



jsp 1강

양 명 속

[now4ever7@gmail.com]

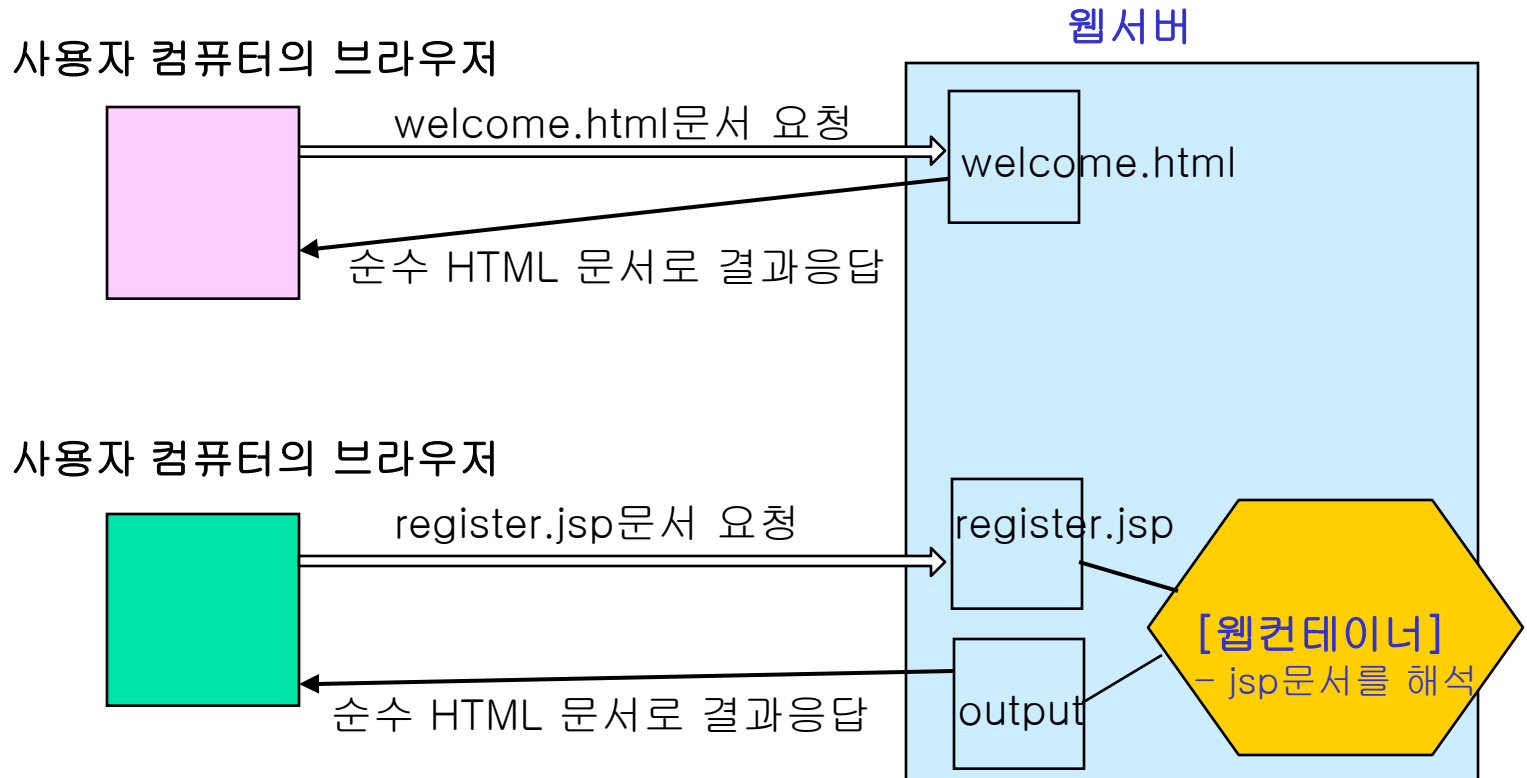


목차

- 개발환경 구축 - 톰캣 설치
- 웹 프로그래밍 기초
 - 서블릿과 JSP
 - 웹 컨테이너
 - 스크립트 방식과 실행코드 방식
- JSP 기본 - JSP 페이지의 구성요소
- 내장 객체
 - request, response, out
- HTTP 프로토콜
- JDBC

동적 웹 페이지

■ 동적 웹 페이지의 JSP

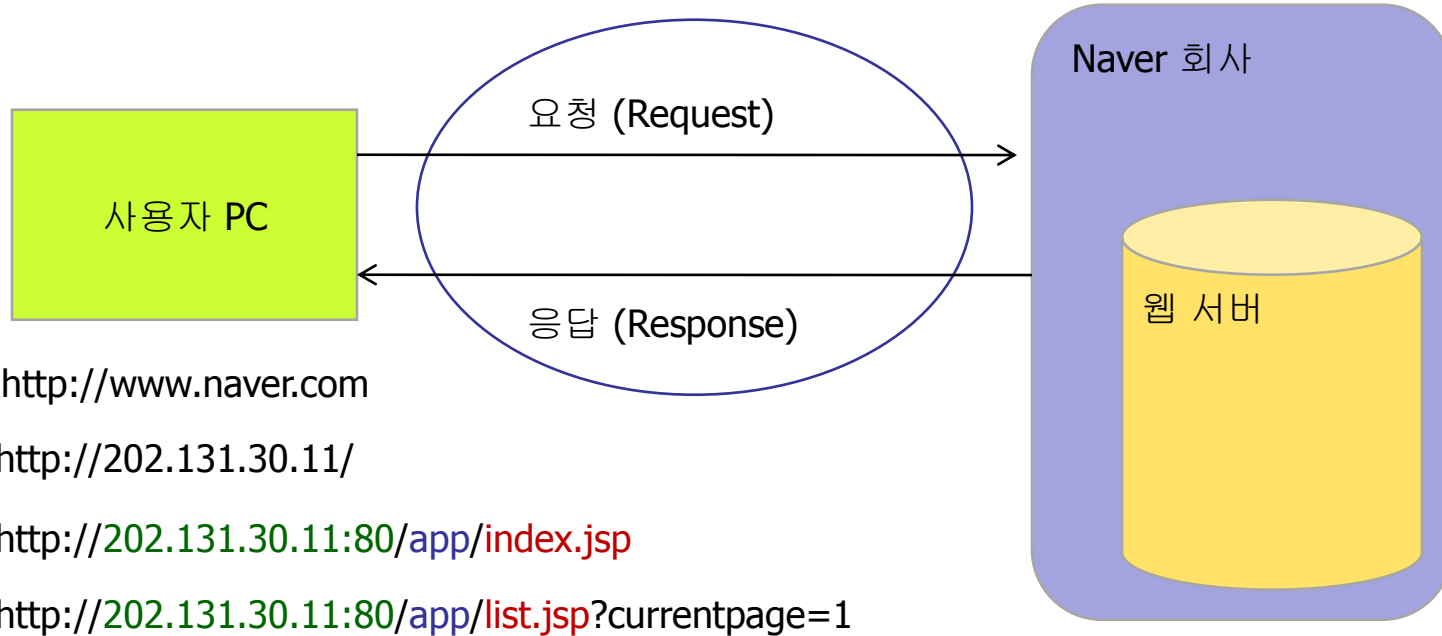


- 웹 서버는 jsp에 대한 요청을 웹 컨테이너로 넘기게 됨
- 이러한 요청을 받은 웹 컨테이너는 해당 jsp 페이지를 찾아서 서블릿(.java 파일 생성)으로 변환하는 파싱(parsing) 과정을 거친 후 컴파일(.class 파일 생성)을 하게 됨 (jsp문서를 해석하고 가공)
- 컴파일된 서블릿(.class)은 최종적으로 웹 브라우저에 응답되어져 사용자는 응답 결과를 보게 됨

웹 페이지 요청

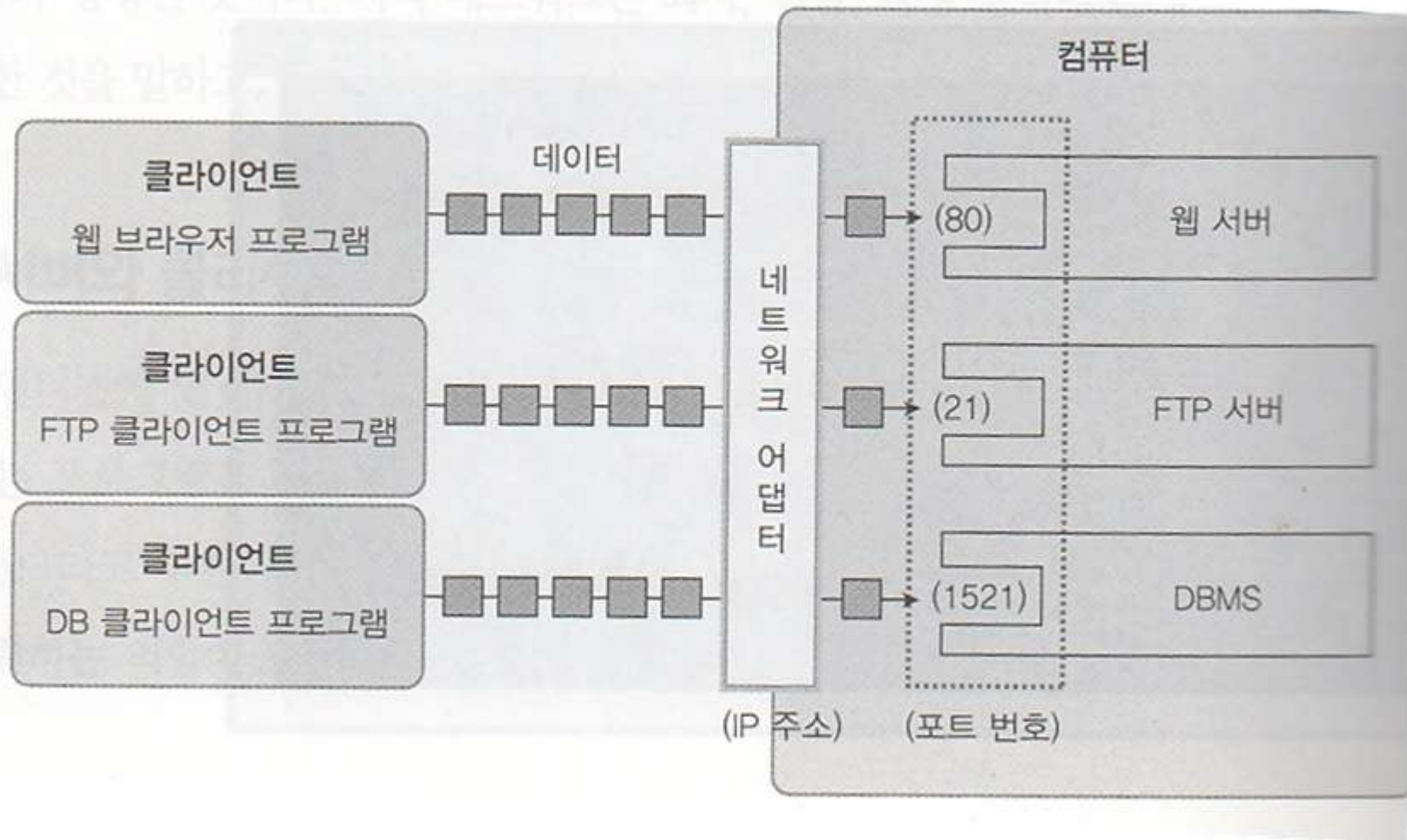
※ port
80 - http (웹 서버)
21 - ftp
25 - smtp

인터넷 망(HTTP)



서버단 : jsp, (database)
클라이언트단(뷰단) : html, javascript

클라이언트/서버 (client/server)



인터넷 주소

• port
21 – ftp, 23 – telnet, 25 – smtp, 110-pop3,
80 – http(웹 서버), 443 - https

- 인터넷 주소 – 세가지 구조로 이뤄져 있음
 - 1.도메인 (아이피)
 - 2.어플리케이션(서버에서 자원의 위치, 폴더)
 - 3.소스페이지(웹 페이지, html, jsp, asp등)
- 예) http://202.131.30.11:80/app/index.jsp?id=hong&age=20
- [1] 아이피 – 서버(컴퓨터)를 찾는 주소
- [2] 포트(PORT) – 호스트(컴퓨터)가 외부와 통신을 하기 위한 통로
 - 컴퓨터 내에 있는 프로그램을 찾는 주소
 - 컴퓨터 안의 프로그램들마다 모두 포트를 가지고 있고, 그 포트를 통해서 이 프로그램에 통신으로 접근
- [3] 어플리케이션 – 웹 프로젝트의 이름(서버에서 자원의 위치, 폴더)
- [4] 소스페이지 – 웹 페이지
- [5] 파라미터 – 이 소스페이지를 호출할 때 넘겨주는 인자값
 - 같은 소스페이지라도 다른 화면을 보여주기 위해 파라미터를 사용



URL 과 웹 어플리케이션 주소

■ URL(Uniform Resource Locator)

- 웹 상에서 서비스를 제공하는 각 서버들에 있는 **파일들의 위치**를 표시하기 위한 것으로 접속할 서비스의 종류, 도메인명, 파일의 위치등을 포함한다
- `http://java.sun.com:80/javase/6/docs/api/index.html`

[프로토콜]://[호스트][:포트][경로][파일명][.확장자][쿼리 문자열]

- 프로토콜 - 서버와 클라이언트가 통신할 때 사용할 프로토콜 입력
 - http, ftp, https 등
- 호스트(도메인) - 클라이언트가 접속할 서버 주소
 - cafe.daum.net 과 같은 호스트 명이나 211.110.9.10 과 같은 ip 주소 사용
- 포트 - 서버와 클라이언트가 통신할 때 사용할 포트
- [경로][파일명][.확장자] - 서버에서 가져올 **자원의 위치**를 입력
- 쿼리 문자열 - 주소 뒤에 추가로 붙는 정보
 - 'parameter' 라고 불리는 데이터를 웹 어플리케이션에 전달할 때 사용됨
 - ? 를 이용하여 경로 부분과 구분
 - 1개 이상의 파라미터 이름과 값을 갖는다.



URL 과 웹 어플리케이션 주소

- 각각의 파라미터는 & 를 이용하여 구분
- 파라미터 이름과 값은 = 을 이용하여 구분

?이름1=값1&이름2=값2&...

http://www.google.com/search?hi=en&q=jsp&aq=f

- **URI** 통합 자원 식별자(Uniform Resource Identifier)

- 존재하는 자원을 식별하기 위한 일반적인 식별자를 규정하기 위한 것으로 URL에서 Http 프로토콜, 호스트명, port번호를 제외한 것

예) <http://java.sun.com/javase/6/docs/api/index.html>

=> URI - </javase/6/docs/api/index.html>



개발환경 구축



웹 프로그래밍 절차

- 1. 개발 환경 구축
- 2. 웹 어플리케이션 코드 개발 및 테스트
- 3. 완성된 웹 어플리케이션을 서비스 환경에 배포
 - 배포(deploy) - jsp/서블릿의 경우 웹 어플리케이션을 war 파일로 묶어서 배포하거나 또는 jsp, 서블릿, 자바 클래스 등의 개별 파일을 배포하기도 함
 - 개발된 결과물을 실제 서비스로 사용되는 서버 장비에 복사하는 과정이 배포 과정



개발환경 구축

- 웹 어플리케이션 개발을 위해 필요한 프로그램
 - JDK – 자바 개발 도구
 - jsp 2.2 표준 – 자바 6 이상의 버전을 필요로 함
 - 웹 컨테이너 – jsp와 서블릿을 실행시켜 주는 컨테이너
 - 톰캣, 제티, GlassFish 등
 - 또는 WAS(Web Application Server)
 - WebLogic, JEUS(국내제작), JBoss 등
 - 코드 편집기
- JDK 설치
 - 환경변수 설정
 - 톰캣과 같은 웹 컨테이너는 JDK 설치 경로를 필요로 함
 - 이때 사용되는 환경 변수의 이름이 JAVA_HOME



개발환경 구축

- 웹 컨테이너
자바해석 + 웹 어플리케이션의 객체 관리
제품) 톰캣(무료), 레진(유료)
- WAS(Web Application Server)
유료) WebLogic(bea사-->오라클인수)
JEUS (국내제작)
무료) JBoss 등

■ 웹 컨테이너 설치

■ 웹 어플리케이션을 실행시켜 줄 웹 컨테이너 설치

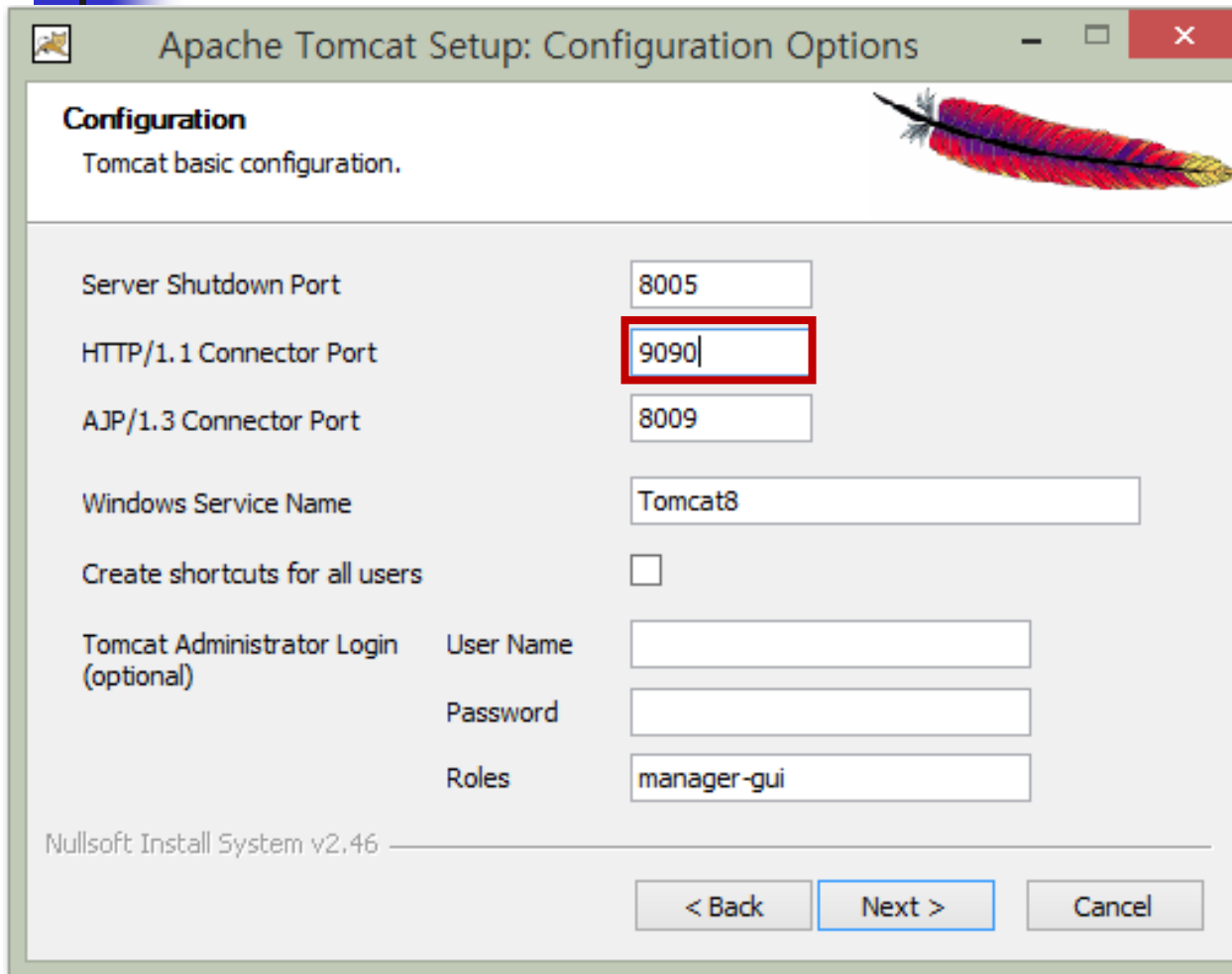
- 톰캣 - 오픈 소스 컨테이너
- 레진 - 상용 컨테이너

■ 톰캣(Tomcat) 9 설치 및 환경 구축하기

- 톰캣이 웹서버 기능도 함
- jsp 2.2 를 지원하는 톰캣 버전은 7.0 이상의 버전
- <http://tomcat.apache.org> 에서 다운
 - installer 다운받기
 - 32-bit/64-bit Windows Service Installer (pgp, md5)
 - 8080 포트는 오라클에서 사용하므로 설치시 port 를 9090으로

