를 모두 구현한 서버를 Enterprise Edition Server 또는 WAS라 부른다. WAS에는 WebLogic, WebSphere, JBoss, Zeus 등 수많은 제품이 있다. Tomcat을 WAS로 부르기도 한다. 이 책에서는 Tomcat 웹 서버를 Tomcat 애플리케이션 서버와 같은 의미로 사용한다.

그림 1-23 웹 애플리케이션 서버



■ 레이어/티어

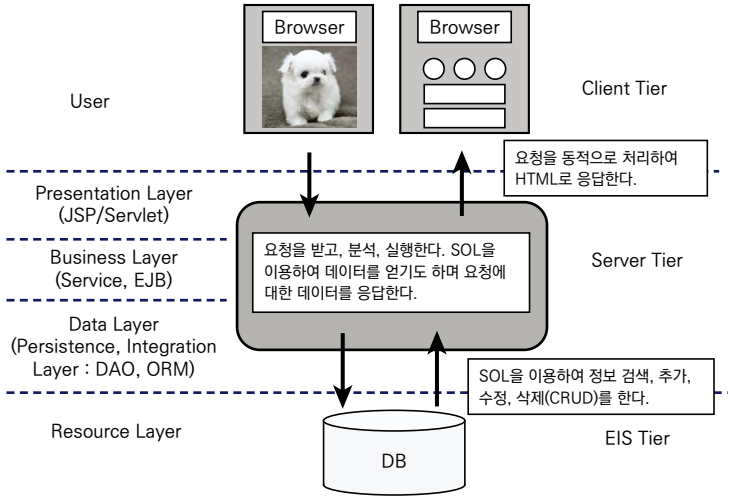
레이어는 프로그램의 역할에 따라 논리적으로, 티어는 시스템의 역할에 따라 물리적으로 나눈 것이다. 혼용되기도 하지만 논리적인 것과 물리적인 것으로 구분된다는 점을 기억하자. 데이터베이스에서 DVD 대출정보를 찾아 웹 브라우저에 출력하는 웹 프로그래밍을 한다고 가정해보자. DVD 대출 프로그램은 데이터베이스에 접근해 정보를 가져오는 부분Data Layer, 고객정보인지 비디오 정보인지 요청을 판단하고 실행하는 부분Business Layer, HTML로 결과를 만들고 응답하는 부분Presentation Layer으로 나눌 수 있다. DVD 대출 프로그램은 역할에 따라 나눌 수 있으며 이를 레이어layer라 한다. 데이터베이스 서버, 웹 서버, 클라이언트인 웹 브라우저등 시스템을 물리적으로 나눌 수 있는데 이것을 티어tier라 한다. 웹 프로그래밍은 기본적으로 3티어다. 레이어는 보통 4개로 나눈다. 이 책에서는 3티어와 4레이어를 기본으로 사용한다.

- 화면 레이어Presentation Layer : JSP와 서블릿이 담당한다. 요청을 받아 비즈니스 레이어에 정보를 요청한다. 결과를 받아 요청한 화면을 동적으로 만들고, 요청한

화면으로 옮긴다.

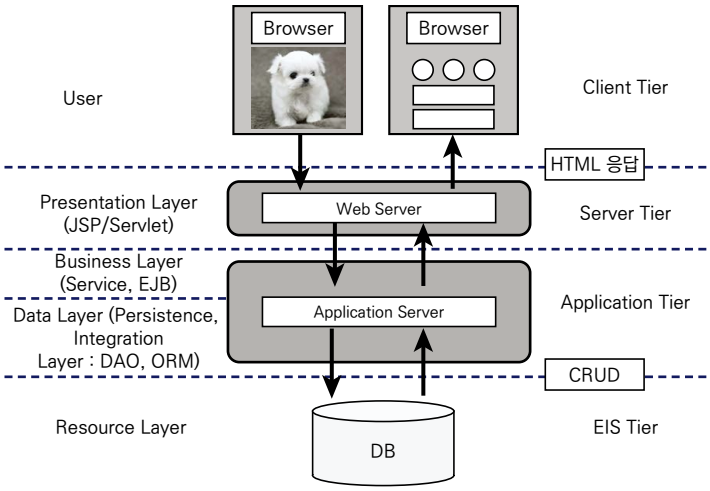
- 데이터 레이어Data Layer: 영속성 레이어Persistence Layer로도 알려진 레이어로, 데이터베이스에 쿼리(CRUD)를 실행한다. 데이터베이스에 접근하여 쿼리를 실행하기 위해 JDBC를 이용하거나, 해당 쿼리에 대해 JDBC 작업을 자동으로 실행해 쉽게 사용할 수 있도록 만든 프레임워크(ORM-iBatis, Hibernate)를 사용한다.
- 비즈니스 레이어Business Layer: 화면 레이어는 데이터 레이어를 이용한다. 비즈니스 레이어에서 얻은 객체를 비즈니스 객체라고 하며, 데이터베이스의 테이블(엔터티)과 관련이 있다. 데이터 레이어에서 얻은 결과로 화면 레이어에서 필요한 비즈니스 객체를 만들어도 된다.

그림 1-24 3티어 구성



애플리케이션 서버를 사용하는 경우는 4티어가 된다. 4티어일 때는 웹 서버와 애플리케이션 서버가 물리적으로 나누어진 티어가 되므로 레이어를 티어라고 부른다.

그림 1-25 4티어 구성



■ DAO(Data Access Object-데이터 접근 객체)

데이터베이스에 관련된 작업(CRUD-Create, Retrieve, Update, Delete: SQL DML+Select)을 전문적으로 담당하는 객체다. DAO 안의 메서드는 모두 데이터베이스와 관련된 작업을 한다. 아래와 같이 CRUD를 실행하는 메서드는 JDBC등을 이용하여 데이터베이스에 접근해서 쿼리를 실행한다. 다른 개발자도 해당 메서드를 호출하면 해당 쿼리를 실행하여 결과를 얻을 수 있다.

public CustUserDto getCustUser(String id) 해당 id에 해당하는 결과를 얻는 쿼리 실행
public int addCustUser(CustUserDto uDto)insert 쿼리 실행
public int updateCustUser(CustUserDto uDto)update 쿼리 실행
public int deleteCustUser(String id)delete 쿼리 실행

■ DTO(Data Transfer Object, 데이터 전송 객체)

데이터베이스의 테이블(엔티티)에 해당하는 객체로 테이블의 컬럼들을 일대일로

저장할 수 있는 멤버필드가 있고 get/set 메서드를 갖는다. 연필들(테이블의 컬럼들)을 안전하게 옮길 수 있도록 만들어진 필통DTO이라고 생각하자. DTO는 로직이 없으며 일반적으로 하나의 DTO가 하나의 행ROW에 해당되고, 대부분 DAO와 같이 사용된다. DAO의 메서드는 DTO를 반환하거나 DTO를 대입한다. 비즈니스 레이어에서 반환하는 Business 객체를 DTO로 보면 된다. 아래 소스에서 CustUserDto가 DTO다. getCustUser() 메서드를 호출하면 테이블에서 해당 행에 해당하는 데이터를 DTO에 담아서 반환한다.

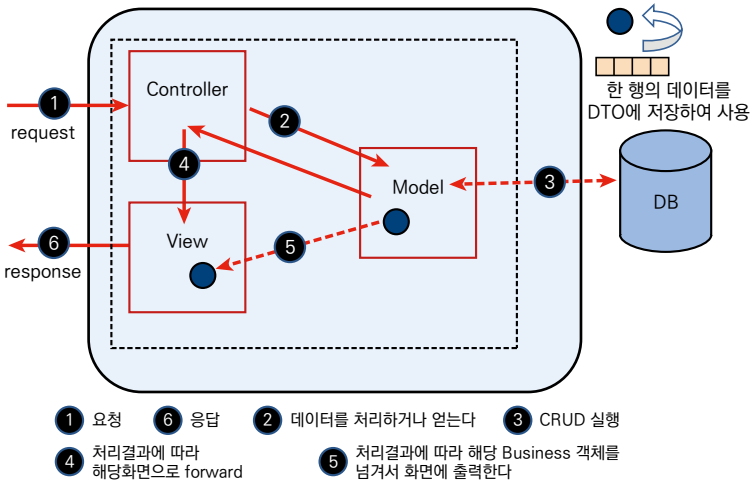
```
public CustUserDto getCustUser(String id)  해당 id에 해당하는 결과를 얻는 쿼리 실행
public int addCustUser(CustUserDto uDto)insert 쿼리 실행
public int updateCustUser(CustUserDto uDto)update 쿼리 실행
```

■ MVC 패턴(Model-View-Controller)

MVC 패턴은 요청을 처리하는 과정에서 발생하는 처리순서, 데이터 전송, 관리 작업과 데이터 출력에 대한 웹애플리케이션 작업을 간단하게 도식화한다. 요청을 받은 컨트롤러는 요청을 분석한다. 요청에 해당하는 모델을 이용하여 비즈니스 로직(데이터 베이스에 관련된 작업)을 실행하고 비즈니스 객체를 얻는다. 해당 뷰(화면)로 제어권을 넘긴다(포워드). 뷰는 받은 비즈니스 객체를 동적으로 처리하고 HTML로 화면을 만들어(렌더링) 웹 브라우저에 응답한다. 아래 그림은 MVC 패턴을 웹어플리케션에서 구현한 MVC Model 2의 처리과정이다. 과정은 다음과 같다.

① 컨트롤에 요청을 한다.
② 컨트롤은 요청을 분석하고 해결할 모델을 찾아서 실행한다.
③ 모델은 쿼리를 실행하고 결과를 얻는다.
④, ⑤ 해당화면으로 이동하여 모델에서 얻은 데이터를 동적으로 처리한다(HTML로 렌더링).
⑥ 브라우저로 요청했던 화면을 보내 응답한다.

그림 1-26 MVC 패턴


Controller
request
Model
View
response
DB
한 행의 데이터를
DTO에 저장하여 사용
1 요청
6 응답
2 데이터를 처리하거나 얻는다
3 CRUD 실행
4 처리결과에 따라
해당화면으로 forward
5 처리결과에 따라 해당 Business 객체를
넘겨서 화면에 출력한다

2 | JSP 실행환경

2.1 JEE 버전과 발표연도

그림 2-1 Java Platform Enterprise Edition(JEE)의 버전과 발표연도

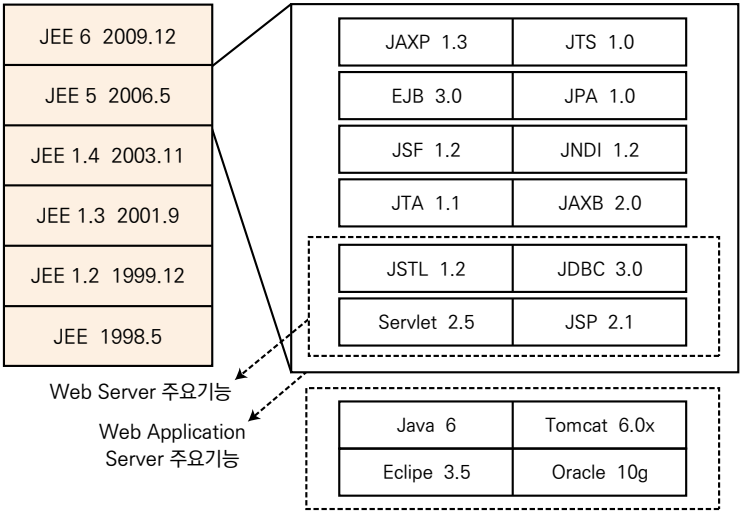


그림 2-1은 "Java Platform Enterprise Edition(줄여서 JEE라고 함)"의 버전과 발표연도다. JEE 5의 기능을 구현한 서버를 Web Application Server(WAS-와스)라고 한다. WAS는 Servlet/JSP를 중심으로 한 웹 서버Web Server와 EJB를 중심으로 한 애플리케이션 서버Application Server-EJB Server로 나눌 수 있다. 이 책에서는 Servlet/JSP를 중심으로 한 웹 서버를 다룬다.

그림 2-2 Java Platform Enterprise Edition(JEE)의 버전과 발표연도

JEE 버전/발표년도	Servlet/JSP 버전	Tomcat 주요버전/년도	Java 최소 버전
JEE 6 2009.12	3.0 / 22	7.0x/	6
JEE 5 2006.5	2.5 / 2.1	6.01x/2008.7	5
J2EE 1.4 2003.11	2.4 / 2.0	5.59/2005 5.0x/2004	5.5(5) 5.0(4)
J2EE 1.3 2001.9	2.3 / 1.2	4.1.12/2003 4.0x/2001	1.3
J2EE 1.2 1999.12	2.2 / 1.1	3.0x/2000	1.2
1997	1.0 /		

그림 2-1은 JEE 버전에 대한 Servlet/JSP 버전과 주요 Tomcat 서버(웹 서버)의 버전을 보여준다. 이 책에서 다룰 JEE 5는 2006년 5월에 발표되었고, Servlet/JSP 버전은 2.5/2.1이다. Tomcat 6.0.0은 2006년에 발표되었지만 2008년 7월의 6.01버전부터 많이 사용되기 시작했다.

