Chapter 03

JSP와의 첫 만남

＊ 학습 목표

- JSP와 서블릿의 관계를 이해하고, JSP 프로그래밍과 관련된 기술을 알아본다.
- JSP 프로그램의 전체 처리 과정을 이해한다.
- JSP 프로그램 개발에 유리한 프로그래밍 모델을 알아본다.
- JSP 기술 변천 과정을 최근 소프트웨어 아키텍처 경향과 함께 살펴본다.
- 간단한 JSP 프로그램을 작성하고 실행하는 방법을 익힌다.

Section

1 JSP 개요

JSP는 자바로 구현된 기술로서, 특히 서블릿이라는 서버 프로그래밍 기술에 기반한 웹 프로그래밍 언어다. 서블릿을 먼저 배워야 JSP를 배울 수 있는 것은 아니지만 기반 기술인 서블릿의 구조와 동작 원리를 이해한다면 JSP를 좀더 쉽게 배울 수 있을 것이다. 이 절에서는 서블릿과 JSP의 관계에 대해 알아보기로 한다.

1 서블릿과 JSP

서블릿(Servlet)은 자바를 이용한 서버 프로그래밍 기술로서, 일반 애플리케이션을 개발하기 위해 만들어진 자바를 웹 환경에서 사용하려고 등장하였다. 서블릿은 초기의 웹 프로그래밍 기술인 CGI(Common Gateway Interface)를 대체하려고 개발되었으나, 느린 처리 속도, 많은 메모리 요구, 불편한 화면 제어 등의 한계로 인해 PHP, ASP 등에 주도권을 넘겨주게 되었다. 그러다 썬에서 서블릿을 기반으로 하면서 PHP와 유사한 형태로 프로그래밍이 가능한 JSP(Java Server Page)를 선보이면서 JSP는 가장 대표적인 웹 프로그래밍 언어로 자리 잡게 되었다.

아래는 동일한 실행 결과를 보여주는 예를 서블릿과 JSP로 구현한 것인데, 프로그램 문법을 전혀 모르더라도 JSP가 서블릿보다 상당히 단순하다는 것을 직관적으로 알 수 있다.

■ 서블릿으로 구현한 코드

```
01 public class HelloWorldServlet extends HttpServlet {
02   public void doGet(HttpServletRequest request,
03                     HttpServletResponse response)
04   throws ServletException, IOException {
05     response.setContentType("text/html; charset=EUC_KR");
06     PrintWriter out = response.getWriter();
```

```
07    out.println("<HTML><HEAD><TITLE>로그인</TITLE></HEAD> </HTML>");
08    out.println("<BODY><H2>Hello World : 헬로월드</H2>");
09    out.println("오늘의 날짜와 시간은 : "+new
10    java.util.Date());
11    out.println("</BODY></HTML>");
12  }
13 }
```

■ JSP로 구현한 코드

```
01 <%@ page contentType="text/html;charset=utf-8" %>
02  <HTML>
03  <HEAD><TITLE>Hello World</TITLE></HEAD>
04  <BODY><H2>Hello World : 헬로월드</H2>
05  오늘의 날짜와 시간은 : <%= new java.util.Date() %>
06  </BODY>
07  </HTML>
```

위 소스를 잠깐 살펴보면 서블릿은 일반적인 자바 클래스의 형태를 취하고 있으며 처리 결과를 웹 화면에 출력하려고 out.println() 메서드를 이용한다는 것을 알 수 있다. 이와 같은 구조는 HTML로 표현해야 하는 내용이 많을수록 서블릿 프로그램이 상당히 비효율적이라는 것을 의미한다(JSP 동작 과정에서 서블릿이 어떤 역할을 하는지는 다음 절에서 자세히 살펴본다).

JSP는 단순하고 직관적이라는 것 외에 서블릿을 기반으로 하기 때문에 자바가 지원하는 기능을 100% 사용할 수 있고 IBM과 오라클 등 세계적인 업체의 강력한 지원을 받고 있어 사후관리가 쉬우며 다양한 운영체제와 개발환경을 이용할 수 있다는 장점이 있다. 또한 여러 오픈소스 프로젝트의 지원도 받고 있다.

서블릿과 JSP의 관계를 이해하는 데 있어서 한 가지 주목할 점이 있다. JSP가 처음 등장할 당시 서블릿을 대체하는 기술로 JSP를 주목했지만, 이제는 서블릿을 대체하는 기술이 아니라 상호보완적인 기술로 이해되고 있다는 점이다. 이를 바탕으로 JSP의 대표적인 특징을 살펴보면 다음과 같다.

❶ 자바의 모든 기능을 사용할 수 있어 발전 가능성이 무한하다.

❷ 서블릿으로 컴파일된 후 메모리에서 처리되기 때문에 많은 사용자의 접속도 원활하게 처리할 수 있다.

❸ JSP 또는 다른 서블릿 간의 데이터를 쉽게 공유할 수 있다.

❹ 빈즈(Beans)라고 하는 자바 컴포넌트를 사용할 수 있다.

❺ 커스텀 태그를 만들어 사용할 수 있으며, JSTL(JSP Standard Tag Library)과 같은 태그 라이브러리를 이용할 수 있다.

❻ 스트러츠, 스프링 @MVC 등 다양한 프레임워크와 결합하여 개발할 수 있다.

❷ JSP 학습에 필요한 기술

JSP는 자바를 기반으로 하는 웹 프로그래밍 기술이므로, JSP를 배우려면 일반적인 웹 프로그래밍 기술과 자바와 관련된 기술 경험이 필요한데, 어느 정도의 요구 수준이 필요한지 아래 표에 간단히 정리하였다.

[표 3-1] 웹 프로그래밍을 위한 기본 기술 경험의 요구 수준

기술	설명	요구 수준
HTML	클라이언트 기술로서, 웹 프로그램의 기본이 되며 시각적인 부분을 담당한다.	• HTML 문서 구조와 기본 태그 • FORM 관련 태그 • HTML5 기본 구조
자바스크립트	웹 화면과 사용자와의 상호작용 및 동적 웹 페이지를 구현할 때 필요한 기술이다.	• 기본 문법 • 객체와 메서드 • 내장 객체 • 이벤트 핸들링
CSS	웹 화면의 레이아웃과 디자인 요소를 구현할 때 필요한 기술이다.	• 스타일시트 정의 및 셀렉터 이해 • DOM 연동에 의한 동적 스타일 제어

다음은 자바 관련 기본 기술이다. JSP는 자바 기반이며 실제로 프로그램을 개발할 때 50% 이상이 순수 자바 프로그램으로 구현된다는 점을 알아야 한다. 무엇보다 자바에 대한 기본이 탄탄할수록 JSP 기반의 프로그래밍이 쉽다는 것을 명심하고 JSP를 배우면서도 자바에 대한 학습을 병행하기 바란다.

[표 3-2] 자바 관련 기본 기술 경험의 요구 수준

기술	설명	요구 수준	
자바	소스코드를 작성하기 위한 프로그래밍 기본 언어로서, Java SE를 기준으로 한다.	• 자바 기본 • 상속, 오버로딩, 오버라이딩 • java.util, java.io 패키지 • 예외 핸들링	• 객체지향 개념 • 인터페이스 구현 • 스레드
JDBC	Java DataBase Connectivity의 약자로서, 자바에서 데이터베이스 프로그래밍을 하기 위한 기술이다.	• JDBC 드라이버 세팅 • PreparedStatement • 기초 SQL문	• ResultSet • 데이터 핸들링
서블릿	JSP의 기본이 되는 자바 기반의 웹 프로그래밍 핵심 기술이다.	• 서블릿 구조 이해 • request, response 처리	• 간단한 서블릿 프로그래밍 • GET/POST 처리

그리고 [표 3-3]에는 추가로 알아두면 유용한 기술을 정리했다. 물론 JSP 프로그래밍을 할 때 꼭 알아야 하는 기술은 아니지만 효율적이고 실무적인 JSP 프로그래밍을 하려면 알고 있는 것이 좋다.

[표 3-3] 고급 웹 프로그래밍을 위한 주변 기술 경험의 요구 수준

기술	필요성	요구 수준
데이터베이스	프로그램의 데이터를 처리하려고 할 때 반드시 필요하다.	• 다양한 SQL문의 사용 • 데이터베이스 연계 프로그래밍 경험 • 데이터베이스 함수 및 내장 프로시저
XML	eXtensible Markup Language의 약자로서, 확장 가능한 구조적 문서 표현을 제공한다. 많은 프로그램에서 데이터 구조를 XML 기반으로 처리한다.	• XML 스키마 및 DTD 이해 • XML DOM 개요
모바일 프로그래밍	최근에는 스마트폰을 중심으로 하는 모바일 기반의 개발이 증가하고 있는 추세다.	• 안드로이드 혹은 아이폰 앱 개발 경험 • 하이브리드 앱 개발 경험
프레임워크	개발자로 하여금 더욱 좋은 프로그램을 만들 수 있도록 미리 제공되는 틀을 말한다.	• 소프트웨어 아키텍처 이해 • 스프링 프레임워크 • 스프링3 @MVC

앞서 언급한 기술들과 요구 수준들만 보고 두려워할 필요는 없지만 가능하다면 관련 기술들을 전반적으로 배워나가길 권한다. 물론 이 책에서는 여러분들이 관련 경험이 부족하다는 전제를 두고 진행할 것이므로 너무 걱정하지 않아도 된다.