c:\ Tomcat 9.0 에 설치

톰캣 설치

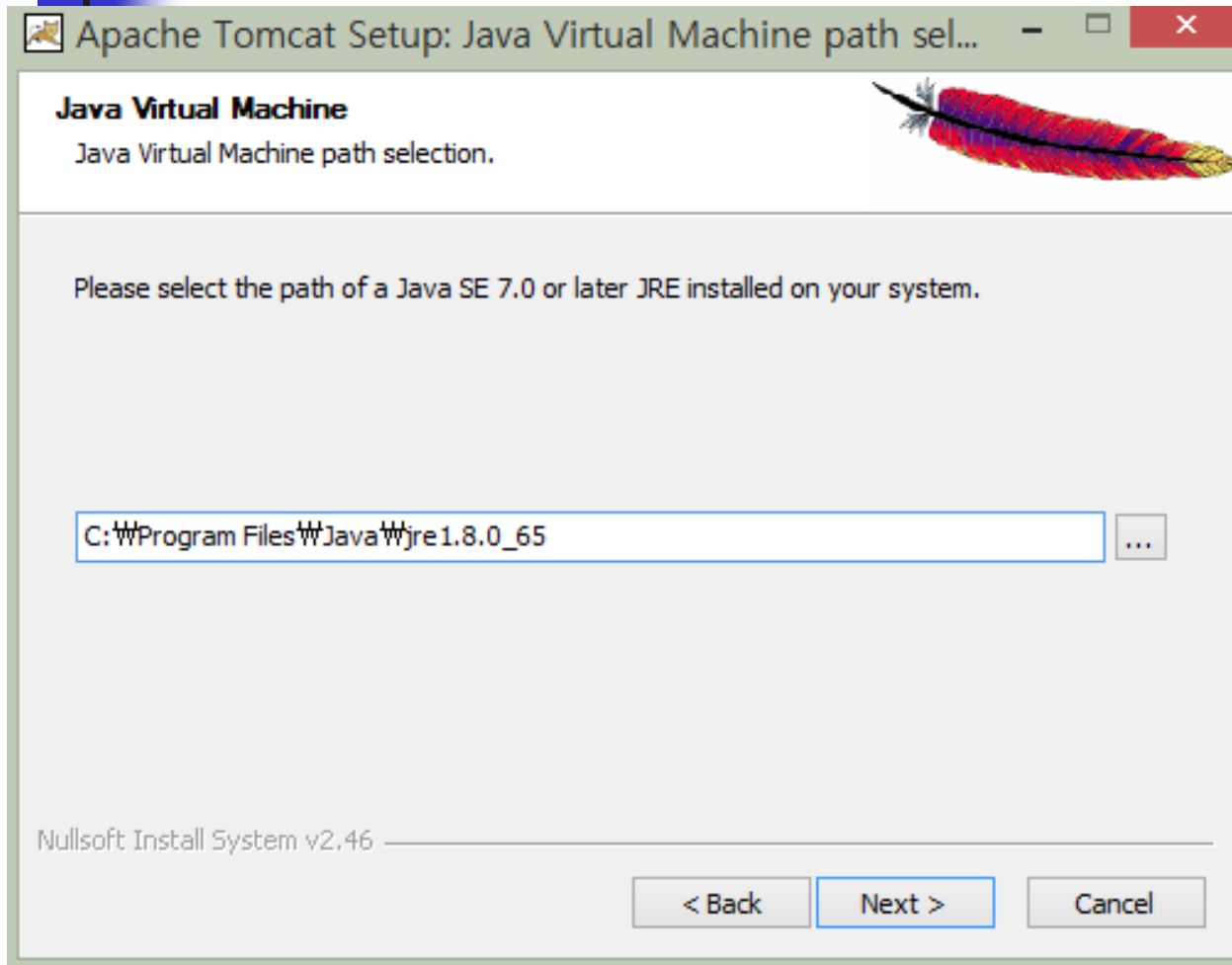


The image shows the 'Apache Tomcat Setup: Configuration Options' window. It has a title bar with a standard Windows icon, a minus button, a maximize button, and a close button. The main content area is titled 'Configuration' and contains the text 'Tomcat basic configuration.' followed by a decorative feather graphic. Below this, there are several configuration options with text labels and input fields:

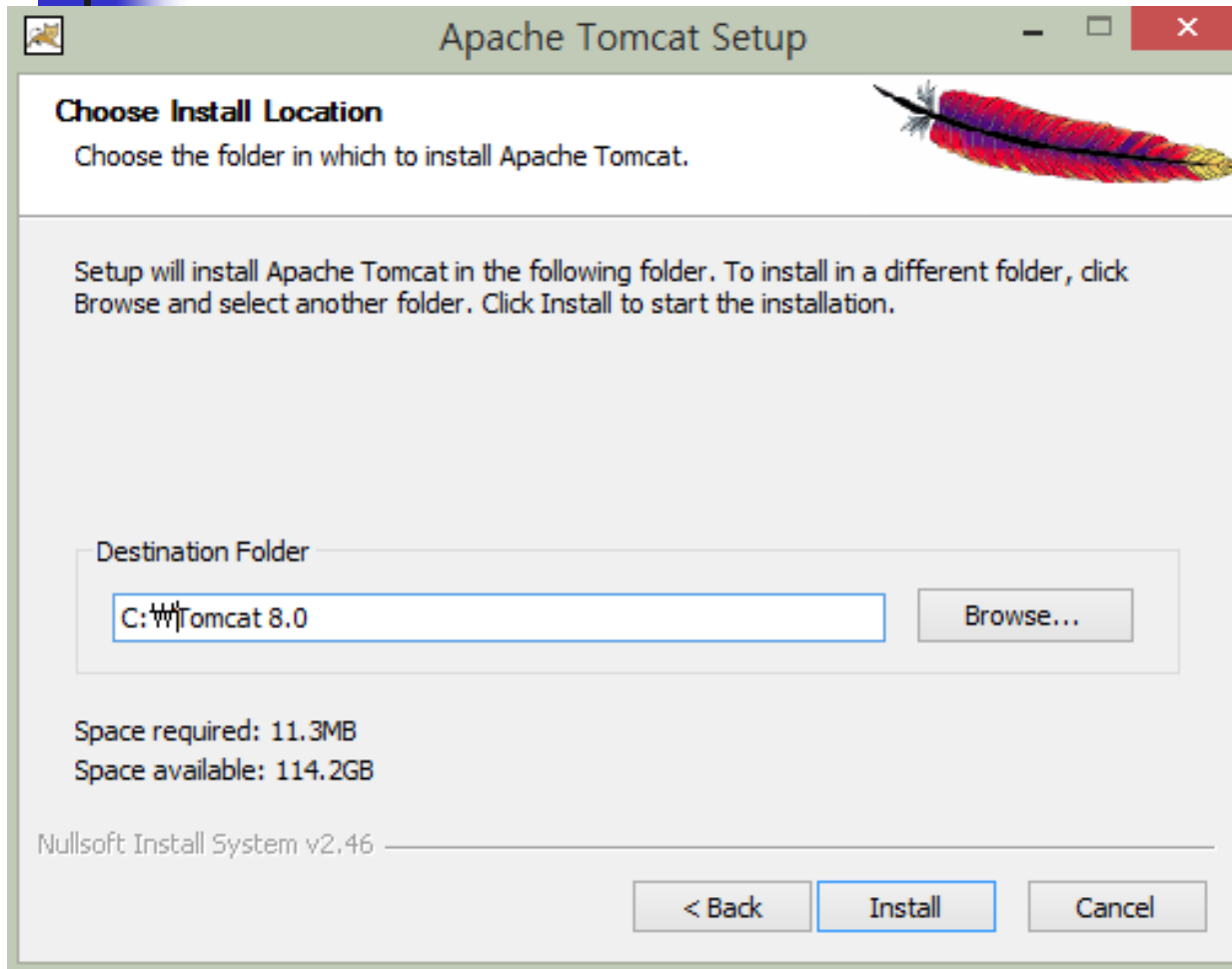
- Server Shutdown Port: 8005
- HTTP/1.1 Connector Port: 9090 (highlighted with a red box)
- AJP/1.3 Connector Port: 8009
- Windows Service Name: Tomcat8
- Create shortcuts for all users: ☐
- Tomcat Administrator Login (optional):
 - User Name:
 - Password:
 - Roles: manager-gui

At the bottom left, it says 'Nullsoft Install System v2.46'. At the bottom right, there are three buttons: '< Back', 'Next >' (highlighted with a blue border), and 'Cancel'.

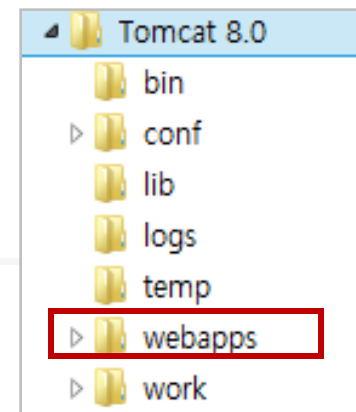
톰캣 설치



톰캣 설치



개발환경 구축



- 톰캣 설치 후 디렉토리
 - bin - 톰캣을 실행하고 종료시키는 스크립트(.bat, .sh) 파일이 위치해 있음
 - conf - server.xml 파일을 포함한 톰캣 설정 파일이 위치
 - lib - 톰캣을 실행하는 데 필요한 라이브러리(jar) 파일이 위치
 - logs - 톰캣 로그파일이 위치
 - temp - 톰캣이 실행되는 동안 임시 파일이 위치
 - **webapps** - 웹 어플리케이션이 위치
 - work - 톰캣이 실행되는 동안 사용되는 작업 파일이 위치
- 웹 어플리케이션은 기본적으로 webapps 디렉토리에 배포됨
 - server.xml 파일을 사용해서 웹 어플리케이션의 경로를 변경할 수 있음

개발환경 구축

• jdbc 오라클 드라이버 복사
C:\app\yang\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar를
c:\java\jdk1.8.0_51\jre\lib\ext와
c:\Tomcat 9.0\lib에 붙여넣기

- 톰캣 설치 후 환경변수 설정
 - 1) JAVA_HOME : JDK 설치 디렉토리
 - c:\Wjava\Wjdk1.8.0_51
 - 2) CATALINA_HOME : 톰캣 설치 디렉토리.
 - 설정하지 않은 경우, 현재 디렉토리를 값으로 사용함
 - c:\WTomcat 9.0
 - 3) PATH 편집
 - 변수명 - path
 - 변수값 - %JAVA_HOME%\bin;%CATALINA_HOME%\bin
 - 4) CLASSPATH 편집 : servlet-api.jar와 jsp-api.jar 추가
 - 변수명 - CLASSPATH
 - 변수값 - c:\Wjava\Wjdk1.8.0_51\Wjre\Wlib\Wext\Wojdbc6.jar;
c:\WTomcat 9.0\Wlib\Wservlet-api.jar;
c:\WTomcat 9.0\Wlib\Wjsp-api.jar;%classpath%;



개발환경 구축

- 5) 웹브라우저에서 테스트
 - 톰캣이 시작된 상태에서 웹 브라우저를 실행하고 주소에 <http://localhost:9090> 입력
 - 또는 <http://localhost:9090/index.jsp>
- 톰캣 실행
 - [시작] - [모든 프로그램] - [Apache Tomcat 9.0] - [Monitor Tomcat] 실행

환경 설정

```
• context.xml
<?xml version='1.0' encoding='utf-8'?>
<Context reloadable="true" privileged="true">
</Context>
```

- 6) servlet-api.jar 를 복사하여
c:\Java\jdk1.8.0_51\jre\lib\ext 에 붙여넣기
- 7) 오라클 드라이버 ojdbc6.jar를 복사하여
c:\Java\jdk1.8.0_51\jre\lib\ext 와
c:\Tomcat 9.0\lib 에 붙여넣기
- 8) c:\Tomcat 9.0\conf\context.xml 파일을 열어서
<Context>를 <Context reloadable="true">로 변경
 - 컨텍스트 변경시 자동으로 재로딩되도록 설정
privileged="true" 추가
 - 톰캣 6.x 버전부터는 서블릿 리로딩에 관련된 추가적인 설정을 해 주어야
Tomcat 시작 시 에러가 발생하지 않으며, 서블릿도 정상적으로 실행된다.
- 9) c:\Tomcat 9.0\conf\server.xml 파일을 열어서 포트 변경
8080 -> 9090으로
(server.xml 파일에 한글 주석 달면 톰캣이 동작 안함)



환경 설정

톰캣 6.0 에서는

10) c:\ Tomcat 6.0\ conf\ web.xml 파일을 열어서
invoker 라는 키워드를 찾아보면 아래 두 가지 부분이 주
석처리가 되어 있는데, 이를 해제해줘야 함

- web.xml

```
<servlet>
  <servlet-name>invoker</servlet-name>
  <servlet-class>org.apache.catalina.servlets.InvokerServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>

<!-- The mapping for the deprecated invoker servlet -->
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

- 보안상 사용자가 정의하는 서블릿은 기본적으로 막혀있기 때문에 서블릿을 개발할 경우에는 이
들 주석을 제거해야 함
- 설정을 바꾼 후 톰캣 켜다 켜기

- 톰캣이 기본적으로 제공하는 웹 화면

Apache Tomcat/9.0.27



If you're seeing this, you've successfully installed Tomcat. Congratulations!



Recommended Reading:

[Security Considerations How-To](#)

[Manager Application How-To](#)

[Clustering/Session Replication How-To](#)

[Server Status](#)

[Manager App](#)

[Host Manager](#)

Developer Quick Start

[Tomcat Setup](#)

[First Web Application](#)

[Realms & AAA](#)

[JDBC DataSources](#)

[Examples](#)

[Servlet Specifications](#)

[Tomcat Versions](#)

Managing Tomcat

For security, access to the [manager webapp](#) is restricted. Users are defined in:

`$CATALINA_HOME/conf/tomcat-users.xml`

In Tomcat 9.0 access to the manager application is split between different users.

[Read more...](#)

[Release Notes](#)

[Changelog](#)

Documentation

[Tomcat 9.0 Documentation](#)

[Tomcat 9.0 Configuration](#)

[Tomcat Wiki](#)

Find additional important configuration information in:

`$CATALINA_HOME/RUNNING.txt`

Developers may be interested in:

[Tomcat 9.0 Bug Database](#)

Getting Help

[FAQ](#) and [Mailing Lists](#)

The following mailing lists are available:

[tomcat-announce](#)

Important announcements, releases, security vulnerability notifications. (Low volume).

[tomcat-users](#)

User support and discussion

[taglibs-user](#)

User support and discussion for [Apache Taglibs](#)

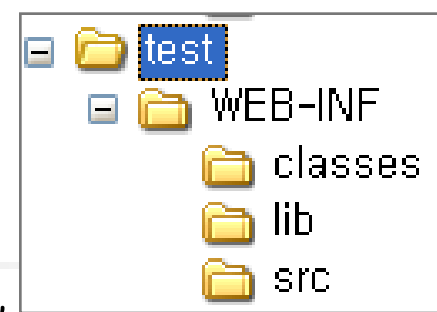
[tomcat-dev](#)



웹 어플리케이션 개발 시작

- 자바에서 웹 어플리케이션 개발
 - 스크립트 방식의 **jsp** 와
 - 실행코드 방식의 **서블릿**을 이용
- 웹 어플리케이션 디렉토리 생성하기
 - 웹 어플리케이션 코드를 보관할 디렉토리가 필요
- 웹 어플리케이션 디렉토리를 만드는 방법
 - [1] [웹 컨테이너 디렉토리]/webapps/ 폴더에 하위 폴더를 추가해서 만드는 방법
 - [2] server.xml 파일에 컨텍스트를 추가해서 웹 어플리케이션을 작성하는 방법

웹 어플리케이션 생성



- [1] 웹 컨테이너에 직접 개발 디렉토리를 생성
 - [웹 컨테이너 디렉토리]/webapps/ 아래에 개발 디렉토리 생성
 - c:\WTomcat 9.0\Wwebapps\Wtestsite
 - testsite 폴더의 하위 폴더로 "WEB-INF" 폴더 만들기
 - "WEB-INF" 폴더의 하위 폴더로 "classes" 폴더 만들기
 - web.xml 파일을 복사해서 WEB-INF 폴더에 붙여넣기 - 톰캣6.0
 - testsite 폴더에 jsp 파일 만들기
 - 톰캣 서비스 시작하고, 다음 URL을 통해서 실행
 - <http://localhost:9090/testsite/now1.jsp>
 - '/testsite'=>웹어플리케이션의 콘텍스트 경로(context path)라고 부름
 - webapps 디렉토리 밑에 생성한 testsite 디렉토리가 하나의 웹 어플리케이션에 해당하고, 이 웹 어플리케이션에 접근할 때 사용되는 URL 경로가 '/testsite/' 로 시작한다

- localhost : 웹서버에서 IP Address 127.0.0.1(loop-back address)로 예약되어 있음
자신의 컴퓨터를 가리키는 가상 IP Address로써 실제 IP Address와 같은 동작을 함 (내부 접속)
- http://localhost/는 localhost라는 도메인을 찾으러 외부 DNS 서버로 나가는 것이 아니라 127.0.0.1이라는 IP 주소를 가리키고 있는 것이므로 곧 바로 자신의 컴퓨터를 찾아감



웹 어플리케이션 생성

- [2] server.xml 파일에 컨텍스트를 추가해서 웹 어플리케이션 작성하기
 - 톰캣 설치 폴더가 아닌 다른 위치에 웹 어플리케이션을 만들고자 할 때 사용됨
 - testsite 폴더를 복사해서 d:\W 에 붙여넣기한 후 폴더이름을 samplesite 으로 변경
 - conf\server.xml 파일을 열어서 "</host"을 찾아 </host> 태그 바로 위의 빈 줄에

```
<Context path="/samplesite" docBase="d:\W\samplesite"/>
```

- 톰캣 켜다 켜고, 웹 브라우저를 열어서
 - <http://localhost:9090/samplesite/now1.jsp> 실행

예제-now1.jsp

소스보기

- 디렉티브, 스크립트릿 코드가 위치한 부분은 공백문자로 표시
- 표현식은 값으로 변환되어 출력
- 디렉티브, 스크립트릿, 표현식을 제외한 나머지 문자는 그대로 출력(원래 jsp 소스 코드는 표시되지 않음)

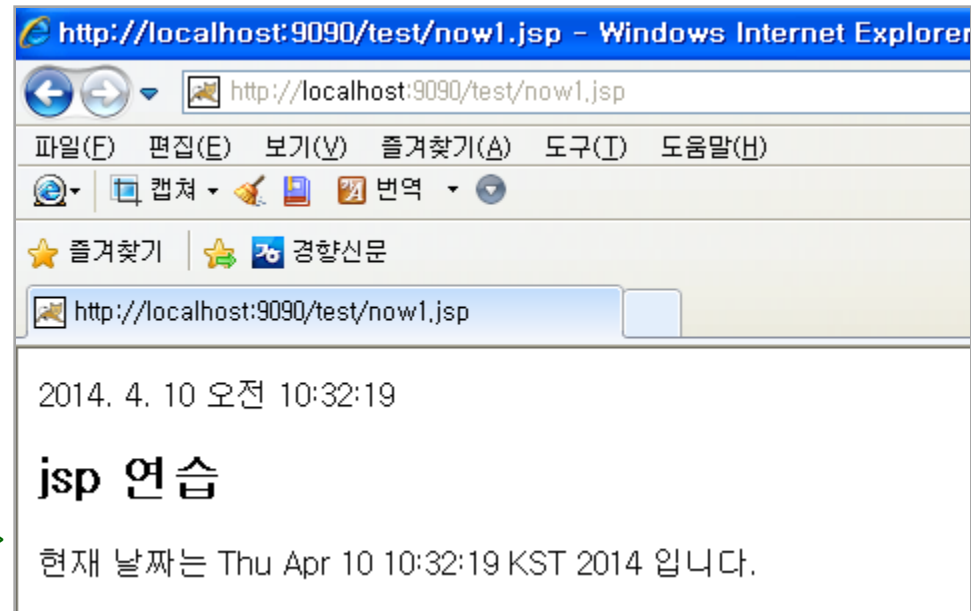
```
<%@page contentType="text/html; charset=euc-kr" %>
<%@page import="java.util.Date" %>
<!-- page 지시자(디렉티브) - page에 대한 정보 설정 -->
```

