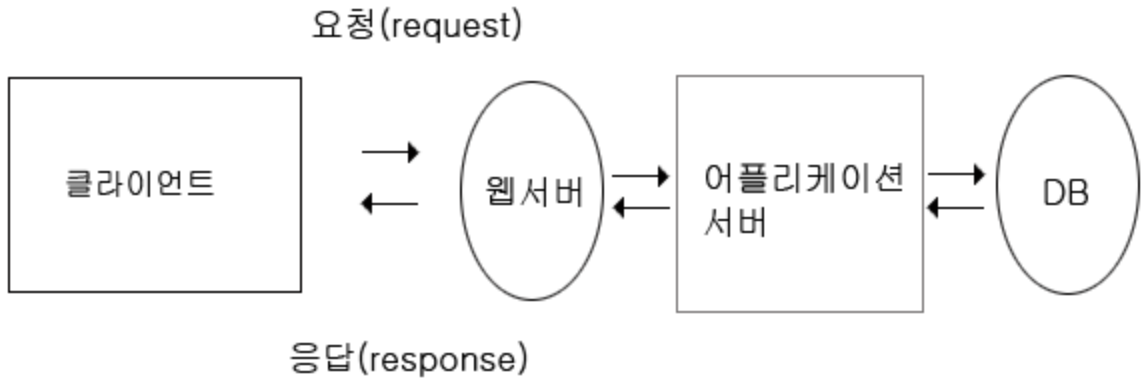
JSP

김용만

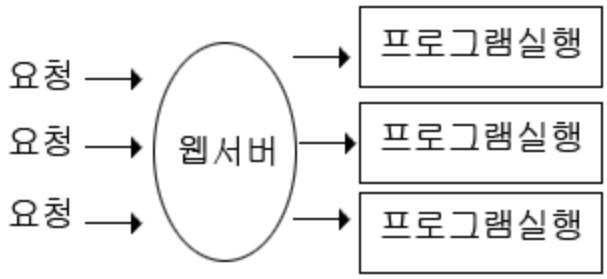
1. 웹어플리케이션 기초

1) 웹어플리케이션의 구성



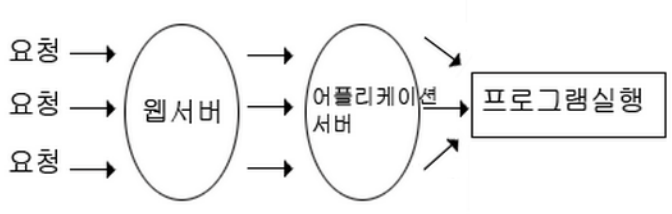
2) 요청 처리 방식

- CGI 방식



여러 요청이 동일한 프로그램을 호출할 때 요청량만큼 프로세서를 생성해서 처리하기 때문에 대량 트래픽 발생시 부하 발생

- Servlet/JSP 사용시 처리 방식



여러 요청이 동일한 프로그램을 호출할 때 요청을 어플리케이션서버내의 컨테이너가 각 요청에 따라 스레드를 생성해 하나의 프로그램을 동작시키는 방식으로 대량의 트래픽 발생해도 CGI 방식에 비해 부하가 덜 발생함

3) Servlet 과 JSP

- Servlet

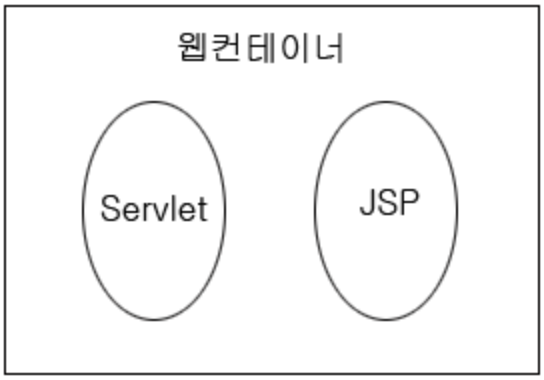
- 자바를 기반으로 하는 웹 개발 표준 언어
- 실행 코드 방식

- JSP

- 자바 기반 스크립트 언어
- 웹 어플리케이션에서 결과 화면을 생성할 때 주로 사용

4) 웹컨테이너

웹어플리케이션을 실행할 수 있는 컨테이너



5) 개발 환경 구축
서버 : 톰캣(http://tomcat.apache.org/)

톰캣의 디렉토리
- bin : 톰캣을 실행하고 종료시키는 스크립트 파일 위치
- conf : 설정 파일이 위치
- lib : 톰캣을 실행하는데 필요한 라이브러리(jar) 파일이 위치
- logs : 톰캣 로그 파일이 위치
- temp : 톰캣이 실행되는 동안 임시 파일이 위치
- webapps : 웹 어플리케이션이 위치

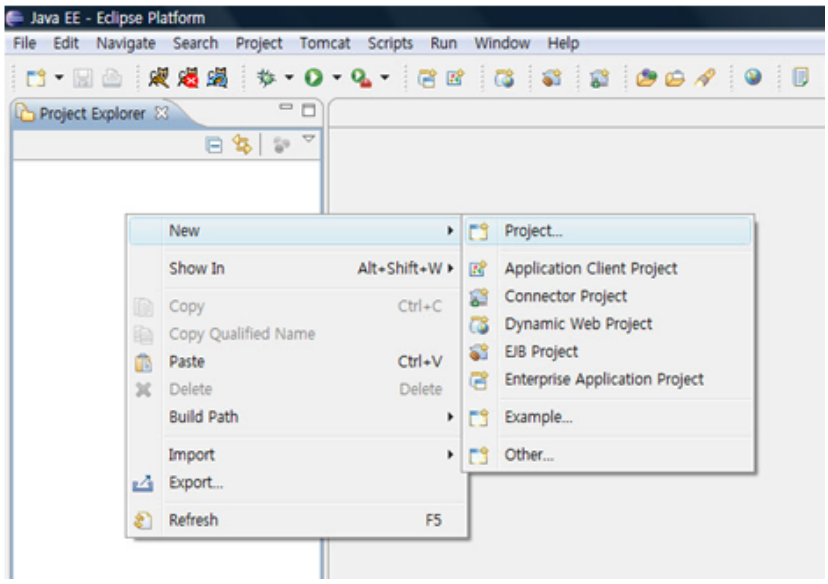
- work : 톰캣이 실행되는 동안 사용되는 작업 파일이 위치

에디터 : 이클립스 (http://www.eclipse.org/)

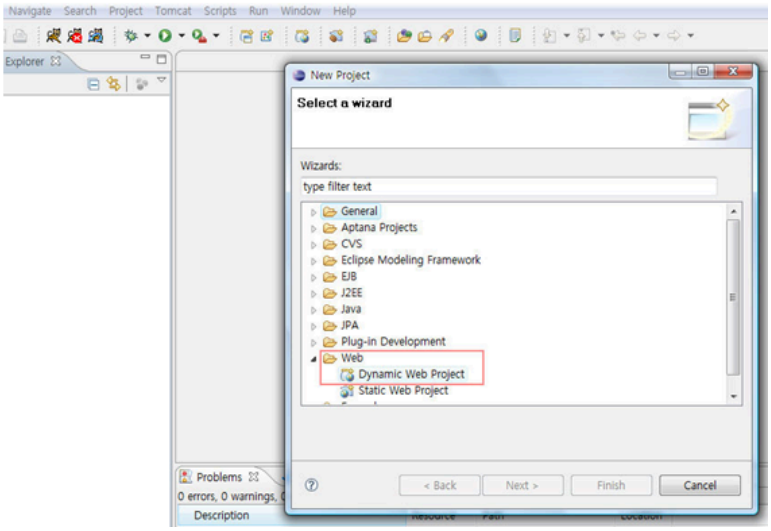
서버와 이클립스 설치 후에 웹어플리케이션 코드를 작성하기 위해 개발 환경을 구축한다. 개발 환경은 다이나믹 웹 프로젝트를 구축해 이클립스에서 서버를 제어하는 방식을 사용한다.