Section

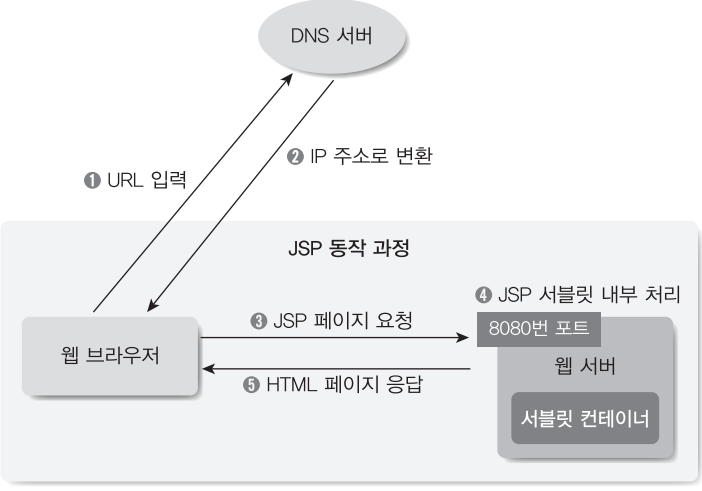
2 JSP 처리 과정의 이해

JSP 프로그램을 작성하기에 앞서 JSP의 처리 과정을 살펴보자. JSP는 HTML과 달리 소스 자체로는 실행할 수 없으며 웹 프로그램의 특성이 있기 때문에 웹 브라우저, 웹 서버, 서블릿 컨테이너, JSP, 서블릿, 빈즈 클래스 등 여러 구성요소 사이의 흐름을 이해해야 한다.

먼저 웹 브라우저와 웹 서버(서블릿 컨테이너)와의 관계부터 이해해보도록 하자(이 책에서는 서블릿 컨테이너가 제공하는 웹 서버 기능을 이용하므로 이를 기준으로 설명하겠다).

❶ JSP 전체 동작 과정

JSP의 전체 동작 과정을 그림으로 그려서 설명하면 다음과 같다.



[그림 3-1] JSP 전체 동작 과정

❶ 웹 브라우저에서 URL을 입력한다.

❷ DNS 서버로부터 입력한 URL을 변환한 IP 주소를 받는다.

❸ 받은 IP 주소의 웹 서버 8080번 포트에 JSP 페이지를 요청한다.

❹ 웹 서버가 요청 내용을 분석하고 서블릿 컨테이너에 요청을 넘겨 처리한다.

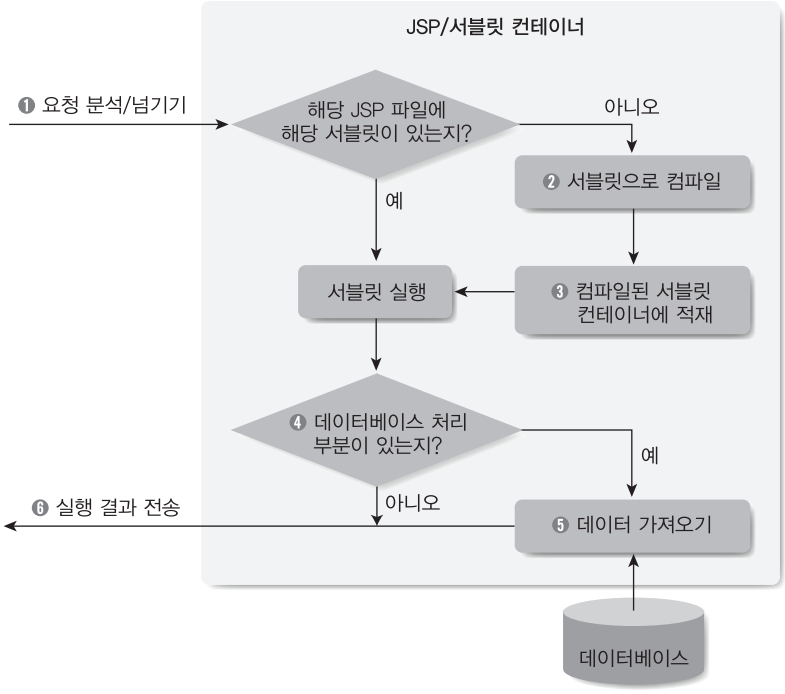
❺ 화면에 보일 내용을 HTML 문서 형태로 웹 브라우저에 전송한다.

여기서 명확하게 알아야 하는 부분은 JSP로 개발된 웹 프로그램이라 하더라도, 사용자에게 최종적으로 전달되는 콘텐츠는 HTML이어야 한다는 점이다. 이와 같은 특징은 모든 웹 프로그램 기술에 동일하게 적용된다. 서블릿 내부에서 처리되는 과정을 좀더 구체적으로 살펴보자.

❷ 서블릿 컨테이너 내부 과정

JSP가 HTML과 같은 일반적인 텍스트 파일 구조인데 비해, 서블릿은 자바 소스로 작성된 클래스 파일 구조로 되어 있다. 서블릿 컨테이너는 JSP 파일을 서블릿 소스로 변환해서 컴파일하는데, 일단 컴파일된 JSP는 단순한 파일이 아니라 컨테이너에서 서블릿 객체로 관리된다. 이러한 과정 때문에 최초 JSP 파일에 접근할 때는 컴파일하는 시간이 많이 소요되지만 일단 컴파일되어 실행된 다음부터는 서블릿 컨테이너가 처리하기 때문에 빠르게 실행된다.

그렇다면 컨테이너 내에서 JSP로 만들어진 프로그램이 처리되는 내부 과정을 살펴보자. 1장에서 HTML을 사용하는 일반 웹 서비스에 대한 요청을 살펴보았는데, JSP의 경우에도 웹 서버까지 요청은 HTML과 동일하고, 웹 서버 이후의 과정에만 차이가 있다.



[그림 3-2] JSP 서블릿 컴파일과 처리 과정

❶ 웹 서버로부터 JSP에 대한 사용자 요청이 컨테이너로 전달된다.

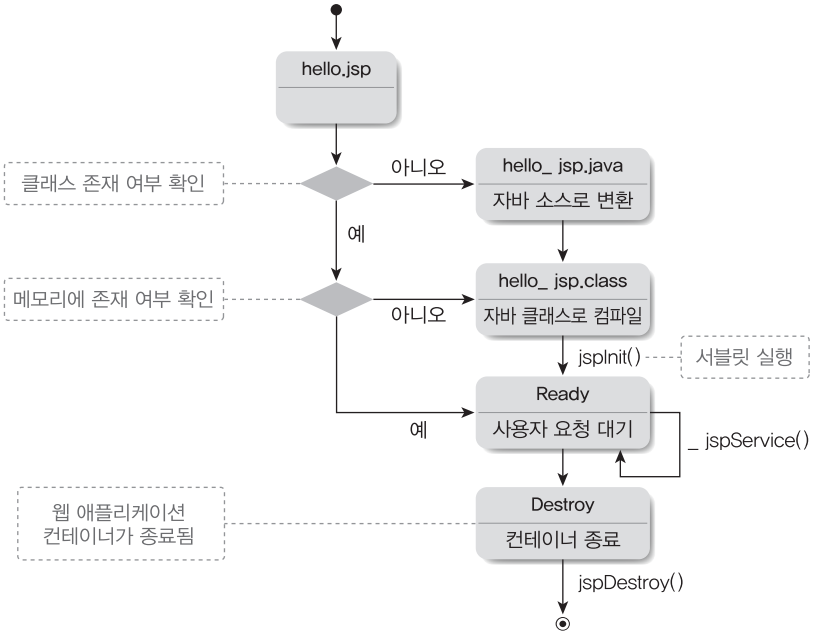
❷ 요청 JSP에 대한 서블릿이 존재하면 다음 단계로 진행하고, 존재하지 않을 경우 JSP를 .java 파일로 변환한 다음 .class 파일로 컴파일한다.

❸ 컴파일된 서블릿 클래스를 컨테이너의 메모리에 적재하고 실행한다.

❹ ~ ❺ 데이터베이스 처리 혹은 별도의 기능을 위한 클래스 호출 등이 있다면 실행하고 결과를 취합해 HTML 형태로 구성한다.

❻ HTML 형태의 결과를 웹 서버를 경유해 사용자 브라우저에 전달한다.

JSP가 서블릿으로 컴파일되고 서블릿으로 실행되는 과정(위의 ❷ 단계 ~ ❸ 단계)을 좀더 자세히 살펴보기로 하자.



[그림 3-3] JSP과 서블릿 변환과 상태변화

JSP 파일은 일반 텍스트를 비롯하여, HTML 코드와 몇몇 특수 태그, 자바 코드가 섞여 있다. 클라이언트 요청에 대해 서블릿 클래스가 존재하지 않을 경우, JSP 파일을 자바 소스 파일로 변환한 후 클래스 파일로 컴파일한다. 컴퓨터에서 실행 가능한 형태인 클래스 파일은 소스가 변경되기 전까지는 메모리에 상주하면서 다시 컴파일되지 않고 서비스된다. 즉 JSP 파일이 수정되기 전까지는 [그림 3-3]의 과정이 수행되지 않는다.

JSP 파일의 실행과 종료에 관계하는 메서드를 생명주기 메서드라고 하는데, 컴파일된 JSP는 서블릿 실행 과정에서 jspInit() 메서드를 통해 Ready 상태가 되고 메모리에 로드된다. 그리고 이후 모든 클라이언트 접속에 따라 _jspService() 메서드만 반복해서 호출된다.