```
<%
    //스크립트릿 - jsp 코드를 넣는 곳
    Date d = new Date();
    String now = d.toLocaleString();
    out.print(now); //out => jsp의 내장 객체, out.print() =>웹 브라우저에 출력하라는 것
    /*자바의 여러 줄 주석*/
    //자바의 한줄 주석
%>
```

out : jsp의 내장 객체, 웹브라우저에 출력

```
<html>
<body>
    <h1>jsp 연습</h1>
    현재 날짜는 <%=d%> 입니다.
    <!-- 표현식 => out.print(d)와 동일 -->
    <!--html 주석-->
    <!--jsp 주석 --%>
</body>
</html>
```

```
<!--http://localhost:9090/testsite/now1.jsp-->
```





웹 프로그래밍 기초



서블릿과 JSP

■ 서블릿(Servlet)

- 자바언어를 개발한 Sun Microsystems에서 웹 개발을 위해 만든 표준
- 서블릿 규약에 따라 만든 클래스를 서블릿이라고 부름
- 서블릿을 만들기 위해서는 자바 코드를 작성하고, 코드를 컴파일해서 클래스 파일로 만들게 됨
 - 서블릿은 실행코드 방식에 속함
 - 서블릿을 이용하여 웹 어플리케이션을 개발할 경우, 화면에 출력되는 데이터를 조금만 바꾸고 싶어도 코드를 수정하고 컴파일하고 클래스를 알맞은 곳에 복사해 주는 작업을 반복해 주어야 했다 => 개발 효율성 떨어뜨림

■ Sun은 서블릿의 단점을 보완하기 위해 스크립트 방식의 표준인 JSP를 만듦

- JSP는 코드를 수정하면 바로 변경 내용이 반영됨
- 2000년 jsp 1.1, 2001년 jsp 1.2 출시
- 그 후 JSP 2.1, 2.2, 현재 JSP 2.3



서블릿과 JSP

■ JSP 표준

- 서블릿 표준을 기반으로 만들어짐
- 내부적으로 JSP 파일이 번역되면 최종 결과물로 서블릿이 만들어짐
 - 서블릿 2.3버전, JSP 1.2 버전이 한 쌍
 - 서블릿 2.4버전, JSP 2.0 버전이 한 쌍
 - 서블릿 2.5버전, JSP 2.1 버전이 한 쌍
 - 서블릿 3.0버전, JSP 2.2 버전이 한 쌍
 - 서블릿 3.1버전, JSP 2.3 버전이 한 쌍 (Java EE7 톰캣 8.0)
 - 서블릿 4.0버전, JSP 2.3 버전이 한 쌍 (Java EE8 톰캣 9.0)

톰캣 버전	Servlet 스펙	JSP 스펙	최소 Java(JDK) 버전
8.0	3.1	2.3	1.7
7.0	3.0	2.2	1.6
6.0	2.5	2.1	1.5
5.5	2.4	2.0	1.4

서블릿 코드의 예

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

@WebServlet("/NowServ") // 서블릿 경로명

```
public class NowServlet extends HttpServlet {
    @Override //get방식으로 요청한 경우 실행되는 메서드
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=euc-kr");
        Date now = new Date();

        PrintWriter writer = response.getWriter();
        writer.println("<html>");
        writer.println("<head><title>현재 시간</title></head>");
        writer.println("<body>");
        writer.println("<b>현재 시간:</b>");
        writer.println(now.toLocaleString());
        writer.println("</body>");
        writer.println("</html>");
        writer.close();
    }
}
```

• 톰캣 7.0 인 경우

```
javac -d ../classes NowServlet.java
```

컴파일 후

<http://localhost:9090/testsite/NowServ>

- 응답문서에 대한
ContentType 지정
- jsp의 페이지 지시자
`<%@page
contentType="text/html;
charset=euc-kr"%>`와 동일

```
response.getWriter();
```

- 브라우저에 출력할 출력 스트림 얻어오기



어노테이션(Annotation, Metadata)

■ 어노테이션

- 클래스나 메서드 등의 선언시에 @를 사용하는 것
 - JDK 5.0 부터 등장
- [1] 컴파일러에게 정보를 알려주거나
- [2] 컴파일할 때와 설치(deployment) 시의 작업을 지정하거나
- [3] 실행할 때 별도의 처리가 필요할 때 사용함
- 어노테이션은 클래스, 메소드, 변수 등 모든 요소에 선언할 수 있음

주석의 형태로 소스코드에 메타데이터(실제 데이터가 아닌 데이터를 위한 데이터)를 삽입하는 것



어노테이션(Annotation, Metadata)

- 자바에 미리 정해져 있는 어노테이션

- `@Override`

- 해당 메서드가 부모 클래스에 있는 메서드를 Override 했다는 것을 명시적으로 선언함
 - 오버라이딩 문법에 맞지 않게 잘못 코딩하면 컴파일러가 에러를 발생함

- `@Deprecated`

- 가급적 사용을 자제해달라는 의미로 사용됨. 예를 들어 메서드 앞에 `@Deprecated`가 붙으면, 이 메소드를 사용하거나 오버라이드 할 경우, 컴파일 할 때 경고가 뜬다.

- `@SuppressWarnings`

- 이 부분에 대해서 경고문을 출력하지 말라는 의미.
 - 예) `@SuppressWarnings("unused")`



JSP란

- JSP(Java **Server Pages**)-스크립트 언어
 - 선 마이크로시스템즈사의 자바 서블릿 기술을 확장시켜 웹 환경에서 자바만으로 **Server Side** 모듈을 개발하기 위한 기술
 - 웹 프로그래밍 언어로, **자바 기반의** 동적인 페이지를 생성하기 위한 **서버에서 실행되는 스크립트 언어**
 - 자바언어를 기반으로 하는 스크립트 언어로서 자바가 제공하는 기능을 그대로 사용할 수 있다.
 - 자바 언어의 특징을 그대로 가지고 있다.
 - HTTP와 같은 프로토콜에 따라 클라이언트의 요청을 처리하고 응답한다.
 - 표현언어, 표현식, 스크립트릿 등 다양한 스크립트 요소와 액션 태그 등을 제공함으로써 보다 쉽게 웹 어플리케이션을 프로그래밍할 수 있도록 도와준다.

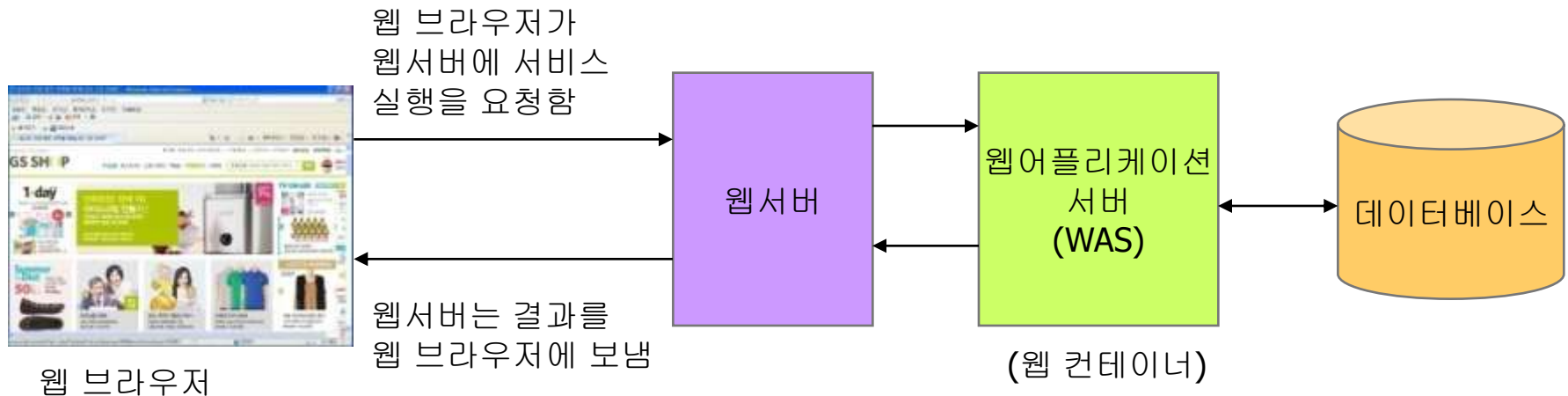
웹 어플리케이션

- 애플리케이션(데스크톱 애플리케이션) - 사용자가 직접 아이콘을 더블 클릭하거나 명령창을 통해 실행시키는 프로그램
- 웹 애플리케이션 - 사용자가 웹 서버를 통해 간접적으로 실행시키는 프로그램

■ 웹 어플리케이션

- 웹을 기반으로 실행되는 어플리케이션
- 예) 웹 브라우저에 <http://www.gssshop.com> 주소 입력=> 웹 어플리케이션에 기능 요청 => 요청 받은 웹 어플리케이션은 요청한 기능에 알맞은 결과 화면을 생성해서 웹 브라우저에 전송

■ 웹 브라우저에 서비스를 제공하기 위해 필요로 하는 구성 요소들



- 웹 서버 - 웹 페이지가 들어 있는 파일(html 파일)을 사용자에게 제공하는 프로그램
- 웹 어플리케이션 서버 - 웹어플리케이션(jsp, 서블릿)을 실행해주는 소프트웨어 엔진

웹 어플리케이션과 웹 프로그래밍

■ 웹 어플리케이션 구축을 위해 필요한 구성 요소들

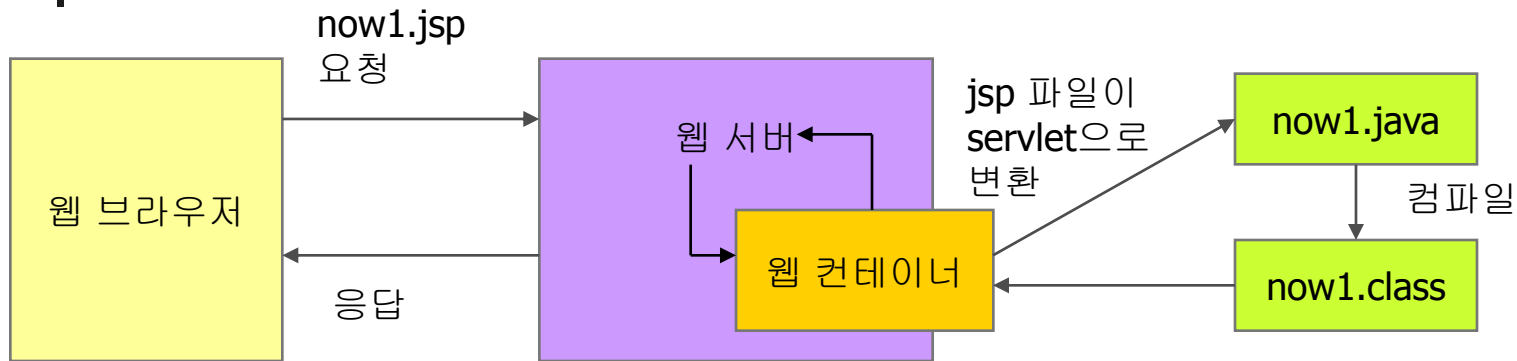
구성요소	역할	주요 제품
웹 서버	웹 브라우저의 요청을 받아서 알맞은 결과를 웹 브라우저에 전송함. 요청된 페이지의 로직 및 데이터베이스와의 연동을 위해 어플리케이션 서버에 이들의 처리를 요청하는 작업을 수행함 주로 정적인 HTML, 이미지, CSS, 자바스크립트를 웹 브라우저에 제공할 때 웹 서버가 사용됨	아파치, IIS
웹 어플리케이션 서버(WAS), 웹 컨테이너	요청된 페이지의 로직 및 데이터베이스와의 연동을 처리하는 부분 게시글 목록, 로그인 처리와 같은 기능을 실행(처리)하고, 그 결과를 응답으로 웹 서버에 전달함	톰캣, 웹로직, 웹스피어, 제우스, JBoss 등
데이터베이스	웹 어플리케이션이 필요로 하는 데이터를 저장함. 예) 회원정보, 게시판 글 데이터 등을 저장함	오라클, MySQL, MS-SQL 등
웹 브라우저	웹에서 클라이언트이며, 사용자의 작업창. 웹 서버에 서비스 실행을 요청하며, 웹 서버의 처리 결과를 사용자에게 보여줌	인터넷 익스플로러, 파이어폭스, 크롬



웹 어플리케이션과 웹 프로그래밍

- 웹 서버(아파치)
 - 정적인 **HTML, CSS** 를 제공하는 데 초점이 맞춰져 있음
 - 클라이언트와 통신, 다중 클라이언트의 접속 관리
- 웹 어플리케이션 서버, 웹 컨테이너(웹로직, 톰캣)
 - **jsp**, 서블릿과 같은 프로그램을 실행하여 결과를 제공하는 데 초점이 맞춰져 있음
 - 비즈니스 로직, 데이터 관리
- => 웹 서버와 웹 어플리케이션 서버를 연동하여 정적인 HTML, CSS, 이미지 파일등은 웹 서버가 제공하도록 하고 JSP나 서블릿에 대한 요청은 웹 서버가 어플리케이션 서버에 전달하도록 구성하는 것이 일반적임

jsp의 동작 구조



jsp 페이지의 내부 처리 과정

- jsp 페이지는 서블릿으로 변환되어 웹 브라우저의 요청에 대한 응답을 HTML 문서로 생성한다.



jsp의 동작 구조

- 웹 브라우저에서 jsp 페이지를 웹 서버로 요청하게 되면, 웹 서버는 jsp에 대한 요청을 웹 컨테이너로 넘기게 됨
- 이러한 요청을 받은 웹 컨테이너는 해당 jsp 페이지를 찾아서 서블릿(.java 파일 생성)으로 변환하는 파싱(parsing) 과정을 거친 후 컴파일(.class 파일 생성)을 하게 됨
- 컴파일된 서블릿(.class)은 최종적으로 웹 브라우저에 응답되어져 사용자는 응답 결과를 보게 됨
 - 이러한 과정은 **해당 jsp 페이지가 최초로 요청되었을 때 단 한번만 실행됨**
 - 이후 같은 페이지에 대한 요청이 있으면 변환된 서블릿 파일로 서비스를 처리함

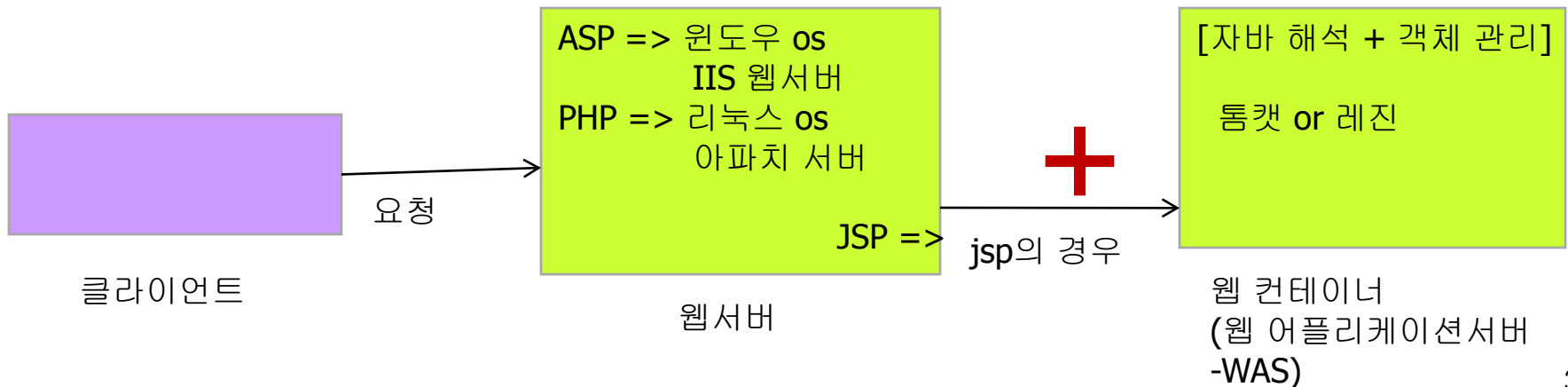


웹 컨테이너

- 웹 컨테이너(Web Container)
 - 웹 어플리케이션을 실행할 수 있는 컨테이너
 - JSP와 서블릿이 실행 가능한 서버
 - 톰캣 - 웹 컨테이너로서 jsp와 서블릿을 지원하고 있음
- JSP를 사용하는 이유
 - 자바 언어를 기반으로 하고 있기 때문에 플랫폼에 상관없이 사용할 수 있다
 - jsp 자체는 asp, php 와 같은 스크립트 언어이기 때문에 쉽게 배울 수 있다.

웹 컨테이너

- JSP는 사실상 서블릿이라 불리는 클래스로 변환되어 실행되는 자바언어의 일부
 - JSP로 웹서비스를 구축하려면 웹서버가 java 언어를 이해해야 하지만, 클래스의 바이트 코드를 이해하는 소프트웨어는 오직 JVM이므로, 대부분의 웹서버는 자바 언어를 해석할 수 있도록 설계되어 있지 않다.
 - 따라서 자바 기반의 웹서비스를 구축하려면 **서블릿 클래스의 해석 및 관련 객체들을 관리**해주는 특정 프로그램의 도움을 받지 않으면 안되는데 이 역할을 해주는 프로그램을 **컨테이너**라고 하며, 톰캣, 레진등이 있다.





웹 컨테이너

- 웹 컨테이너 : 예) 톰캣

- 서블릿을 관리

- 서블릿에 대한 요청을 받고 응답을 해주는 중간 역할
 - 클라이언트와 서블릿간의 요청과 답변을 전달해줌
 - 요청을 넘겨받은 컨테이너는 `HttpRequest`와 `HttpResponse` 객체를 만들어 서블릿의 `doPost()`, `doGet()` 메서드 중 하나를 호출한다.

- 생명주기 관리

- 서블릿 클래스를 로딩하여 인스턴스화
 - 초기화 메소드를 호출
 - 요청이 들어오면 적절한 서블릿 메소드를 호출

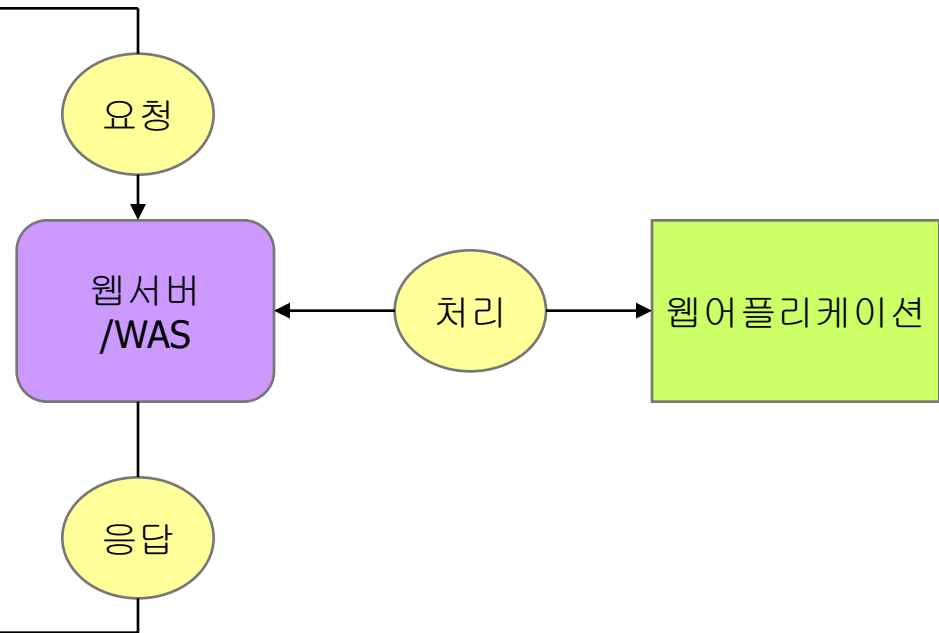
웹 어플리케이션 서버 방식

- 웹 어플리케이션이 실행되는 과정
 - 요청-처리-응답의 3단계 과정

1) 웹 브라우저는 웹 서버에 어떤 기능을 원하는지 요청한다.



2) 웹 서버는 웹 어플리케이션을 실행하여 웹 브라우저가 요청한 기능을 수행한 후, 결과를 웹 브라우저에 응답한다.

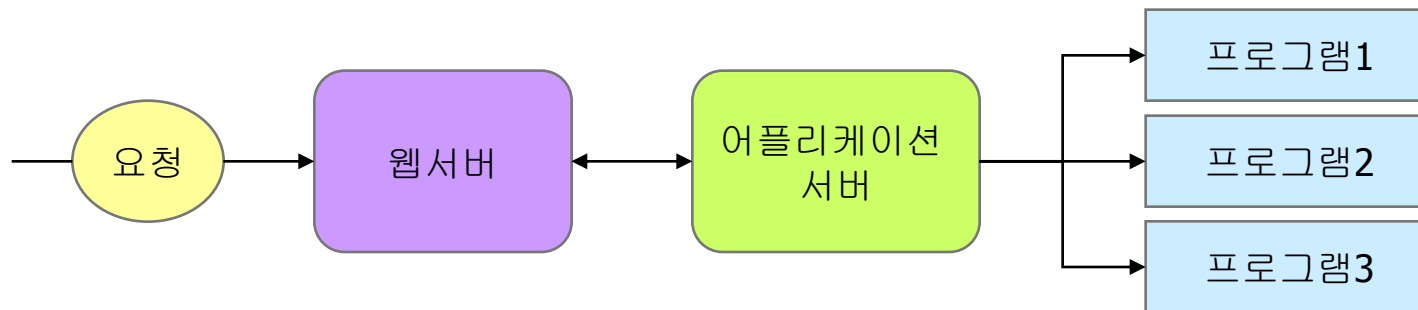


3) 웹 브라우저는 웹 서버로부터의 응답 결과를 출력한다.



CGI방식 vs 웹 어플리케이션 서버 방식

- 과정 2)에서 웹 서버는 웹 어플리케이션 프로그램을 사용해서 웹 브라우저의 요청을 처리함
- 이때 웹 서버가 웹 어플리케이션 프로그램을 실행하는 방식에 따라서 2가지 형태로 웹 어플리케이션 동작 방식을 구분할 수 있음
 - CGI방식 - 웹 서버가 어플리케이션 프로그램을 직접 실행하는 구조
 - 펄(Perl), C/C++ 언어 사용
 - 어플리케이션 서버 방식 - 웹 서버가 직접 프로그램을 호출하기 보다는 웹 어플리케이션 서버를 통해서 간접적으로 웹 어플리케이션 프로그램을 실행함
 - asp, jsp, asp.net 등

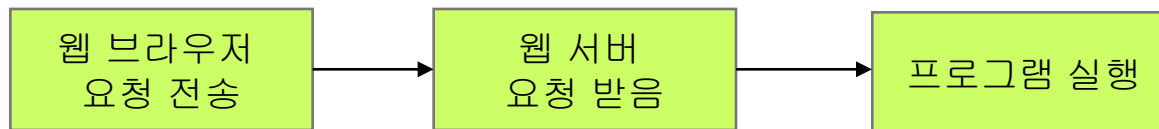


어플리케이션 서버 방식의 요청 처리

스크립트 방식과 실행코드 방식

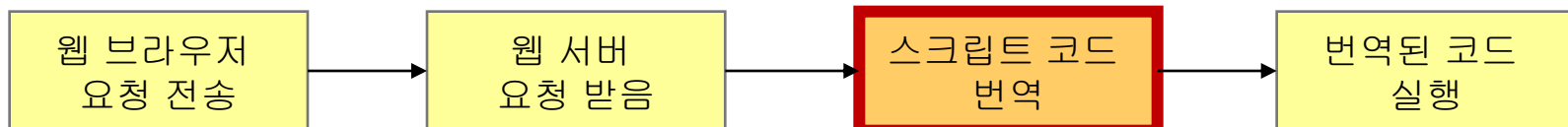
- 웹 어플리케이션 프로그래밍은 구현하는 방식에 따라 실행코드 방식과 스크립트 방식으로 구분

비교 항목	실행 코드 방식	스크립트 방식
코드 형태	컴파일 된 실행 프로그램	컴파일되지 않은 스크립트 코드
실행 방식	컴파일 된 기계어 코드 직접 실행	스크립트 코드를 해석한 뒤 실행
코드 변경	소스 코드를 다시 컴파일해야 함	스크립트 코드만 고치면 됨
종류	C기반 CGI 프로그램, 서블릿	JSP , ASP.NET, asp, php, ruby 등



실행코드 방식의 경우

미리 컴파일된 실행 프로그램을 사용자의 요청에 따라 실행함



스크립트 방식의 경우 실행 방식

사용자의 요청이 있을 때 스크립트 코드를 번역해서 번역된 코드를 실행함



스크립트 방식과 실행코드 방식

■ 스크립트 방식

- 스크립트 코드의 번역은 해당 페이지가 최초로 요청되었을 때 한 번만 실행됨
- 그 이후에 해당 페이지의 요청이 있는 경우에는 번역된 코드가 실행됨
- asp, jsp 등의 웹 어플리케이션 서버 방식
- CGI 방식의 실행코드 방식을 사용하는 것보다 전체적인 성능이 뛰어남

■ 실행코드 방식

- CGI 방식, 서블릿



스크립트 언어

- 클라이언트 사이드 스크립트(Client Side Script)
 - 클라이언트 쪽에서 처리
 - 예) JavaScript, VBScript
- 서버 사이드 스크립트(Server Side Script)
 - 서버에서 처리
 - 예) ASP, JSP, PHP

Java EE 기술들

- <http://www.oracle.com/technetwork/java/index.html>
=> Software Downloads 에서 Java EE and GalssFish 클릭 => technologies 탭 클릭 => Java EE에 속한 하위 기술 목록이 출력됨
- <http://www.oracle.com/technetwork/java/javaee/tech/javaee6technologies-1955512.html>
- Java EE 기술 목록에서 웹 애플리케이션 관련 기술을 확인
 - 서블릿/ JSP의 기술 사양 버전 확인
- Java EE와 서블릿/JSP의 버전
 - Java EE는 기능 확장이 쉽고, 이기종 간의 이식이 쉬우며, 신뢰성과 보안성이 높고, 트랜잭션 관리와 분산 기능을 쉽게 구현할 수 있는 기술을 제공
 - Java EE 기술 사양은 한 가지 기술을 정의한 것이 아니라 기업용 애플리케이션 개발에 필요한 여러가지 복합적인 기술들을 정의하고 모아 놓은 것임
 - Java EE 기술 중에서 서블릿, JSP는 웹을 기반으로 한 클라이언트/서버 기술을 정의

Web Application Technologies » Read more
Java Servlet 3.0
JavaServer Faces 2.0
JavaServer Pages 2.2/Expression Language 2.2
Standard Tag Library for JavaServer Pages (JSTL) 1.2

[Java EE 6 기술 목록]



WAS의 이해

- 애플리케이션 서버(Application Server)
 - 클라이언트/서버 시스템 구조에서 서버쪽 애플리케이션의 생성과 실행, 소멸을 관리하는 프로그램
- WAS(웹 애플리케이션 서버, Web Application Server)
 - 서블릿과 서블릿 컨테이너와 같이 웹 기술을 기반으로 동작되는 애플리케이션 서버
- [1] 자바에서 말하는 WAS
 - Java EE 기술 사양을 준수하여 만든 서버
 - Java EE 구현체
 - 상용 제품 - 티맥스소프트의 **JEUS**, 오라클의 **WebLogic**, IBM의 **웹스피어**, 레드햇의 **JBoss** 등
 - 무료, 오픈소스 - 레드햇의 **JBoss AS**, 오라클의 **GlassFish**, 아파치 재단의 **Geronimo** 등

WAS의 이해

- [2] 서블릿 컨테이너(웹 컨테이너)
 - Java EE 기술 중에서 서블릿, JSP 등 웹 관련 부분만 구현한 서버
 - 아파치 재단의 톰캣, Caucho의 resin, 오픈 프로젝트 Jetty 등
 - 서블릿이나 JSP 프로그래밍을 할 때는 사용하는 제품의 버전에 맞추어 API를 사용해야 함

JavaEE	Servlet/JSP	Tomcat	JBoss	WebLogic	JEUS
JavaEE 6	Servlet 3.0 JSP 2.2	7.0.x	7.x(all) 6.x(almost)	12.x	7.x
JavaEE 5	Servlet 2.5 JSP 2.1	6.0.x	5.x	10.x	6.x
J2EE 1.4	Servlet 2.4 JSP 2.0	5.5.x	4.x	9.x	5.x
J2EE 1.3	Servlet 2.3 JSP 1.2	4.1.x		7.x, 8.x	4.x
J2EE 1.2	Servlet 2.2 JSP 1.1	3.3.x		6.x	3.x

* a(모든 기술 구현함), almost(일부 기술 미구현)

[Java EE 와 Servlet/JSP 버전, 구현체 버전]



JSP 기본 - JSP 페이지의 구성요소



JSP 기본 코드 구조

- JSP의 주된 목적

- 웹 브라우저에서 보여줄 HTML 문서를 생성하는 것

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

설정부분