그림 2-3 JSE, JEE, JSP, Servlet, Tomcat을 연도별로 기술 발달

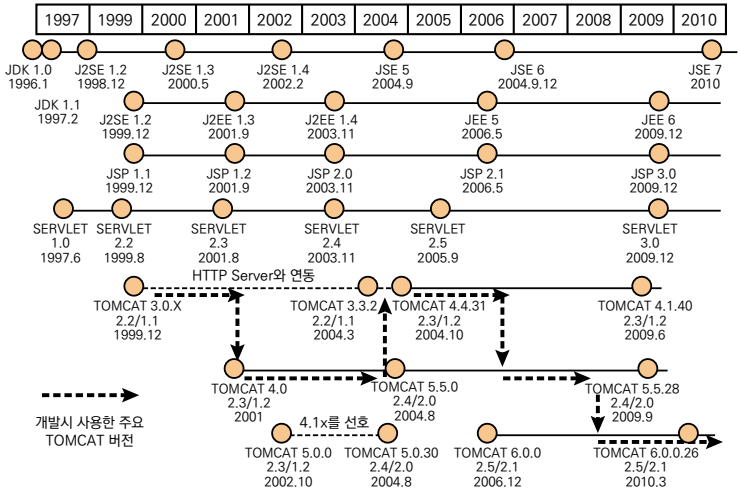


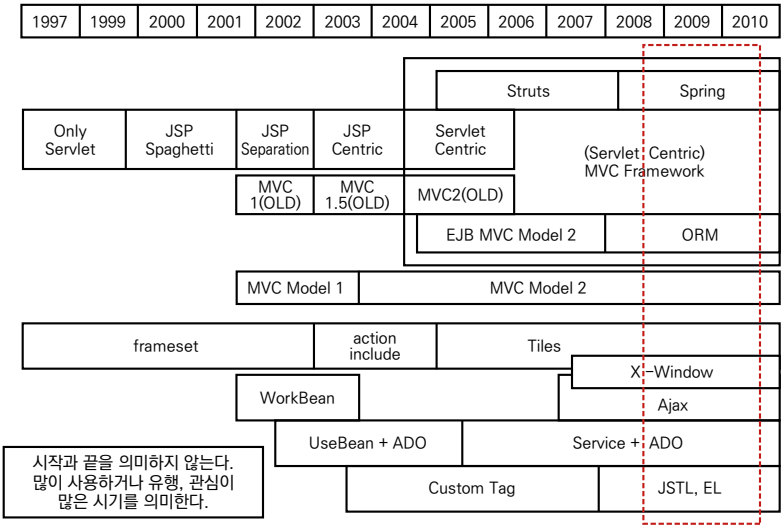
그림 2-3은 JSE, JEE, JSP, Servlet, Tomcat을 버전별, 연도별로 표시한 것이다. Servlet/JSP 버전(JEE 버전)이 먼저 나오고 그 버전을 구현한 Tomcat이 계속해서 발전된 점을 알 수 있다. 일반적으로 개발 시 최신 버전을 사용하지 않는다. 버그나 문제점이 거의 제거된 안전한stable 버전을 사용한다.

2.1.1 연도별 개발 유형

그림 2-4는 연도별 개발 유형을 간단하게 정리한 것이다. 연도별 표시는 시작이나 끝이 아니라 많이 사용하거나 유행, 관심이 많은 시기를 가리킨다. 예를 들어 2008년~2010년의 주요 개발기술은 다음과 같다.

- MVC Model 2 형태
- Servlet을 Controller로 사용하는 Servlet Centric 타입의 프레임워크를 사용
- Spring 프레임워크 사용
- 화면 모듈Module화는 Tiles 프레임워크을 사용
- ORM(iBatis, Hibernate, JPA) + Service + Dao를 사용
- JSTL+EL을 이용하여 화면을 출력
- 비동기 요청처리를 위해 AJAX사용
- 화려하거나 기능이 많은 화면출력용
- X-Window 계열(Flex, MiPlatform)을 사용
- Spring+ORM 를 이용한 CBD 개발

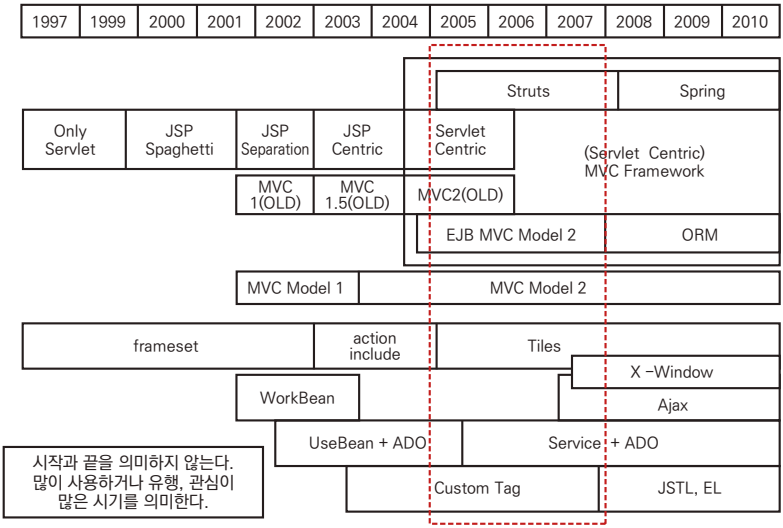
그림 2-4 2008년~2010년의 주요 개발 기술



다음은 2005년~2007년 주요 개발 기술이다.

- MVC Model 2 형태
- Servlet을 Controller로 사용하는 Servlet Centric 타입의 프레임워크를 사용
- Struts 프레임워크 사용
- 화면 Module화는 Tiles 프레임워크을 사용
- Struts + EJB + Service(EJB를 숨기고 Service를 호출)를 사용
- Struts 태그나 Custom 태그를 사용
- XHTML과 JS를 많이 사용
- Struts + EJB 를 이용한 CBD 개발

그림 2-5 2005 ~ 2007년 주요 개발 기술이다



이 표 역시 유행이나 관심 시기에 중점을 두고 표기했다. JSTL을 사용하면서도 Custom Tag를 사용한다. JSP가 controller라고 해서 모두 MVC Model 1은 아니다. Frameset이나 Action include도 사용하고 있다. 단지 Tiles와 같은 모듈화용 프레임워크를 많이 사용하는 추세인 것을 의미한다.

그림 2-6 JSP/Servlet 구성

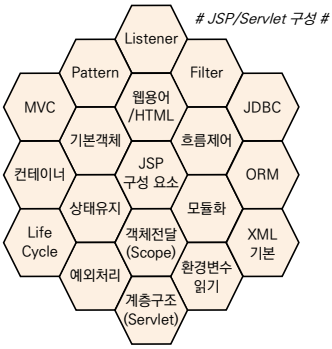


그림 2-6은 Servlet/JSP를 구성하는 주요 요소를 정리한 것이다. 많이 알수록 개발을 쉽고 정확하게 할 수 있다.

1. 웹 용어/HTML
2. JSP 구성요소
3. 기본 객체, 흐름제어, 객체전달Scope
4. 모듈화, 상태유지
5. JDBC

위의 5가지를 알면 웬만한 개발을 할 수 있다. 그렇지만 성능을 좋게 하거나, 빠르고 정확하게 개발하거나, 큰 프로젝트를 수행하려면 다음의 5가지도 확실하게 알아야 한다.

6. 컨테이너 구조와 작동원리, 라이프싸이클Life Cycle
7. 계층구조
8. 환경읽기, XML 기본
9. MVC
10. 간단한 패턴

2.2 설치와 환경설정

2.2.1 JDK 7.X 설치와 환경설정

Java 프로그래밍 환경을 구현하기 위해 JDK1.6을 다운로드하고 설치하자.

1. 웹 브라우저를 실행하고, 주소창에 다음과 같이 입력한다.

    - 웹 주소: http://www.oracle.com/technetwork/java/index.html

2. 메뉴바에서 [DOWNLOADS]-[Java for Developers]를 선택한다.

3. Java의 [DOWNLOAD] 버튼을 클릭한다.

4. [Accept License Agreement]를 체크한 후, 자신의 PC에서 사용하는 운영체제에 맞는 JDK를 다운로드한다.

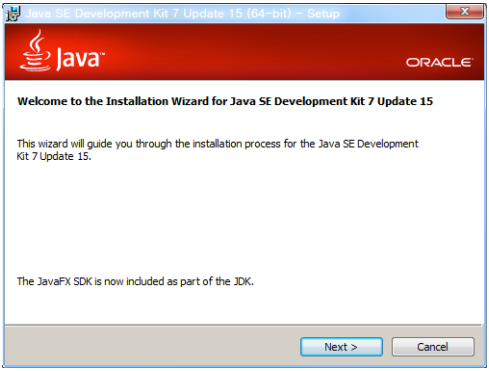
Java SE Development Kit 7u15

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

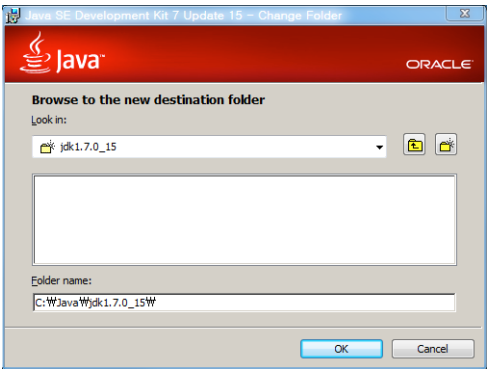
Product / File Description	File Size	Download
Linux x86	106.64 MB	jdk-7u15-linux-i586.rpm
Linux x86	92.97 MB	jdk-7u15-linux-i586.tar.gz
Linux x64	104.77 MB	jdk-7u15-linux-x64.rpm
Linux x64	91.68 MB	jdk-7u15-linux-x64.tar.gz
Mac OS X x64	143.75 MB	jdk-7u15-macosx-x64.dmg
Solaris x86 (SVR4 package)	135.52 MB	jdk-7u15-solaris-i586.tar.Z
Solaris x86	91.94 MB	jdk-7u15-solaris-i586.tar.gz
Solaris SPARC (SVR4 package)	135.92 MB	jdk-7u15-solaris-sparc.tar.Z
Solaris SPARC	95.26 MB	jdk-7u15-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	22.92 MB	jdk-7u15-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.59 MB	jdk-7u15-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	22.53 MB	jdk-7u15-solaris-x64.tar.Z
Solaris x64	14.96 MB	jdk-7u15-solaris-x64.tar.gz
Windows x86	88.75 MB	jdk-7u15-windows-i586.exe
Windows x64	90.4 MB	jdk-7u15-windows-x64.exe

5. 다운로드한 파일을 실행한다.

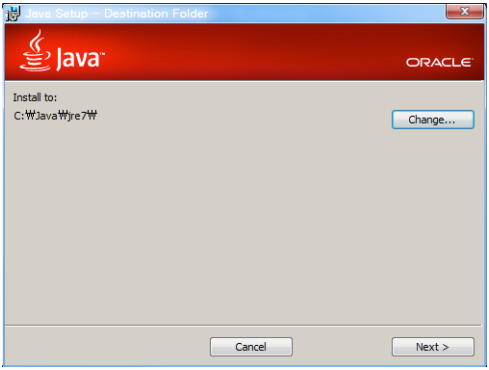


6. 다음과 같은 화면이 나타나면, [Change] 버튼을 선택하여, 설치하기 원하는 위치를 지정한다.

필자는 "C\Java"에 설치할 것이다.



7. JDK가 설치되면, 다음과 같이 JRE를 설치 관련 창이 보여질 것이다. [Change] 버튼을 선택하여 원하는 위치에 저장한다. 필자는 "C:\Java\jre"에 설치할 것이다.

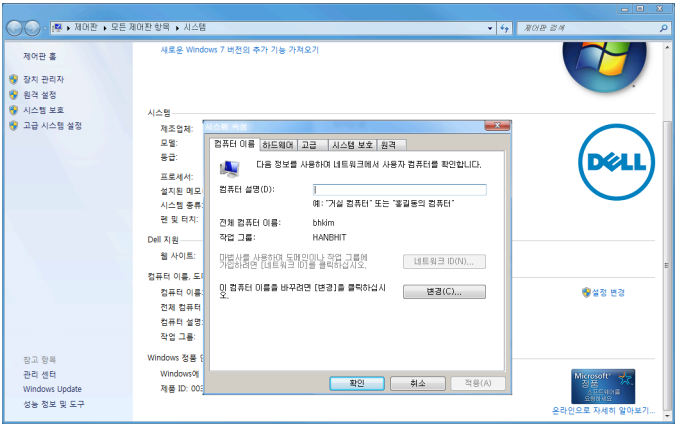


8. 다음과 같은 창이 나타나면 JDK 설치가 끝난 것이다.