컨테이너가 종료될 때에는 jspDestroy() 메서드를 통해 메모리에서 언로드되고, 다시 컨테이너가 시작될 경우 최초 클라이언트 요청에 대한 클래스가 있기 때문에 재컴파일 과정은 거치지 않고, jspInit() 메서드 호출을 통해 메모리에 로드하는 과정을 거친다.

여기까지 JSP 처리 과정과 관련해서 제법 상세한 내용까지 살펴보았다. 아직까지 이 모두를 이해하기를 어렵겠지만, 아래 내용만은 꼭 기억해두길 바란다.

❶ JSP는 일반 텍스트 파일로 되어 있다(텍스트 파일은 컴퓨터가 이해할 수 없다. 즉 실행 가능한 프로그램이 아니며 특정 동작을 할 수 없다).

❷ JSP는 HTML 코드와 몇몇 특수한 태그, 그리고 자바 코드가 섞여 있다.

❸ 사용자가 요청할 경우 JSP는 컨테이너(톰캣)에 의해 서블릿 형태의 .java 소스로 변환되고 컴파일된다.

❹ 컴파일된 .class는 컴퓨터에서 실행할 수 있는 형태로 특정한 기능을 수행할 수 있게 된다. 이후 소스 변경 전까지 해당 파일은 메모리에 상주하면서 다시 컴파일되지 않고 서비스된다.

Section

3 JSP 프로그램 기술 변천

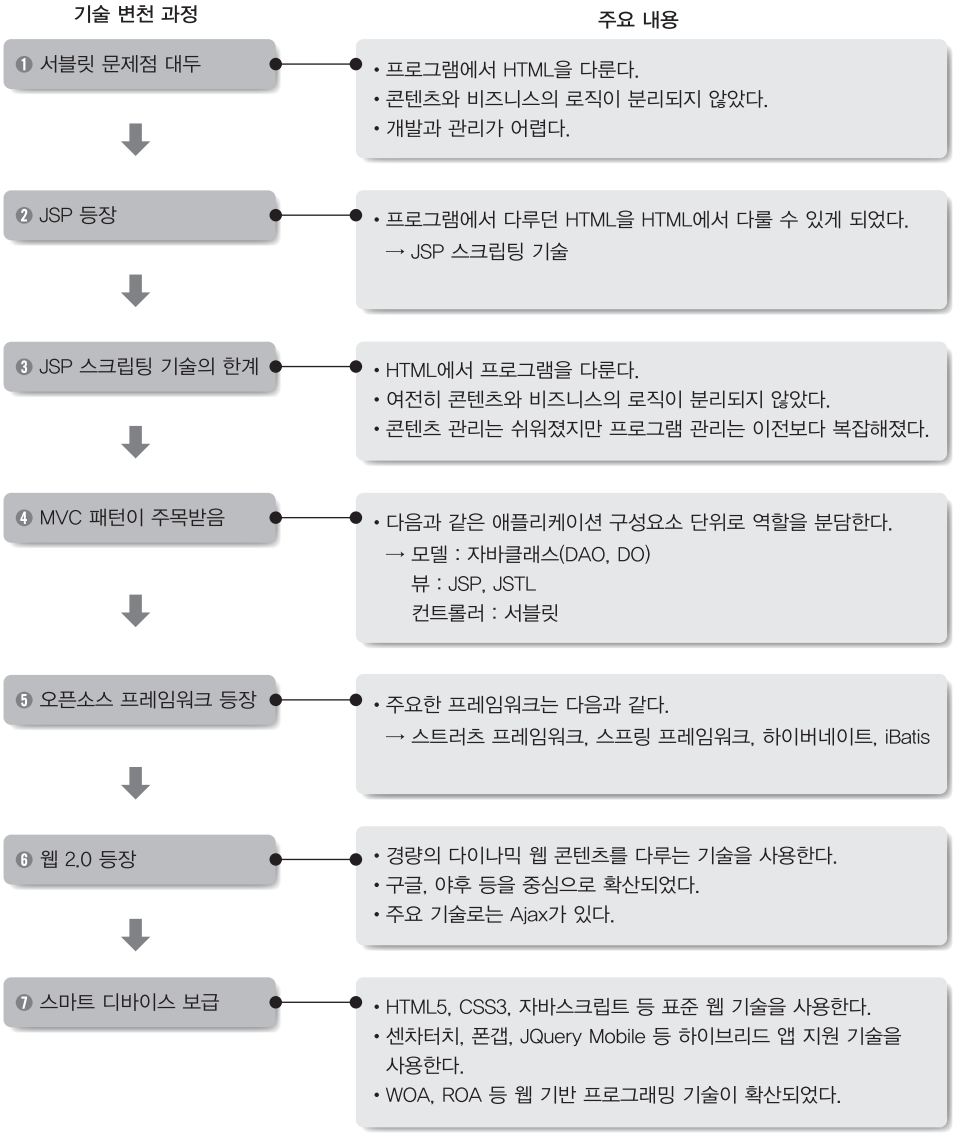
사용자 입장에서는 편리하고 기능이 많아야 좋은 프로그램이겠지만, 개발자 입장에서는 개발 생산성, 확장성, 유지보수가 수월해야 좋은 프로그램이다. 세상의 모든 것에는 유행이 있듯이, 프로그래밍 분야에서도 유행이 되는 기술과 개발 모델이 있다.

현재 유행하는 JSP 웹 프로그래밍 모델은 MVC 패턴이라고 불리는 JPS 모델-2다. 현업에서는 이 모델을 절대적으로 가장 많이 사용하고 있다. 이러한 개발 모델을 지금 당장은 이해하기 어렵겠지만, 최근의 프로그래밍 경향을 이해하는 측면에서 기본적인 부분만 살펴보도록 한다. 또한 이 절에서는 JSP 프로그램의 기술 변천 과정을 살펴보고, JSP 프로그램 개발에 유리한 프로그램 유형을 최근 소프트웨어 아키텍처 경향과 함께 알아본다.

예전에는 MVC 패턴을 기존 방법에 대한 새로운 접근이라는 의미로 모델-2라는 표현을 썼지만 요즘은 그냥 MVC 패턴이라고 더 많이 부른다.

1 서블릿과 JSP 기술 변천

JSP를 더욱 효율적으로 사용하려면 서블릿에 대한 얘기를 빼놓을 수 없다. 원칙적인 순서라면 서블릿을 먼저 익히고 JSP로 나아가는 것이 좋지만 초보 웹 개발자에게는 이런 순서가 더 어렵기 때문에, 이 책에서는 JSP를 먼저 다루고 서블릿으로 넘어가는 방법을 택했다. 이 책에서 익힐 서블릿의 수준은 JSP의 이해를 돕고 실전 프로젝트에서 프레임워크를 이해하려는 정도임을 알아두고 JSP의 기술 변천 과정부터 살펴보자.



[그림 3-4] 서블릿과 JSP의 기술 변천

JSP가 등장한 가장 큰 이유는 서블릿의 한계인 HTML 표현상의 문제를 해결하기 위함이었다. 서블릿은 프로그램 내에서 HTML을 처리하기 때문에 간단한 태그를 변경할 때도 재컴파일 해야 하는 문제가 있었다. 이러한 이유로 웹 디자이너는 마음대로 화면을 수정할 수 없었고, 복잡한 HTML 소스를 프로그래머가 서블릿 클래스 내에서 관리해야 하는 문제가 뒤따랐

다. 이처럼 비즈니스 로직과 콘텐츠가 하나의 소스에 있다는 점은 개발과 관리 면에서 여러 문제점을 안겨주었고, 콘텐츠와 비즈니스 로직을 분리하기 위한 일환으로 JSP가 등장하게 되었다.

물론, JSP 이전에는 별도로 만들어진 템플릿 엔진 등을 이용하여 이러한 문제를 부분적으로 해결할 수 있었지만 표준화되지 못했으며, 사용 방법의 복잡성으로 인해 일부에서만 활용되었다.

JSP가 등장한 초기에는 JSP가 기존 서블릿의 문제점을 모두 해결한 것처럼 보였다. 그러나 많은 프로그램이 웹 기반으로 개발되고 대규모 시스템이 등장함에 따라 JSP 스크립팅 기술의 한계가 드러나기 시작했다. 즉, 콘텐츠와 비즈니스 로직은 어느 정도 분리되기는 했으나 JSP에서 프로그램을 직접 다루는 부분이 늘어나게 되어 이전보다 더 복잡하고 관리가 어려운 상황이 되었다.

예를 들어, 어떤 화면에서 사용자 이벤트(버튼 클릭이나 링크 클릭)가 발생하면 이를 받아 처리하는 JSP 파일과 처리 결과를 보여줄 JSP 파일이 필요하다(로그인 폼 → 로그인 처리 → JSP(빈즈 사용) → 로그인 후 화면). 결국 기능을 추가할수록 요청을 처리해야 하는 JSP 파일이 중복적으로 많이 생성되고 각각의 JSP 파일에서 비즈니스 로직을 처리하기 때문에 관리는 더욱 어려워진다. 이렇듯 초기의 JSP 개발 모델인 JSP 모델-1은 코드와 프로그램이 뒤섞인 구조였기 때문에 신속한 유지보수와 서비스 확장이 어려웠다. 이를 해결하려고 제시된 것이 바로 JSP 모델-2, 현재 MVC 패턴이라고 불리는 모델이다.