```
<html>
```

```
<head><title>HTML 문서의 제목</title></head>
```

```
<body>
```

```
<%
```

```
    String name = "홍길동";
```

```
    int age = 20;
```

```
%>
```

```
<b><%=name%></b> (<%=age%>) 세 입니다.
```

```
</body>
```

```
</html>
```

생성부분



JSP 기본 코드 구조

- 설정부분 : JSP 페이지에 대한 설정 정보를 지정
 - JSP 페이지가 생성하는 문서의 타입
 - JSP 페이지에서 사용할 커스텀 태그
 - JSP 페이지에서 사용할 자바 클래스 지정
 - `<%@ page..%>` - page 디렉티브
 - JSP페이지에 대한 정보를 나타낼 때 사용
- 생성부분 : HTML 코드 및 JSP 스크립트



JSP 페이지의 구성요소

- 디렉티브(Directive) : 지시어
- 스크립트: 스크립트릿(Scriptlet), 표현식, 선언부
- 기본 객체 - **response, request, out, session** 등(별도의 선언 과정 없이 사용가능, 내장 객체)
- 정적인 데이터
- 표준 액션 태그(Action Tag)
 - XML의 태그와 같은 모양을 취하며, JSP페이지에서 특별한 기능을 제공함
- 표현언어(Expression Language)
 - JSP 2.0부터 추가, 스크립트릿과 표현식 대신에 쉽고 간단하게 사용할 수 있음
- 커스텀 태그와 표준 태그 라이브러리(JSTL)
 - 커스텀 태그 - JSP를 확장시켜 주는 기능, 액션 태그와 마찬가지로 태그 형태로 기능을 제공함
 - 표준 태그 라이브러리 - 커스텀 태그 중에서 자주 사용되는 것들을 별도로 표준화한 태그 라이브러리



디렉티브(Directive)

- 디렉티브(Directive)-지시어
 - JSP 페이지에 대한 설정 정보를 지정할 때 사용

```
<%@ 디렉티브이름 속성1="값1", 속성2="값2", ...%>
```

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

- JSP가 제공하는 디렉티브
 - **page** : JSP 페이지에 대한 정보를 지정, JSP가 생성하는 문서의 타입, 출력 버퍼의 크기, 에러 페이지 등
 - **taglib** : JSP 페이지에서 사용할 태그 라이브러리를 지정
 - **include** : JSP 페이지의 특정 영역에 다른 문서를 포함시킴

디렉티브에 사용할 수 있는 속성

속성명	속성의 기본값	사용법	속성 설명
contentType	"text/html;charset=ISO-8859-1"	"text/html;charset=euc-kr"	jsp 페이지가 생성할 문서의 타입을 지정
import		"java.util.*,java.io.*"	
info			페이지를 설명해주는 문자열을 지정
language	"java"	"java"	사용할 언어 지정
session	"true"	"true"(기본값) "false"	HttpSession을 사용할지 여부를 지정
buffer	"8kb"	"8kb"(기본값) "none"	jsp 페이지의 출력 버퍼의 크기를 지정
autoflush	"true"	"true"(기본값) "false"	출력버퍼가 다 찰 경우에 저장되어 있는 내용의 처리를 설정
errorPage		"/error.jsp"	에러발생시 에러를 처리할 페이지를 지정
isErrorPage	"false"	"false"(기본값) "true"	해당 페이지를 에러페이지로 지정
pageENCODING		"ISO-8859-1" "euc-kr"	해당 페이지의 문자 인코딩을 지정
extends		JSP엔진에 의해 자동으로 설정	상속받을 클래스를 지정
isThreadSafe	"true"	"true"(기본값) "false"	현 페이지에 다중 스레드를 허용할지 여부를 설정



page 디렉티브 속성

■ contentType

- JSP 페이지가 생성할 응답 문서의 타입을 지정
- 한글로 구성된 HTML 문서를 생성하는 경우

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

■ import

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

유일하게 중복 사용이 가능한
속성

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



include 디렉티브

- `<%@ include %>`

- jsp에서는 여러 jsp 페이지에서 공통적으로 포함하는 내용이 있을 때, 이러한 내용을 매번 입력하지 않고, 별도의 파일로 저장해 두었다가 필요한 jsp 페이지 내에 삽입할 수 있는 기능을 제공
- include 디렉티브 - 공통적으로 포함될 내용을 가진 파일을 해당 jsp 페이지 내에 삽입하는 기능을 제공함

```
<%@include file="로컬 URL" %>
```

- include 디렉티브의 처리 과정 - 정적
 - include 디렉티브를 사용한 jsp 페이지가 컴파일 되는 과정에서 include 되는 jsp 페이지의 소스 내용을 그대로 포함해서 컴파일을 하게 됨
 - include 되는 파일의 결과가 포함되는 것이 아니라 단순히 파일의 내용이 텍스트로 include 디렉티브가 위치한 자리에 그대로 복사되는 것
 - 두 개의 파일이 한 페이지로 합쳐지고, 한 페이지로 인식되어 변환되고 컴파일됨



include 지시어 – copyright.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.util.*" %>
<hr>
<%
    Date date = new Date();
%>
페이지 하단 <br>
현재일자 : <%=date.toLocaleString()%>
<br><br>
copyright &copy; testsite.com all right reserved.
```



include 지시어-now1.jsp

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

```
<h2>include 지시자 이용</h2>
```

```
<%@include file="copyright.jsp" %>
```

```
    <!-- include 지시자 : 특정 문서를 포함시킬 때 사용  
        페이지의 소스를 포함시킴
```

```
-->
```

```
<h2>include 액션 태그 이용</h2>
```

```
<jsp:include page="copyright.jsp" />
```

```
    <!--include 액션 태그
```

```
        소스가 아닌 페이지의 실행결과를 포함시킴
```

```
-->
```

include 지시자 이용

페이지 하단

현재일자 : 2014. 4. 10 오전 11:30:22

copyright © testsite.com all right reserved.

include 액션 태그 이용

페이지 하단

현재일자 : 2014. 4. 10 오전 11:30:22

copyright © testsite.com all right reserved.



taglib 디렉티브

- `<%@ taglib %>`
 - 표현언어(EL), JSTL, 커스텀 태그를 jsp 페이지 내에 사용할 때 사용되어짐

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

```
...
```

```
<c:set var="aInt" value="123" />
```



스크립트 요소

- 스크립트 요소

- JSP 프로그래밍에서 로직을 수행하는데 필요한 부분
- 스크립트 코드를 사용해서 프로그램이 수행해야 하는 기능을 구현할 수 있음
 - 스크립트릿
 - 표현식
 - 선언부

스크립트 요소

스크립트릿은 jsp 페이지가 servlet으로 변환되고 이 페이지가 호출될 때 _jspService() 메서드 안에 선언됨

■ 스크립트릿

- JSP 페이지에서 자바 코드를 실행할 때 사용되는 코드의 블록

```
<%  
    자바코드1;  
    자바코드2;  
    ...  
%>
```

■ 표현식

- jsp페이지에서 웹 브라우저에 출력할 부분을 표현하기 위한 것
- 스크립트릿 코드 내에서 표현식을 쓸 수 없음
- 대신 스크립트릿내에서 출력할 부분은 내장객체인 out 객체를 사용해서 출력함

```
<%= 출력할 값 %>
```

```
<%  
    out.println("출력할 값");  
%>
```



스크립트 요소

■ 선언부

- JSP 페이지의 스크립트릿이나 표현식에서 사용할 수 있는 멤버변수나 메서드를 작성할 때 사용됨

```
<%!  
    public 리턴타입 메서드이름(매개변수){  
        자바코드...  
    }  
%>
```



주석

- html 주석

- <!-- html 주석 -->

- html 주석 내에 표현식 및 스크립트릿을 사용하면 그 코드들은 실행됨

- html 주석은 화면에 표시하지 않으나 코드는 실행됨

- jsp 주석

- <%-- jsp 주석 --%>

- jsp 페이지에서만 사용됨

- 화면에 표시되지도 않고, 실행되지도 않는다.

- java 주석

- <%

- //한줄 주석

- /*여러 줄 주석*/

- %>

선언부, 스크립트릿, 표현식-test01.jsp

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

```
<h1>선언부, 스크립트릿, 표현식</h1>
```

```
<%! //선언부 - 멤버변수나 메서드를 선언하는 부분
```

```
String id ="hong"; //멤버변수
```

```
//메서드
```

```
public int add(int a, int b){
```

```
    return a+b;
```

```
}
```

```
%>
```

```
<% //스크립트릿 - 자바 코드 사용, 여기에서 선언된 변수는 모두 지역변수임.
```

```
String str="hello jsp!!"; //지역변수
```

```
out.println("<hr color=pink>");
```

```
out.println("스크립트릿 부분입니다.");
```

```
out.println("<br>멤버변수 id: "+ id);
```

```
out.println("<br>지역변수 str: "+ str);
```

```
out.println("<br><br><b>add()메서드 호출</b>");
```

```
int n1 = 10, n2 = 20;
```

```
int sum = add(n1, n2);
```

```
out.println("<br>두수의 합"+sum);
```

```
%>
```

```
<hr color=green>
```

```
<b><%=str%></b> //표현식
```

- jsp가 서블릿으로 변환된 자바파일 찾기
- D:\ Tomcat 9.0\ work\ Catalina\ localhost

선언부, 스크립트릿, 표현식

스크립트릿 부분입니다.
멤버변수 id: hong
지역변수 str: hello jsp!!

add()메서드 호출
두수의 합30

hello jsp!!