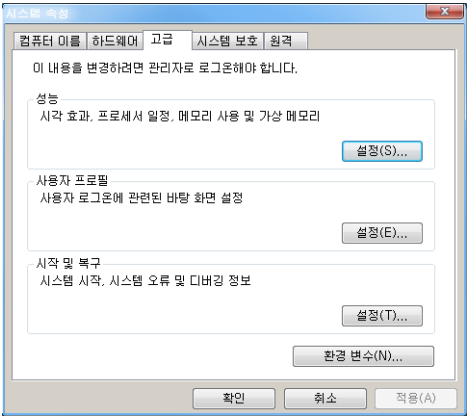
이제 환경변수를 설정하고, 설치가 제대로 되었는지 확인해보자.

9. [시작]-[제어판]-[시스템]을 선택한다(또는 [시작]-[컴퓨터]에서 마우스 오른쪽 버튼을 클릭하여 [속성]을 선택한다).

10. '제어판 홈' 창에서 [설정 변경]을 클릭하면 시스템 창이 나타난다.



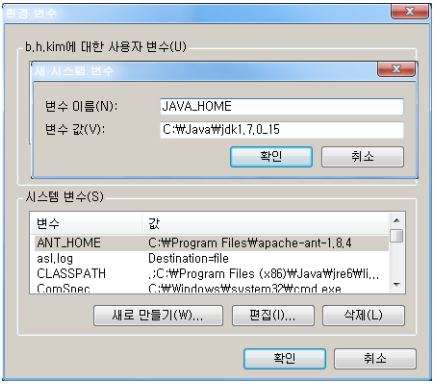
11. 시스템 창에서 [고급] 탭에 있는 [환경 변수]를 클릭한다.



12. 시스템 변수에서 [새로 만들기] 버튼을 선택하여 환경변수를 입력한다. 새 시스템 변수에 '변수이름'과 '변수값'을 다음과 같이 입력한다.

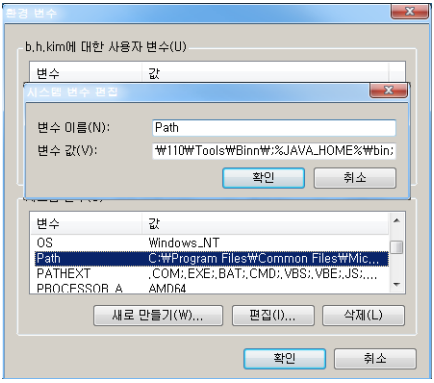
- 변수 이름: JAVA_HOME

- 변수값: C:\Java\jdk1.7.0_15



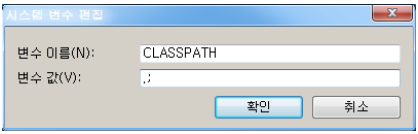
13. JAVA_HOME 변수를 적용하기 위해 시스템 변수를 클릭하고 [편집] 버튼을 눌러, 'Path' 값에 다음을 추가한다.

- 변수 이름: Path
- 변수값: %JAVA_HOME%\bin;



14. Java Library를 사용하기 위한 [새로 만들기] 버튼을 클릭한 다음, CLASSPATH를 추가한다.

- 변수이름: CLASSPATH
- 변수값: .;



15. 환경변수가 정확하게 설정되었는지 확인하기 위해 명령프롬프트를 실행한다. [모든 프로그램]-[시작]-[보조프로그램]-[명령 프롬프트] 또는 [시작]-[실행]에서 'cmd' 명령어를 입력한다.

16. 지금까지 설치한 자바 버전이 맞는지 확인한다.

```
C:\>java -version  // 입력 후 Enter
```

17. 환경설정이 이상 없이 잘 적용되었는지 확인한다.

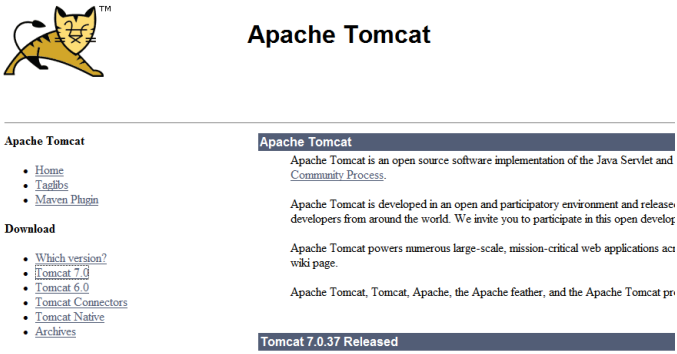
```
C:\>javap java.lang.Object   // 입력 후 Enter
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:₩Users₩b.h.kim>java -version
java version "1.7.0_15"
Java(TM) SE Runtime Environment (build 1.7.0_15-b03)
Java HotSpot(TM) Client VM (build 23.7-b01, mixed mode, sharing)

C:₩Users₩b.h.kim>javap java.lang.Object
Compiled from "Object.java"
public class java.lang.Object {
  public java.lang.Object();
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  public boolean equals(java.lang.Object);
  protected native java.lang.Object clone() throws java.lang.CloneNotSupportedException;
  public java.lang.String toString();
  public final native void notify();
  public final native void notifyAll();
  public final native void wait(long) throws java.lang.InterruptedException;
  public final void wait(long, int) throws java.lang.InterruptedException;
  public final void wait() throws java.lang.InterruptedException;
  protected void finalize() throws java.lang.Throwable;
  static {};
}

C:₩Users₩b.h.kim>
```

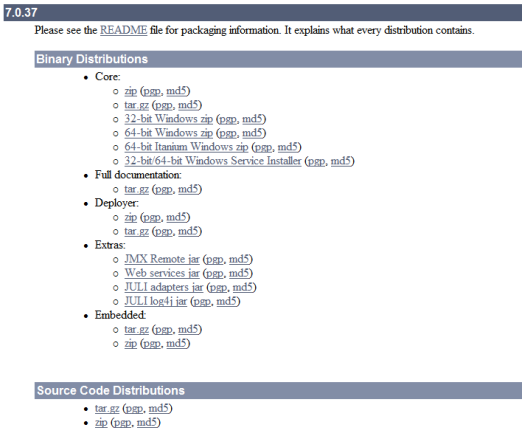
2.2.2 Tomcat 7.X 설치와 환경설정

JSP/Servlet 실행을 위한 웹 서버 중, 오픈 소스면서 가장 많이 사용하는 Apache Tomcat을 설치한다.

1. Apache Tomcat 웹 사이트(http://tomcat.apache.org/)로 이동한 다음, [Download] 항목에 있는 [Tomcat 7.X]를 클릭한다.

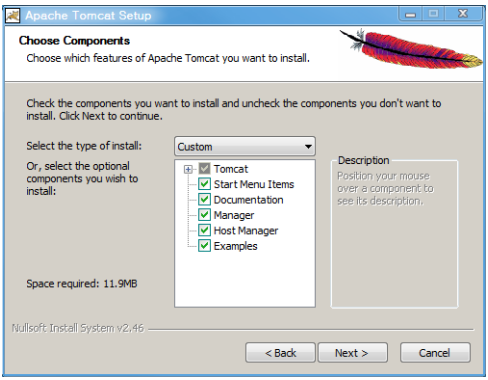


2. [Binary Distributions]-[Core]에 있는 리스트 중 [32-bit/64-bit Windows Service Installer]를 클릭하여 실행 파일을 다운받는다.



3. 받은 파일을 실행한다. 설치창이 나타나면 [Next] 버튼을 클릭한다. 그러면 라이센스창이 보여지는데, [I Agree] 버튼을 선택하여 라이센스를 얻는다.

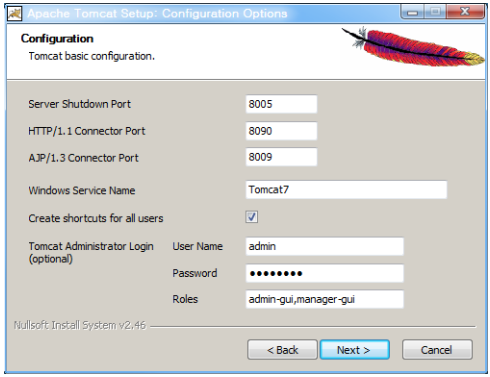
4. 'Choose Comprenets' 창이 나타나면, 옵션을 모두 선택한 다음 [Next] 버튼을 클릭한다.



5. 'Configuration' 창이 나타나면, 다음과 같이 옵션을 설정한다.

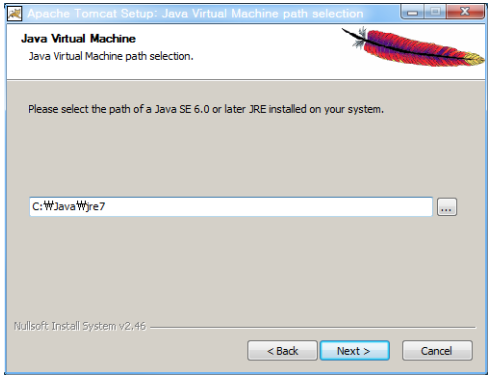
- HTTP/1.1 Connector Port: 8080
- User Name: admin

- Password: 11111111



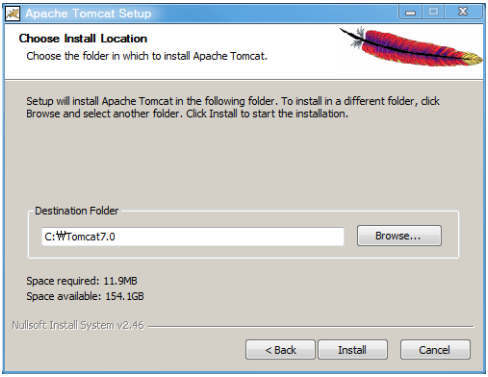
6. 'Java Virtual Machine' 창이 나타나면, JRE 위치를 선택한다.

- C:\Java\jre7

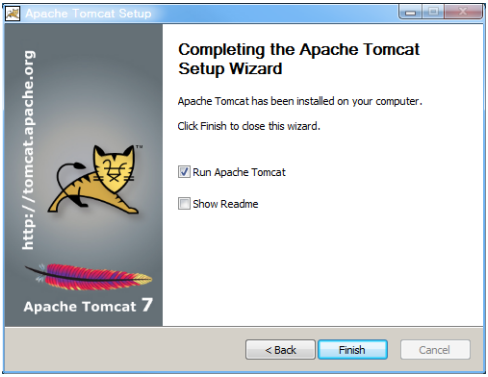


7. 'Choose Install Location' 창이 나타나면, Apache Tomcat을 설치할 폴더를 선택한다. 폴더 이름에 공백이 들어가지 않게 변경해야 한다. 설정이 끝나면 [Install] 버튼을 클릭하여 설치를 진행한다.

● Destination Folder: C:\Tomcat7.0



8. Apache Tomcat 7.0 설치완료 창이 나타나면 Run Apache Tomcat을 체크한 다음 [Finish]버튼을 눌러 Apache Tomcat Server를 테스트한다.



9. Apache Tomcat 7.0 Server가 실행된다. Apache Tomcat 7.0이 실행되는지 확인하려면 윈도우 작업표시줄 맨 오른쪽의 아이콘( )에 녹색 또는 빨간색이 표시되는지 살펴본다.

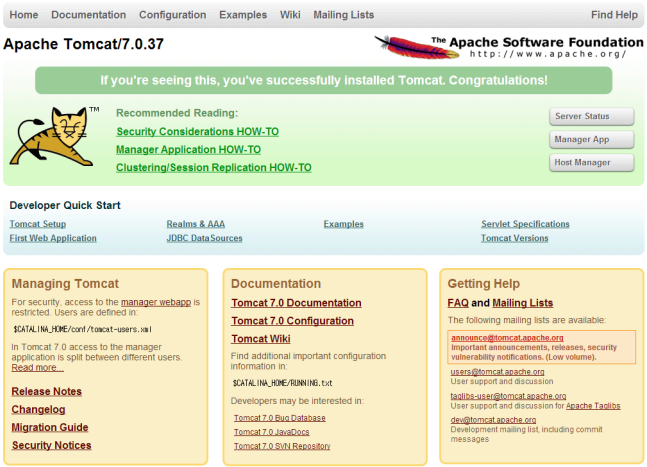
● 녹색: Apache Tomcat 7.0 Server 실행 상태

- 빨간색: Apache Tomcat 7.0 Server 중지 상태

10. 마지막으로 Apache Tomcat 7.0 Server가 정상적으로 설치되었는지 확인한다.