다이나믹 웹 프로젝트 방식으로 프로젝트 생성

이클립스 Project Explorer 창에 마우스 포인터를 위치하고 오른쪽 마우스 버튼을 누른다.
New > Project 선택

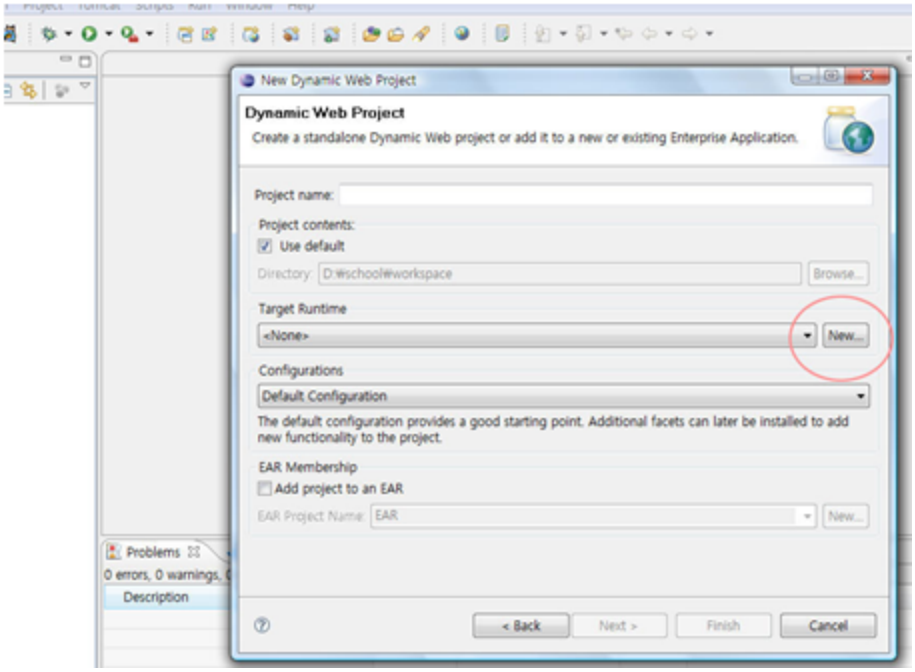


아래 New Project 창에서 web> Dynamic Web Project 선택 하고 next 버튼 클릭

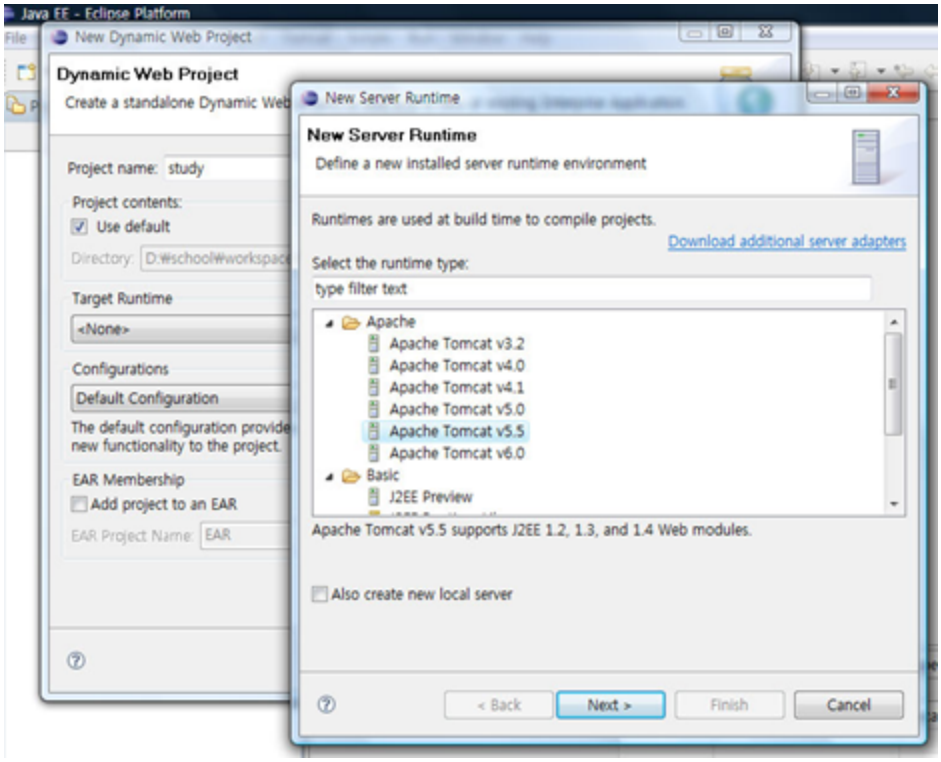


Dynamic Web Project 창이 뜨면 Project name에 Project명을 쓰고 Target Runtime을 설정하기

위해 New 버튼을 클릭한다.

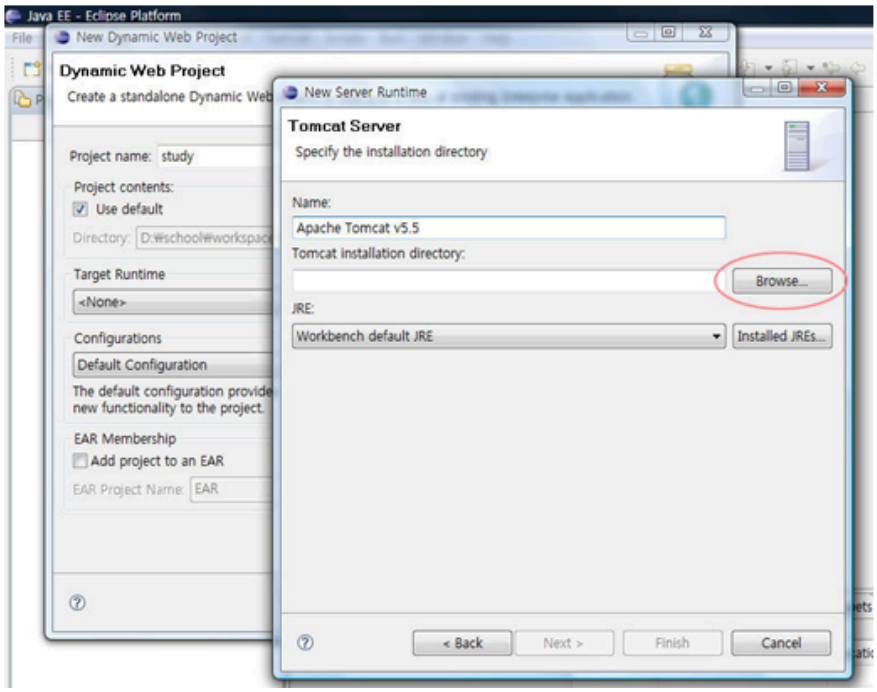


New Server Runtime 창에서 Apache 선택하고 Apache Tomcat 버전을 선택한 후 New 버튼 클릭

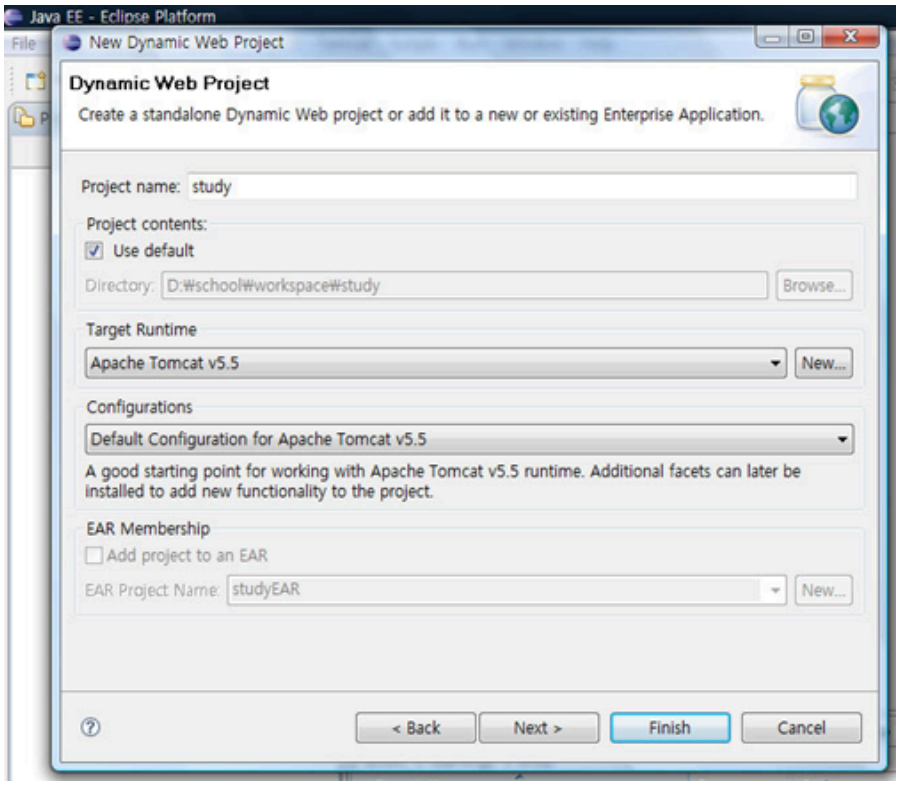


Tomcat Server 창에서 Tomcat의 디렉토리를 등록한다. Browse를 클릭하고 컴퓨터에 설치한

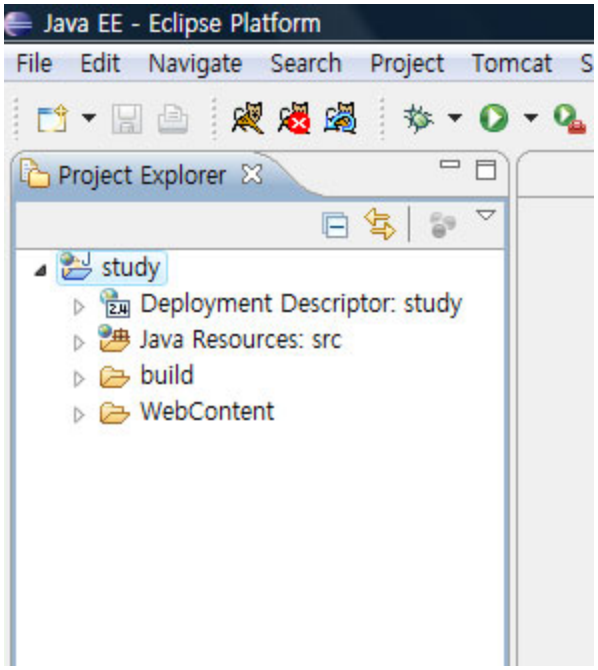
Tomcat 의 경로를 검색한다.



next 클릭한다.