내장 객체



내장 객체

- JSP 내장객체
 - 별다른 선언과정과 객체 생성 없이 사용할 수 있는 9개의 객체들을 웹 컨테이너가 제공함
- 내장객체의 4가지 범주
 - JSP 페이지 입출력 관련 내장 객체
 - JSP 페이지 외부 환경 정보 제공 내장 객체
 - JSP 페이지 서블릿 관련 내장 객체
 - JSP 페이지 예외 관련 내장 객체

JSP 내장객체

내장 객체	리턴 타입	설명
<code>request</code>	<code>javax.servlet.http.HttpServletRequest</code> 또는 <code>javax.servlet.ServletRequest</code>	클라이언트의 요청 정보를 저장하고 있는 객체이다
<code>response</code>	<code>javax.servlet.http.HttpServletResponse</code> 또는 <code>javax.servlet.ServletResponse</code>	클라이언트의 요청에 대한 응답 정보를 저장하고 있는 객체
<code>out</code>	<code>javax.servlet.jsp.JspWriter</code>	JSP 페이지 출력할 내용을 가지고 있는 출력 스트림 객체
<code>session</code>	<code>javax.servlet.http.HttpSession</code>	세션 정보를 저장하고 있는 객체
<code>application</code>	<code>javax.servlet.ServletContext</code>	웹 어플리케이션 Context의 정보를 저장하고 있는 객체
<code>pageContext</code>	<code>javax.servlet.jsp.PageContext</code>	JSP 페이지에 대한 정보를 저장하고 있는 객체
<code>page</code>	<code>java.lang.Object</code>	JSP 페이지를 구현한 자바 클래스 객체
<code>config</code>	<code>javax.servlet.ServletConfig</code>	JSP 페이지에 대한 설정정보를 저장하고 있는 객체
<code>exception</code>	<code>java.lang.Throwable</code>	JSP 페이지에서 예외 발생시에만 사용되는 객체



request 내장객체

■ request 객체

- 웹 브라우저에서 jsp 페이지로 전달되는 정보의 모임
- HTTP 헤더와 HTTP 바디로 구성되어 있음
- 웹 컨테이너는 요청된 HTTP 메시지를 통해 HttpServletRequest 객체를 얻어내서 그것으로부터 **사용자의 요구사항을 얻어냄**
- JSP 페이지에서는 HttpServletRequest 객체를 request 객체명으로 사용함



request 내장객체

- request 객체의 요청 파라미터 메서드

메서드	설명
String getParameter (name)	이름이 name인 파라미터에 할당된 값을 리턴하며, 지정된 파라미터 값이 없으면 null값을 리턴함
String[] getParameterValues (name)	이름이 name인 파라미터의 모든 값을 String 배열로 리턴함 Checkbox 에서 주로 사용됨
Enumeration getParameterNames ()	요청에 사용된 모든 파라미터 이름을 java.util Enumeration 타입으로 리턴함



HTML 폼과 요청 파라미터의 처리

- 웹 브라우저 폼에 입력한 값 처리
- 웹 브라우저는 폼에 입력한 정보를 파라미터로 전송함
 - request 기본 객체는 웹 브라우저가 전송한 파라미터를 읽어올 수 있는 메서드 제공

예제-test02.jsp

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<html>
<head><title>폼 생성</title></head>
<body>
<h1>request 객체 연습</h1>
<form action="test02_ok.jsp" method="post">
이름: <input type="text" name="name" > <br>
주소: <input type="text" name="address" > <br>
좋아하는 동물:
    <input type="checkbox" name="pet" value="dog">강아지
    <input type="checkbox" name="pet" value="cat">고양이
    <input type="checkbox" name="pet" value="pig">돼지
동의합니다<input type="checkbox" name="agree">
<br><br>
    <input type="submit" value="전송">
</form> <br>
<a href="test03.jsp?no=7&name=홍길동">test03 페이지로 이동하기</a><!--get방식-->
</body>
</html>
```

request 객체 연습

이름 :

주소 :

좋아하는 동물 : ☒ 강아지 ☒ 고양이 ☐ 돼지

동의합니다 ☐

[test03 페이지로 이동하기](#)

예제 - test02_ok.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<%@ page import="java.util.Enumeration" %>
<%@ page import="java.util.Map" %>
<%
```

```
    request.setCharacterEncoding("euc-kr");
```

```
%>
```

```
<html>
```

```
<head><title>요청 파라미터 출력</title></head>
```

```
<body>
```

```
<b>request.getParameter() 메소드 사용</b><br>
```

```
name 파라미터 = <%= request.getParameter("name") %> <br>
```

```
address 파라미터 = <%= request.getParameter("address") %>
```

```
<p>
```

```
<b>request.getParameterValues() 메소드 사용</b><br>
```

```
<%
```

```
    String[] values = request.getParameterValues("pet");
```

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

```
        for (int i = 0 ; i < values.length ; i++) {
```

```
%>
```

```
            <%= values[i] %>
```

```
<%
```

```
        }//for
```

```
    }//if
```

```
%>
```

request.getParameter() 메소드 사용

name 파라미터 = 홍길동

address 파라미터 = 마포구 서교동

request.getParameterValues() 메소드 사용

dog cat

request.getParameterNames() 메소드 사용

pet address name

request.getParameterMap() 메소드 사용

name = 홍길동

POST방식으로 넘긴 요청 파라미터에 대한 한글 인코딩 처리



예제

<p>

request.getParameterNames() 메소드 사용

<%

```
Enumeration paramEnum = request.getParameterNames();
while(paramEnum.hasMoreElements()) {
    String name = (String)paramEnum.nextElement();
```

%>

<%= name %>

<%

}

%>

<p>

request.getParameterMap() 메소드 사용

<%

```
Map parameterMap = request.getParameterMap();
String[] nameParam = (String[])parameterMap.get("name");
if (nameParam != null) {
```

%>

name = <%= nameParam[0] %>

<%

}

%>

</body>

</html>

예제 - test03.jsp

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

```
<%
```

```
//get방식으로 넘긴 파라미터 읽어오기
```

```
//http://localhost:9090/testsite/test03.jsp?no=7&name=홍길동
```

```
String no = request.getParameter("no");
```

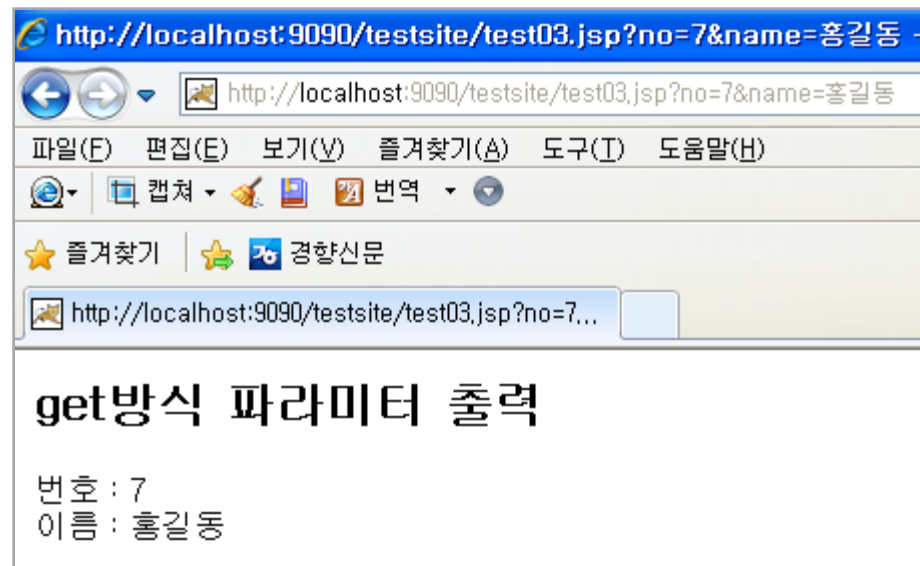
```
String name = request.getParameter("name");
```

```
%>
```

```
<h2>get방식 파라미터 출력</h2>
```

```
번호 : <%=no%><br>
```

```
이름 : <%=name%>
```





jsp 페이지에서의 한글 처리 문제

- [1] 서버에서 웹 브라우저에 응답되는 페이지의 화면 출력 시 한글 처리

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

- 서블릿에서의 경우

```
response.setContentType("text/html; charset=EUC-KR");
```

- [2] 웹 브라우저에서 서버로 넘어오는 파라미터 값에 한글이 있는 경우(POST 방식)의 한글 처리

```
<% request.setCharacterEncoding("euc-kr"); %>
```



jsp 페이지에서의 한글 처리 문제

- GET 방식 한글처리 추가
 - get방식으로 보낸 데이터가 한글인 경우 처리하기 위해서
 - server.xml에 URLEncoder="EUC-KR" 추가

```
<Connector port="9090" protocol="HTTP/1.1"  
    connectionTimeout="20000"  
    redirectPort="8443"  
    URLEncoder="EUC-KR" />
```

```
c:\ Tomcat 9.0\ conf\ server.xml
```



response 내장 객체

■ response 객체

- 웹 브라우저로 응답할 **응답 정보**를 가지고 있음
- 웹 브라우저에 보내는 응답 정보는 **HttpServletResponse** 객체로 jsp에서는 **response** 객체로 사용함
- 응답 정보와 관련하여 주로 헤더 정보 입력, 페이지 리다이렉트 등의 기능 제공
- response 내장 객체의 메서드

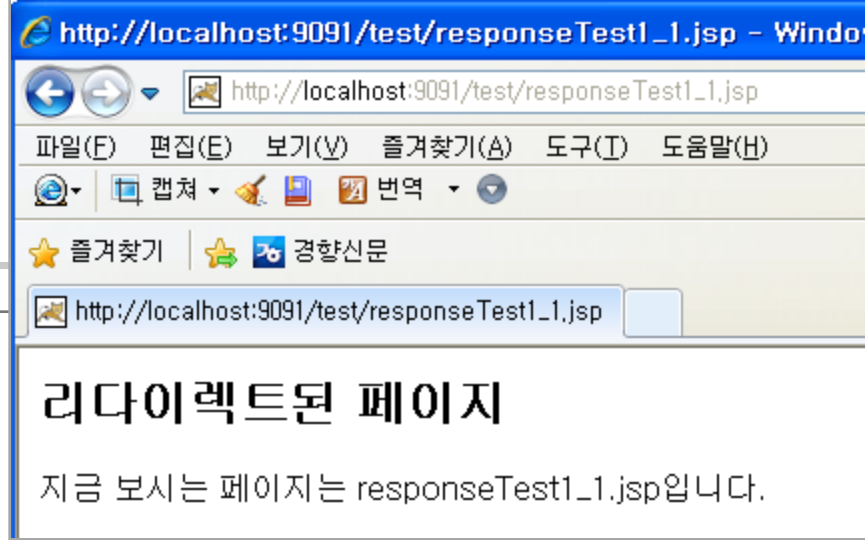
메서드	설명
void setHeader(name, value)	name에 해당하는 속성을 value값으로 설정한다.
void setContentType (type)	요청의 결과로 보여질 페이지의 contentType을 설정한다
void sendRedirect (url)	지정된 url로 페이지가 이동한다.

예제

```
<%@ page contentType="text/html; charset=euc-kr"%>
<h2>Response내장객체 예제</h2>
현재 페이지는 responseTest1.jsp 페이지입니다.

<%
response.sendRedirect("responseTest1_1.jsp"); //get방식
%>
```

```
<%@ page contentType="text/html; charset=euc-kr"%>
<html>
<body>
<h2>리다이렉트된 페이지</h2>
지금 보시는 페이지는 responseTest1_1.jsp입니다.
</body>
</html>
```





out 내장객체

■ out 객체

- jsp 페이지가 생성한 결과를 웹 브라우저에 전송해 주는 출력 스트림
- jsp 페이지가 웹 브라우저에게 보내는 모든 정보는 out 객체로 통해서 전송됨
- out 객체는 javax.servlet.jsp.**jspWriter** 클래스 타입으로 jsp에서는 out 객체로 사용됨
- `println()`메서드
 - 웹 브라우저에 출력을 하기 위한 메서드
 - 표현식 `<%=코드%>` 와 `<% out.println(코드)%>`는 동일



out 내장객체

■ out 객체의 메서드

메서드	설명
boolean isAutoFlush()	출력 버퍼가 다 찼을 때 처리 여부를 결정하는 것으로 자동으로 flush 할 경우에는 true를 리턴하고, 그렇지 않을 경우 false 리턴
int getBufferSize()	출력 버퍼의 전체 크기를 리턴
int getRemaining()	현재 남아 있는 출력 버퍼의 크기를 리턴
void clearBuffer()	현재 출력 버퍼에 저장되어 있는 내용을 웹 브라우저에 전송하지 않고 비운다
String println(str)	주어진 str값을 웹 브라우저에 출력함. 이 때 줄바꿈은 적용되지 않음
void flush()	현재 출력 버퍼에 저장되어 있는 내용을 웹 브라우저에 전송하고 비운다.
void close()	현재 출력 버퍼에 저장되어 있는 내용을 웹 브라우저에 전송하고 출력 스트림을 닫는다.

회원가입 페이지

http://localhost:9090/test/lec/register.jsp

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

캡처 번역

즐거찾기 | 경향신문

회원가입페이지입니다

회원가입

아이디 :

비밀번호 :

이름 :

휴대폰 번호 :



http://localhost:9090/test/lec/register_ok.jsp

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

캡처 번역

즐거찾기 | 경향신문

http://localhost:9090/test/lec/register_ok.jsp

====사용자가 입력한 값====

아이디 : hong

비밀번호 : 123

이름 : 홍길동

전화번호 : 010-100-2000

데이터베이스에 저장합니다



회원가입 페이지-register.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML> <HEAD>
  <TITLE> 회원가입 페이지입니다 </TITLE>
<script type="text/javascript">
  function send(){
    if(window.document.frm1.id.value==""){
      alert("아이디를 입력하세요!");
      window.document.frm1.id.focus();
      return false;
    }
    if(frm1.pwd.value.length==0){
      alert("비밀번호를 입력하세요!");
      frm1.pwd.focus();
      return false;
    }
    if(!frm1.name.value){
      alert("이름을 입력하세요!");
      frm1.name.focus();
      return false;
    }
  }
```



회원가입 페이지

```
        return true;
    }
</script>
</HEAD>
<BODY>
<h3>회원가입</h3>
    <form name="frm1" method="post" action="register_ok.jsp" onsubmit="return send()">
        아이디 : <input type="text" name="id" maxlength="10"><br>
        비밀번호 : <input type="password" name="pwd" maxlength="10"><br>
        이름 : <input type="text" name="name" ><br>
        휴대폰 번호 : <input type="text" name="hp"><br><br>

        <input type="submit" value="전송">
        <input type="reset" value="취소">
    </form>
</BODY>
</HTML>
```



register_ok.jsp

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