    [웹 브라우저 실행] – [http://localhost:8090 주소 입력]

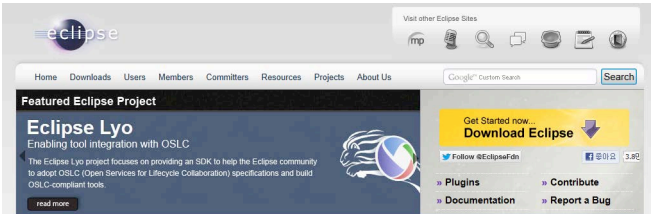
    다음과 같이 고양이 화면이 보이면, Apache Tomcat Server가 제대로 설치된 것이다.



2.2.3 이클립스 4.X 설치와 환경설정

JSP/Servlet 기술이 적용된 Web Application 개발에 필요한 프로그램인 이클립스를 다운받아 설치한다.

1. 이클립스 홈 페이지(http://www.eclipse.org)에 접속한 후, [Downloads]를 클릭한다.



2. 사용하는 운영체제가 32bit인지 64bit인지 확인한 다음 이클립스를 내려받자. 여기서는 일반적으로 많이 사용하는 32bit 기준으로 진행한다.

   • Eclipse IDE for Java EE Developers. 221MB [Windows 32 Bit]를 선택한다.

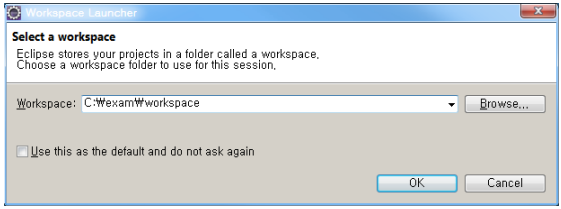
3. 다음과 같은 화면이 나타나면 녹색 화살표를 클릭한다.

4. 압축되어 있는 이클립스 파일을 사용할 위치에 푼다. 필자는 다음 위치에 압축을 풀었다.

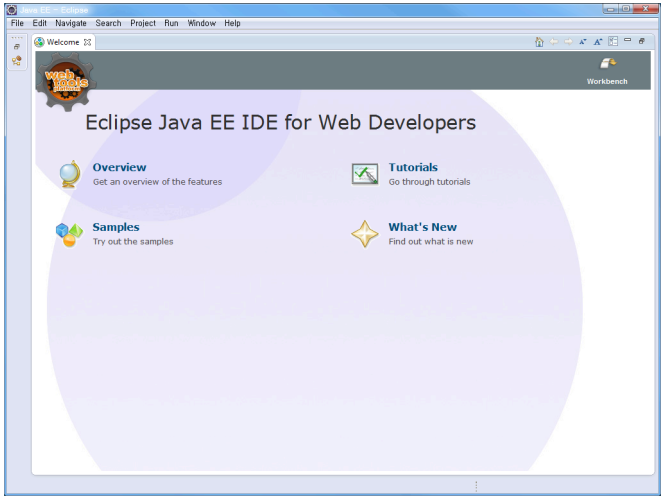
   • 이클립스 파일 압축해제 위치: C:\Java\

5. 이클립스가 설치된 폴더(C:\Java\eclipse)에서 'eclipse.exe'를 실행한다. 이클립스를 실행하면 workspace를 입력하라는 화면이 나타난다. 프로그램을 작성할 위치를 입력한 다음 [OK] 버튼을 클릭한다.

   • Workspace : C:\exam\workspace



6. 다음과 같이 화면이 나오면 이클립스가 정상적으로 설치된 것이다.

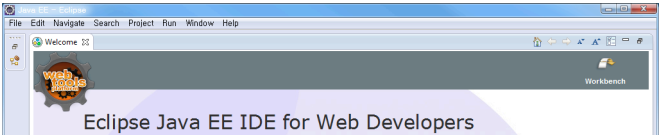


2.2.4 Tomcat과 이클립스 연동

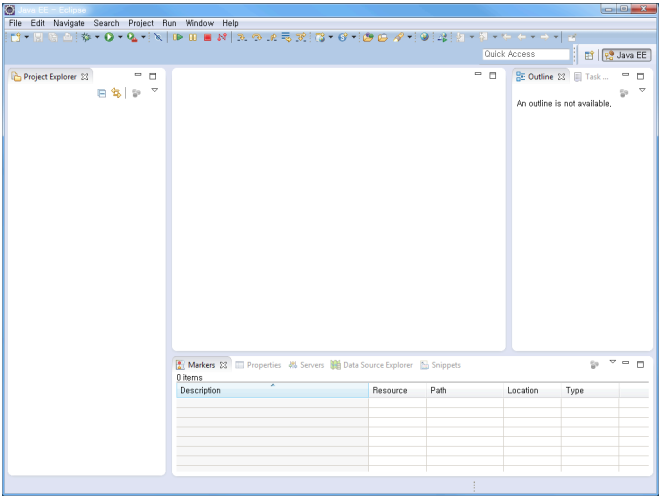
Apache Tomcat을 이클립스에 적용하여 사용하는 방법을 알아보자. 이클립스를 실행하고 Web Application 구현 시 개발할 작업공간workspace을 지정한다.

- Workspace: C:\exam\workspace

1. 처음 실행하면 Welcome tab이 생성되고 다음과 같은 화면이 나타난다. Welcome 옆의 [X] 버튼을 클릭하여 화면을 닫는다.

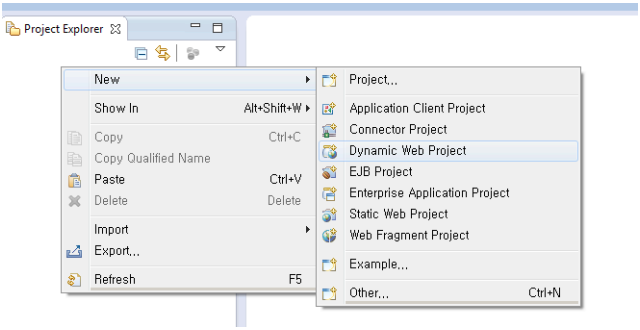


2. 다음과 같은 화면이 나오면 Web Application 적용 환경이 설정된 것이다.



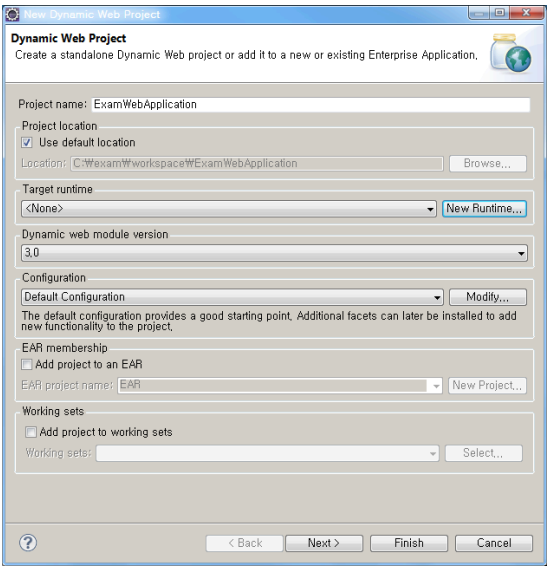
3. [Project Explorer] 영역에서 마우스 오른쪽 버튼을 클릭한 후, [New]-[Dynamic

Web Project]를 선택한다.



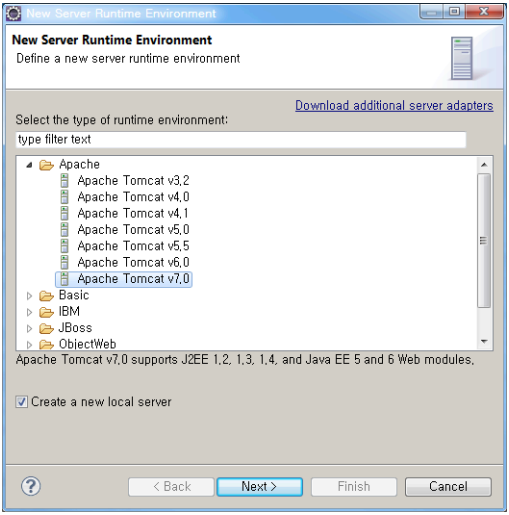
4. 'Dynamic Web Project' 창이 나타나면, ExamWebApplication 프로젝트를 구성하기 위한 환경을 설정한다. 그리고 [New Runtime] 버튼을 클릭하여 Apache Tomcat 7.0을 적용한다.

- Project name: ExamWebApplication



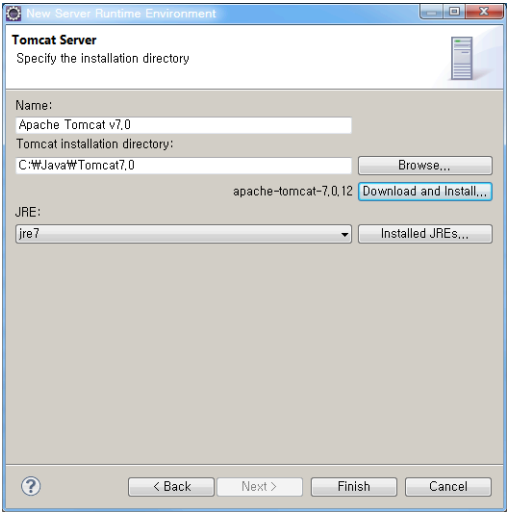
5. 설치된 Apache Tomcat 버전을 선택한 후, [NEXT] 버튼을 클릭한다.

- Apache Tomcat v7.0 선택
- Create a new local server 체크

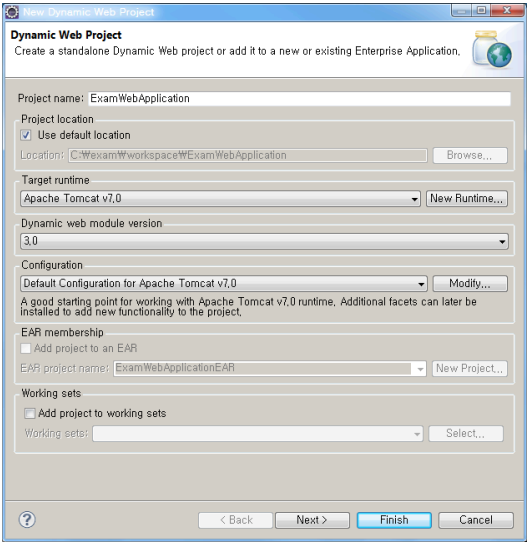


6. Apache Tomcat v7.0의 설치 경로를 지정하고 Apache Tomcat 설치 디렉터리를 설정한다. 그런 다음 [Finish] 버튼을 클릭하여 Apache Tomcat 적용을 마친다.

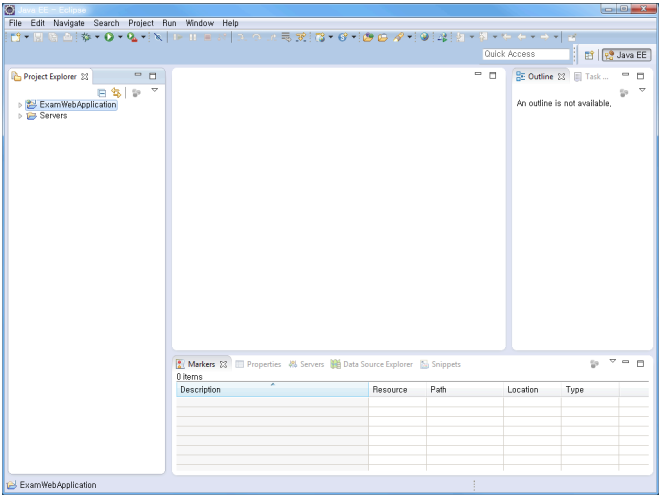
- Tomcat installation directory: C:\Java\Tomcat7.0
- JRE: jdk1.7_15 선택



7. 다음 그림과 같이 환경이 설정되었는지 확인한 후 [Finish]버튼을 선택하여 웹 애플리케이션을 생성한다.

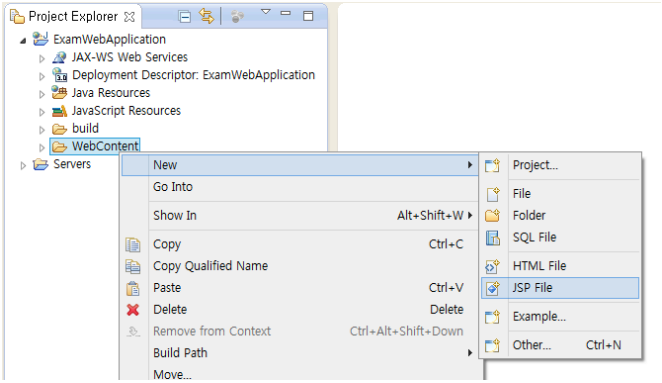


8. 다음 그림과 같이 생성되었다면 웹 애플리케이션이 정상적으로 만들어진 것이다.