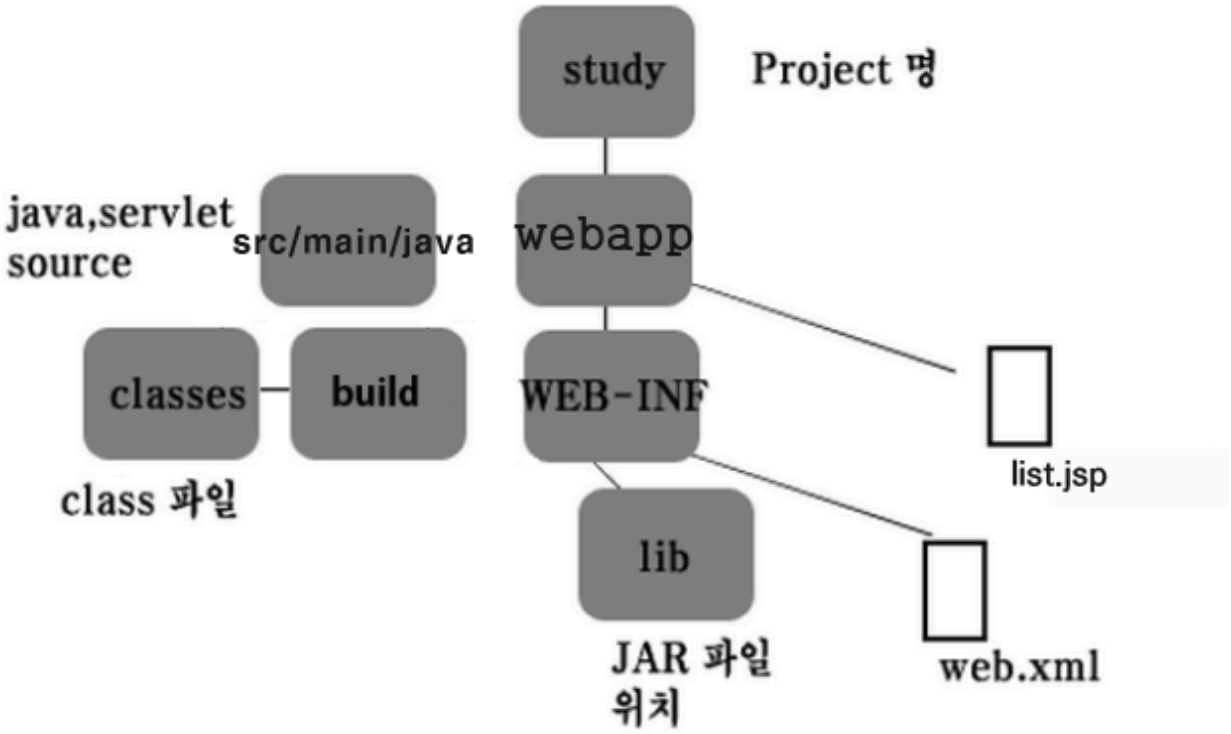


Finish 클릭. 아래와 같이 프로젝트가 만들어졌다. Project명 : study

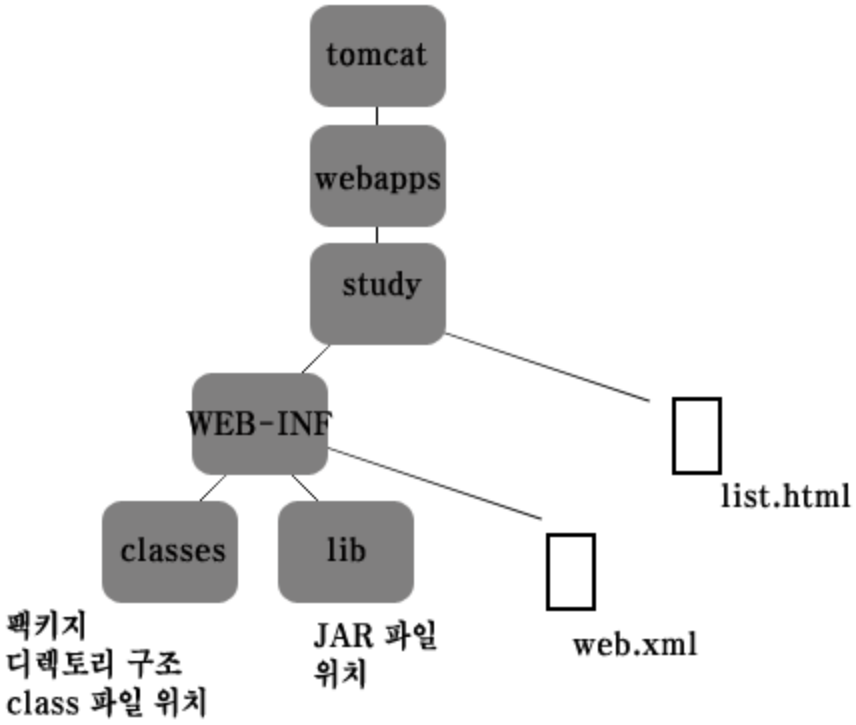


6) 구축된 프로젝트 (다이나믹 웹 프로젝트)

이클립스의 다이나믹 웹 프로젝트를 이용해 개발 환경 구축



6) 배포 환경(톰캣 배포시 환경)



2. Servlet

1) 서블릿의 개요

서블릿(Servlet) : 자바 기반의 웹 프로그래밍 기술로 자바 언어의 모든 기능을 사용 할 수 있으며 쓰레드 기반의 빠른 처리 속도를 자랑하나, 프로그램 내에서 화면 코딩을 제어해야 하는 문제로 인해 유지 보수에 많은 문제가 있다. 이러한 문제점을 개선한것이 JSP로 JSP는 내부적으로 서블릿 기술에 기반하고 있다.

2) HTTP 프로토콜과 HTTP 메서드

- HTTP 프로토콜

HTTP은 비 연결(Connectionless : 클라이언트의 요청에 응답한 후 바로 연결을 끊음)과 비 상태(Stateless : 서버의 상태가 어떤지 간에 상관없이 요청을 함)의 특징

- HTTP 메소드(method)

- Get : 쿼리문자열로 전송

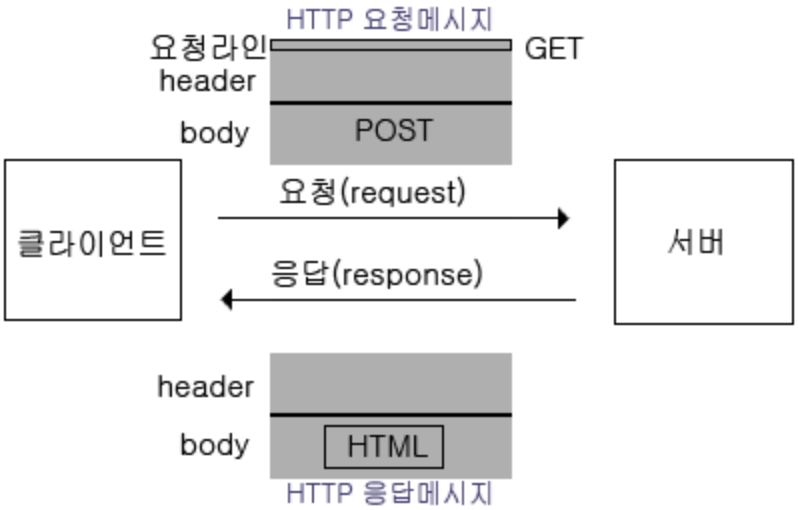
요청라인의 URL뒤에 데이터를 연결하여 전송
256byte가 전송 한계

ex)
http://localhost:8080/servletMain/hello
http://localhost:8080/servletMain/hello?id=dragon