```
<%
```

```
    request.setCharacterEncoding("euc-kr");
```

```
    String id=request.getParameter("id");
```

```
    String pwd=request.getParameter("pwd");
```

```
    String name=request.getParameter("name");
```

```
    String hp=request.getParameter("hp");
```

```
    out.print("====사용자가 입력한 값====");
```

```
    out.print("아이디 : " + id + "<br>");
```

```
    out.print("비밀번호 : " + pwd + "<br>");
```

```
    out.print("이름 : " + name + "<br>");
```

```
    out.print("전화번호 : " + hp + "<br>");
```

```
    out.print("데이터베이스에 저장합니다");
```

```
%>
```

실습

상호 :

업종 : ☒ 한식 ☐ 양식 ☐ 일식

지역 :

☒ 주차 ☒ 쿠폰 ☐ 시식평

위치 :

=====사용자가 입력한 값=====

상호 : 놀부 보쌈

업종 : 한식

지역 : 서울

위치 : 서울시 구로구 구로동 150번지 놀부 보쌈 1호점

기타 옵션 : 주차, 쿠폰

데이터베이스에 저장합니다

과제

← → http://125,133,193,80:9090/test/lec/formTest5.jsp

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

← 캡처 → 번역

★ 즐겨찾기 ★ 경향신문

New Document

아이디 : hong

비밀번호: ●●●●

취미 : ☒ 축구 ☒ 낚시

성별 : ☒ 남자 ☐ 여자

자기소개 : 안녕하세요
홍길동입니다

학년 : 1학년

사용언어 :

html
jsp
java

전송 취소

=> 다중 선택 가능

← → http://125,133,193,80:9090/test/lec/formTest5_ok.jsp

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

← 캡처 → 번역

★ 즐겨찾기 ★ 경향신문

http://125,133,193,80:9090/test/lec/formTest5...

=====사용자가 입력한 값=====

아이디 : hong

비밀번호 : 1234

번호 : 히든필드값

자기소개 : 안녕하세요 홍길동입니다

학년 : 1학년

성별 : M

취미 : 축구, 낚시

프로그래밍 언어 : html java

데이터베이스에 저장합니다



HTTP

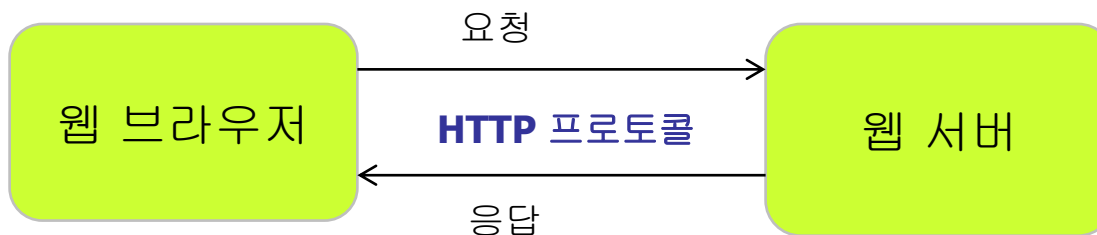
HTTP 프로토콜의 이해

프로토콜(Protocol) - 통신규약, 규칙

- 통신 시스템이 데이터를 교환하기 위해 사용하는 통신 규칙
- 컴퓨터간에 정보를 주고받을 때의 통신방법에 대한 규칙과 약속

■ HTTP 프로토콜

- 웹 브라우저와 웹 서버 사이의 데이터 통신 규칙
- 웹 페이지의 링크를 클릭하면 웹 브라우저는 HTTP 요청 형식에 따라 웹 서버에 데이터를 보냄
- 웹 서버는 웹 브라우저가 보낸 데이터를 분석하여 요청 받은 일을 처리하여 응답함
- 즉, Html 파일을 요청하면 해당 파일을 찾아서 보내주고, 이미지 파일을 요청하면 찾아서 보내줌
- 이때 보내는 데이터는 HTTP 응답 형식에 맞추어 보냄





HTTP 프로토콜

■ HTTP

- http 는 tcp/ip 위에서 돌아감
- http는 웹에서만 사용하는 프로토콜
- stateless, connectless의 특징을 갖는 프로토콜
- http는 tcp/ip를 기반으로 하여, tcp/ip를 이용해서 한 지점에서 다른 지점으로 요청과 응답을 전송함
- http의 구조 - 요청(request)/응답(response)의 끊임없는 주고 받음
 - 클라이언트는 요청하고 서버는 여기에 응답한다

■ HTML

- HTTP 응답 안에 html 컨텐츠가 데이터로 포함되어 있다

HTTP 프로토콜

■ 구글 크롬 - 도구 - 개발자 도구

Developer Tools - http://121.173.144.120:9090/testsite/now1.jsp?name=123

Elements Network Sources Timeline Profiles Resources Audits Console

Preserve log Disable cache

Name Path

now1.jsp?name=123 /testsite

Headers Preview Response Cookies Timing

Remote Address: 121.173.144.120:9090
Request URL: http://121.173.144.120:9090/testsite/now1.jsp?name=123
Request Method: GET
Status Code: 200 OK

▼ Request Headers view parsed

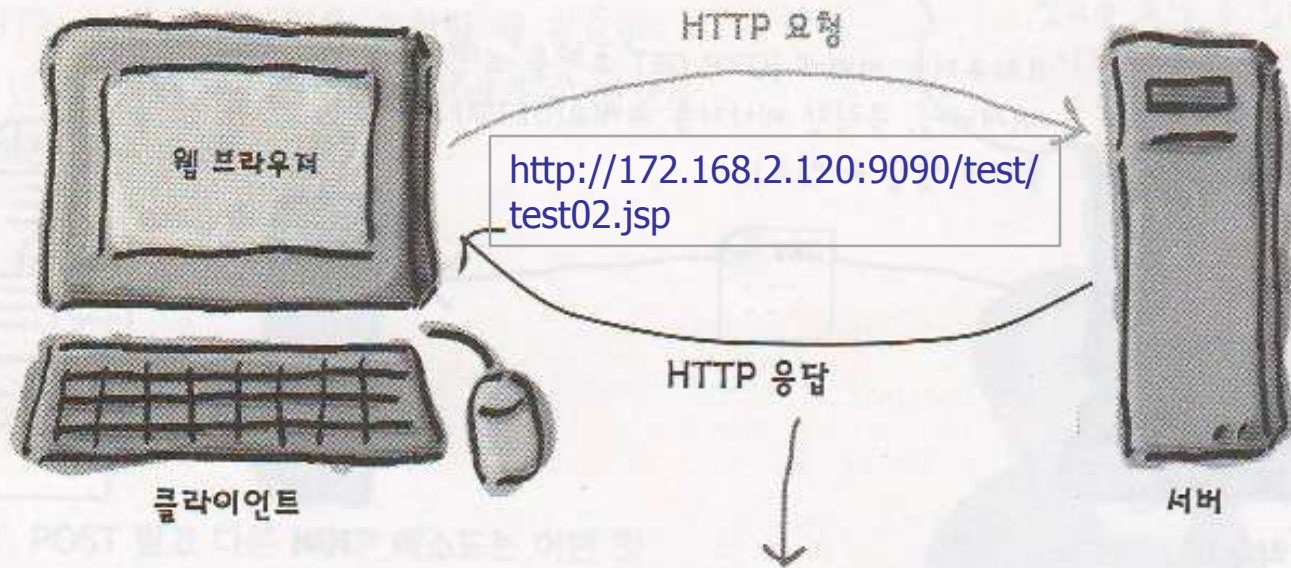
GET /testsite/now1.jsp?name=123 HTTP/1.1
Host: 121.173.144.120:9090
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept-Encoding: gzip, deflate, sdch
Accept-Language: ko-KR;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: JSESSIONID=37531E6BCD95C425CFC145D5CEE7A55C

▼ Query String Parameters view parsed

name=123

▼ Response Headers view parsed

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html; charset=euc-kr
Content-Length: 788
Date: Tue, 16 Sep 2014 11:11:18 GMT



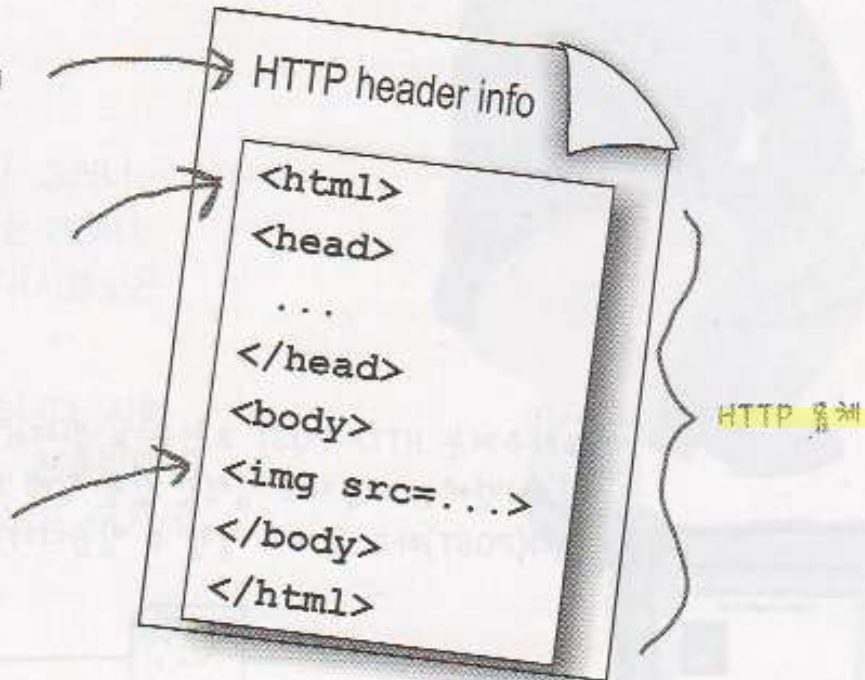
- 요청의 주요 구성요소
- HTTP메소드 (GET, POST)
- 접근하고자 하는 페이지(url)
- 폼 파라미터

- 응답의 주요 구성요소
- 상태코드
- 컨텐츠 타입 (텍스트, 그림, html 등)
- 컨텐츠(html, 코드, 이미지 등)

HTTP 헤더

브라우저가 `<html>` 태그를 만나면, "이제 페이지를 출력할 때가 됐군" 이라고 생각합니다.

브라우저가 `` 태그를 만나면, "음 .. 그림이란 말이네. 그림 그림을 가져와야지" 라고 말한 후, `` 태그에 적힌 주소에서 이미지를 가져오기 위하여 HTTP 요청을 하나 더 만듭니다.



GET

사용자가 링크를
클릭합니다.



사용자



브라우저

브라우저는 서버에 HTTP GET 요청을 보냅니다(서버님, 클릭한 페이지를 주세요(GET)라고
풀어서 말할 수 있죠).



서버



POST

폼에 정보를 입력하고
서밋(Submit) 버튼을
클릭합니다.



사용자

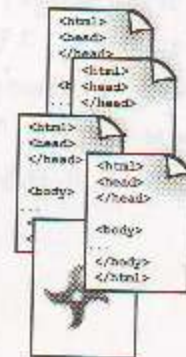


브라우저

브라우저는 HTTP POST 요청을 보냅니다(서버
님, 이번에는 사용자가 입력한 값을 함께 보냈
니다(POST)라고 풀어서 말할 수 있습니다).



서버



GET-단순한 요청은 get

요청 라인

HTTP 메소드

웹 서버 상 자원에 대한 경로

파라미터는 URL 바로 다음? 문자를 구분자로 하여 기술합니다. 개별 파라미터는 & 값으로 구분하죠.

브라우저가 요청한 프로토콜 버전

GET /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1

Host: www.wickedlysmart.com

User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/20030624 Netscape/7.1

Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

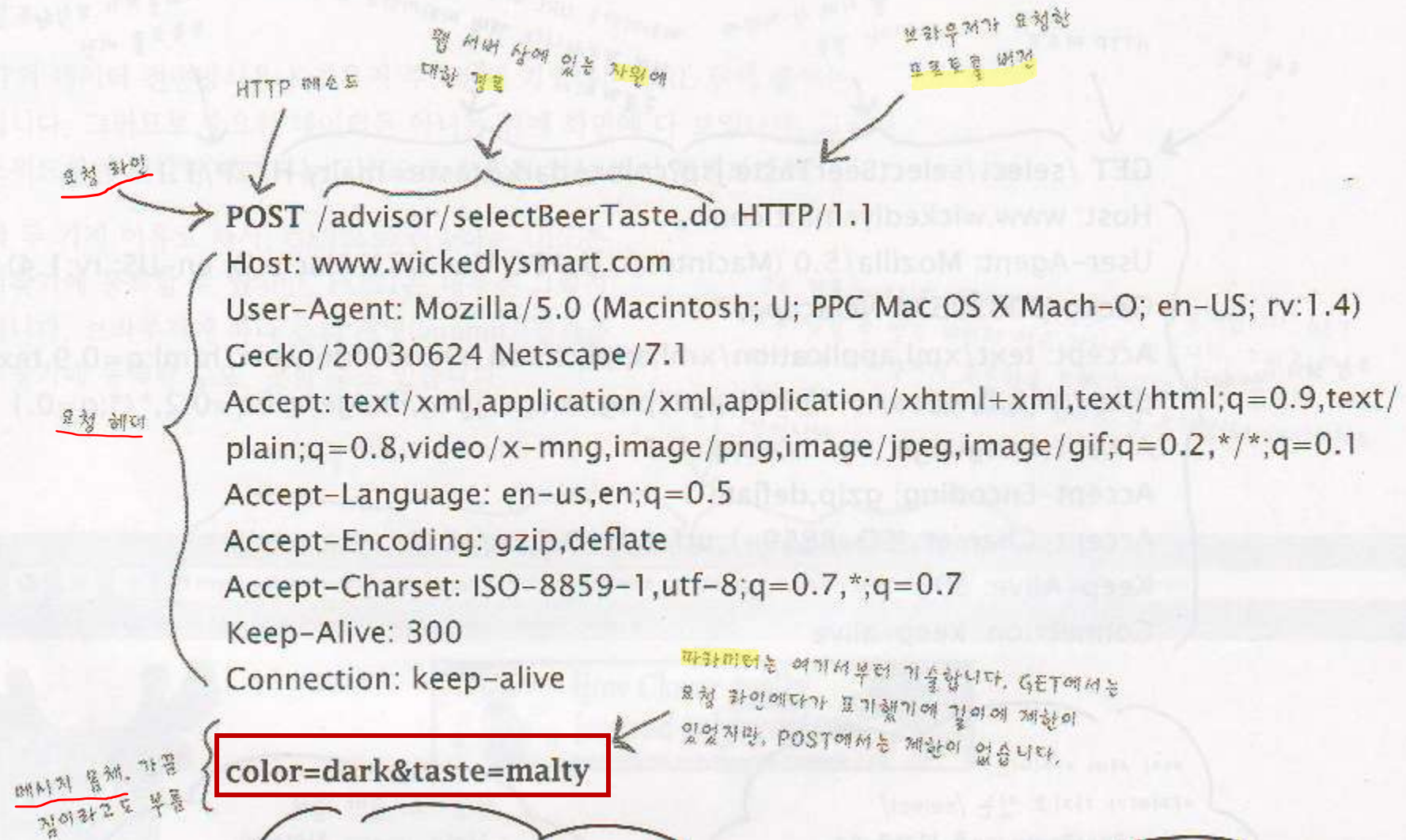
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

요청 헤더(Header)

Post-사용자가 입력한 정보를 함께 보내려면 post





Http Method

■ Get

- HTTP header(요청 라인)에 정보를 실어 보냄.
- URL뒤에 붙는다.
- 적은 양의 데이터 전송에 좋고, 전달 속도가 빠르지만 256byte가 한계

■ Post

- HTTP의 body(메시지 몸체)에 정보를 실어보냄.
- 데이터 사이즈에 제한이 없다.
- 보안에 좋다.

HTTP 응답

웹 서버가 사용한 프로토콜 버전
HTTP 상태 코드
상태 코드에 대한 텍스트 버전

HTTP 응답 헤더

HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=0AAB6C8DE415E2E5F307CF334BFCA0C1; Path=/testEL
Content-Type: text/html
Content-Length: 397
Date: Wed, 19 Nov 2003 03:25:40 GMT
Server: Apache-Coyote/1.1
Connection: close

Content-type의 값을 보통 MIME 타입이라고 부릅니다. MIME 타입이란 브라우저에게 "지금 서버가 이러 이러한 데이터를 보내려고 하니, 화면에 보여줄 준비를 하시오"라는 정보입니다. 서버가 보내주는 MIME 타입은 클라이언트가 보낸 요청의 헤더 중 Accept란에 기술되어 있는 값과 관련 있습니다(앞 페이지 POST의 헤더 정보를 다시 한번 보세요).

문체에는 컨텐츠가 들어갈 때, 지금 여기에 HTML 페이지가 들어 있습니다.

<html>
...
</html>

한번에 다 보도록 하지요



사용자가 주소
창에 URL을
입력합니다.



브라우저는 이 정보로 HTTP GET
요청을 만듭니다.

```
GET /test1/Beer1.html HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh...
...
```

HTTP GET을 서버로
보냅니다.



서버

서버는 요청된 페이지를
서버에서 찾고...

HTTP 응답을
작성합니다.

```
HTTP/1.1 200 OK
Set-Cookie: ...
...
<html><body>
<h1 align=center>Beer Login Page</h1>
<form>
  Select a beer type or buy beer
  making supplies?<p>
  <input type=radio name=select
  value=Select> Select a beer<br>
  <input type=radio name=select
  value=Buy> Buy supplies<br><br>
  <center>
    <input type=SUBMIT>
  </center>
</form>
</body></html>
```

Beer1.html

```
<html><body>
<h1 align=center>Beer Login Page</h1>
<form>
  Select a beer type or buy beer
  making supplies?<p>
  <input type=radio name=select
  value=Select> Select a beer<br>
  <input type=radio name=select
  value=Buy> Buy supplies<br><br>
  <center>
    <input type=SUBMIT>
  </center>
</form>
</body></html>
```

HTTP 응답을
클라이언트로
보냅니다.



클라이언트

브라우저는 HTML을
화면에 표시합니다.



Beer Login Page

Select a beer type or buy beer making supplies?

☐ Select a beer
☐ Buy supplies

Submit

아하~ 성공했군요
이제 맥주 한잔 할
수 있겠군요~





GET/ POST

- GET 요청의 데이터 전달 형식

url주소?파라미터명1=값1&파라미터명2=값2

http://abc.com:9090/test/now1.jsp?**name=홍길동&age=20**

■ GET

- 웹 브라우저의 주소창에 URL을 입력하거나
- 웹 페이지에서 링크를 클릭하는 경우에는 GET요청을 서버에 보냄
- 입력 폼의 method 속성값이 get인 경우

■ POST

- 사용자가 입력한 정보를 함께 보내려면 post
- 폼에 정보를 입력하고 submit 버튼을 클릭하는 경우



JDBC



JDBC

■ JDBC(Java Database Connectivity)

- 자바 프로그램과 데이터베이스를 연결하는 프로그래밍 방식
- 자바언어로 데이터베이스에 접근할 때 사용되는 API로 java.sql 패키지 의미함
- java 프로그램은 JDBC를 통해 데이터베이스에 연결하여 데이터를 검색하고, 입력, 수정, 삭제할 수 있음
- 데이터베이스에 접근할 경우 내부적으로 JDBC를 사용함

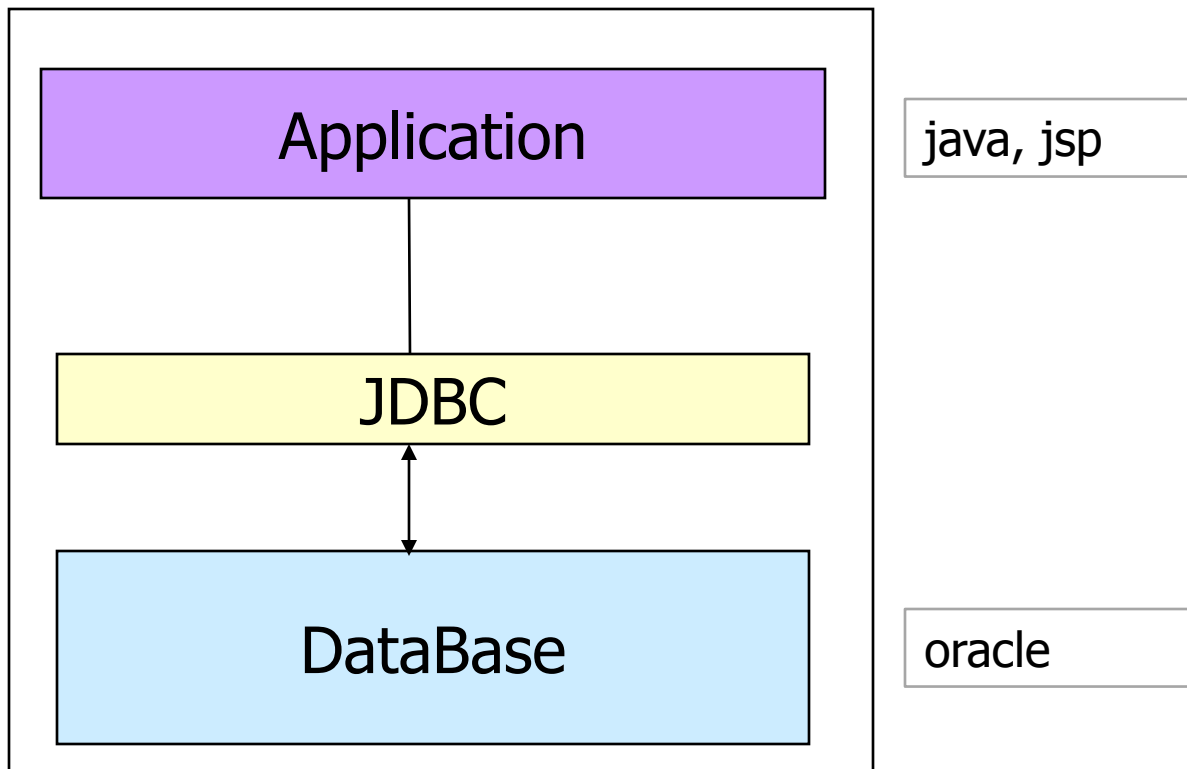
■ 드라이버 설치(ojdbc5.jar, ojdbc6.jar)

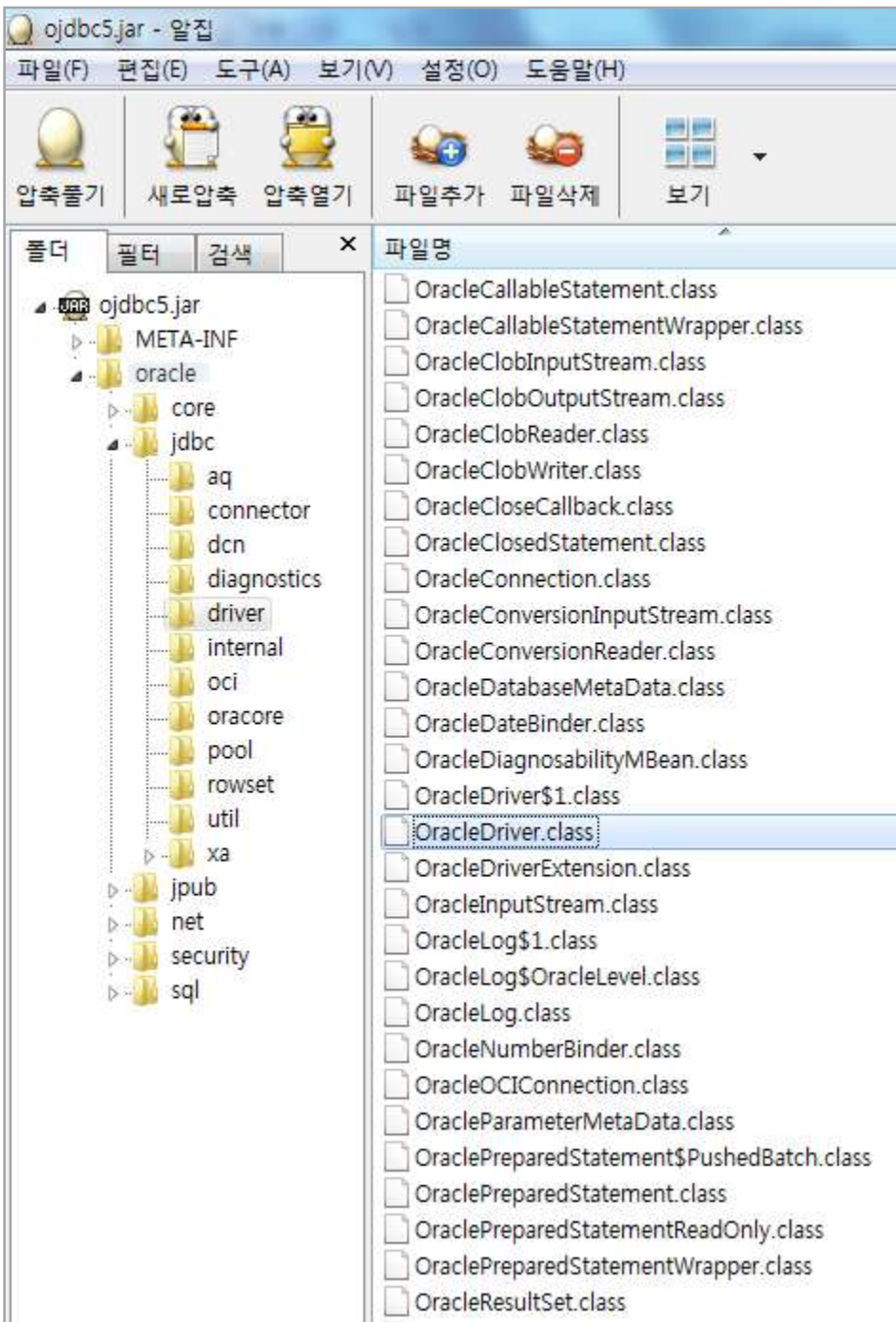
- 자바 프로그램에게, 연결해서 사용할 데이터베이스 프로그램의 사용방법을 알려주는 것
- C:\Wapp\Wyang\Wproduct\W11.2.0\Wdbhome_1\Wjdbc\Wlib\Wojdbc6.jar 를 복사하여
c:\Wjava\Wjdk1.8.0_51\Wjre\Wlib\Wext 에 붙여넣기

JDBC

■ JDBC

- 데이터에 대한 연결성을 제공(연결통로)
- 필요한 데이터들에 대한 접근을 제공하는 역할





<http://docs.oracle.com/javase/8/docs/api/index.html>

ojdbc5.jar

oracle.jdbc.driver.OracleDriver

java.sql 패키지의 인터페이스들을 DBMS 벤더(오라클, ms sql, mysql 등)들이 상속받아 JDBC Driver(클래스 파일들)를 구현.

Interfaces

Array
Blob
CallableStatement
Clob
Connection
DatabaseMetaData
Driver
NClob
ParameterMetaData
PreparedStatement
Ref
ResultSet
ResultSetMetaData
RowId
Savepoint
SQLData
SQLInput
SQLOutput
SQLXML
Statement
Struct
Wrapper



설정정보 확인

- 설정정보 확인

- C:\Wapp\Wyang\product\11.2.0\dbhome_1\NETWORK\ADMIN\tnsnames.ora, listener.ora => host명과

- 자바소스에서

String url = "jdbc:oracle:thin:@192.168.0.105:1521:orcl";

"jdbc:oracle:thin:@<HOST>:<PORT>:<SID>"

- <HOST>명을 일치시켜야 함

JDBC 프로그래밍 순서

• url 형식

프로토콜 : 서브프로토콜 : SID

jdbc:oracle:thin:@localhost:1521:orcl

- 1. 데이터베이스와 연결하는 드라이버 클래스 찾기(드라이버 로딩)
 - Class.forName("oracle.jdbc.driver.OracleDriver");
- 2. 드라이버 클래스를 통해 데이터베이스 서버와 연결하는 Connection 객체 생성
 - String url="jdbc:oracle:thin:@localhost:1521:orcl";
 - String id="javauser1", pwd="java";
 - Connection con = DriverManager.getConnection(url, id, pwd);
- 3. 작업을 처리할 Statement, PreparedStatement, CallableStatement 객체 생성
 - Statement stmt = con.createStatement();
 - 또는 PreparedStatement pstmt = con.prepareStatement("쿼리문")
- 4. Statement/PreparedStatement를 통해 쿼리문 전송
 - (1) insert, delete, update 문인 경우
 - int cnt = stmt.executeUpdate()
 - (2) select 문인 경우
 - ResultSet rs = stmt.executeQuery()
- 5. ResultSet 객체를 통한 Query 결과 처리
- 6. 접속 종료(자원 반납)
 - rs.close(); stmt.close(); con.close();
 - null체크해서 close()해주고, finally 블록에서 구현

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

(1) OracleDriver를 메모리에 load => OracleDriver 객체 생성

(2) 드라이버 객체를 DriverManager에 등록

- DriverManager : 드라이버들을 관리 등록함 (Vector로 되어 있어서 드라이버를 Vector에 담고 있으면서 DB와 연결함)

JDBC 프로그래밍 순서

▶	#	NO	NAME	TEL
	1	1	홍길동	010-100-2000
	2	2	김연아	010-200-5000
	3	3	윤아	010-300-7000
	4	5	이승기	010-111-2222

- 5. select 문일 경우
 - ResultSet의 논리적 커서를 이동시키면서 각 컬럼의 데이터를 꺼내온다.
 - `boolean b=rs.next()` : 커서 이동, 커서가 위치한 지점에 레코드가 있으면 `true`를 리턴, 없으면 `false`를 리턴한다.
 - 커서는 맨 처음에 첫 번째 행의 직전에 위치하고 있다가, `next()`가 호출되면 다음 행으로 이동한다.
- `rs.get~(컬럼인덱스) / rs.get~(컬럼명)` 메소드 : 데이터를 꺼내온다.
 - get뒤에는 컬럼의 데이터 유형에 맞는 자료형을 기재
 - number 인 경우 `rs.getInt(1);`
 - varchar2인 경우 `rs.getString(2);`
 - date인 경우 `rs.getDate("regdate");`