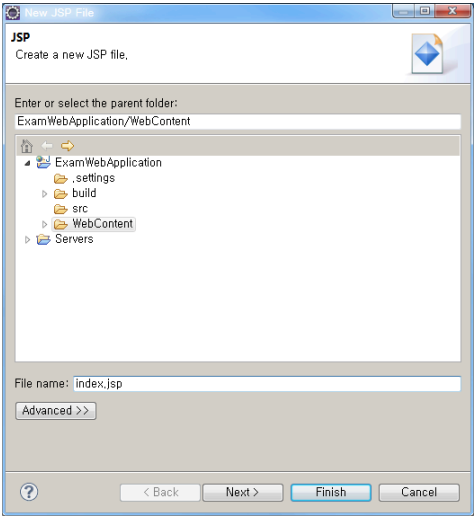


9. 이제 만들어진 웹 애플리케이션을 실행하기 위한, index.jsp 페이지를 생성한다.

- [ExamWebApplication]-[WebContent]에서 마우스 오른쪽 버튼 클릭 한 후 [New]-[JSP file]을 선택한다.



10. File name에 'index.jsp'를 입력하여 jsp파일을 생성한 후, [Finish] 버튼을 클릭하여 파일 생성을 완료한다.



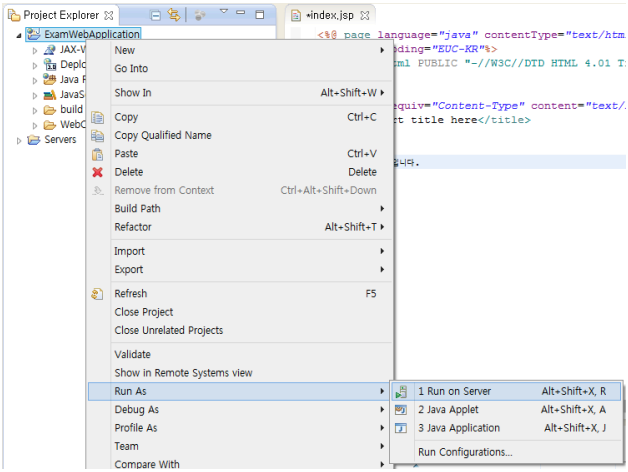
11. 생성된 파일의 〈body〉와 〈/body〉 사이에 다음 내용을 입력하고 저장한다.

- 입력할 내용: JSP 페이지 테스트입니다.

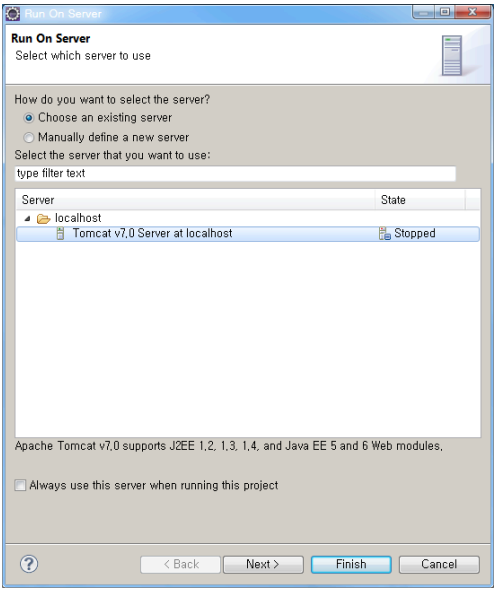
```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
JSP 페이지 테스트입니다.
</body>
</html>
```

12. 생성된 웹 애플리케이션을 실행해본다.

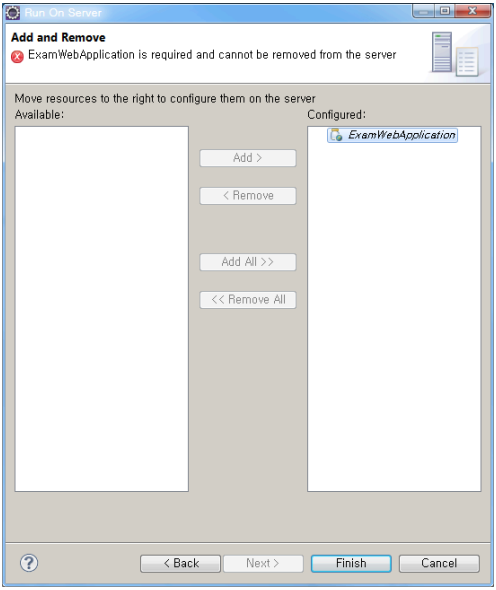
- [ExamWebApplication]에서 마우스 오른쪽 버튼을 클릭한 후, [Run As]-[Run on Server] 선택한다.



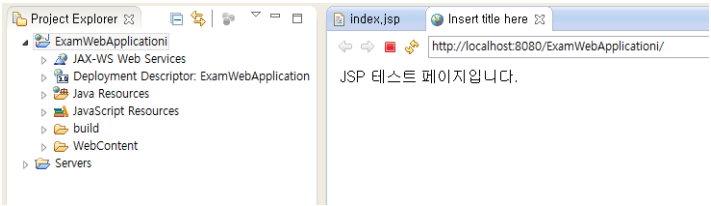
13. 실행할 웹 애플리케이션 선택한다. [Next]버튼을 클릭한다.



14. Configured 영역에 있는 적용할 project를 선택한 다음 [Finish] 버튼을 눌러 실행한다.



15. 다음과 같은 결과가 보인다면 이클립스에 Apache Tomcat이 정상적으로 설정된 것이다.



NOTE_ 실행 시, 에러가 발생하면, Tomcat 서버가 실행되고 있는지 살펴본다(실행 창에 'services.msc'를 입력하면 서비스관리 창이 나타난다. Apache Tomcat 속성으로 들어가서 서비스가 실행되고 있는지 확인한다). 서비스가 시작 상태라면 중지 상태로 만들고 [시작 유형]을 '수동'으로 설정한다.

2.3 이클립스 활용방법

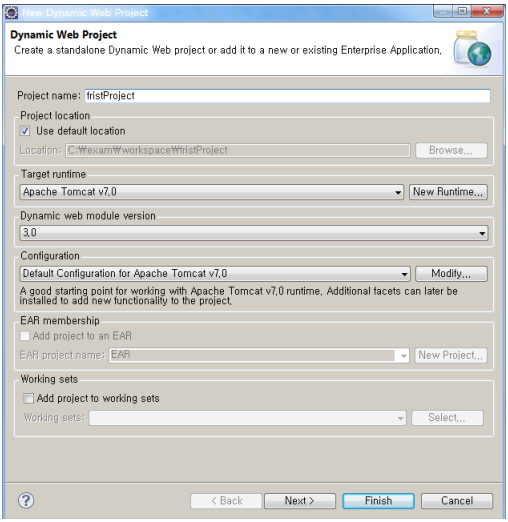
2.3.1 JSP 작성하기

1. JSP 작성용 웹 프로젝트를 만든다.

   • [File]-[New]-[Dynamic Web Project]

2. 웹 프로젝트의 이름을 입력한다. 실행환경(Target runtime), 버전(Dynamic web module version), 컨테이너 환경(Configuration)을 확인하고 [Finish] 버튼을 누른다.

   • Project name: fristProject

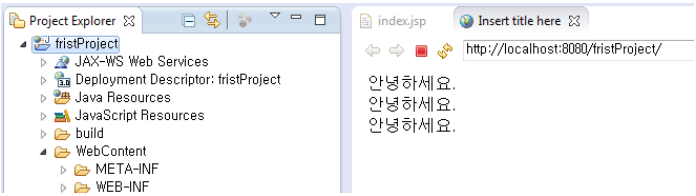


3. JSP 파일을 만든다. [프로젝트]에서 오른쪽 마우스를 이용하여 [New]-[JSP]를 선택한다. 파일 이름을 “index.jsp”로 하고 〈body〉와 〈/body〉 사이에 다음과 같이 추가한다.

```
<%@ page contentType="text/html; charset=EUC-KR"%>
<html>
<head>
 <title>first</title>
</head>
<body>
 <%
 for(int i=0; i<3 ;i++){
    out.println("안녕하세요.<br/>");
 }
 %>
</body>
</html>
```

5. 프로젝트를 실행하면, 다음과 같은 실행결과 화면이 나타날 것이다

- [프로젝트]에서 오른쪽 마우스를 이용하여 [Run AS]-[Run on Server]를 선택

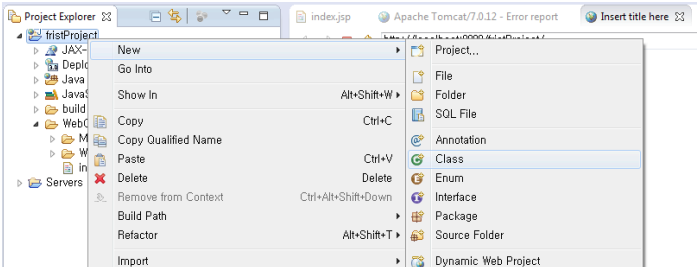


2.3.2 JSP의 useBean 액션 태그 만들기

1. cust.jsp를 만들고 id, name, height를 입력할 수 있는 〈input〉 태그를 만든다.

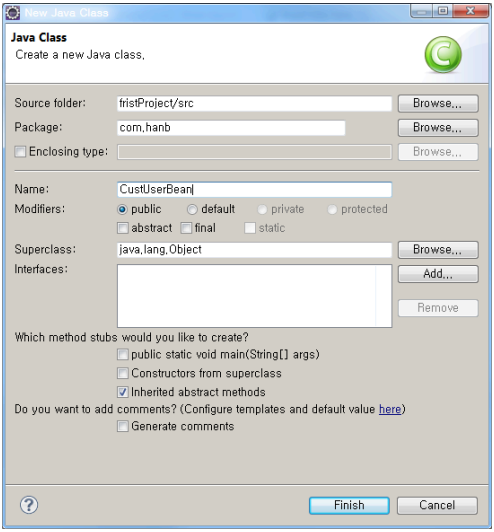
```
<%@ page contentType="text/html; charset=EUC-KR"%>
<html>
<head>
  <title>useBean</title>
</head>
<body>
  <form action="custshow.jsp" method='post'>
  <table border="1" bgcolor="#ffeeee">
    <col width="200"/><col width="400"/>
    <tr>
      <td>아이디</td>
      <td><input type='text' name='id' size="60"></td>
    </tr>
    <tr>
      <td>이   름</td>
      <td><input type='text' name='name' size="60"></td>
    </tr>
    <tr>
      <td>키</td>
      <td><input type='text' name='height' size="60"></td>
    </tr>
    <tr>
      <td colspan="2"><input type='submit' value="보내기"></td>
    </tr>
  </table>
  </form>
  <a href='index.jsp'>HOME</a>
</body>
</html>
```

2. CustomerBean 클래스를 만든다.



패키지[Package]와 클래스 이름(Name)을 입력하고 [Finish] 버튼을 클릭한다.

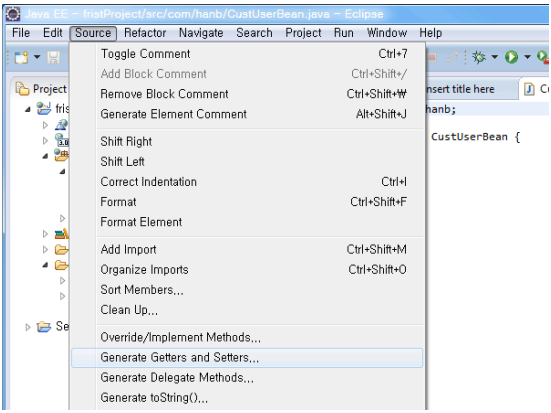
- Package: com.hanb
- Name: CustUserBean



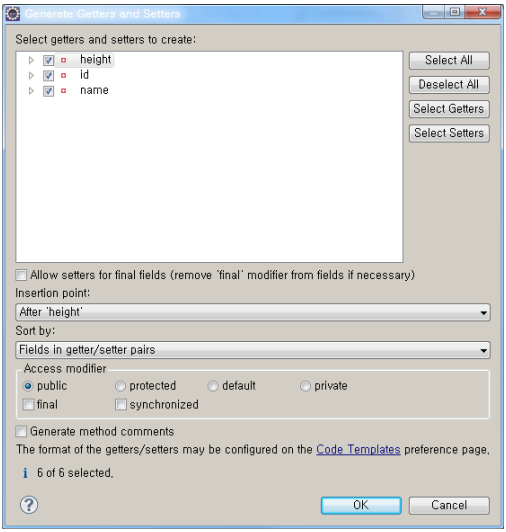
3. 멤버필드에 id, name, height을 입력한다.

```
package com.hanb;
public class CustUserBean {
   private String id;
   private String name;
   private int height;
}
```

4. 생성자와 get/set 메서드를 만든다. [Source]-[Generate Getters and Setters]를 클릭한다.