MVC 패턴의 기본 개념은 사용자에게 보일 페이지(View)와 데이터 처리(Model), 그리고 이들 상호 간의 흐름을 제어(Controller) 하는 모듈을 분리하는 것이다. 이로써 더욱 쉽게 웹 애플리케이션을 확장하고 유지보수를 할 수 있다. MVC 패턴의 구현은 패턴에서 요구하는 규칙에 따라 개발자가 직접 구조를 설계할 수도 있고, 스프링3 @MVC, 스트러츠와 같은 검증된 오픈소스 프레임워크를 사용할 수도 있다.

저자 한마디

MVC 패턴

패턴이라는 용어는 소프트웨어 디자인 패턴에서 유래한 것이다. 소프트웨어 디자인 패턴은 특정 목적에 대해 검증된 설계 모델을 말하는데, 목적이 동일한 모든 곳에서 적용될 수 있는 형태를 말한다. MVC(Model-View-Controller) 패턴은 객체지향 언어 중 하나인 스몰토크(Smalltalk)에서 사용자 인터페이스 개발에 사용된 것으로, 오랫동안 그 효용성이 입증되어왔으며 웹 개발에도 적합한 것으로 인정받고 있다.

이 책에서는 최대한 JSP 구성요소만을 사용하여 비교적 간단한 형태의 MVC 패턴을 적용해서 프로그래밍할 것이고, 패턴 구조에 따른 프로그램은 체험학습에서 다룬다. 이 책의 대부분의 학습은 JSP 문법과 실습에 대한 이해를 돕고자 단일 JSP 페이지 기반에서 학습하게 되니 참고하기 바란다.

이후에 어느 정도 자바 프로그래밍에 자신이 있고 서블릿, 필터 등 웹 애플리케이션 프로그래밍 기술을 잘 알고 난 이후에는 자신만의 컨트롤러를 구현해보는 것도 좋은 방법이다. 하지만 잘못 개발할 경우 호환성이나 생각하지 못한 문제가 발생할 수 있다. 따라서 가급적이면 스프링3 @MVC나 스트러츠와 같이 검증된 오픈소스 프레임워크를 이용하기 바란다.

❷ JSP 프로그램 모델 유형 비교

JSP는 어떻게 프로그래밍 하느냐에 따라 JSP의 장점을 살릴 수도, 그렇지 못할 수도 있기 때문에 처음의 프로그래밍을 어떻게 하느냐가 매우 중요하다. 일단 JSP 프로그램 유형을 크게 두 가지로 구분하면 다음과 같다.

❶ 유형-1 : 빈즈 클래스를 사용하지 않고 스크립트릿만을 사용하는 방법

❷ 유형-2 : 빈즈 클래스와 〈jsp:useBean〉 액션을 적극적으로 사용하는 방법

두 소스코드의 실행 결과는 모두 동일하지만, 어떤 구현 절차를 거치느냐는 앞으로 프로그래밍하는 데 매우 중요한 영향을 끼친다.

유형-1 계산기 프로그램(calc.jsp)

```
01 <%@ page contentType="text/html;charset=UTF-8" %>
02
03 <%
04        // 변수 설정
05        int result = 0;
06
07        // 웹 페이지 요청이 POST인 경우에만 수행
          // 즉 폼(Form)을 통해 전달된 것만 수행
08        // 초기 로딩 시 오류 방지
09        if(request.getMethod().equals("POST")) {
10
```

```
11            // 연산자를 가져옴
12            String op = request.getParameter("operator");
13
14            // 문자열 형태로 전달된 인자들을 int로 변환함
15            int num1 = Integer.parseInt(request.getParameter("num1"));
16            int num2 = Integer.parseInt(request.getParameter("num2"));
17
18            // 각 연산자별 처리
19            if(op.equals("+")) {
20                    result = num1+num2;
21            }
22            else if(op.equals("-")) {
23                    result = num1-num2;
24            }
25            else if(op.equals("*")) {
26                    result = num1 * num2;
27            }
28            else if(op.equals("/")) {
29                    result = num1 / num2;
30            }
31        }
32 %>
33
34 <HTML>
35 <HEAD>
36 <TITLE> 계산기 </TITLE>
37 </HEAD>
38
39 <BODY>
40 <CENTER>
41 <H3>계산기</H3>
42 <HR>
43 <form name=form1 method=post>
44 <INPUT TYPE="text" NAME="num1" width=200 size="5">
45 <SELECT NAME="operator">
46        <option selected>+</option>
47        <option>-</option>
48        <option>*</option>
```

```
49        <option>/</option>
50 </SELECT>
51
52 <INPUT TYPE="text" NAME="num2" width=200 size="5">
53 <input type="submit" value="계산" name="B1">
54 <input type="reset" value="다시 입력" name="B2">
55 </form>
56 <HR>
57
58 계산결과 : <%=result %>
59 </BODY>
60 </HTML>
```

유형 -2 계산기 프로그램(calc.jsp)

```
01 <%@ page contentType="text/html;charset=UTF-8" %>
02 <jsp:useBean id="calc" scope="page" class="jspbook.ch03.CalcBean" />
03 <jsp:setProperty name="calc" property="*" />
04 <% calc.calculate(); %>
05
06 <HTML>
07 <HEAD>
08 <TITLE> 계산기 </TITLE>
09 </HEAD>
10
11 <BODY>
12 <CENTER>
13 <H3>계산기</H3>
14 <HR>
15 <form name=form1 method=post>
16 <INPUT TYPE="text" NAME="num1" width=200 size="5">
17 <SELECT NAME="operator">
18 <option selected>+</option>
19        <option>-</option>
20        <option>*</option>
21        <option>/</option>
```

```
22 </SELECT>
23
24 <INPUT TYPE="text" NAME="num2" width=200 size="5">
25 <input type="submit" value="계산" name="B1">
26 <input type="reset" value="다시 입력" name="B2">
27 </form>
28 <HR>
29
30 계산결과 : <jsp:getProperty name="calc" property="result" />
31 </BODY>
32 </HTML>
```

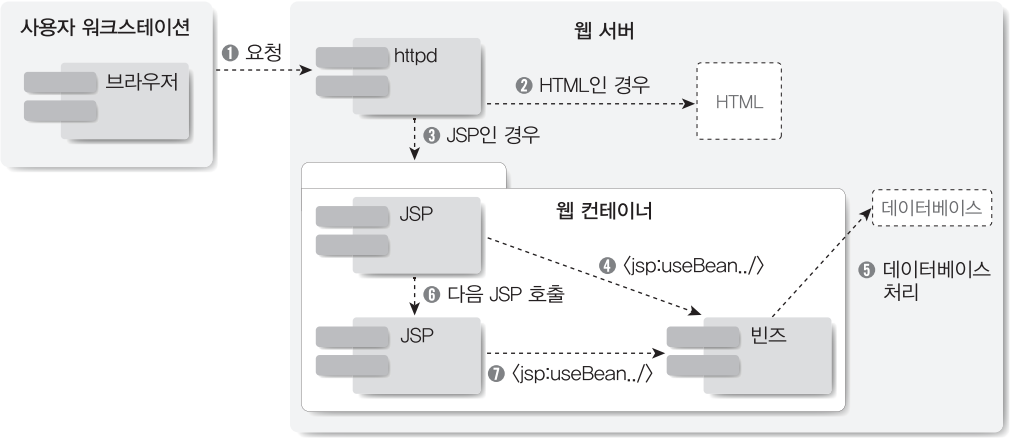
언뜻 보기에도 빈즈를 사용한 [유형-2]의 소스코드가 훨씬 간단해보인다. 빈즈에 대한 개념은 6장에서 자세히 배울 것이니 여기서는 빈즈 사용의 장점만 간단히 알아두고 넘어가자. [유형-1]의 경우 외부 클래스를 사용하는 부분도 스크립트릿에서 사용하므로 개발 과정에서 JSP 파일에만 집중하면 되고, 비교적 배우기가 쉽다는 장점이 있지만 소스코드가 복잡해진다는 단점이 생긴다.

스크립트릿 : JSP 파일에 자바 프로그램 문법을 사용할 수 있는 JSP 구문을 말한다.

위의 소스코드는 간단한 계산기 JSP 소스다. 따라서 워낙 간단하여 빈즈 없이 구현해도 무방하지만, 실무에서 사용하는 소스코드는 이 소스코드와는 비교할 수 없을 만큼 복잡하기 때문에 빈즈를 이용하지 않고 구현하기에는 한계가 많다. 빈즈를 사용하면 프로그램으로 구현되는 기능들을 캡슐화하여 재활용하기가 쉽고, JSP 파일 간의 공유도 쉽다는 장점이 있다.

아직 JSP를 배우지 않았기 때문에 여기서는 구조적인 차이점을 중심으로 살펴볼 것이고 별도의 소스작성이나 실행은 다루지 않는다. 물론 이 책의 내용과 관련된 자료를 제공하는 웹 사이트(http://www.hanb.co.kr/exam/4068)에서 소스코드를 제공할 것이므로 2부 학습 이후 JSP 실행에 익숙해지면 직접 한번 실행해 봐도 좋을 것이다.

빈즈를 이용한 JSP 처리 과정



[그림 3-5] 빈즈를 사용한 JSP 처리 과정

❶ 웹 브라우저 요청에 대해서 웹 서버가 http 프로토콜에 따라 사용자 요청을 처리한다.

❷ ~ ❸ 웹 서버는 HTML과 JSP인 경우를 구분해서 처리한다.

❹ JSP의 경우 useBean 액션을 이용해서 빈즈 클래스와 연동한다.

❺ 필요한 경우 데이터베이스 관련 작업 등을 처리한다.