HTML 페이지
<form action="/servletMain/lauch" **method="get"**>
...
</form>

- Post : 요청 몸체 데이터로 전송
  전송하는 데이터 사이즈의 제한이 없음

HTML 페이지
<form action="/servletMain/lauch" **method="post"**>
...
</form>

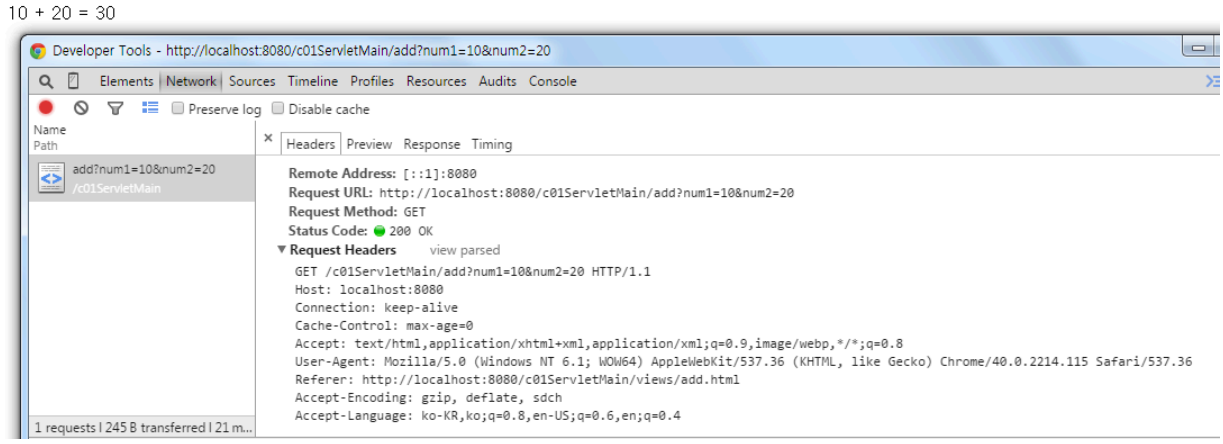


GET 방식 전송 과 POST 방식 전송

- GET 방식 전송

get 방식으로 이용한 파라미터 전송시, 요청 데이터

http://localhost:8080/c01ServletMain/add?num1=10&num2=20



<크롬 요소 검사를 이용한 header 정보 보기>

GET /c01ServletMain/add?num1=10&num2=20 HTTP/1.1
Host: localhost:8080
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) Chrome/40.0.2214.115 Safari/537.36
Referer: http://localhost:8080/c01ServletMain/views/add.html
Accept-Encoding: gzip, deflate, sdch
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4

파라미터 값에 한글이 포함되어 있을 때 아래와 같이 인코딩되어 서버에 전송

name=%ED%99%8D%EA%B8%B8%EB%8F%99

[톰캣에서 GET방식 파라미터 값을 위한 인코딩 처리]

톰캣설치디렉터리/conf/server.xml 에서 아래 코드 수정

<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1"
redirectPort="8443"
URIEncoding=”euc-kr”/>

URIEncoding=”euc-kr”로 지정하면 GET 방식으로 전송된 파라미터를 읽을 때 항상 URIEncoding 속성에 지정한 캐릭터 셋을 지정

<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1"

redirectPort="8443"
useBodyEncodingForURI=”true”/>

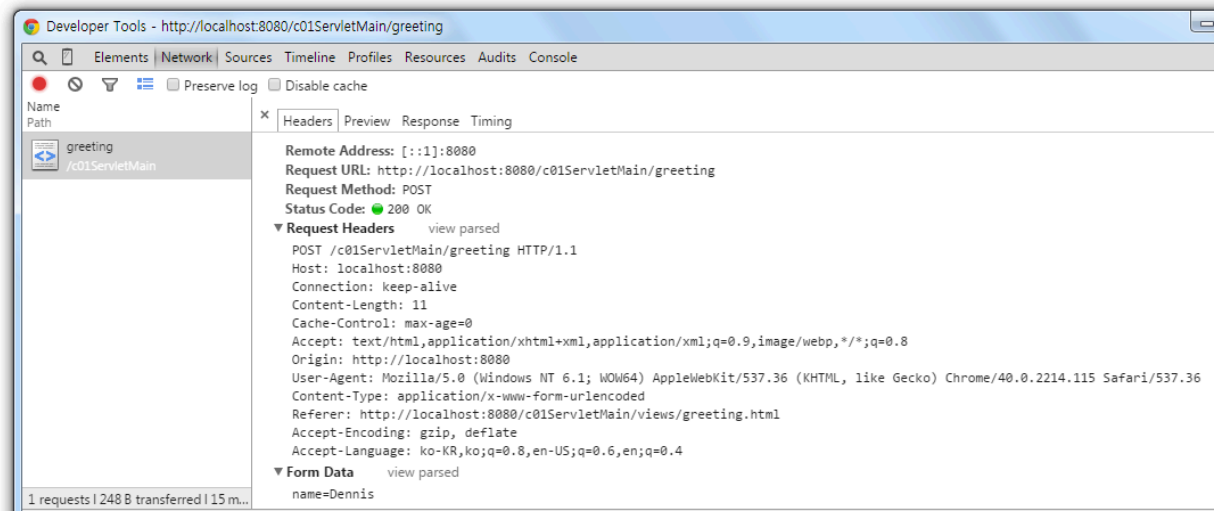
useBodyEncodingForURI=”true”로 지정하면 GET 방식으로 전달된 파라미터값을 읽어올 때 **request.setCharacterEncoding()** 메서드로 지정한 캐릭터 셋이 적용됨

- POST 방식 전송

post 방식으로 이용한 파라미터 전송시, 요청 데이터

http://localhost:8080/c01ServletMain/greeting

Dennis 님의 방문을 환영합니다.



<크롬 요소 검사를 이용한 header 정보 보기>

POST /c01ServletMain/greeting HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 11
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.115 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost:8080/c01ServletMain/views/greeting.html
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4

name=Dennis

파라미터 값에 한글이 포함되어 있을 경우

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{
                try {
                        response.setContentType("text/html;charset=utf-8");

                        request.setCharacterEncoding("utf-8");

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

3) Servlet의 패키지 및 주요 객체

패키지

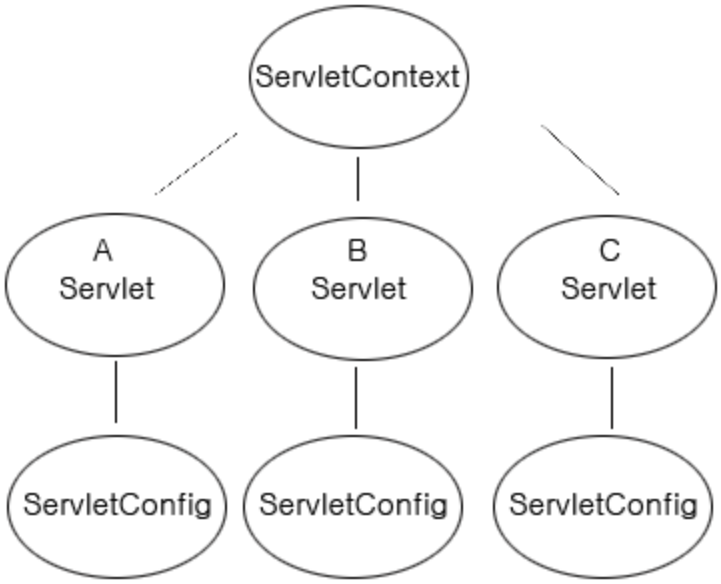
- javax.servlet : 패키지는 프로토콜에 독립적인 서블릿을 만들기 위한 클래스
- javax.servlet.http : 패키지는 HTTP 프로토콜의 고유한 기능(GET, POST 등)을 제공

주요객체

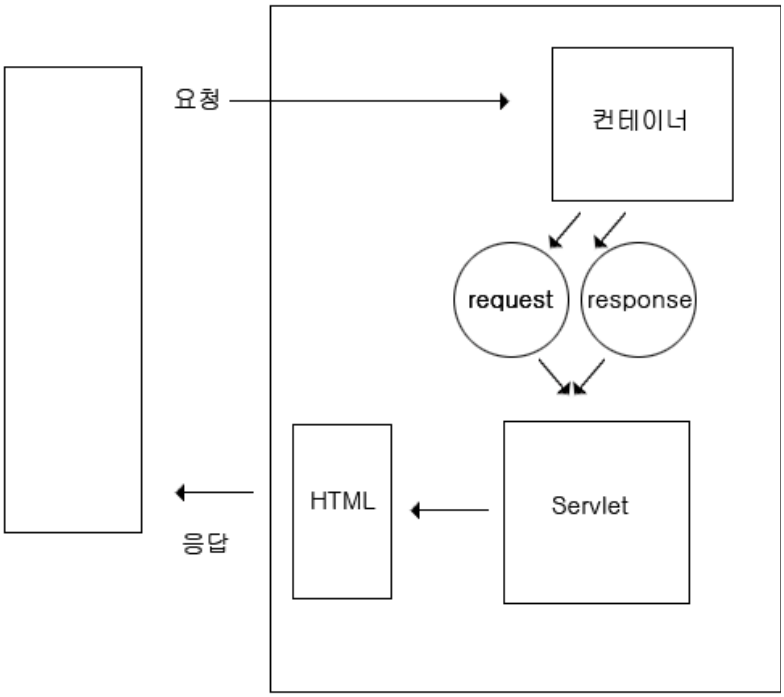
- HttpServletRequest : HTTP 요청에 대한 기능 제공
- HttpServletResponse: HTTP 응답에 대한 기능을 제공