```
rs.getInt("no");  
rs.getString("name");
```



기본 SQL문

```
select [칼럼명 또는 표현식]  
from [테이블명, 뷰명]  
where 원하는 조건;
```

```
select * from mem where id='hong';
```

```
INSERT INTO table [(column1, column2, ...)]  
VALUES (value1, value2, ....)
```

```
INSERT INTO dept2 (dcode, dname, pdept , area)  
VALUES (9000, '특판1팀', '영업부', '임시지역');
```

```
insert into mem(no, id, pwd, name, hp)  
values (mem_seq.nextval, 'hong', '1','홍길동', '010-100-2000') ;
```



기본 SQL문

```
UPDATE table  
SET column = value  
WHERE 조건;
```

```
UPDATE professor  
SET bonus = 100, email='h@nate.com'  
WHERE position='조교수';
```

```
DELETE FROM table  
WHERE 조건;
```

```
DELETE FROM dept2  
WHERE dcode between 9000 and 9100;
```



register_ok.jsp – DB처리

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

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

```
<%
```

```
    request.setCharacterEncoding("euc-kr");
```

```
    String id=request.getParameter("id");
```

```
    String pwd=request.getParameter("pwd");
```

```
    String name=request.getParameter("name");
```

```
    String hp=request.getParameter("hp");
```

```
    String url ="jdbc:oracle:thin:@220-00:1521:ORCL";
```

```
    String uid = "javauser1", upwd="java";
```

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

```
    Connection con = DriverManager.getConnection(url, uid, upwd);
```

```
    out.print("DB연결 성공!<br>");
```

```
    String sql = "insert into mem(no, id, pwd, name, hp)" +  
        " values (mem_seq.nextval, ?,?,?,?)" ;
```

```
    PreparedStatement pstmt=con.prepareStatement(sql);
```

```
    pstmt.setString(1, id);
```

```
    pstmt.setString(2, pwd);
```

```
    pstmt.setString(3, name);
```

```
    pstmt.setString(4, hp);
```

-오라클 드라이버 **ojdbc6.jar**를 복사하여
c:\ Tomcat 7.0\ lib 에 붙여넣기해야
ClassNotFoundException 발생 안함

- 톰캣 켜다 켜야 함



register_ok.jsp – DB처리

```
int result = pstmt.executeUpdate();
if(result>0){
    out.print("<script>");
    out.print("alert('등록 성공!!');");
    out.print("location.href='main.jsp';");
    out.print("</script>");
}else{
    out.print("<script>");
    out.print("alert('등록 실패!!');");
    out.print("history.back()");
    out.print("</script>");
}

pstmt.close();
con.close();

%>
```

http://localhost:9091/jdbcTest/pd/pdWrite.jsp

상품 등록

상품명	<input type="text"/>
가격	<input type="text"/>
<input type="button" value="등록"/> <input type="button" value="취소"/>	

[상품 목록](#)

http://localhost:9091/jdbcTest/pd/pdList.jsp

상품 목록

번호	상품명	가격	등록일
6	usb	20,000원	2013-03-02
5	노트북	1,500,000원	2013-03-02
4	외장하드	190,000원	2013-03-02
3	모니터	270,000원	2013-03-02
2	키보드	30,000원	2013-03-02

[상품 등록](#)

http://localhost:9091/jdbcTest/pd/pdDetail.jsp?no=5

상품 상세보기

5을 클릭하였습니다

번호 : 5
 상품명 : 노트북
 가격 : 1,500,000원
 등록일 : 2013-03-02 22:29:54.0

[목록](#) | [수정](#) | [삭제](#)

http://localhost:9091/jdbcTest/pd/pdEdit.jsp?no=5

상품 수정

상품명	<input type="text" value="노트북"/>
가격	<input type="text" value="1500000"/>
<input type="button" value="수정"/> <input type="button" value="취소"/>	

[상품 목록](#)



table

```
--drop table pd;  
create table pd  
(  
  no number primary key,  
  pdName varchar2(50) not null,  
  price number null,  
  regdate date default sysdate  
);
```

```
--drop sequence pd_seq;  
create sequence pd_seq  
increment by 1  
start with 1  
nocache;
```

pdWrite_ok.jsp

```
<%@page import="java.sql.*"%>
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<% //pdWrite.jsp 페이지에서 등록버튼을 누르면 post방식으로 submit
    //=> 사용자가 입력한 값을 읽어와서 db에 insert한다

//1. post방식으로 넘긴 요청 파라미터에 대한 인코딩 처리(한글처리)
request.setCharacterEncoding("euc-kr");

//2. 사용자가 입력한 값 읽어오기(post방식으로 넘긴 파라미터 읽어오기)
String pdName = request.getParameter("pdName");
String price = request.getParameter("price");

//3. db작업 - insert처리
Connection con=null;
PreparedStatement ps=null;
try{
    //[1] JDBC 드라이버 로딩
    Class.forName("oracle.jdbc.driver.OracleDriver");
    System.out.println("드라이버 로딩 성공!!<br>");

    //[2] DB와 연결 - Connection객체 생성
    String url="jdbc:oracle:thin:@yang2:1521:orcl";
    String uid="javauser1", upwd="java";
    con=DriverManager.getConnection(url, uid, upwd);
```


pdWrite_ok.jsp

```
System.out.println("db연결 성공!!<br>");

//[3] DB 에 sql 문을 전송하기 위한 PreparedStatement객체 생성
String sql="insert into pd(no, pdname, price) values(pd_seq.nextval,?,?)";
ps = con.prepareStatement(sql);
ps.setString(1, pdName);
ps.setInt(2, Integer.parseInt(price));

//[4] sql문 전송 - 실행
int n = ps.executeUpdate();

//[5] 결과 처리-화면 출력
if(n>0){
    System.out.println("입력성공!");
    //get방식으로 넘기기 => link 걸기
    response.sendRedirect("pdList.jsp");
}else{
    out.print("입력 실패!!!");
}
}catch(ClassNotFoundException e){
    System.out.println("class not found!!!" +e+"<br>");
}catch(SQLException e){
    System.out.println("입력시 sql 예외발생:" +e+"<br>");
}
```

```
finally{
    try{
        if(ps!=null)ps.close();
        if(con!=null)con.close();
    }catch(SQLException e){
        e.printStackTrace();
    }
}
%>
```

pdList.jsp

```
<%@page import="java.sql.*, java.text.*"%>
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<% //pdWrite_ok.jsp 에서 입력 성공하면 get방식으로 이동된다
    //pdWrite.jsp 에서 상품목록 링크를 누르면 get방식으로 이동된다
    //http://localhost:9090/jdbcTest/pd/pdList.jsp
    //=> pd 테이블에서 전체 내용을 조회해서 화면 출력하자
    //1. get방식 한글처리
    //2. 파라미터값 읽어오기
    //3. db 작업 - select
    DecimalFormat df = new DecimalFormat("#,###");
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

    Connection con=null;
    PreparedStatement ps=null;
    ResultSet rs=null;
    try{
        //[1] 드라이버 로딩
        Class.forName("oracle.jdbc.driver.OracleDriver");
        System.out.println("드라이버 로딩 성공!");

        //[2] 연결 - con
        String url="jdbc:oracle:thin:@yang2:1521:orcl";
        String uid="javauser1", upwd="java";
        con = DriverManager.getConnection(url, uid, upwd);
        System.out.println("db연결 성공!");
```

```
//[3] sql문장 작성 - ps
String sql="select * from pd order by no desc";
ps=con.prepareStatement(sql);
```

```
//[4] 실행
rs = ps.executeQuery();
```

```
%>
<h2>상품 목록</h2>
<table width="500" border="1">
    <tr><th>번호</th><th>상품명</th><th>가격</th><th>등록일</th></tr>
```

```
<!------- 반복문 시작 -->
```

```
<%
while(rs.next()){
    int no=rs.getInt("no");
    String pdName=rs.getString("pdname");
    int price=rs.getInt("price");
    Timestamp regdate = rs.getTimestamp("regdate");
```

```

    <tr>
        <td align="center"><%=no %></td>
        <td>
            <a href="pdDetail.jsp?no=<%=no%>">
                <%=pdName%>
            </a>
        </td>
        <td align="right"><%=df.format(price)%>원</td>
        <td align="center"><%=sdf.format(regdate)%></td>
    </tr>
```

```
<%
} //while %>
```

```
<!------- 반복문 끝 -->
```

```
</table>
```

```
<br><br>
```

```
<a href="pdWrite.jsp">상품 등록</a>
```

```
</body>
```

```
</html>
```

```
<%
```

```
    }catch(ClassNotFoundException e){
```

```
        System.out.println("class not found!!!" +e+"<br>");
```

```
    }catch(SQLException e){
```

```
        System.out.println("상품 전체 조회시 sql 예외발생!" + e);
```

```
        e.printStackTrace();
```

```
    }finally{
```

```
        try{
```

```
            if(rs!=null)rs.close();
```

```
            if(ps!=null)ps.close();
```

```
            if(con!=null)con.close();
```

```
        }catch(SQLException e){
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
%>
```



pdDetail.jsp

```
<%@page import="java.text.DecimalFormat"%>
<%@page import="java.sql.*"%>
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<script type="text/javascript">
    function del(no){
        var delYn = confirm("삭제하시겠습니까?");
        //if(delYn==true){
        if(delYn){
            location.href="pdDelete_ok.jsp?no="+no;
        }//if
    }
</script>
<%
    //pdList.jsp에서 상품명을 클릭하면 이동됨, get 방식으로 이동한 것
    //http://localhost:9090/mystudy/pd/pdDetail.jsp?no=21
    //파라미터로 넘어온 no에 해당하는 레코드(상품 하나)를 조회해서 화면 출력
    //1. get - 한글 처리
    //2. get방식으로 넘긴 파라미터 읽어오기
    String no = request.getParameter("no");
```

```

DecimalFormat df = new DecimalFormat("#,###");

//3.db 작업 - select
Connection con=null;
PreparedStatement ps=null;
ResultSet rs=null;

String pdname="";
int price=0;
Timestamp regdate=null;
try{
    //[1] 드라이버 로딩
    Class.forName("oracle.jdbc.driver.OracleDriver");
    out.print("드라이버 로딩 성공!");

    //[2] 연결 - con
    String url="jdbc:oracle:thin:@yang2:1521:orcl";
    String uid="javauser1", upwd="java";
    con = DriverManager.getConnection(url, uid, upwd);
    out.print("db연결 성공!");

    //[3] sql문장 처리- ps
    String sql="select * from pd where no=?";
    ps=con.prepareStatement(sql);
    ps.setInt(1, Integer.parseInt(no));

    //[4] 실행
    rs = ps.executeQuery();

```

//[5] 화면 처리

```
if(rs.next()){  
    pdname=rs.getString("pdname");  
    price=rs.getInt("price");  
    regdate = rs.getTimestamp("regdate");
```

}//if

```
}catch(ClassNotFoundException e){
```

```
    e.printStackTrace();
```

```
}catch(SQLException e){
```

```
    e.printStackTrace();
```

```
}finally{
```

```
    try{
```

```
        if(rs!=null)rs.close();    if(ps!=null)ps.close();    if(con!=null)con.close();
```

```
    }catch(SQLException e){        e.printStackTrace();    }
```

```
}//try
```

%>

<h2>상품 상세보기</h2>

<%=no %>을 클릭하였습니다

번호 : <%=no %>

상품명 : <%=pdname %>

가격 : <%=df.format(price)%>원

등록일 : <%=regdate %>

목록 |

<a href="pdEdit.jsp?no=<%=no%>">수정 |

<a href="#" onclick="del(<%=no%>)">삭제



pdDelete_ok.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@page import="java.sql.*"%>
<% //1. get- 한글 처리
```

```
    //2. get방식으로 넘긴 파라미터 읽어오기
    String no=request.getParameter("no");
```

```
    //3. db작업 - update
    Connection con=null;
    PreparedStatement ps=null;
    try{    //[1] 드라이버 로딩
        Class.forName("oracle.jdbc.driver.OracleDriver");
        out.print("드라이버 로딩 성공!!<br>");
```

```
        //[2] 연결 - Connection객체 생성
        String url="jdbc:oracle:thin:@yang2:1521:orcl";
        String uid="javauser1", upwd="java";
```



```
con=DriverManager.getConnection(url, uid, upwd);  
out.print("db연결 성공!!<br>");
```

```
//[3] sql 문장 처리 - PreparedStatement객체 생성  
String sql="delete from pd where no=?";  
ps = con.prepareStatement(sql);  
ps.setInt(1, Integer.parseInt(no));
```

```
//[4] 실행
```

```
int n = ps.executeUpdate();
```

```
//[5] 결과 처리-화면 출력
```

```
if(n>0){
```

```
    out.print("삭제성공!");
```

```
    response.sendRedirect("pdList.jsp");
```

```
}else{
```

```
    out.print("삭제 실패!!!");
```

```
}
```

```
}catch(ClassNotFoundException e){
```

```
    e.printStackTrace();
```

```
}catch(SQLException e){
```

```
    e.printStackTrace();
```

```
}finally{
```

```
    try{
```

```
        if(ps!=null)ps.close();
```

```
        if(con!=null)con.close();
```

```
    }catch(SQLException e){
```

```
        e.printStackTrace();    }
```

```
}
```

```
%>
```



pdEdit.jsp

```
<%@page import="java.sql.*"%>
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%
    //pdDetail.jsp에서 수정 링크버튼을 눌러서 이동됨 => get 방식으로 이동
    //http://localhost:9090/mystudy/pd/pdEdit.jsp?no=21
    //no에 해당하는 상품을 조회해서 화면에 출력한다
    //1. get - 한글처리
    //2. get방식으로 넘긴 파라미터 읽어오기
    String no=request.getParameter("no");

    //3. db 작업 - select
    Connection con=null;
    PreparedStatement ps=null;
    ResultSet rs=null;

    String pdname="";
    int price=0;
    Timestamp regdate=null;
    try{
        //[1] 드라이버 로딩
        Class.forName("oracle.jdbc.driver.OracleDriver");
        //out.print("드라이버 로딩 성공!");
```



```
<html><body>
<h2>상품 수정</h2>
<form name="frmPd" method="post" action="pdEdit_ok.jsp">
  <!-- pdEdit_ok.jsp 페이지에서 no가 필요하므로 hidden 필드에 값을 넣어주자 -->
  <input type="hidden" name="no" value="<%=no%>">
  <table width="400" border="1">
    <tr>
      <td width="100">상품명</td>
      <td><input type="text" name="pdName" size="30" value="<%=pdname%>"></td>
    </tr>
    <tr>
      <td>가격</td>
      <td><input type="text" name="price" value="<%=price%>"></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input type="submit" value="수정">
        <input type="reset" value="취소">
      </td>
    </tr>
  </table>
</form>

<br><br>
<a href="pdList.jsp">상품 목록</a>

</body>
</html>
```



pdEdit_ok.jsp

```
<%@page import="java.sql.*"%>
<%@ page language="java" contentType="text/html; charset=EUC-KR"    pageEncoding="EUC-KR"%>
<% //pdEdit.jsp에서 [수정]버튼을 클릭해서 SUBMIT됨(이동됨) => post 방식
    //http://localhost:9090/mystudy/pd/pdEdit_ok.jsp
    //사용자가 입력한 값(파라미터)을 읽어와서 update 한다
    //1. post - 한글 처리
    request.setCharacterEncoding("euc-kr");
    //2. post방식으로 넘긴 파라미터 읽어오기(사용자가 입력한 값)
    String no=request.getParameter("no");
    String pdName = request.getParameter("pdName");
    String price = request.getParameter("price");
    //3. db작업 - update
    Connection con=null;
    PreparedStatement ps=null;
    try{
        //[1] 드라이버 로딩
        Class.forName("oracle.jdbc.driver.OracleDriver");
        out.print("드라이버 로딩 성공!!<br>");
        //[2] 연결 - Connection객체 생성
        String url="jdbc:oracle:thin:@yang2:1521:orcl";
        String uid="javauser1", upwd="java";
        con=DriverManager.getConnection(url, uid, upwd);
        out.print("db연결 성공!!<br>");
```

```

//[3] sql 문장 처리 - PreparedStatement객체 생성
String sql="update pd set pdname=?, price=? where no=?";
ps = con.prepareStatement(sql);
ps.setString(1, pdName);
ps.setInt(2, Integer.parseInt(price));
ps.setInt(3, Integer.parseInt(no));

//[4] 실행
int n = ps.executeUpdate();

//[5] 결과 처리-화면 출력
if(n>0){
    out.print("수정 성공!");
    response.sendRedirect("pdDetail.jsp?no="+no);
}else{
    out.print("수정 실패!!!");
}
}catch(ClassNotFoundException e){
    e.printStackTrace();
}catch(SQLException e){
    e.printStackTrace();
}finally{
    try{
        if(ps!=null)ps.close();
        if(con!=null)con.close();
    }catch(SQLException e){
        e.printStackTrace();    }
}
}

```



web.xml

- web.xml(DD, Deployment Descriptor)
 - 배포 서술자
 - 서블릿과 jsp를 어떻게 실행하느냐에 관한 많은 정보들이 들어 있다
 - URL과 서블릿을 매핑
 - 보안 역할 설정, 오류 페이지 설정, 항목 라이브러리, 초기화 구성 및 관련 정보 설정 등
 - DD - 작성한 소스코드를 바꾸지 않고, 중요한 것들을 수정할 수 있다



인코딩 및 유니코드

■ 인코딩

- 문자코드를 컴퓨터가 이해할 수 있는 0과1의 바이너리 값을 가지는 연속적인 비트 형태로 대응시켜주는 작업
- 컴퓨터가 이해할 수 있는 코드 형태로 만들어 주는 것

■ 문자 코드(Character code)