❻ 처리한 결과를 보여주는 JSP를 호출한다.

❼ 필요한 경우 결과를 보여주는 JSP에서도 빈즈를 사용할 수도 있다.

빈즈와 관련해서는 6장에서 자세히 배운다. 여기서는 JSP 프로그램의 유형을 머릿속에 그려 보고 빈즈를 이용한 JSP 프로그램의 유용성을 이해하기만 하면 된다. 빈즈를 사용하면 프로그램 개발에 훨씬 유리하지만 그렇다고 웹 프로그램의 구조적 문제를 모두 해결할 수 있는 것은 아니다.

Section

04 기본실습 JSP 프로그래밍 : Hello World JSP

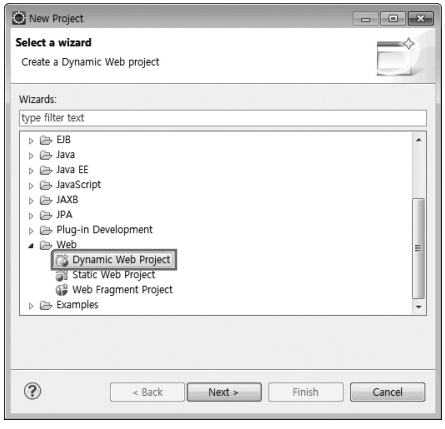
준비 과정이 길었던 만큼 기대가 클 것이다. 이제부터 JSP 프로그래밍을 시작해보자. 이 프로그래밍은 JSP 맛보기 학습이라고 생각하면 된다. 전체 흐름을 파악하는 것이 목적이므로 세부적인 내용에 연연할 필요는 없다. 이클립스에서 JSP 프로그래밍을 하기 위한 절차를 살펴보고, 익숙해질 수 있도록 두세 번 정도 반복해서 따라하기를 권한다.

❶ 이클립스 프로젝트 생성

이클립스를 실행하고 [File]→[New]→[Project]를 선택하면 미리 정의된 특정 유형의 프로젝트 템플릿을 이용하여 프로젝트를 생성할 수 있다.

다이나믹 웹 프로젝트 생성하기

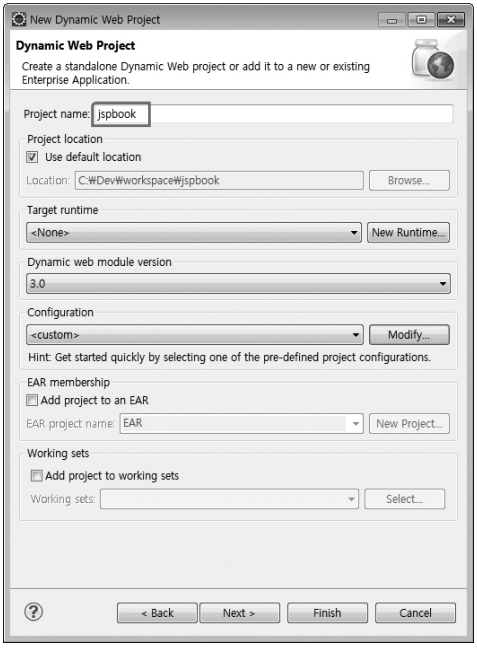
트리 메뉴 중 [Web]을 선택하고 [Dynamic Web Project]를 선택하면 JSP 개발을 위한 프로젝트가 생성된다. 참고로 아래의 [Static Web Project]는 HTML만으로 구성된 정적인 웹 페이지 개발을 위한 템플릿이다.



[그림 3-6] 다이나믹 웹 프로젝트 생성

기본 정보 설정하기

〈Next〉 버튼을 누르면 프로젝트 기본 정보 설정 화면이 나온다. [Project name]에 jspbook이라고 입력하고 나머지 항목은 기본 설정을 사용한다. [Target Runtime]은 개발한 JSP 등의 실행을 위한 서버를 설정하는 것인데, 추후에 설정할 것이다. [Configuration] 항목은 프로젝트의 기본 설정 값을 미리 정의해놓고 사용하는 것인데, 고급 웹 프로그래밍을 위해 필요한 항목이다. 이 책에서 다루는 범위 안에서는 기본 값인 Custom을 사용한다. 〈Next〉 버튼을 누른다.



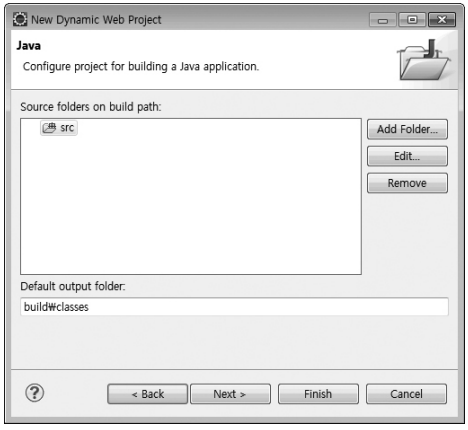
[그림 3-7] 프로젝트 기본 정보 설정

지금 생성하는 jspbook 프로젝트는 앞으로 이 책에서 다루는 모든 소스를 관리할 프로젝트이므로, 이후 예제에서는 프로젝트를 새롭게 생성할 필요가 없다. 만일 개인적으로 관리하고 싶은 소스들이 있다면 별도의 프로젝트를 만들어 사용하면 된다.

소스 폴더 설정하기

다음 단계는 프로젝트에서 사용할 소스 폴더를 지정하는 부분이다. 특별한 경우가 아니면 기본 설정 값인 [src] 폴더를 그냥 사용한다. 컴파일된 자바 클래스는 [build₩classes] 폴더에 위치하는데, 여기서 말하는 소스는 자바 프로그램에 대한 소스 파일 위치를 말하는 것으로,

JSP 파일과는 달리 순수한 자바 코드로 작성된 소스 파일의 위치다.



[그림 3-8] 소스 폴더 설정

웹 모듈 설정하기

마지막 단계는 웹 모듈에 대한 설정이다. 여기서 설정하는 정보들은 우리가 개발한 웹 애플리케이션을 톰캣에 등록하기 위한 항목이다. 즉 톰캣에는 여러 애플리케이션(프로젝트)이 동시에 등록되어 실행될 수 있으므로, 이를 구분하기 위한 설정이라 볼 수 있다. 더욱 자세한 내용은 '9장. 웹 애플리케이션 아키텍처'에서 살펴볼 것이다.

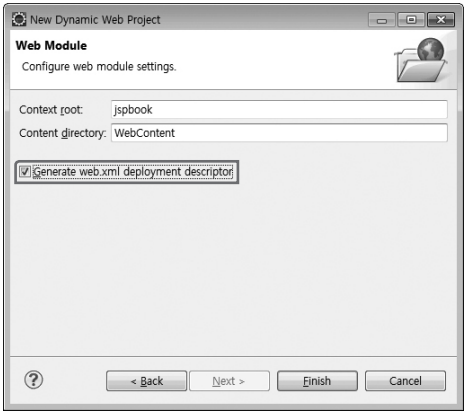
■ Context Root

웹 애플리케이션의 메인 접속 경로를 설정하는 곳이다. jspbook이라고 설정할 경우 브라우저를 통한 접속 경로(URL)는 http://localhost:8080/jspbook이 된다. 웹 애플리케이션은 하나의 컨텍스트로 관리되며, 동일 컨텍스트 내에서만 정보가 공유되는 특징이 있다. 보통 큰 단위의 애플리케이션군으로 컨텍스트를 설정한다. 예를 들어 포털의 경우 카페, 블로그, 메일 등을 별도 컨텍스트로 분리한다고 볼 수 있다.

■ Content Directory

웹 애플리케이션 디렉터리 구조에서 JSP, HTML, 그림 파일 등 콘텐츠가 위치할 디렉터리를 설정한다. 기본 값인 [WebContent] 폴더를 사용한다. 실제 이클립스 프로젝트상에는 여러 폴더가 있지만, 그중 [WebContent] 폴더에만 웹을 통해 서비스되는 내용들이 위치한다고 보면 된다.

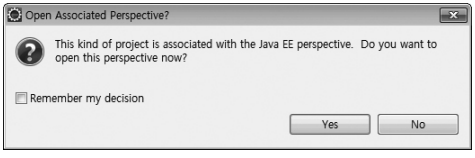
웹 모듈을 설정할 때 Generate web.xml deployment descriptor 항목은 체크해주도록 한다. web.xml 파일은 웹 애플리케이션과 관련된 설정 값을 기록하는 파일로써, 서블릿, 필터, 리스너, 초기화 매개변수 등의 정보를 기록한다. web.xml과 관련된 사항은 이후에 다시 살펴볼 것이다.