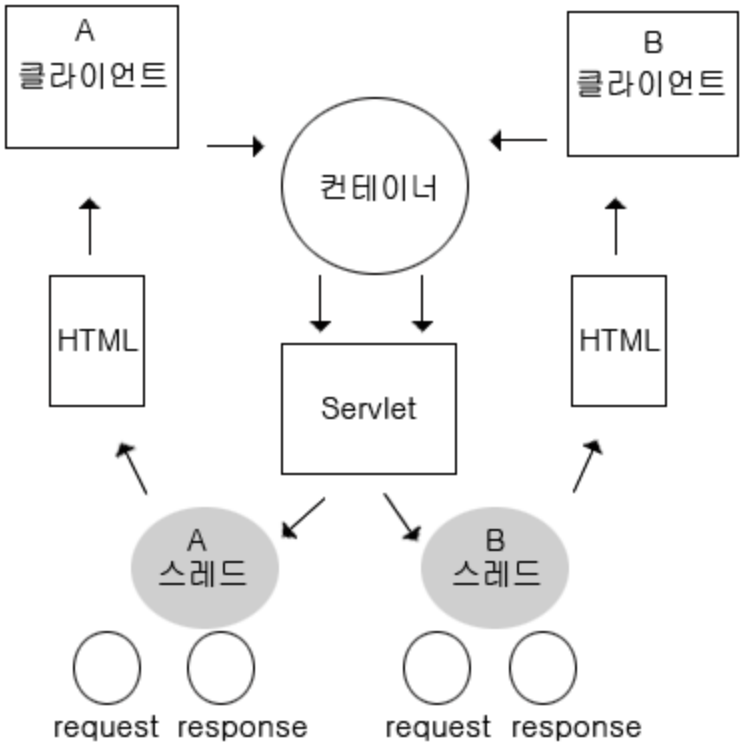
- ServletConfig 객체 : 서블릿 당 ServletConfig 객체 하나
  서블릿 배포시 설정된 정보를 서블릿으로 넘겨줌

- ServletContext 객체 : 웹 어플리케이션 당 하나의 ServletContext 객체 하나
  웹 어플리케이션의 파라미터 정보를 읽어보기 위하여 사용
  서버 정보를 파악하기 위하여 사용(컨테이너의 이름 및 버전, 지원하는 API 정보)

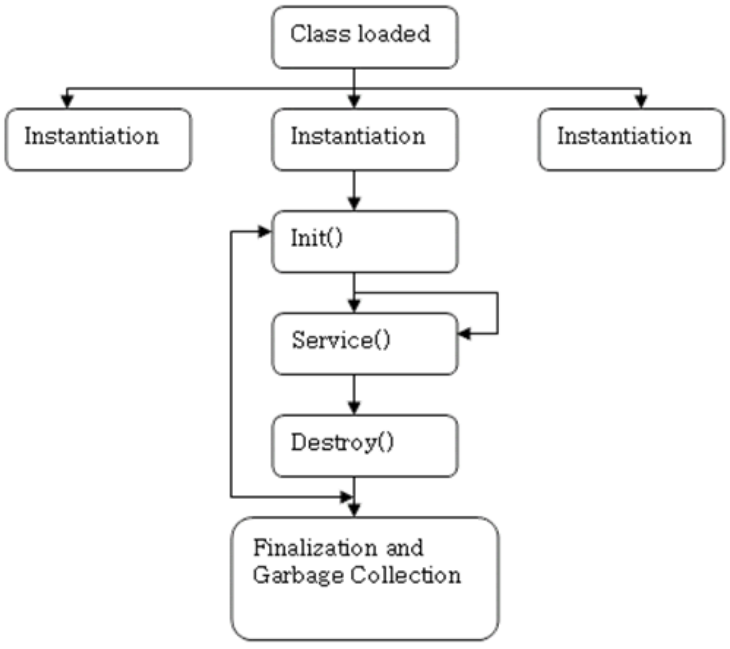

ServletContext
A
Servlet
B
Servlet
C
Servlet
ServletConfig
ServletConfig
ServletConfig

4) 서블릿 동작 원리


요청
컨테이너
request
response
HTML
Servlet
응답



5) 라이프 사이클



■ init()

- 컨테이너 에서 서블릿 객체를 생성한 다음에 호출한다. service() 이전에 실행
- 서블릿을 초기화
- 초기화할 내용(DB 접속 등)이 있는 경우 재정의

■ service()
- 클라이언트의 요청 후 컨테이너에서 쓰레드를 이용하여 호출
- 요청의 HTTP 메소드(GET, POST등)를 참조하여 해당 메소드(doGet(), doPost() 등) 호출 판단
- 거의 재정의 하지 않음

■ destroy()
- 서블릿 종료시 호출

6) 서블릿 매핑

■ 배포 서술자 (DD, Deployment Descriptor)
- 서블릿 컨테이너에 서블릿 배포 시 사용하는 XML 문서
- URL과 서블릿 매핑 정보 포함
- 보안역할 설정, 오류 페이지 설정, 초기화 구성 및 관련 정보 설정 등

■ URL 매핑을 위한 항목
- <servlet> 서블릿 내부명과 완전한 클래스명과의 매핑정보
- <servlet-mapping> 서블릿 내부명과 URL 명과의 매핑정보

3. JSP

1) 디렉티브

디렉티브 구문 : –<%@ 디렉티브이름 속성1="값1" 속성2="값2" ... %>

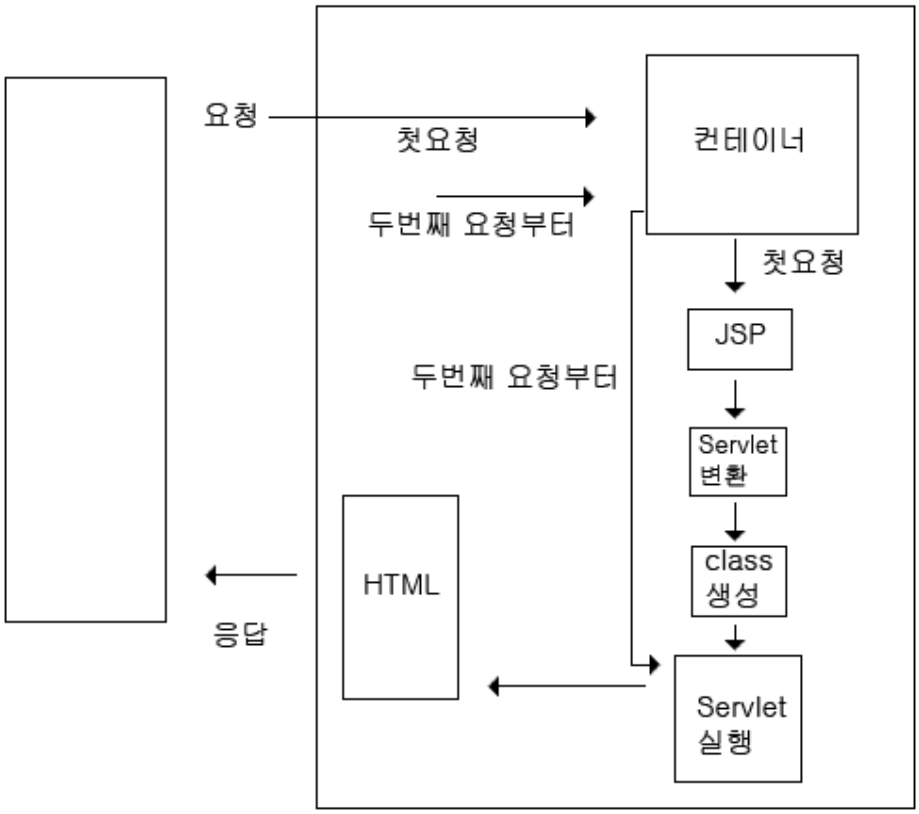
- page : JSP 페이지에 대한 정보를 지정, 문서의 타입, 출력 버퍼의 크기, 에러 페이지 등 정보 지정
- taglib : 사용할 태그 라이브러리를 지정
- include : 다른 문서를 포함

- page 디렉티브 주요 속성

language	JSP 스크립트 코드에서 사용되는 프로그래밍 언어
contentType	JSP가 생성할 문서의 타입을 지정
pageEncoding	JSP 페이지 자체의 캐릭터 인코딩 지정