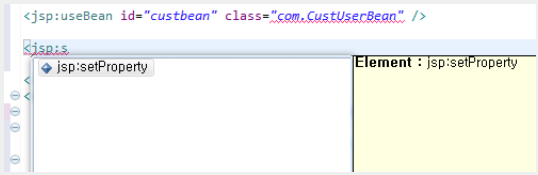
'Generate and Settera' 창에서 [Select All] 버튼을 클릭한 후 [OK] 버튼을 클릭한다.



5. custshow.jsp를 만들고 〈jsp:useBean〉 태그를 입력한다. class=""에 패키지까지만 입력해도 들어가야 할 클래스를 찾을 수 있다. 다음과 같이 입력한다

〈jsp:useBean id="custbean" class="com.CustUserBean" /〉

NOTE_ 〈jsp:s에서 기다리거나 [CTRL+SPACE BAR]를 누르면 다음 그림처럼 〈jsp:setProperty〉태그를 선택할 수 있다.



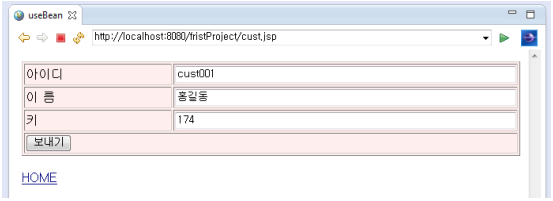
6. ‘custshow.jsp’에서 id, name, height 파라미터를 useBean 액션 태그로 받는다. 받을 때는 setProperty를 이용하고 출력할 때는 getProperty를 이용한다.

```
<%@ page contentType="text/html; charset=EUC-KR"%>
<% request.setCharacterEncoding("euc-kr"); %>
<html>
<head>
  <title>useBean</title>
</head>
  <jsp:useBean id="custbean" class="com.hanb.CustUserBean"/>
  <jsp:setProperty name="custbean" property="id" />
  <jsp:setProperty name="custbean" property="name" />
  <jsp:setProperty name="custbean" property="height" />

<body>
  <table border="1" bgcolor="#ffeeee">
  <col width="200"/><col width="400"/>
    <tr>
      <td>아이디</td>
      <td><jsp:getProperty name="custbean" property="id" /></td>
    </tr>
    <tr>
      <td>이   름</td>
      <td><jsp:getProperty name="custbean" property="name" /></td>
    </tr>
    <tr>
      <td>키</td>
      <td><jsp:getProperty name="custbean" property="height" /></td>
    </tr>
  </table>
  <a href='cust.jsp'>입력하기</a>
  </body>
```

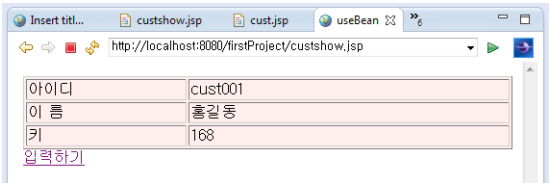
```
</html>
```

7. 프로젝트를 실행한 후 id, name, height를 입력하고 custshow.jsp로 전송한다.

   - “http://localhost:8080/fristProject/cust.jsp” 주소가 맞는지 확인한다.



id, name, height 파라미터를 setProperty로 받아서 getProperty로 출력한다.



3 | 웹 개발 환경

3.1 컨텍스트

컨텍스트Context는 웹 프로그래밍을 위한 물리적인 디렉터리로 실행단위, 배포단위(디렉터리)다. 웹 프로그래밍을 하면서, 정해진 컨텍스트 구조(그림 3-1)를 지키지 않으면 수많은 오류를 만나게 된다. 우리가 사용하는 이클립스 웹 개발환경은 컨텍스트와 비슷한 구조로 되어있다(그림 3-2). 즉, 개발할 때는 이클립스 개발환경 구조, 배포되어 실행될 때는 컨텍스트 구조가 된다.

그림 3-1 컨텍스트 구조

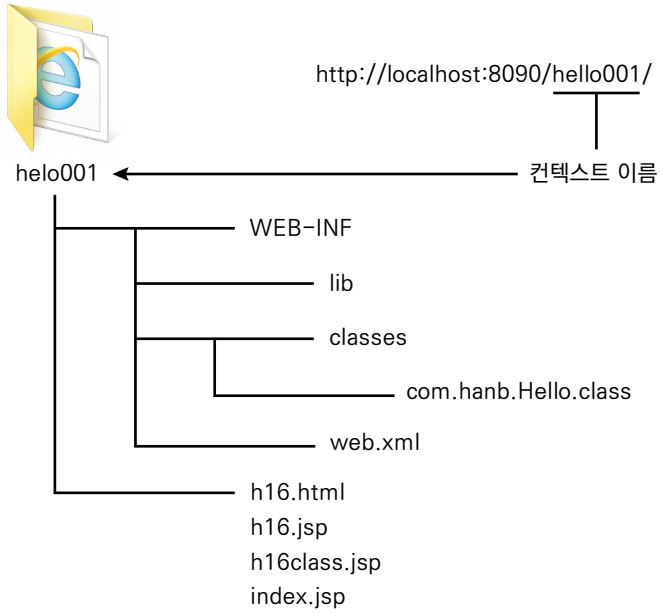
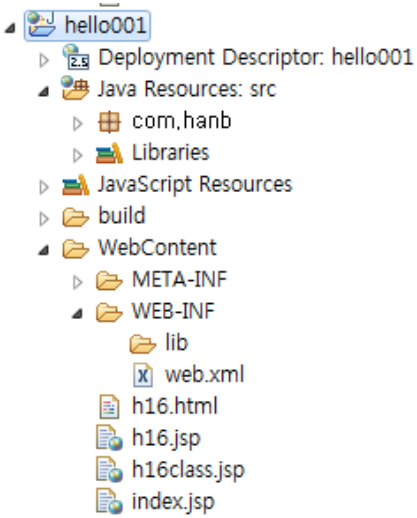


그림 3-2 이클립스 웹 개발환경



컨텍스트 hello001에 있는 JSP, 서블릿, 클래스, 라이브러리는 다음과 같은 구조로 되어 있어야 한다. 그렇지 않으면 실행할 수가 없다.

hello001 : 컨텍스트

+ HTML, JSP, Image를 놓는다.

+ WEB-INF 디렉터리

+ lib : ~.jar로 끝나는 라이브러리를 포함함

  라이브러리는 클래스의 묶음임

+ classes : 패키지 컴파일된 클래스(~.class). 서블릿(Servlet)

  웹 클래스는 반드시 패키지가 있어야 함

+ web.xml : 환경변수 설정을 위한 XMl 파일

web.xml은 컨텍스트 환경을 설명하는 배포설명자(DD[Deployment Descriptor])로, 실행될 때 컨테이너가 가장 먼저 읽는다.

코드 3-1 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>hello001</display-name>
   <welcome-file-list>
     <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

"http://localhost:8090/hello001"를 실행하면 "http://localhost:8090/hello001/index.jsp"로 실행된다. 여기서 hello001은 컨텍스트의 이름이다. 컨테이너는 이름변경, 경로변경, 환경변수 설정 등 실행환경이 규정된 web.xml을 가장 먼저 읽는다.

3.2 경로

request.getXXX() 메서드를 이용하면 경로 관련 정보를 구체적으로 얻을 수 있다. 다음 웹 브라우저 경로에 대한 상세 정보는 표 3-1과 같다.

- 웹 브라우저 경로: http://localhost:8090/headers/requestinfor.jsp?id=cust001&name=jack

표 3-1 웹 브라우저 경로에 대한 상세 정보

request 메서드	요청내용	상세정보
getRequestURL()	요청 URL	http://localhost:8090/headers/requestinfor.jsp
getRequestURI()	요청 URI	/headers/requestinfor.jsp
getServerName()	서버이름	localhost
getServerPort()	서버포트	8090
getScheme()	프로토콜	http
getContextPath()	/컨텍스트이름	/headers
getServletPath()	/JSP 파일 이름	/requestinfor.jsp
getQueryString()	파라미터	id=cust001&name=jack

3.3 정적문서와 동적문서

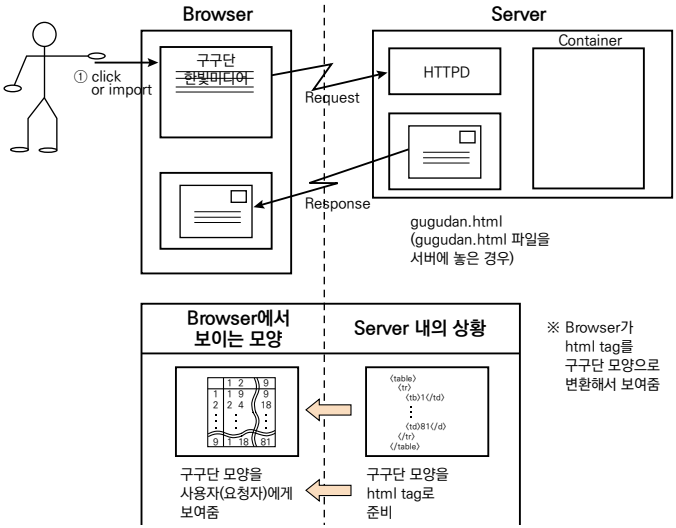
실행 결과만 보면 사용자는 html과 jsp의 차이를 알 수 없다. 간단한 개념차이부터 짚고 넘어가자.

확장자가 html이나 htm으로 끝나는 문서는 정적문서다. 스크립트(자바스크립트, VB스크립트, 액션스크립트)는 클라이언트사이드 동적문서로 웹 브라우저에서 실행된다. 서버사이드 문서는 동적문서로, 확장자가 jsp(또는 class)로 끝난다. JSP/Servlet 웹 프로그래밍은 서버사이드 동적문서로 서버와 컨테이너가 반드시 필요하다.

인터넷으로 하나의 문서를 여러 사람에게 보여주려면, 서버가 필요하다. 클라이언트사이드 문서는 서버가 없어도 실행하거나 볼 수 있다. 웹 브라우저가 HTML을 정해진 형태로 변환하여 보여주는데, 〈table〉 태그가 있으면 테이블 모양으로 변환해주는 약속(프로토콜)이 있기 때문이다. 반면 서버사이드 문서(jsp/servlet)는 서버와 jsp/servlet을 실행해서 html로 변환해주는 컨테이너가 반드시 필요하다.

gugudan.html이 있는 서버에 사용자가 구구단을 요청request하면, 서버는 html로 만든 구구단을 사용자에게 그대로 보내(response, 또는 '응답'이라고 한다)준다. 서버가 보내준 html이 웹 브라우저와 만나 구구단 표가 된다.

그림 3-3 서버사이드 동작 흐름도



gugudan.jsp를 서버의 컨텍스트에 갖다 놓으면, gugudan.jsp는 서버의 컨테이너에 배포된 것이다. 사용자가 gugudan.jsp를 요청하면 gugudan.jsp는 gugudan_jsp.java와 비슷한 형태의 자바클래스가 되고, 이것이 실행되어 구구단을 html로 변경한 후 사용자에게 보내준다.

사용자는 결과만 보았을 때 gugudan.html과 gugudan.jsp의 차이를 알 수 없다. gugudan.html은 구구단 표를 html로 미리 만들어 놓은 것이고(정적), jsp는 프로그래밍에 의해 구구단이 html로 만들어진 것이다(동적). 〈h〉는 글씨크기를 조절하는 html 태그로 가장 큰 〈h1〉부터 가장 작은 〈h6〉까지 있다. 다음 예제를 통해 정적문서와 동적문서의 차이를 좀 더 이해해 보자.

예제 3-2 정적문서 HTML 예제(h16.html)

```
01: <html>
02: <head>
03:   <title>글씨 크기</title>
04: </head>
06: <body>
07:   글씨크기-정적문서 HTML 예제입니다.<hr/>
09:   <h1>안녕하세요. h1 크기입니다.</h1><br/>
10:   <h2>안녕하세요. h2 크기입니다.</h2><br/>
11:   <h3>안녕하세요. h3 크기입니다.</h3><br/>
12:   <h4>안녕하세요. h4 크기입니다.</h4><br/>
13:   <h5>안녕하세요. h5 크기입니다.</h5><br/>
14:   <h6>안녕하세요. h6 크기입니다.</h6><br/>
15:   <a href='index.jsp'>HOME</a>
16: </body>
17: </html>
```

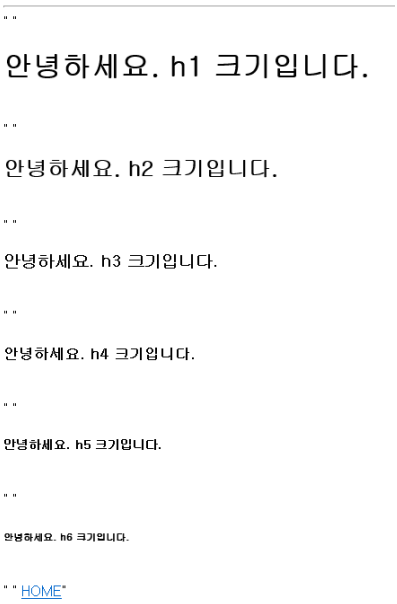
1번째 줄: 〈html〉태그는 〈head〉, 〈body〉 태그로 구성된다. 〈head〉에는 제목과 자바 스크립트가 들어있다. 〈body〉는 응답으로 보여줄 내용을 포함하고 있다.

```
<html>
  <head>
    제목, 자바스크립트
  </head>
  <body>
    중요 내용
  </body>
</html>
```

9~14번째 줄: 〈h1〉이 글씨가 가장 크고 〈h6〉가 가장 작다.