[그림 3-9] 웹 모듈 설정

퍼스펙티브 변경하기

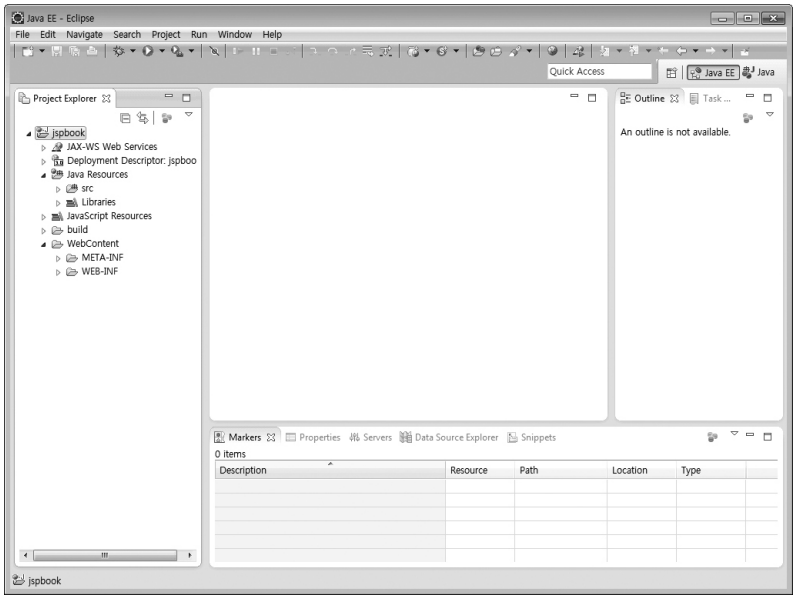
〈Finish〉 버튼을 누르면 퍼스펙티브 전환과 관련된 대화창이 나온다. 〈Yes〉 버튼을 눌러 Java EE 퍼스펙티브로 변경해주도록 한다. 퍼스펙티브는 앞서 잠시 설명한 것처럼 이클립스 뷰를 특정 목적의 개발에 적절한 형태로 배치한 것을 말한다. Java EE 퍼스펙티브는 웹 개발에 가장 적합한 퍼스펙티브다.



[그림 3-10] 퍼스펙티브 변경

생성된 프로젝트 확인하기

[그림 3-11]과 같이 프로젝트 탐색기에 jspbook 프로젝트가 나타나면 정상적으로 생성된 것이다.



[그림 3-11] 생성된 프로젝트 확인

지금까지 이클립스에서 웹 애플리케이션을 개발하기 위한 프로젝트 생성을 살펴보았다. 앞서 언급했듯이 이 책에서는 jspbook 프로젝트에 폴더 단위로 각 장별 소스를 관리할 것이다. 개인적인 연습이나 소스 관리를 할 때는 별도의 프로젝트를 만들어 사용하도록 한다.

❷ Hello World 프로그램 소스 작성

이제 생성된 프로젝트에서 간단한 JSP 프로그램을 작성해보자. 일반 편집기의 경우 모든 코드를 손으로 입력해야 하지만, 이클립스에서는 많은 부분의 코드가 자동으로 생성된다. 이런 방식은 장점도 있지만, 별로 필요 없는 복잡한 코드를 함께 만들어낸다는 문제점도 동반한다.

이클립스에서 소스를 작성하는 기본 과정은 항상 동일하므로, 3장과 4장에서만 상세하게 설명하고, 이후에는 소스만 설명하겠다. 처음 접하게 될 JSP 프로그램은 오늘 날짜와 시간을 나타내는 간단한 Hello World 프로그램이다.

폴더 생성

먼저 프로젝트 탐색기의 [WebContent] 폴더를 선택하고 〈마우스 오른쪽〉 버튼을 눌러 [New] →[Folder]를 실행하고 Folder name에 ch03을 입력해 [ch03] 폴더를 만든다.

[그림 3-12] ch03 폴더 생성

JSP 생성

생성된 [ch03] 폴더에서 다시 〈마우스 오른쪽〉 버튼을 클릭하고 [New]→[JSP File]을 선택한다. 만일 JSP가 보이지 않으면 [Other]→[Web]→[JSP File]을 선택하면 된다.

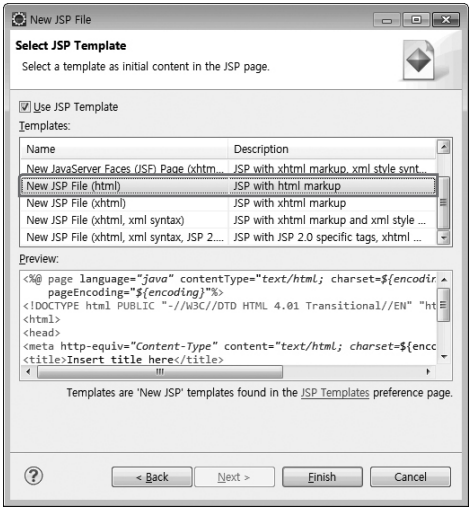
❶ JSP 파일 이름 지정하기

[그림 3-13]과 같은 화면에서 파일 이름을 HelloWorld.jsp라고 입력한다. 반드시 공백 없이 붙여서 대소문자를 구분해서 입력해야 한다. 이때 폴더 선택창에 ch03이 선택되어 있는지도 확인해본다.

[그림 3-13] JSP 파일 이름 지정

❷ 템플릿 코드 지정하기

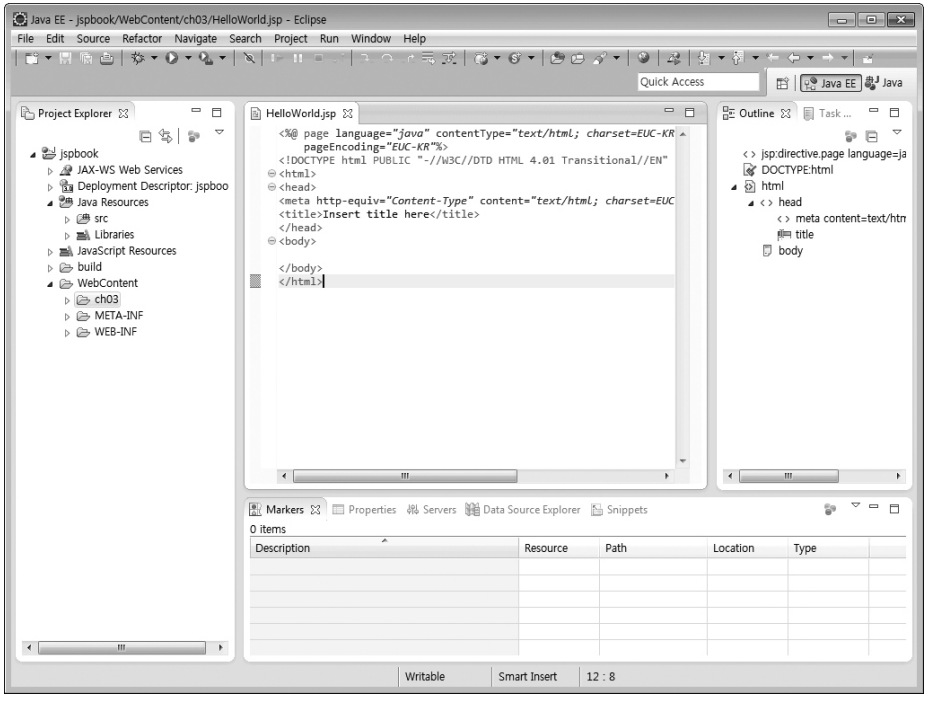
〈Next〉 버튼을 누르면 기본으로 생성할 템플릿 코드를 지정하는 화면이 나온다. 여러 방식의 템플릿을 사용할 수 있으며, 환경설정에서 자신만의 템플릿도 만들어서 관리를 할 수 있다. 앞으로 특별한 언급이 없는 한 기본 항목인 New JSP File(html) 템플릿을 이용한다.



[그림 3-14] 템플릿 선택

❸ 생성된 코드 확인하기

〈Finish〉 버튼을 누르면 [그림 3-15]와 같이 기본으로 생성된 코드를 확인할 수 있다. 오른쪽 [아웃라인 뷰(outline view)]에는 HTML 태그들이 트리 구조로 구조화되어 있다. '아웃라인 뷰'는 코드가 길어질 경우, 원하는 위치로 이동하거나 전체 구조를 살펴보는 데 도움이 된다.



[그림 3-15] 생성된 기본 코드

저자 한마디

가독성을 담보한 핵심 코드 작성

이 책에서는 소스의 가독성을 위해 가급적 핵심 코드를 중심으로 담았다. 따라서 책에 있는 소스코드의 page 지시어 부분과 HTML의 〈HEAD〉 〈/HEAD〉 태그 내의 내용이 실제 이클립스에서 생성한 코드 사이에 약간의 차이가 있을 수 있다. 그러나 이들 내용은 대부분 자동으로 작성되는 부분이므로 실제 실행에는 아무런 문제가 없다. 따라서 책의 소스와 이클립스 코드가 조금 다르다고 혼란스러워할 필요가 없으며, 책과 동일하게 이클립스 코드를 수정할 필요도 없다. 앞으로는 해당 부분에 추가적인 설명이 필요한 경우에만 관련 사항을 다루도록 한다.

소스코드 작성

기본으로 생성된 코드에 다음과 같은 내용을 추가한다.

예제 3-1 헬로월드(HelloWorld.jsp)

```
01 <%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
02 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.
    w3.org/TR/html4/loose.dtd">
03 <HTML>
04 <HEAD>
05 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
06 <TITLE>HelloWorld</TITLE>
07 </HEAD>
08 <BODY>
09 <center>
10 <H2>HelloWorld : 헬로월드</H2>
11 <HR>
12 현재 날짜와 시간은 : <%=new java.util.Date() %>
13 </center>
14 </BODY>
15 </HTML>
```

소스코드 분석

처음 코드를 보면 "어? 그냥 HTML이네." 하고 생각할 수 있다. 1행과 12행을 제외하면 전부 HTML 코드기 때문이다.