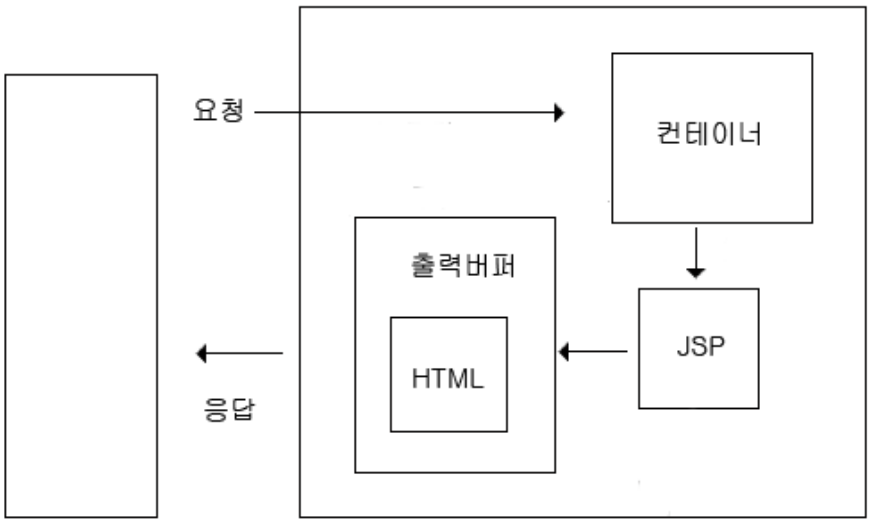
import	JSP 페이지에서 사용할 자바 클래스를 지정
session	JSP 페이지가 세션을 사용할 지의 여부를 지정
info	JSP 페이지에 대한 설명을 입력한다.
errorPage	에러가 발생할 때 보여 줄 페이지를 지정
isErrorPage	에러 페이지인지의 여부를 지정
buffer	버퍼 사용 여부 및 크기 지정(기본값 : 8kb)
autoFlush	버퍼가 다 찼을 때 처리 방식 지정 true - 버퍼가 다 찼을 경우 버퍼를 플러시하고 계속해서 작업 false - 버퍼가 다 찼을 경우 예외를 발생시키고 작업 중지
isELIgnored	표현 언어(EL)지원 여부 지정. true - 미지원, false - 지원
trimDirectiveWhitespaces	출력결과에서 템플릿 텍스트의 공백 문자를 제거할지 여부 지정

- JSP 구동 원리



- 출력 버퍼와 응답

출력 버퍼 - JSP가 생성한 응답 결과를 임시로 저장



출력 버퍼의 장점

- 데이터 전송 성능 향상
- 버퍼가 다 차기 전까지 헤더 변경 가능
- JSP 실행 도중 버퍼를 비우고 새 내용 전송 가능

- include 디렉티브

코드 차원에서 포함
구문 : <%@ include file="포함할파일" %>
활용

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

2) 스크립트

동적으로 출력 결과를 생성하기 위해 사용

스크립트 요소

- 선언부(Declaration) - 변수 선언, 메서드 선언 <%! %>
- 스크립트릿(Scriptlet) - 자바 코드를 실행- 변수 선언, 연산, 제어문 사용, 출력 등 <% %>
- 표현식(Expression) - 연산, 출력(변수의 값, 메서드의 결과값)<%= %>

3) 기본객체

JSP가 제공하는 기본 객체

기본객체	실제 타입	설명
request	javax.servlet.http.HttpServletRequest	클라이언트의 요청 정보를 저장한다.
response	javax.servlet.http.HttpServletResponse	응답 정보를 저장한다.
pageContext	javax.servlet.jsp.PageContext	JSP페이지에 대한 정보를 저장한다.
session	javax.servlet.http.HttpSession	HTTP 세션 정보를 저장한다.
application	javax.servlet.ServletContext	웹 어플리케이션에 대한 정보를 저장한다.
out	javax.servlet.jsp.JspWriter	JSP 페이지가 생성하는 결과를 출력할 때 사용되는 출력 스트림이다.

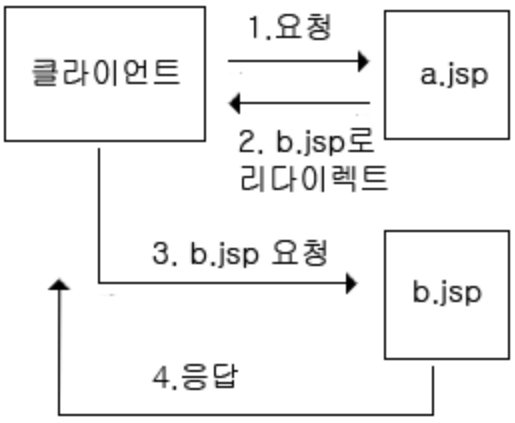
config	javax.servlet.ServletConfig	JSP 페이지에 대한 설정 정보를 저장한다.
page	java.lang.Object	JSP 페이지를 구현한 자바 클래스 인스턴스이다.
exception	java.lang.Throwable	예외 객체. 에러 페이지에서만 사용된다.

- request : 웹 브라우저가 웹 서버에 전송한 요청 관련 정보 제공

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

- response : 웹 브라우저에 전송하는 응답 정보 설정

주요 기능
- 헤더 정보 입력
- 리다이렉트 처리



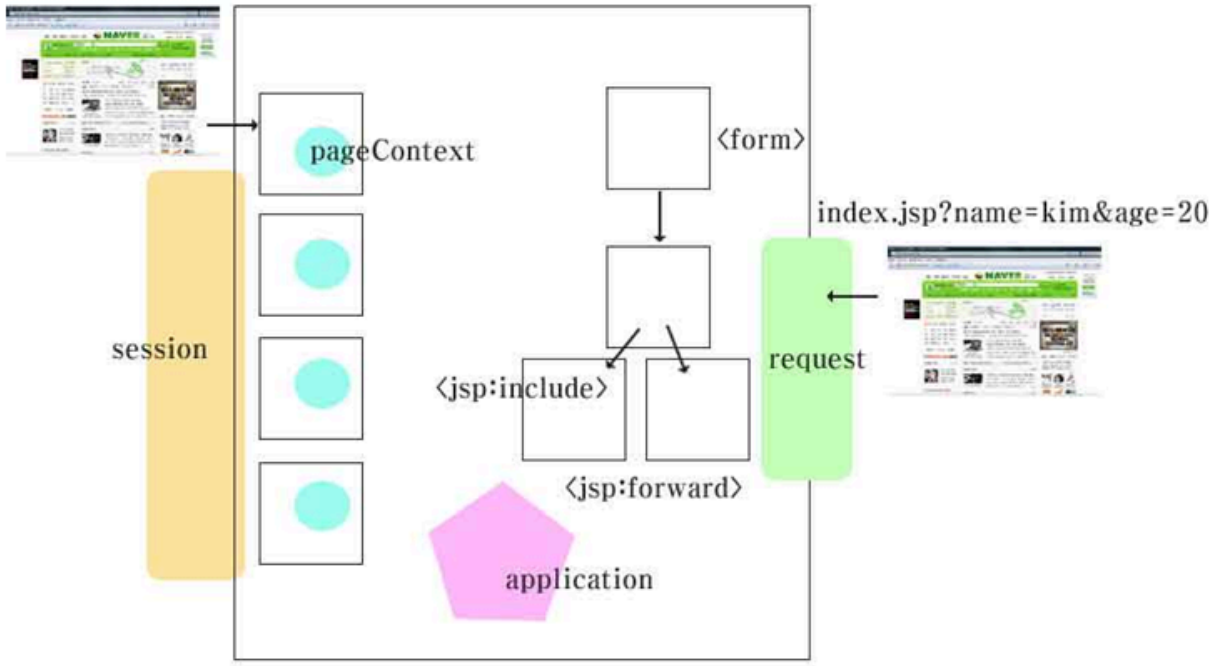
- out : JSP 페이지가 생성하는 모든 내용은 out 기본 객체를 통해서 전송

- pageContext : 다른 기본 객체에 대한 접근 메서드 제공

- application : 웹 어플리케이션에 대한 정보를 저장

4) 기본 객체와 영역

기본 객체	영역
pageContext	page : 하나의 JSP 페이지를 처리할 때 사용되는 영역
request	request : 하나의 HTTP 요청을 처리할 때 사용되는 영역
session	session : 하나의 웹 브라우저와 관련된 영역
application	application : 하나의 웹 어플리케이션과 관련된 영역



5) 액션 태그

<jsp:include> 액션 태그

다른 JSP 페이지의 '실행' 결과를 현재 위치에 삽입

<jsp:include> 액션 태그의 기본 사용 방법

```
<jsp:include page="포함할 페이지" flush="true"/>
```

page : 포함할 JSP 페이지
flush : 지정한 JSP 페이지를 실행하기 전에 출력버퍼를 플러시 할지의 여부를 지정.
true이면 출력버퍼를 플러시하고, false이면 하지 않음

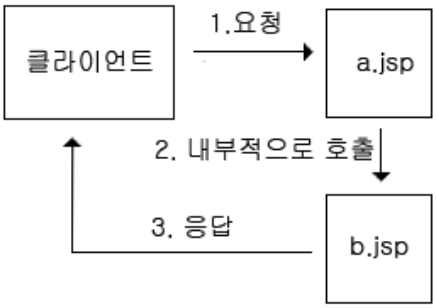
<jsp:include/> 액션 태그와 **include** 디렉티브의 비교

비교항목	**<jsp:include/>**	**include** 디렉티브
처리시간	요청 시간에 처리	JSP 파일을 자바 소스로 변환할 때 처리
기능	별도의 파일로 요청 처리 흐름을 이동	현재 파일에 삽입시킴
데이터 전달 방법	request 기본 객체나 <jsp:param>을 이용한 파라미터 전달	페이지 내의 변수를 선언한 후, 변수에 값 지정
용도	화면의 레이아웃의 일부분을 모듈화할 때 주로 사용된다.	다수의 JSP 페이지에서 공통으로 사용되는 변수를 지정하는 코드나 저작권과 같은 문장을 포함