15번째 줄: 'HOME'을 선택하면 index.jsp를 보여준다.

그림 3-4 예제 3-2 실행 결과 화면



예제 3-3 동적문서 JSP 예제(h16.jsp)

```
01: <%@ page contentType="text/html; charset=EUC-KR" pageEncoding="EUC-KR"%>
02: <%--한글이 깨지지 않도록 한다.--%>
03: <% request.setCharacterEncoding("euc-kr"); %>
04: <html>
05: <head>
08:   <title>글씨 크기</title>
09: </head>
```

```
10: <body>
11:   글씨크기-동적문서 JSP 예제입니다.<hr/>
12:   <%
13:     for(int i=1; i<=6 ;i++){
14:     %>
15:       <%="<h"+i+">"%>안녕하세요. h<%=i%> 크기입니다.<%="</h"+i+">"%>
16:       <br/>
17:     <%
18:     }
19:   %>
20:   <a href='index.jsp'>HOME</a>
21: </body>
22: </html>
```

- "html이냐 자바코드가 html이 되느냐"에 중요한 차이가 있다.
- 컨테이너 안에서 자바코드가 html로 변환된 다음 사용자에게 전송된다.
- <%="<h"+1+">"%>는 <h1> html 태그가 된다.

1번째 줄: <%@ page contentType="text/html; charset=EUC-KR" %>에서 @ page는 jsp의 글자타입 등 페이지의 특징을 선언하는 페이지 디렉티브 page directive다.

2번째 줄: <%-- --%>는 프로그래밍에 영향을 주지 않는 jsp 코멘트다.

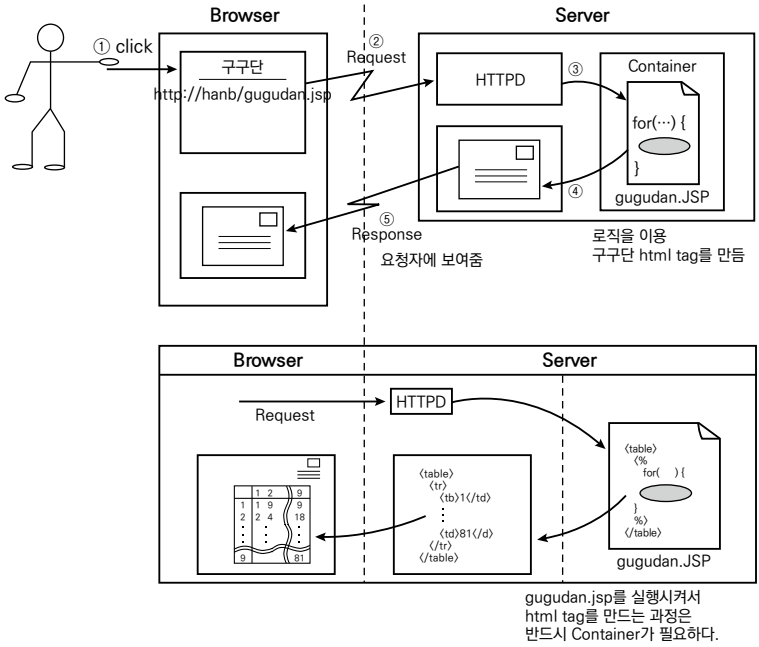
3번째 줄: <% request.setCharacterEncoding("euc-kr"); %>는 jsp에서 한글이 깨지지 않게 설정한 부분이다. 여기까지는 페이지의 특징을 선언하는 부분이다.

12번째 줄: <% %>는 스크립트렛scriptlet(자바코드부분)으로 자바 프로그래밍을 할 수 있는 부분이다. for 반복문을 1에서 6까지, 1씩 올리면서 실행하고

있다. 〈% %〉 사이는 자바를 사용할 수 있는 부분이고 그 외는 html 부분이다.

15번째 줄: 〈%= %〉는 익스프레션expression(또는 표현식)으로, html로 변환한다. 〈%= (1+2)%〉는 3이 된다. html이냐 자바코드가 html이 되느냐에 중요한 차이가 있다. 먼저 jsp가 컨테이너 안에서 자바코드를 실행한 다음, 결과인 html을 사용자에게 보낸다. "〈h"+1+"〉"이 익스프레션에서 〈h1〉이 되므로 "〈h1〉안녕하세요 h1 크기입니다.〈/h1〉〈br/〉"이 되고, i가 증가하여 2가 되면서 "〈h2〉안녕하세요 h2 크기입니다.〈/h2〉〈br/〉"가 된다. 이렇게 〈h6〉까지 반복되어 모두 html로 변환된 다음 사용자에게 전송된다.

그림 3-5 JSP 작동순서



JSP 작동순서를 정리해보자.

①, ② 사용자가 서버에 있는 h6.jsp를 요청(http://ip:port/context/h16.jsp)한다.
③ HTTPD(컨테이너에서 처리해야할 것이 있는 가를 판단하는 역할)가 확장자 jsp를 보고 컨테이너에게 작업을 요청한다.
④ 자바코드를 실행하여 html로 만든다.
⑤ 생성된 html을 사용자에게 보낸다(응답).

JSP의 작동원리를 이해하기 위해 h16class.jsp를 살펴보자. JSP의 페이지 디렉티브 중 import="com.hanb.Hello"는 자바의 import com.hanb.Hello;와 동일하다. com.hanb 패키지의 Hello 클래스를 사용하겠다는 뜻이다. 〈%=hello.hx(i)%〉에서 i=1 일 때 hx(1)가 실행되면 〈%=“〈h1〉안녕하세요. h1 크기입니다. 〈/h1〉〈br/〉”%〉가 된다. 다시 html 문자열 ‘〈h1〉안녕하세요. h1 크기입니다. 〈/h1〉〈br/〉’이 되고, 사용자에게 전달되면 큰 글씨로 ‘안녕하세요. h1 크기입니다.’로 보인다.

예제 3-4 동적문서 JSP/Class 예제(h16class.jsp)

```
01: <%@ page contentType="text/html; charset=EUC-KR"%>
02: <%@ page import="com.hanb.Hello"%>
03: <% request.setCharacterEncoding("euc-kr"); %>
04: <html>
05: <head>
06:   <title>글씨 크기</title></head>
07: <body>
08:   글씨크기-동적문서 JSP/Class 예제입니다.<hr/>
09:   <%
10:     Hello hello=new Hello();
11:     for(int i=1; i<=6 ;i++){
12:     %>
```

```
13:       <%=hello.hx(i)%>
14:     <%
15:     }
16:   %>
17:   <a href='index.jsp'>HOME</a>
18: </body>
19: </html>
```

9번째 줄: 자바코드영역. 스크립트릿

10번째 줄: Hello 객체를 생성한다.

13번째 줄: HTML 영역에서 익스프레션을 이용하여 문자열을 HTML로 출력한다. i가 1일 때 <%=hello.hx(1)%>에서 hx(1) 메서드는 <h1>안녕하세요. h1 크기입니다.</h1>
을 반환한다. 그리고 <%="<h1>안녕하세요. h1 크기입니다.</h1>
"%>은 <h1>안녕하세요. h1 크기입니다.</h1>
을 출력한다. 같은 방법으로 h2~h6까지 반복한다.

예제 3-5 웹 프로그래밍의 클래스 형식 예제(Hello.java)

```
1: package com.hanb;  //웹 프로그래밍->패키지 필수
2: public class Hello {
3:  public String hx(int i){
4:     //String t=String.format("<h%d>안녕하세요. h%d 크기입니다.</h%d>
       <br/>",i,i,i);
5:   String t=String.format("<h%1$d>안녕하세요. h%1$d 크기입니다.</h%1$d>
       <br/>",i);
6:     return t;
7:  }
8: }//class
```

1번째 줄: 웹 프로그래밍에서 클래스는 반드시 패키지 형태로 선언해야 한다. jsp에서 사용할 때는 〈%@ page import="com.hanb.Hello"%〉와 같이 페이지 디렉티브를 이용하여 포함한다.

4번째 줄: String.format()을 이용하여 문자열 출력형식을 결정한다. %d의 개수와 파라미터(i) 개수가 동일하므로 $가 필요하지 않다. i가 1일 때 파라미터 i가 3개이므로 %d대신 1이 대입되어 〈h1〉안녕하세요. h1 크기입니다.〈/h1〉〈br/〉가 된다.

5번째 줄: String.format()을 이용하여 문자열 출력형식을 결정한다. %d는 정수, %s는 문자열을 의미한다. 출력할 파라미터와 출력 형식%의 개수가 다르면 $를 사용한다. $1은 첫 번째 파라미터 i의 위치를 의미한다. i가 1일 때 파라미터 i가 1개 이므로 %1$d대신 1이 대입되어 〈h1〉안녕하세요. h1 크기입니다.〈/h1〉〈br/〉가 된다.

3.4 JSP 스크립트 요소

JSP는 html 태그부분과 스크립트부분으로 구성된다. 스크립트부분은 컨테이너에서 html 태그로 변환된 후 사용자에게 전달된다. 사용자가 보는 결과화면에서는 정적문서인지 혹은 동적문서인지 알 수 없다.

표 3-2 스크립트 요소

표 기	이 름	사용용도
〈%! %〉	선언(declaration)	한 페이지 내에서 사용하려고 선언한 메서드나 필드
〈% %〉	스크립트렛(실행-scriptlet)	자바코드를 실행하는 부분
〈%= %〉	익스프레션(표현-expression)	메서드 실행결과나 연산결과를 문자열로 변환

표 3-3 스크립트 요소 예

표 기	이 름	의 미

〈%! %〉	〈%!
	public String dehx(int i)
	{
	..
	}
	%〉
		dehx() 메서드를 선언한다.

〈% %〉	〈%
	Hello hello=new Hello();
	for(int i=1; i〈=6; i++){
	...
	}
	%〉
		객체 생성, 메서드호출, 로직 실행을 한다.

〈%= %〉	〈%=dehx(1)%〉, 〈%= (1+2)%〉	메서드 실행결과식이나 연산결과를 html 문자열로 만든다. 〈h1〉안녕하세요. h1 크기입니다.〈/h1〉과 3이 된다.

예제 3-6 HTML의 스크립트 사용 예(h16sample.jsp)

```
01: <%@ page contentType="text/html; charset=EUC-KR"%>
02: <%@ page import="com.hanb.Hello"%>
03: <% request.setCharacterEncoding("euc-kr"); %>
04: <html>
05: <head>
06:   <title>글씨 크기</title>
07: </head>
08: <%!  //선언
09: public String dehx(int i){
10:  return
11:  String.format("<h%1$d>안녕하세요. h%1$d 크기입니다.</h%1$d><br/>",i);
12: }
13: %>
```

```
14: <body>
15:   글씨크기-동적 문서 JSP/Class 예제입니다.<hr/>
16:   <%   //스크립트릿
17:   Hello hello=new Hello(); //객체 생성 후 메서드 호출
18:   for(int i=1; i<=6 ;i++){
19:     out.print(hello.hx(i));
20:   }
21:
22:   %>
23:   <hr/>
24:   <%  //익스프레션
25:      for(int i=1; i<=6 ;i++){
26:      %>
27:        <%=dehx(i)%>
28:      <%
29:      }
30:   %>
31:   <a/ href='index.jsp'>HOME</a>
32: </body>
33: </html>
```

2번째 줄: Hello 클래스를 사용하기 위해 페이지 디렉티브에 포함한다.

8번째 줄: JSP 한 페이지 내에서 사용할 메서드를 선언declaration한다.

9~12번째 줄: 입력된 파라미터 i값에 따라 〈h1〉~〈h6〉크기의 문자열을 만든다. i가 1일 때 %1$d 대신 1이 대입되어 '〈h1〉안녕하세요. h1 크기입니다.〈/h1〉〈br/〉'를 반환한다.

20번째 줄: 출력 메서드다. 익스프레션을 이용하면 자바 영역이 끝나야 하기 때문에 %〉〈%가 필요하다. 그래서 %〉〈%=hello.hx(i)%〉〈%와 동일하다.