- 문자들의 집합과 이 문자들을 나타내기 위해 정한 숫자 (문자코드)들을 1대1로 연결시켜 놓은 것
- 예) ASCII 문자 코드
 - A : 65, a : 97



문자코드

■ EUC-KR 인코딩

- 유닉스 운영체제에서 영어는 KS C 5636(ASCII문자에 대한 표준)을, 한글은 KS C 5601을 사용하는 것
- ASCII 문자코드는 1바이트로 표현,
- 한글 문자코드는 2바이트로 표현



유니코드

■ 유니코드

- 인간이 사용하는 모든 언어를 표현할 수 있도록 하기 위하여 만들어짐
- 기존 언어의 인코딩 체계를 모두 포함할 수 있도록 고안된 커다란 문자 집합
- 두 개의 대표적 문자 인코딩이 있음

■ UTF-8

- ASCII 문자코드는 1바이트로 인코딩, 다른 문자들은 2바이트나 그 이상으로 인코딩하는 방식
- 한글은 3바이트로 인코딩
- 기존의 ASCII 문자코드 체계와 그대로 호환이 가능
 - 인터넷상에서 문서를 교환하기 위한 기본적인 인코딩으로 환영받음
 - XML 문서 : 디폴트로 UTF-8 인코딩 사용

■ UTF-16

- 간단하게 2 바이트를 사용하여 모든 문자 코드를 표현



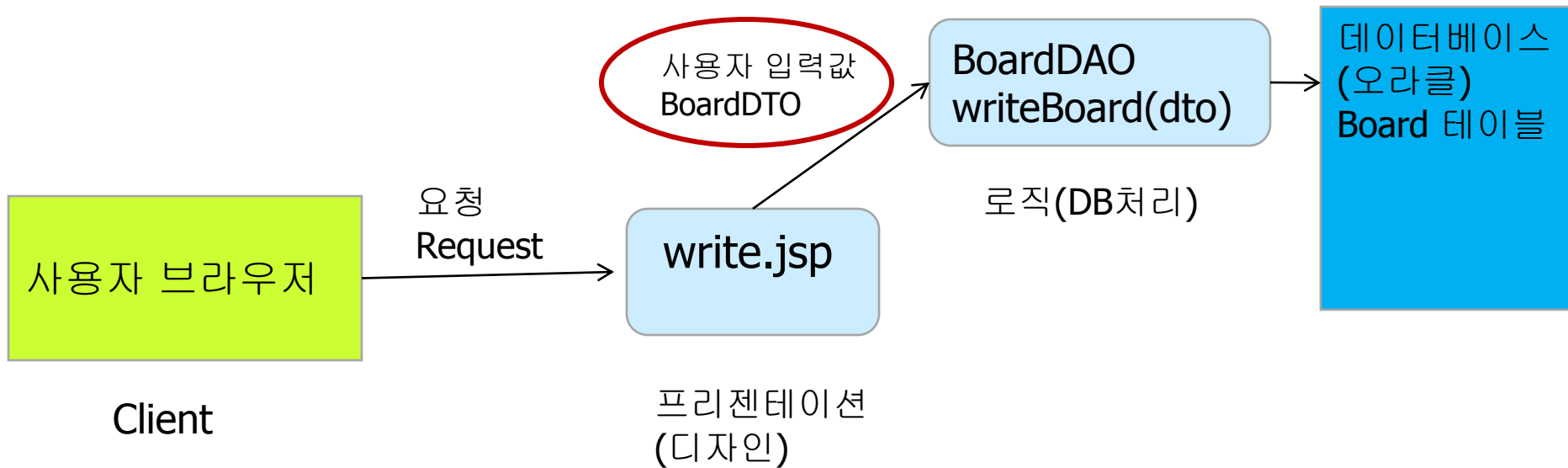
DAO, DTO 이용



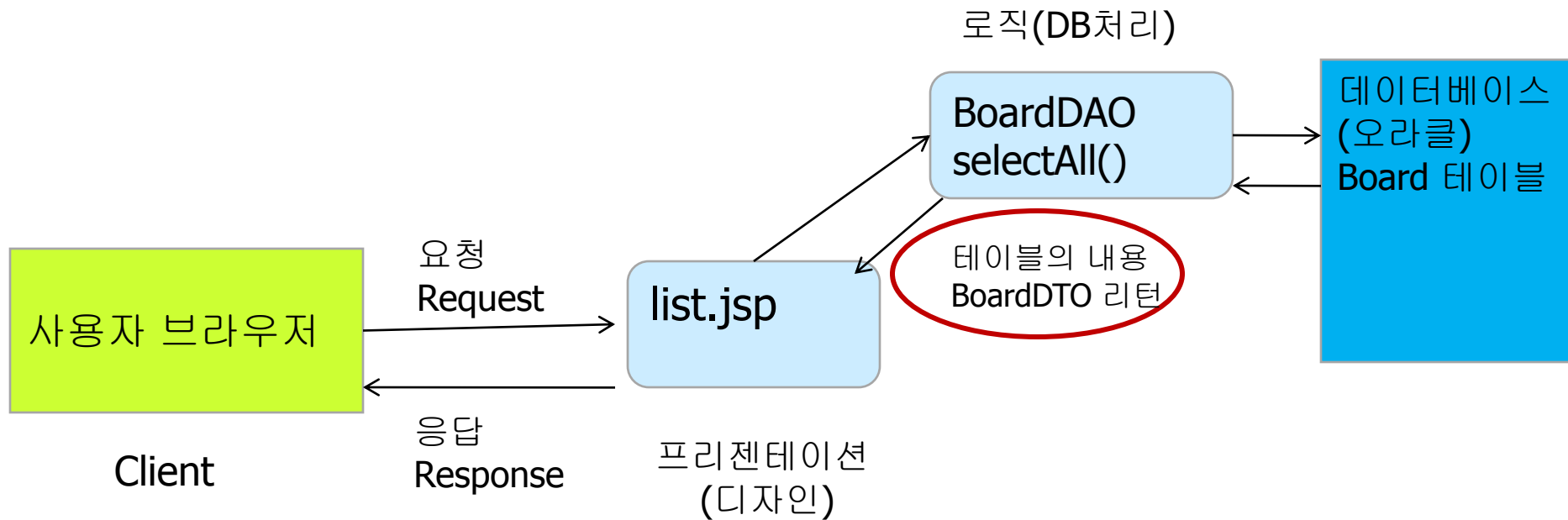
DAO(Data Access Object)

- DAO(Data Access Object)
 - 데이터베이스 작업을 전담하는 객체
 - CRUD
 - C : create, insert
 - R : read, select
 - U : update
 - D : delete
- DTO(Data Transfer Object), VO(Value Object), Bean
 - 객체를 표현한 한 단위
 - 데이터를 전달하는 단위
- 빈즈 규약(캡슐화된 객체)
 - 멤버변수는 private으로
 - 멤버변수에 대한 접근은 getter/setter로

DAO 이용 - 입력처리



DAO 이용 - 조회





PdDTO

```
public class PdDTO {  
    //멤버변수 - private  
    private int no;  
    private String pdName;  
    private int price;  
    private Timestamp regdate;  
    //getter/setter - public  
    public int getNo() {  
        return no;  
    }  
    public void setNo(int no) {  
        this.no = no;  
    }  
    ...  
  
    @Override  
    public String toString() {  
        return "PdDTO [no=" + no + ", pdName=" + pdName + ", price=" + price  
            + ", regdate=" + regdate + "];"  
    }  
}
```

```
public class ConnectionPoolMgr {
    public ConnectionPoolMgr(){
        //1. 드라이버 로딩
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("드라이버 로딩 성공!");
        } catch (ClassNotFoundException e) {
            System.out.println("드라이버 로딩 실패!!");
            e.printStackTrace();
        }
    }

    public Connection getConnection() throws SQLException{
        //2. db에 연결하는 연결객체 생성-Connection
        Connection con=null;
        String url="jdbc:oracle:thin:@yang-hp:1521:orcl8";
        String uid="javauser1", upwd="java";
        con = DriverManager.getConnection(url, uid, upwd);
        System.out.println("db연결 : con="+con);

        return con;
    }

    public void dbClose(PreparedStatement ps, Connection con) throws SQLException{
        if(ps!=null) ps.close();
        if(con!=null)con.close();
    }
}
```



```
public void dbClose(ResultSet rs, PreparedStatement ps,  
                    Connection con) throws SQLException{  
    if(rs!=null)rs.close();  
    if(ps!=null) ps.close();  
    if(con!=null)con.close();  
}
```

ConnectionPoolMgr

```
public void dbClose(CallableStatement cs, Connection con) throws SQLException{  
    if(cs!=null) cs.close();  
    if(con!=null)con.close();  
}  
}
```

```
public class PdDAO {  
    private ConnectionPoolMgr pool;  
    //생성자  
    public PdDAO(){  
        pool=new ConnectionPoolMgr();  
    }  
    public int insertPd(PdDTO pdDto) throws SQLException{  
        Connection con=null;  
        PreparedStatement ps=null;  
        int n=0;  
        try{  
            //[1], [2] 드라이버 로딩(생성자에서), 연결객체 생성(메서드 호출)  
            con = pool.getConnection();  
  
            //[3] sql문장을 처리하는 PreparedStatement객체 생성  
            String sql="insert into pd(no, pdName, price)"  
                +" values(pd_seq.nextval, ?, ?)";  
            ps = con.prepareStatement(sql);  
            ps.setString(1, pdDto.getPdName());  
            ps.setInt(2, pdDto.getPrice());  
  
            //[4] 실행  
            n= ps.executeUpdate();  
            System.out.println("상품 등록 여부, n="+n +", 입력값 pdDto : " + pdDto);  
        }finally{  
            pool.dbClose(ps, con);  
        }  
        return n;  
    }  
}
```

```
public ArrayList<PdDTO> selectAll() throws SQLException{           //상품 전체 목록 조회
    Connection con=null;
    PreparedStatement ps=null;
    ResultSet rs=null;
    //여러개의 레코드를 하나의 컬렉션에 담아서 리턴
    ArrayList<PdDTO> alist = new ArrayList<PdDTO>();
    try{
        //[1][2]con
        con = pool.getConnection();

        //[3] ps
        String sql="select * from pd order by no desc";
        ps=con.prepareStatement(sql);

        //[4] 실행
        rs = ps.executeQuery();
        while(rs.next()){
            int no = rs.getInt("no");
            String pdName = rs.getString("pdname");
            int price = rs.getInt("price");
            Timestamp regdate = rs.getTimestamp("regdate");

            //하나의 레코드(여러 칼럼)를 하나의 Bean으로 묶는다
            PdDTO pdDto = new PdDTO();
            pdDto.setNo(no);
            pdDto.setPdName(pdName);
            pdDto.setPrice(price);
            pdDto.setRegdate(regdate);
        }
    }
}
```

```

        alist.add(pdDto);
    } //while
    System.out.println("상품 목록 조회 , alist.size()="+alist.size());
} finally{
    pool.dbClose(rs, ps, con);
}
return alist;
}

public PdDTO selectByNo(int no) throws SQLException{ //상세보기 - no에 해당하는 상품 조회
    Connection con=null;
    PreparedStatement ps=null;
    ResultSet rs=null;

    //하나의 레코드를 Bean으로 묶어서 리턴
    PdDTO pdDto = new PdDTO();
    try{
        //[1][2] con
        con=pool.getConnection();

        //[3] ps
        String sql="select * from pd where no=?";
        ps=con.prepareStatement(sql);
        ps.setInt(1, no);

        //[4] 실행
        rs = ps.executeQuery();
        if(rs.next()){
            String pdName = rs.getString("pdname");

```

```

        int price = rs.getInt("price");
        Timestamp regdate = rs.getTimestamp("regdate");

        pdDto.setNo(no);
        pdDto.setPdName(pdName);
        pdDto.setPrice(price);
        pdDto.setRegdate(regdate);
    } //if
    System.out.println("상품 상세보기, pdDto="+pdDto+", 입력값 no=" + no);
} finally{
    pool.dbClose(rs, ps, con);
}
return pdDto;
}

public int updatePd(PdDTO pdDto) throws SQLException{           //상품 수정 처리
    Connection con=null;
    PreparedStatement ps=null;
    int n = 0;
    try{
        //[1][2] con
        con=pool.getConnection();

        //[3] ps
        String sql="update pd set pdname=?, price=?" +" where no=?";
        ps=con.prepareStatement(sql);
        ps.setString(1, pdDto.getPdName());
        ps.setInt(2, pdDto.getPrice());
        ps.setInt(3, pdDto.getNo());
    }
}

```



```

<%@page import="java.sql.SQLException"%>
<%@page import="com.mystudy.pd.model.PdDTO"%>
<%@page import="java.util.ArrayList"%>
<%@page import="com.mystudy.pd.model.PdDAO"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="java.text.DecimalFormat"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html><head><meta charset="UTF-8"><title>pdList.jsp</title></head>
<body>
    <h1>상품 목록</h1>
    <table width="500" border="1">
        <tr><th>번호</th><th>상품명</th><th>가격</th><th>등록일</th></tr>
    <%
        //가격-천단위 구분기호 처리하기 위해
        DecimalFormat df = new DecimalFormat("#,###");
        //날짜 - 년, 월, 일만 표시
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

        //db에서 pd테이블의 전체 레코드를 조회해서 화면에 출력하기
        //db작업 - select
        PdDAO pdDao=new PdDAO();
        try{
            ArrayList<PdDTO> alist=pdDao.selectAll();
            //----반복 시작
            for(int i=0;i<alist.size();i++){
                PdDTO dto =alist.get(i);
            }
        }
    %>

```

```

        <tr align="center">
            <td><%=dto.getNo() %></td>
            <td>
                <a href="pdDetail.jsp?no=<%=dto.getNo()%>">
                    <%=dto.getPdName() %>

                </a>
            </td>
            <td align="right">
                <%=df.format(dto.getPrice()) %>원</td>
            <td><%=sdf.format(dto.getRegdate()) %></td>
        </tr>

    <%
        }//for
        //-----반복 끝
    }catch(SQLException e){
        System.out.println("상품 목록 조회, sql error:"+e);
    }
%>
</table>
<br>
<a href="pdWrite.jsp">상품 등록</a>
</body>
</html>
```


<%

```
//사용자가 입력한 정보를 읽어와서 db에 저장하자
//1. post방식으로 넘긴 요청 파라미터 읽어오기
//한글 인코딩 처리
request.setCharacterEncoding("utf-8");
```

```
String pdName = request.getParameter("pdName");
String price = request.getParameter("price");
```

```
//2. db에 저장하기
PdDAO pdDao = new PdDAO();
```

```
//dto에 값을 셋팅해서 매개변수로 넘겨준다
PdDTO pdDto = new PdDTO();
pdDto.setPdName(pdName);
pdDto.setPrice(Integer.parseInt(price));
try{
    int n = pdDao.insertPd(pdDto);
    if(n>0){ //입력 성공한 경우
        response.sendRedirect("pdList.jsp");
    }else{ //실패
        System.out.print("상품 등록 실패!!");
        response.sendRedirect("pdWrite.jsp");
    }
}catch(SQLException e){
    System.out.println("상품 입력, sql error:"+e);
}
```

%>

pdDetail.jsp

```
<%
//파라미터인 no를 읽어와서 그 no 에 해당하는 상품을 조회한 후 화면 출력하기
//http://localhost:9090/mystudy/pd/pdDetail.jsp?no=7
//1. 파라미터 읽어오기
String no= request.getParameter("no");
if(no==null || no.isEmpty()){
    System.out.println("파라미터인 no가 없습니다. 잘못된 url경로!");
    return;
}
DecimalFormat df = new DecimalFormat("#,###");

//2. db작업 - select
PdDAO pdDao = new PdDAO();
PdDTO dto=null;
try{
    dto=pdDao.selectByNo(Integer.parseInt(no));
}catch(SQLException e){
    System.out.println("상품 상세보기, sql error:"+e);
}
%>
<!DOCTYPE html><html><head><meta charset="UTF-8"><title>pdDetail.jsp</title>
<script type="text/javascript" src="../jquery/jquery-1.11.2.min.js"></script>
<script type="text/javascript">
```

```

function del(no){
    var flag=confirm("삭제하시겠습니까?");//[확인]을 선택하면 true, [취소]를 선택하면 false를 리턴
    if(flag){ //확인을 선택한 경우 삭제처리 페이지로 이동
        location.href="pdDelete_ok.jsp?no="+no;
    }
}

$(function(){
    $('#aDel').click(function(){
        if(confirm("삭제하시겠습니까?")){
            location.href="pdDelete_ok.jsp?no=<%=no%>";
        }
    });
});
</script></head>
<body>
<h1>상품 상세보기</h1>
번호 : <%=no %> <br>
상품명 : <%=dto.getPdName() %><br>
가격 : <%=df.format(dto.getPrice()) %>원<br>
등록일 : <%= dto.getRegdate() %>
<br><br>
<a href="pdList.jsp">목록</a> |
<a href="pdEdit.jsp?no=<%=no%>">수정</a> |
<a href="#" onclick="del(<%=no%>)">삭제</a>
<a href="#" id="aDel">삭제2</a>
</body></html>

```

pdEdit.jsp

<%

//파라미터인 no를 읽어와서 db에서 no에 해당하는 상품을 조회한 후 화면 출력하기

//1. 파라미터 읽어오기

String no= request.getParameter("no");

if(no==null || no.isEmpty()){

 System.out.println("파라미터인 no가 없습니다. 잘못된 url경로!");

 return;

}

DecimalFormat df = new DecimalFormat("#,###");

PdDAO pdDao = new PdDAO();

PdDTO dto=null;

try{

 dto=pdDao.selectByNo(Integer.parseInt(no));

}catch(SQLException e){

 System.out.println("상품 수정시 조회, sql error:"+e);

}

%>

```
<!DOCTYPE html><html><head><meta charset="UTF-8"><title>pdEdit.jsp</title></head>
<body>
  <h1>상품 수정</h1>
  <form name="frm" method="post" action="pdEdit_ok.jsp">
    <!-- pdEdit_ok.jsp에서 update하려면 no가 필요하므로 hidden 필드에 담아서 보낸다 -->
    <input type="hidden" name="no" value="<%=no%>">
    상품명 : <input type="text" name="pdName" maxlength="50"
              value="<%=dto.getPdName()%>"><br>
    가격 : <input type="text" name="price" value="<%=dto.getPrice()%>">
           <br><br>
    <input type="submit" value="수정">
    <input type="reset" value="취소">
  </form>
  <br>
  <a href="pdList.jsp">상품 목록</a>
</body>
</html>
```

```
<% //파라미터들을 읽어와서 pd테이블에 update한다
//1. 파라미터 읽어오기 - post, 파라미터에 대한 한글 인코딩 처리
request.setCharacterEncoding("utf-8");
String no = request.getParameter("no");
String pdName = request.getParameter("pdName");
String price = request.getParameter("price");
if(no==null || no.isEmpty()){
    System.out.println("잘못된 url경로입니다");
    return;
}
//2. db작업 - update
PdDAO pdDao = new PdDAO();
PdDTO dto = new PdDTO();
dto.setNo(Integer.parseInt(no));
dto.setPdName(pdName);
dto.setPrice(Integer.parseInt(price));
try{
    int n = pdDao.updatePd(dto);
    if(n>0){
        System.out.println("수정 성공");
        response.sendRedirect("pdDetail.jsp?no="+no);
    }else{
        System.out.println("수정 실패");
        response.sendRedirect("pdEdit.jsp?no="+no);
    }
} catch(SQLException e){
    System.out.println("상품 수정, sql error:"+e);
}
%>
```

<%

```
//파라미터인 no를 읽어와서 pd 테이블에서 삭제처리한다
//http://localhost:9090/mystudy/pd/pdDelete_ok.jsp?no=4
//1. 파라미터 읽어오기 - get
String no = request.getParameter("no");
if(no==null || no.isEmpty()){
    System.out.println("잘못된 url 경로입니다");
    return;
}

//2. db작업 - delete
PdDAO pdDao = new PdDAO();
try{
    int n = pdDao.deletePd(Integer.parseInt(no));

    if(n>0){
        System.out.println("상품 삭제 성공");
        response.sendRedirect("pdList.jsp");
    }else{
        System.out.println("상품 삭제 실패");
        response.sendRedirect("pdDetail.jsp?no="+no);
    }
}catch(SQLException e){
    System.out.println("상품 삭제, sql error:"+e);
}
```

%>