<jsp:forward> 액션 태그

하나의 JSP 페이지에서 다른 JSP 페이지로 요청 처리를 전달할 때 사용

동작 방식



<jsp:forward> 액션 태그의 사용법

```
<jsp:forward page="이동할 페이지"/>
```

forward 와 **redirect** 의 특징 비교

메서드	특징
forward <jsp:forward />	같은 웹서버, 같은 웹 애플리케이션 디렉토리에 속하는 웹 자원만 호출 가능 request 내장 객체를 통해 데이터를 전달
redirect response.sendRedirect()	다른 웹 서버에 있는 웹 자원도 호출할 수 있음 호출할 JSP페이지의 URL 뒤에 데이터를 붙여서 전달

6) 에러 처리

- page 디렉티브를 이용한 에러 처리

에러 페이지 지정
–<%@ page **errorPage =** "예외발생시보여질JSP지정" %>

에러 페이지 작성
–<%@ page **isErrorPage = "true"** %>
isErrorPage 속성이 true인 경우 에러 페이지로 지정

- 응답 상태 코드 별 에러 페이지 지정

web.xml 파일에서 설정

```
<error-page>
        <error-code>에러코드</error-code>
        <location>에러페이지의 URI</location>
</error-page>
```

주요 HTTP Status Code 표

HTTP Status Code	메시지
200	OK, 에러 없이 전송이 성공
403	Forbidden(금지) 서버가 허용하지 않는 웹 페이지나 미디어를 사용자가 요청할 때
404	Not Found, 문서를 찾을 수 없음. 이 에러는 클라이언트가 요청한 문서를 찾지 못한 경우에 발생. URL을 다시 잘 보고 주소가 올바로 입력되었는지를 확인.
500	Internal Server Error(서버 내부 오류). 이 에러는 웹 서버가 요청사항을 수행할 수 없을 경우에 발생.

- 예외 타입 별 에러 페이지 지정

web.xml 파일에서 설정

```
<error-page>
        <exception-type>예외클래스명</exception-type>
        <location>에러페이지의 URI</location>
</error-page>
```

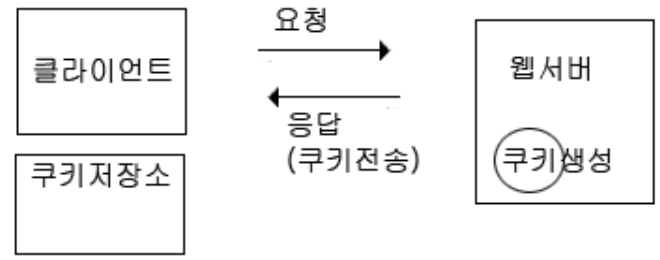
- 에러 페이지 우선 순위

1.page 디렉티브의 errorPage 속성에서 지정한 에러 페이지를 보여준다.
2.JSP 페이지에서 발생한 예외 타입이 web.xml 파일의 <exception-type>에서 지정한 예외 타입과 동일한 경우 지정한 에러 페이지를 보여준다.
3.JSP 페이지에서 발생한 에러 코드가 web.xml 파일의 <error-code>에서 지정한 에러 코드와 동일한 경우 지정한 에러 페이지를 보여준다.
4.아무것도 해당되지 않을 경우 웹 컨테이너가 제공하는 기본 에러 페이지를 보여준다.

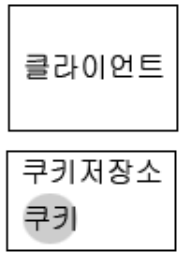
7) 쿠키

’쿠키(cookie)’는 웹 브라우저가 보관하고 있는 데이터로서 웹 서버에 요청을 보낼 때 함께 전송

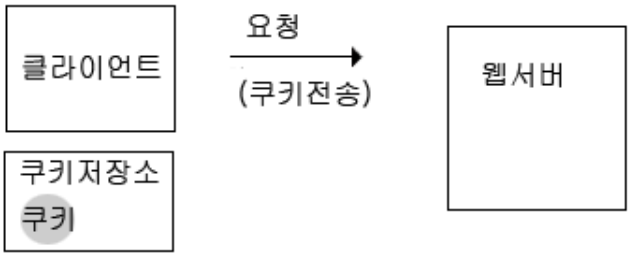
1 쿠키생성



2 쿠키저장



3 쿠키전송



구성 요소

- 이름 - 각각의 쿠키를 구별하는 데 사용되는 이름
- 값 - 쿠키의 이름과 관련된 값
- 유효시간 - 쿠키의 유지 시간
- 도메인 - 쿠키를 전송할 도메인
- 경로 - 쿠키를 전송할 요청 경로

쿠키 생성하기

```
쿠키 생성
Cookie cookie = new Cookie(cookieName,cookieValue);
생성된 쿠키를 클라이언로 전송
response.addCookie(cookie);
```

Cookie 클래스가 제공하는 메서드

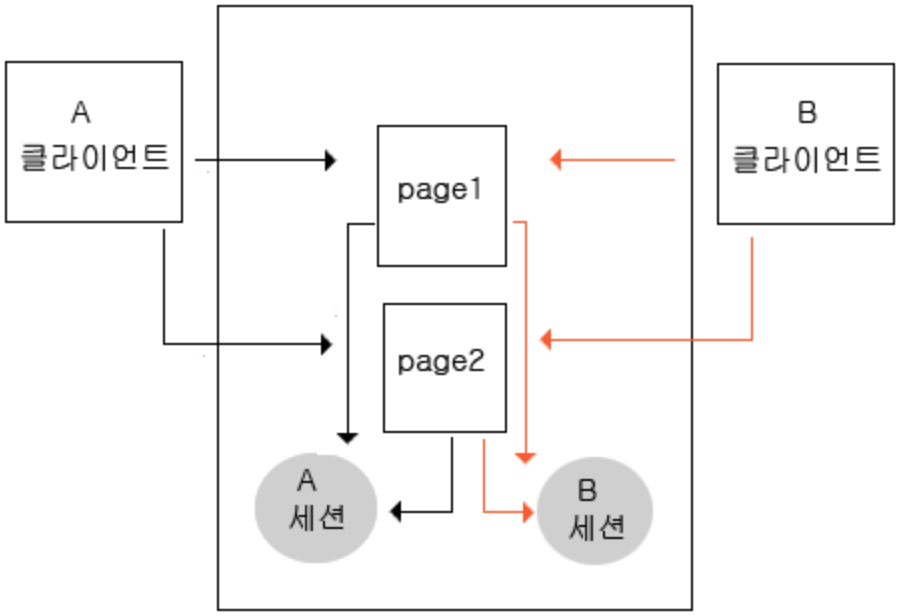
메서드	리턴타입	설명
getName()	String	쿠키의 이름을 구함
getValue()	String	쿠키의 값을 구함
setValue(String value)	void	쿠키의 값을 지정
setDomain(String pattern)	void	이 쿠키가 전송될 서버의 도메인을 지정
getDomain()	String	쿠키의 도메인을 구함
setPath(String uri)	void	쿠키를 전송할 경로를 지정
getPath()	String	쿠키의 전송 경로를 구함
setMaxAge(int expiry)	void	쿠키의 유효 시간을 초 단위로 지정. 음수를 입력할 경우 웹 브라우저를 닫을 때 쿠키가 함께 삭제
getMageAge()	int	쿠키의 유효 시간을 구함

쿠키값 읽어오기

```
Cookie[] cookies = request.getCookies();
```

8) 세션

웹 컨테이너에서 클라이언트의 정보를 보관할 때 사용
오직 서버에서만 생성
클라이언트마다 세션이 생성



세션의 주요 메서드

메서드	리턴 타입	설명
getId()	String	세션의 고유 ID를 구함
getCreationTime()	long	세션이 생성된 시간을 구함. 시간은 1970년1월1일 이후 흘러간 시간을 의미. 단위는 1/1000초
getLastAccessedTime()	long	웹 브라우저가 가장 마지막에 섹션에 접근한 시간을 구함. 시간은 1970년1월1일 이후 흘러간 시간을 의미. 단위는 1/1000초
setMaxInactiveInterval(int second)	void	세션 유효시간을 초단위로 설정
getMaxinactiveInterval()	int	설정된 세션 유효시간을 반환

4. 자바빈 **& <jsp:useBean>** 액션 태그

- 자바빈(JavaBeans)

웹 프로그래밍에서 데이터의 표현을 목적으로 사용

```
public class Member {
/* 값을 저장하는 필드 */
```

```
private String name;

/* 필드의 값을 읽어오는 값 */
public String getName() {
   return name;
}
/* 필드의 값을 변경하는 값 */
public void setName(String name) {
   this.name = name;
}
}
```

<jsp:useBean> 태그

```
<jsp:useBean id="빈이름" class="자바클래스이름" scope="범위"/>
```

JSP에서 자바빈 객체를 생성할 때 사용

- id : jsp 페이지에서 자바빈 객체에 접근할 때 사용할 이름을 명시함
- class : 패키지 이름을 포함한 자바빈 클래스의 완전한 이름을 입력함
- scope : 자바빈 객체가 저장될 영역을 지정.(page,request,session,application 중 하나 , 기본값은 page)