01행: page 지시어로 현재 JSP 페이지 형식 등을 지정한다.

page 지시어는 JSP 파일에 기본적으로 들어가야 하는 내용인데, 2부에서 배우게 될 것이므로 여기서는 자세한 설명은 생략한다. 한글 설정이 UTF-8로 나오는지 확인하고, 만일 EUC-KR로 나오면 2장에서 설명한 한글 인코딩과 관련된 설정 마지막 부분을 참고해서 수정한 후 파일을 다시 생성해야 한다. 물론 텍스트를 직접 수정해도 되지만, 향후 자동 생성되는 코드를 위해 Preference를 설정해주는 것이 좋다.

12행: 오늘 날짜와 시간을 출력하는 부분이다.

사실 예제에서의 JSP 코드는 〈%= %〉 부분이 전부며, 현재 시간과 날짜를 출력하는 코드로 이루어져 있다. 여기서 〈%= %〉를 'JSP의 표현식'이라고 하는데, 문자열 형태로 출력할 수 있는 간단한 내용의 출력 구문을 말한다. 코드의 내용은 Date 클래스의 인스턴스를 생성한 다음 출력하는 것으로, 자바 프로그램에서는 콘솔에 결과가 출력되지만, JSP에서는 웹 브라우저에 결과가 출력된다는 데 그 차이가 있다.

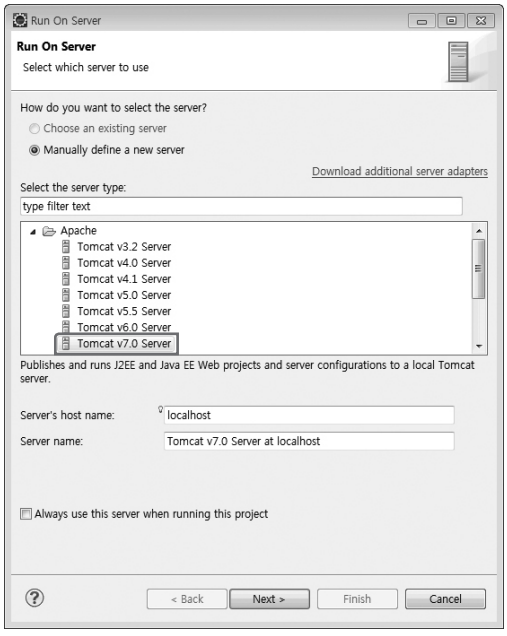
❸ 서버 설정 및 실행

앞서 배운 것처럼 JSP는 서버 기반의 웹 프로그래밍 기술이므로 JSP 프로그램을 실행하려면 반드시 서버가 있어야 한다. 서버 설정은 처음 한 번만 해두면 되고, 만일 나중에 문제가 되어 서버 설정을 삭제하거나 톰캣을 다시 설치하는 등 변화가 있을 경우에는 별도로 추가하거나 변경할 수 있다.

이클립스가 아닌 편집기에서 작업한 경우라면 톰캣을 실행하고 웹 브라우저에서 URL을 입력해 결과를 확인할 수 있다. 그러나 이클립스의 경우 자체적으로 톰캣을 관리하기 때문에 별도로 실행할 필요가 없다.

서버 설정하기

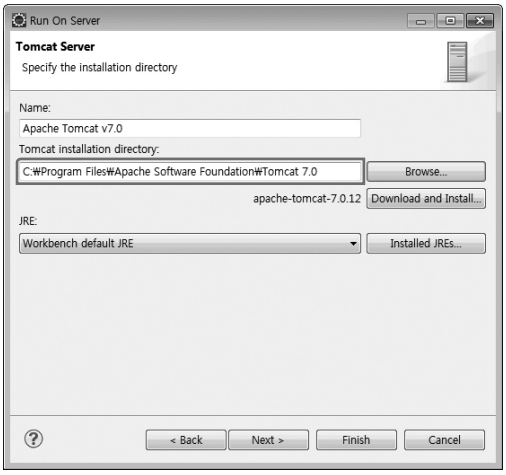
프로젝트 탐색기의 [ch03] 폴더 아래에 있는 HelloWorld.jsp 파일을 선택하고 〈마우스 오른쪽〉 버튼 클릭해 [Run As]→[Run on Server]를 선택하면 [그림 3-16]과 같이 서버 설정을 위한 화면이 나온다. [Apache]→[Tomcat v7.0 Server]를 선택한다. 화면을 보면 짐작이 가겠지만 이클립스에서는 다양한 회사의 서버 제품군을 지원한다. 따라서 굳이 톰캣이 아니더라도 서버만 제대로 설정되어 있다면 실습에는 아무런 문제가 없다. 〈Next〉 버튼을 누른다.



[그림 3-16] 서버 설정

톰캣 폴더 지정하기

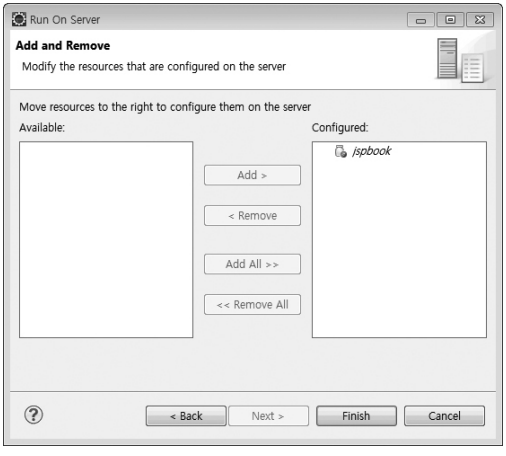
다음 화면에서는 톰캣이 설치된 폴더를 지정한다. 〈Browse〉 버튼을 눌러 톰캣이 설치된 폴더를 지정한다. 톰캣 기본 설치 경로는 [컴퓨터]→[C드라이브]→[Program Files]→[Apache Software Foundation₩Tomcat 7.0]이다. 직접 입력하지 말고 반드시 〈Browse〉를 이용해 폴더를 찾아 선택하고 〈Next〉 버튼을 누른다.



[그림 3-17] 톰캣 폴더 지정

실행할 프로젝트 선택하기

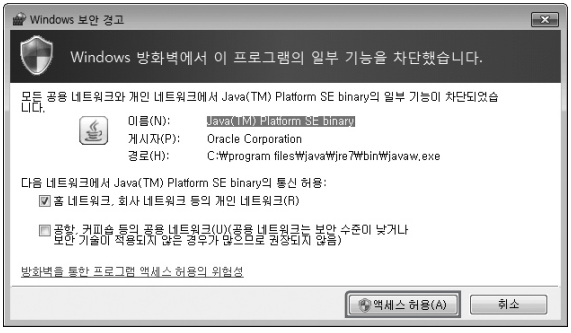
다음 화면은 실행에 포함시킬 프로젝트, 즉 웹 애플리케이션을 선택하는 화면이다. 기본적으로 현재 실행하는 프로젝트는 [Configured projects] 목록에 있다. 만일 여러 웹 애플리케이션 프로젝트가 있다면 이곳에서 추가/삭제를 할 수 있다. 예를 들어 3개의 프로젝트가 있는데, 매번 테스트를 위해 현재 테스트가 필요없는 나머지 프로젝트를 동시에 실행시킬 필요는 없다. 이 경우 필요 없는 프로젝트는 [Remove]한 다음 실행하면 더욱 빠르게 실행할 수 있다.



[그림 3-18] 실행 프로젝트 선택

보안 경고 해제하기

〈Finish〉 버튼을 누르면 톰캣이 시작되고, HelloWorld.jsp가 실행된다. 이때 윈도우 보안 경고가 나올 수 있는데 〈액세스 허용〉 버튼을 클릭한다.

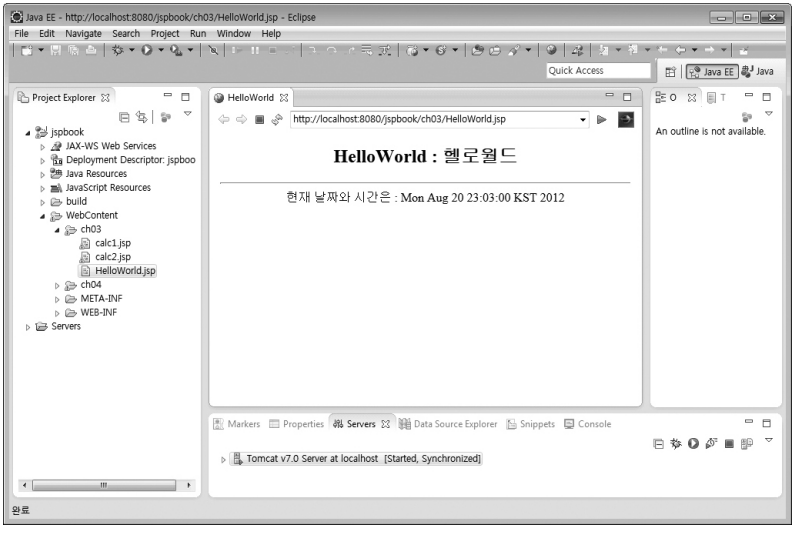


[그림 3-19] 윈도우 보안 경고

정상적인 경우 [그림 3-20]와 같이 이클립스에 포함된 웹 브라우저가 실행되면서 결과가 출력된다.

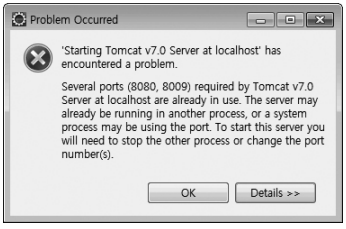
실행 결과 확인하기

이상이 없다면 다음과 같은 실행 결과를 확인할 수 있다.