24~30번째 줄: 스크립트 안은 자바코드 영역이다. 반복문 for를 이용하여 익스프레션 안의 dehx()를 호출한다. 익스프레션은 스크립트가 끝나야 한다.

- 스크립트(〈% %〉)안에 있던 자바코드는 모두 _jspService() 메서드 안으로 들어간다.
- 약속된 스크립트 태그들이 자동으로 html 코드로 변환되어 실행된다.
- JSP는 간단한 표기(태그)나 선언을 이용하여 원하는 것을 쉽게 만들 수 있게 한다.

jsp는 컨테이너에 의해 자동으로 클래스로 변환된다. jsp는 java server page로 서버에서 실행되는 자바로 만든 페이지(웹용 클래스)를 말한다. 따라서 jsp는 클래스로 변환된다. [소스-h16sample.jsp]와 [소스-h16sample_jsp.java]를 비교해 보자. 패키지는 자동으로 정해진 org.apache.jsp(Tomcat)로 선언되고 클래스가 import된다. 선언(〈%! %〉)에 선언되었던 dehx()메서드는 멤버메서드로 선언된다. 스크립트(〈% %〉)안에 있던 자바코드가 모두 _jspService() 메서드 안으로 들어간다는 점이 중요하다. 또한 〈%=%〉는 out.print()로 변환된다. 간단하게 사용하는 스크립트렛(〈% %〉)이나 표현(〈%=%〉), 선언(〈%! %〉)이 복잡한 코드로 변환되고 실행되게 하여 내부로직을 숨기는 추상적(구체적의 반대) 웹 프로그래밍을 할 수 있다.

다음 코드는 h16sample.jsp가 Tomcat 서버에서 클래스(서블릿)로 자동 변환된 코드다. jsp가 아래와 같이 변화되어 실행된다는 점을 알면 된다.

예제 3-7 JSP의 스크립트 사용 예(h16sample_jsp.java)

```
001: package org.apache.jsp;
003: import javax.servlet.*;
004: import javax.servlet.http.*;
005: import javax.servlet.jsp.*;
```

```
006: import com.hanb.Hello;
007:
008: public final class h16sample_jsp
                    extends org.apache.jasper.runtime.HttpJspBase
009:     implements org.apache.jasper.runtime.JspSourceDependent {
010:
011:   //선언
012: public String dehx(int i){
013:   return
014:   String.format("<h%1$d>안녕하세요. h%1$d 크기입니다.</h%1$d><br/>",i);
015: }
...
036:   public void _jspService(HttpServletRequest request,
                                HttpServletResponse response)
037:     throws java.io.IOException, ServletException {
038:
043:      JspWriter out = null;
050:      response.setContentType("text/html; charset=EUC-KR");
057:      out = pageContext.getOut();
062:    request.setCharacterEncoding("euc-kr");
..
071:    //스크립트릿
072: Hello hello=new Hello(); //객체 생성 후 메서드 호출
073:   for(int i=1; i<=6 ;i++){
077:     out.print(hello.hx(i));
078:   }
079:
080:      out.write("\r\n");
081:      out.write("<hr/> \r\n");
082:   // 익스프레션
083:   for(int i=1; i<=6 ;i++){
084:
```

```
085:      out.write("\r\n");
086:      out.write("\t  ");
087:      out.print(dehx(i));
088:      out.write("    \r\n");
089:      out.write("\t");
...
095:  }  //_jspService
```

1번째 줄: Tomcat 서버에서 jsp는 'org.apache.jsp.jsp'가 _jsp.java로 변환된다. Tomcat 서버에서는 org.apache.jsp가 기본 패키지다.

3~5번째 줄: jsp에서 사용되는 기본 객체 패키지다.

6번째 줄: jsp의 페이지 디렉티브의 〈%@ page import="com.hanb.Hello"%〉가 변한된 것이다.

8번째 줄: 컨테이너에서 이루어지는 작업은 HttpJspBase가 하므로 상속을 하여 자세한 내용은 숨겨지고 jsp에서 작성했던 작업만 보인다.

12~15번째 줄: 〈%! %〉 선언 영역에서 선언했던 dehx() 메서드가 클래스에서 선언된다.

36번째 줄: 모든 스크립트(〈%%〉)안의 코드는 자동으로 _jspService() 메서드 안에 들어간다.

43~57번째 줄: 클래스에서 필요한 기본 객체들이 자동으로 선언되고 준비된다.

72~78번째 줄: 반복문 for 문을 실행하여 원하는 문자열을 출력한다.

87번째 줄: 선언영역에 선언했던 메서드 dehx()를 호출한다.

구구단을 이용하여 정적문서(클라이언트쪽 문서)와 동적문서(서버쪽 문서)의 차이

점 및 서버쪽 문서의 동작원리를 확인하자. 정적문서 gugudan.html은 구구단의 결과를 이미 테이블로 만들어 놓은 것이다. 〈table〉안에 〈tr〉과 〈td〉를 이용하여 정사각형의 테이블을 만들고 1X1부터 9X9까지의 81칸의 공간에 곱셈결과를 넣은 것이다. 그러나 gugudan.jsp는 컨테이너 안에서 중첩 for를 이용하여 곱셈을 한 다음, 〈table〉안에 〈tr〉과 〈td〉 태그를 이용하여 html 구구단표를 만들어 사용자에게 보낸다. 결과 화면에서는 이런 차이점을 알 수 없다.

HTML은 구구단을 만들어 테이블 형식으로 출력한다. 10X10 테이블을 만들고 헤더, 단수, 구구단 결과를 테이블로 만들므로 색상 바꾸기도 어렵다. jsp는 중첩 for를 이용하고 곱셈의 결과를 출력한다. 중첩 for를 사용할 정도의 실력이라면 간단하게 구구단을 만들 수 있으며 코딩 분량도 적다. 색상도 쉽게 바꿀 수 있다.

예제 3-8 HTML 형식의 구구단(gugudan.html)

```
001: <html>
002: <head>
003: <title>구구단</title>
004: </head>
005: <body>
006: <center>
007: <p>구구단</p><hr/>
008: <table border='0' bgcolor='#aaaaaa' width='50%'>
009:   <tr>
010:   <td bgcolor='#ffffaa'>X</td>
011:          <td bgcolor='#ffffaa'>1단</td>
012:          <td bgcolor='#ffffaa'>2단</td>
013:          <td bgcolor='#ffffaa'>3단</td>
014:          <td bgcolor='#ffffaa'>4단</td>
015:          <td bgcolor='#ffffaa'>5단</td>
016:          <td bgcolor='#ffffaa'>6단</td>
017:          <td bgcolor='#ffffaa'>7단</td>
```

```
018:         <td bgcolor='#ffffaa'>8단</td>
019:         <td bgcolor='#ffffaa'>9단</td>
020:     </tr>
021:     <tr>
022:         <td bgcolor='#ffffaa'>1단</td>
023:         <td bgcolor='#dddddd'>1</td>
024:         <td bgcolor='#dddddd'>2</td>
025:         <td bgcolor='#dddddd'>3</td>
026:         <td bgcolor='#dddddd'>4</td>
027:         <td bgcolor='#dddddd'>5</td>
028:         <td bgcolor='#dddddd'>6</td>
029:         <td bgcolor='#dddddd'>7</td>
030:         <td bgcolor='#dddddd'>8</td>
031:         <td bgcolor='#dddddd'>9</td>
032:     </tr>
033:     <tr>
034:         <td bgcolor='#ffffaa'>2단</td>
035:         <td bgcolor='#ddffaa'>2</td>
036:         <td bgcolor='#ddffaa'>4</td>
037:         <td bgcolor='#ddffaa'>6</td>
038:         <td bgcolor='#ddffaa'>8</td>
039:         <td bgcolor='#ddffaa'>10</td>
040:         <td bgcolor='#ddffaa'>12</td>
041:         <td bgcolor='#ddffaa'>14</td>
042:         <td bgcolor='#ddffaa'>16</td>
043:         <td bgcolor='#ddffaa'>18</td>
044:     </tr>
...
117:     <tr>
118:  <td bgcolor='#ffffaa'>9단</td>
119:         <td bgcolor='#dddddd'>9</td>
120:         <td bgcolor='#dddddd'>18</td>
```

```
121:          <td bgcolor='#dddddd'>27</td>
122:          <td bgcolor='#dddddd'>36</td>
123:          <td bgcolor='#dddddd'>45</td>
124:          <td bgcolor='#dddddd'>54</td>
125:          <td bgcolor='#dddddd'>63</td>
126:          <td bgcolor='#dddddd'>72</td>
127:          <td bgcolor='#dddddd'>81</td>
128:      </tr>
129:   </table>
130:   <a href='index.jsp'>HOME</a>
131:   </center>
132: </body>
133: </html>
```

예제 3-8를 실행한 다음, 웹 브라우저에서 소스보기를 하면 3-7과 동일한 것을 알 수 있다. 서버의 컨테이너가 jsp 로직을 실행한 다음 html로 만들고, 클라이언트인 웹 브라우저로 보낸다. html로 구구단을 모두 만든 후 웹 브라우저에 html로 출력하므로 내용이 변경되지 않는다. 그래서 정적문서라고 한다. jsp는 컨테이너에서 로직을 실행하고 그 결과를 html로 만들어 나간다. html 변환이 완료되면 웹 브라우저로 보낸다. 로직에 의해 결과가 변경될 수 있고, 로직을 실행하면서 html로 변환되기 때문에 jsp는 동적문서다.

예제 3-9 JSP 형식의 구구단(gugudan.jsp)

```
01: <%@ page contentType="text/html; charset=EUC-KR" %>
02: <%--한글이 깨지지 않도록 한다.--%>
03: <% request.setCharacterEncoding("euc-kr"); %>
04: <html>
05: <head>
06:   <title>구구단</title>
07: </head>
```

```
08: <body>
09:   <center>
10:   <p>구구단</p><hr/>
11:   <table border='0' bgcolor='#aaaaaa' width='50%'>
12:   <%
13:   for( int i=0; i<10; i++){  //행(row)에 대한 for
14:     %><tr>
15:     <%for( int j=0; j<10; j++){  //열(column)에 대한 for
16:       if(i==0){
17:         if(j==0){
18:         %><td bgcolor='#ffffaa'>X</td>
19:         <%
20:         }else{
21:         %><td bgcolor='#ffffaa'><%= j %>단</td>
22:         <%
23:         }
24:        }else{
25:         if(j==0){
26:         %><td bgcolor='#ffffaa'><%= i %>단</td>
27:         <%
28:         }else{
29:         %><td bgcolor='<%=( i%2==0)?"#ddffaa":"#dddddd" %>'><%= i*j %></td>
30:         <%
31:         }
32:       }
33:     }
34:     %></tr>
35:     <%
36:   }
37:   %>
38:   </table>
39:   <a href='index.jsp'>HOME</a>
```

```
40:  </center>
41: </body>
42: </html>
```

11번째 줄: 경계선이 없고 배경색bgcolor이 '#aaaaaa'이면서 전체 화면의 50%를 차지하는 테이블을 만든다.

13~14, 34번째 줄: 테이블의 행(row)을 10개 만든다. 행은 <tr></tr>로 만든다.

15번째 줄: <tr></tr> 안에 열(column-<td></td>)을 10개 만든다. <tr><td></td></tr>과 같이 <tr>안의 <td>를 셀이라고 부른다.

16~19번째 줄: 첫 행의 첫 열(0,0 셀)위치에 X를 출력한다.

20~23번째 줄: 첫 행의 다른 열(0,1셀 ~0,9셀)에는 '1단'~'9단'을 출력한다. 정리하면 다음과 같다.

X 1단 2단 3단 ... 9단

25~27번째 줄: 1행부터 9행에 대해 첫 열(j==0)에 1단~9단을 출력한다.

28~31번째 줄: 첫 열을 제외한 1행부터 9행에 대해 행과 열의 곱(i=1~9, j=1~9)으로 구구단을 만든다. 만약 i=5라면 "5단 5 10 15 … 45"와 같이 5단을 출력한다. 그리고 홀수단이므로 셀의 배경색은 '#dddddd'가 된다.

NOTE_ <% %>는 열고 닫기가 매우 어렵다. 자바코드는 반드시 스크립트릿 영역에 있어야 한다. html 영역에서 문자열을 출력할 때는 익스프레션(<%=%>)을 사용한다.

그림 3-6 구구단 결과화면

구구단

X	1단	2단	3단	4단	5단	6단	7단	8단	9단
1단	1	2	3	4	5	6	7	8	9
2단	2	4	6	8	10	12	14	16	18
3단	3	6	9	12	15	18	21	24	27
4단	4	8	12	16	20	24	28	32	36
5단	5	10	15	20	25	30	35	40	45
6단	6	12	18	24	30	36	42	48	54
7단	7	14	21	28	35	42	49	56	63
8단	8	16	24	32	40	48	56	64	72
9단	9	18	27	36	45	54	63	72	81

HOME