[그림 3-20] 실행 결과 확인

만약 [그림 3-21]과 같은 톰캣 포트 충돌 오류가 발생하는 경우에는 윈도우에 이미 톰캣이 실행되어 있는 것이니, 윈도우 트레이에 있는 톰캣 아이콘을 클릭해 실행되어 있는 톰캣을 종료한다. 앞서 말한 대로 톰캣을 설치할 때는 Starup Type이 Manual로 되어 있어야 윈도우가 시작될 때 톰캣이 자동으로 시작되지 않는다.



[그림 3-21] 톰캣 포트 충돌 오류

한편 이클립스에 포함된 웹 브라우저를 이클립스 [Window]→[Web Browser]에 등록된 다른 웹 브라우저로 변경할 수도 있다. 톰캣이 실행된 상태에서 별도의 웹 브라우저(인터넷 익스플로러, 파이어폭스, 크롬 등)를 실행하고 http://localhost:8080/jspbook/ch03/HelloWorld.jsp를 입력해도 동일한 결과를 확인할 수 있다. 그러나 이클립스에서 톰캣을 실행하지 않은 상태에서는 확인할 수 없으니 주의하자.

[그림 3-22] 외부 브라우저(크롬)를 이용한 실행 결과

너무 간단한 예제라 JSP라는 느낌이 안 왔을지도 모르겠다. 하지만 JSP 개발을 위한 이클립스 사용 방법과 JSP의 기본적인 구조 정도는 충분히 경험했다고 봐도 된다.

민감한 독자라면 HelloWorld.jsp를 처음 실행할 때 조금 느리다는 느낌을 받을 수 있는데, 브라우저를 종료하고 다시 접속하거나 다른 컴퓨터에서 접속하면 다시 빨라진다. 처음 톰캣을 시작할 때 시간이 소요되는데, 그 다음 JSP 파일이 요청될 경우 톰캣에서 해당 JSP에 대한

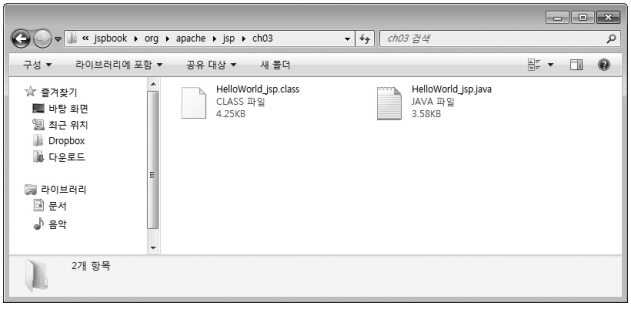
서블릿이 존재하는지 확인하고, 존재하지 않을 경우 JSP를 서블릿으로 변환한 후(.java 파일 생성), 생성한 서블릿을 컴파일하여(.class 파일 생성) 결과를 전송하기 때문이다.

앞서도 설명했지만, 서블릿으로 변환된 JSP는 재컴파일되지 않고 서블릿에서 서비스하기 때문에 빠른 속도가 보장된다. 소규모에서는 성능이 다소 떨어진다고 느낄 수 있지만, 사용자가 많아져도 성능 저하가 크지 않기 때문에 대형 사이트를 구축할 때는 대부분 JSP를 사용하고 있다.

❹ 서블릿으로 변환된 소스 확인

2절에서 JSP가 서블릿으로 변환된다고 했는데, 정확하게는 일반적인 서블릿 코드의 형태는 아니고 톰캣에서 변환한 서블릿의 형태라고 이해하는 것이 좋겠다. 그렇다면 서블릿은 어떻게 생긴 걸까? 간혹 더욱 정확한 디버깅을 할 때 서블릿으로 변환된 소스를 접할 수 있으므로, 이에 대해 알아두는 것이 좋다. 여기서는 톰캣에 의해 변환되어 생성된 서블릿 파일과 위치만 살펴보고 서블릿으로 변환된 소스에 대해서는 '6장. JSP 내장객체'에서 자세히 다룰 것이다.

서블릿으로 변환된 소스는 [c:₩dev₩workspace₩.metadata₩.plugins₩org.eclipse.wst.server.core₩tmp0₩work₩Catalina₩localhost₩jspbook₩org₩apache₩jsp₩ch03] 폴더에 위치한다(경로상 폴더 [tmp0]의 경우 숫자가 바뀔 수도 있으며 c:₩dev₩workspace는 각자 설정한 디렉터리에 따라 다르다). [그림 3-23]과 같이 HelloWorld_jsp.java 파일과 클래스를 확인할 수 있는데, 자바 파일을 일반 편집기로 열면 작성한 JSP 파일 내용이 서블릿 형태로 바뀐 것을 볼 수 있다.



[그림 3-23] JSP가 변환된 자바 파일이 있는 폴더

서블릿으로 변경된 소스는 JSP 태그 이외에 프로그램적인 요소에서 오류가 발생했을 때도 유용하게 활용할 수 있다. 그 이유는 웹 페이지에 표시되는 소스코드 행이 JSP 소스코드 행이 아니라 서블릿으로 변경된 소스코드 행이기 때문이다. 한편, JSP 파일을 변경했지만 혹시라도 캐시 문제로 인해서 이전 내용이 계속 출력되거나 문제가 있을 경우에 자동 생성된 서블릿 파일을 지우면 JSP가 새로 컴파일되어 확실하게 새로운 내용을 적용시킬 수 있으므로 위치를 기억해두면 여러모로 도움이 된다. 한 번에 서블릿을 제대로 이해하려고 노력하지 않아도 된다. 이 소스는 이후에 서블릿을 살펴본 다음 다시 한 번 살펴볼 것이다.

요약

1 JSP란

❶ JSP는 Java Server Page의 약자로서, 자바를 기반으로 만들어진 웹 프로그래밍 기술이다. 구조적으로는 서블릿(Servlet)에 기반하고 있으며 내부적으로 JSP가 서블릿 형태로 변환되는 구조다.

❷ JSP는 HTML처럼 화면 위주의 프로그래밍을 지원하며, 스크립트릿, 빈즈 액션 등 자바 프로그램과 연동할 수 있는 구조로 되어 있다.

❸ JSP를 학습하려면 HTML, 자바스크립트, CSS와 같은 기본 기술들과 다양한 자바 관련 응용 기술이 요구된다.

2 JSP 내부 동작 구조

❶ JSP는 자바 서블릿 기반의 웹 프로그래밍 기술로서, JSP 소스 자체로는 실행할 수 없는 상태다.

❷ 톰캣은 서블릿 컨테이너로서, JSP를 서블릿 형태의 자바 소스로 변환하고 자바 클래스로 컴파일하고 서블릿으로 로딩한 다음 사용자 요청에 응답하는 구조다.

❸ 이때 한번 컴파일된 JSP는 소스 수정 전까지 다시 컴파일되지 않고 메모리상에 적재된 서블릿으로 처리된다.

❹ 개발자가 작성한 JSP 소스는 변환된 소스의 _jspService() 메서드에 들어간다.

3 JSP 프로그래밍 기술 변천

❶ 초기 자바 기반 웹 프로그램은 서블릿 중심이었으나, 웹이 화면의 중심이라는 특성으로 인해 프로그램 내에서 화면을 처리하는 데 여러 문제를 야기한다.

❷ 이후 화면을 중심으로 자바 프로그램을 연동할 수 있는 JSP가 등장하게 되었으나 JSP 내에서 과도한 스크립트릿을 사용하는 문제로 프로그램 개발 구조의 변화가 요구된다.

❸ MVC 패턴은 웹 프로그램을 역할에 따라 기술 요소를 구분해서 구현하는 프로그래밍 모델로 이후 웹 프로그램 개발의 기본 모델로 정착되었다.

4 MVC 패턴

❶ MVC는 Model-View-Controller의 약자로 프로그램의 전체 구성요소를 데이터, 화면, 컨트롤의 세 가지 기능적 요소에 초점을 둔 프로그래밍 방식이다.

❷ MVC 패턴에 따르면 웹 프로그램의 경우 Model은 자바 클래스로 구현되며 보통 DO, DAO와 같은 데이터 처리 클래스들로 구성된다.

❸ View는 화면 처리와 관련된 것으로 기본적으로 JSP 영역이며 HTML, CSS 외의 표현 언어, 커스텀 태그 등이 사용된다.

❹ 스크립트릿, 〈jsp:useBean〉 등 JSP 페이지에서 프로그램을 다루는 부분에서는 사용을 권장하지 않는다.

❺ 컨트롤러(Controller)는 모델(Model)과 뷰(View)를 연결하는 역할을 수행한다. 사용자 요청을 받아들이고 데이터를 처리한 후 결과를 뷰에 전달한다.

❻ MVC 패턴은 프로그램 모델이므로 직접 구조를 설계해도 되지만 스트러츠 프레임워크 혹은 스프링 @MVC 등 검증된 프레임워크를 사용하는 것이 좋다.

[연습문제]

1 웹 브라우저로부터 서블릿 컨테이너에 이르는 JSP 동작 과정을 그림과 함께 설명하시오.

2 JSP 서블릿 컴파일러 처리 과정을 플로우 차트로 설명하시오.

3 MVC 패턴이 나오게 된 배경을 간단히 설명하고 MVC 패턴의 3가지 구성요소가 무엇인지 설명하시오.

4 testprj라는 이름의 Dynamic Web Project를 생성한 뒤 [File]→[New]를 이용해 test.jsp라는 파일을 프로젝트에 추가하시오.