<jsp:setProperty액션 태그>

```
<jsp:setProperty name="자바빈" property="이름" value="값" />
```

자바빈 객체의 프로퍼티 값 설정

- name : 프로퍼티의 값을 변경할 자바빈 객체의 이름. <jsp:useBean> 액션 태그의 id 속성에서 지정한 값을 사용
- property : 값을 지정할 프로퍼티의 이름
- value : 프로퍼티의 값. 표현식을 사용할 수 있음

<jsp:getPropety> 액션 태그

```
<jsp:getProperty name="자바빈" property="이름" >
```

프로퍼티의 값을 출력하기 위해 사용

- name : <jsp:useBean>의 id 속성에서 지정한 자바빈 객체의 이름
- property : 출력할 프로퍼티의 이름

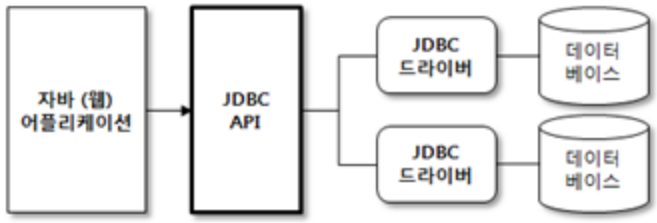
4. 파일 업로드

cos 라이브러리 사용
http://www.servlets.com

6. JSP 와 DB연동

1) JDBC

Java Database Connectivity
자바에서 DB 프로그래밍을 하기 위해 사용되는 API
JDBC API 사용 어플리케이션의 기본 구성

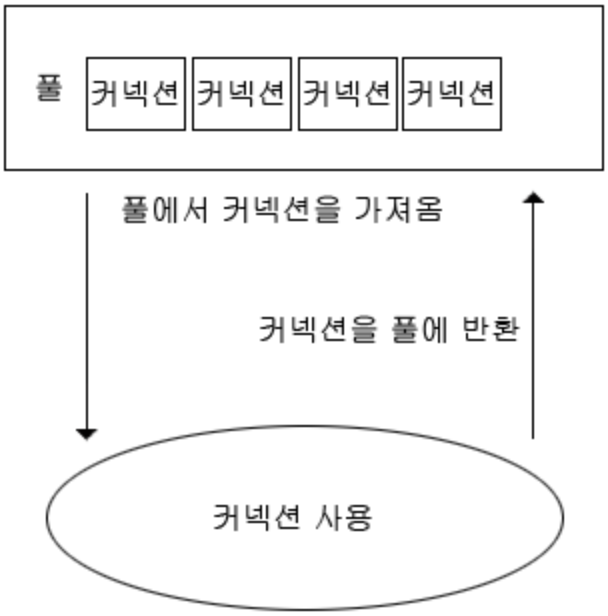


2) 커넥션 풀(Connection Pool) 이란?

데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool) 속에 저장해 두고 있다가 필요할 때에 커넥션을 풀(pool)에서 가져다 쓰고 다시 풀(pool)에 반환하는 기법을 의미

3) 커넥션 풀의 장점

1) 풀 속에 미리 커넥션이 생성되어 있기 때문에 커넥션을 생성하는데 시간이 소비되지 않음
2) 커넥션을 재사용하기 때문에 생성되는 커넥션 수가 많지 않음
3) 커넥션을 생성하고 제거하는데 필요한 시간이 소요되지 않기 때문에 어플리케이션의 실행 속도가 빨라지며 한번에 생성될 수 있는 커넥션 수를 제어하기 때문에 동시 접속자 수가 많아도 웹 어플리케이션이 쉽게 다운되지 않음



4) 자카르타 DBCP API와 JNDI를 이용한 커넥션 풀 사용하기

(1) 데이터베이스 연동 프로그램의 문제점

데이터베이스에 연결하기 위해서 매번 커넥션(Connection) 객체를 생성할 때는 많은 시스템 자원이 요구됨.
(객체 메모리 적재시 메모리에 객체를 할당할 자리 생성, 초기화 작업, 객체 미사용시 객체를 삭제하는 작업 등등)

(2) 자카르타 DBCP API 와 DataSource

- 컨넥션 풀을 사용하기 위해 아파치 그룹에서 제공하는 자카르타 DBCP 이용. 자카르타의 DBCP는 커넥션 풀링 기능을 제공하고 사용되지 않는 커넥션을 해제시켜주는 기능도 포함.

- DataSource : javax.sql.DataSource
데이터베이스에 접근하기 위한 표준 인터페이스, 특히 DBCP를 이용하기 위한 표준방식

(3) JNDI (Java Naming and Directory Interface) : 사용자가 원하는 리소스/서비스를찾기 위한 방법 등록하고

DBCP API를 이용한 커넥션 풀을 구성할 때 데이터 베이스 커넥션에 대한 정보를 자바코드에 직접 하드 코딩하는 것 보다 외부파일에 실제 정보를 기록해 두고 자바코드에는 그 정보를 읽어올 수 있는 name(이름값)만을 기록해두면 데이터 베이스에 대한 정보가 변경된다 하더라도 모든 자바코드를 수정하는 것이 아니라 해당 정보를 기록해 놓은 외부파일만 수정하면 되기 때문에 간편하게 사용할 수 있다.

(4) Eclipse Dynamic Web Project 에서 DBCP API를 이용한 커넥션 풀 설정

- 자카르타(Jakarta) DBCP API 관련 jar 파일 설치 (jakarta.apache.org)

commons-collections-3.1.jar
commons-dbcp-1.2.1.jar
commons-pool-1.2.jar

설치 위치 :작업하고 있는 프로젝트>WEB-INF>lib 에 상기 jar파일을 설치함

- 데이터베이스 설정 및 컨넥션 풀 설정 정보 기록

설치 위치 : 컨텍스트>META-INF 에 context.xml 생성

```
<?xml version="1.0" encoding="UTF-8"?>
<Context >

    <Resource name="jdbc/java"
     auth="container"
     type="javax.sql.DataSource"
     username="scott"
     password="tiger"
     driverClassName="oracle.jdbc.driver.OracleDriver"
     factory="org.apache.commons.dbcp.BasicDataSourceFactory"
     url="jdbc:oracle:thin:@localhost:1521:java"
     maxActive="100"
        maxIdle="10" />
  </Context>
```

- web.xml 파일에 추가 될 내용

컨텍스트>WEB-INF/web.xml 파일을 열어 web-app의 하위 요소로 아래와 같이 추가 기재

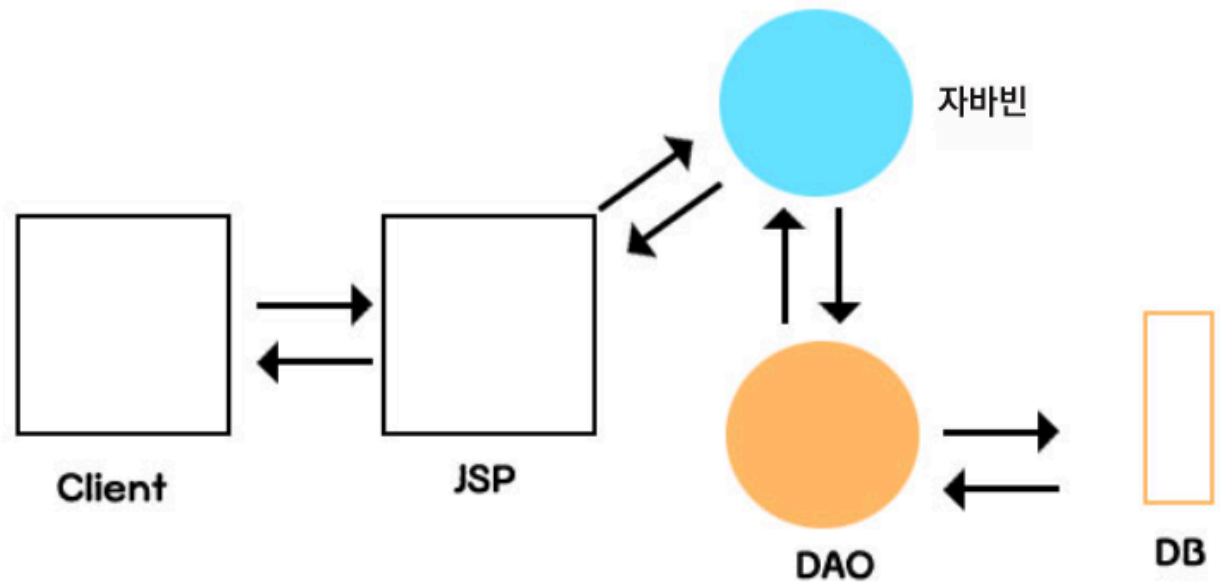
```
  <web-app>

     <resource-ref>
     <description>DB Connection</description>
     <res-ref-name>jdbc/java</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
    </resource-ref>
```

</web-app>

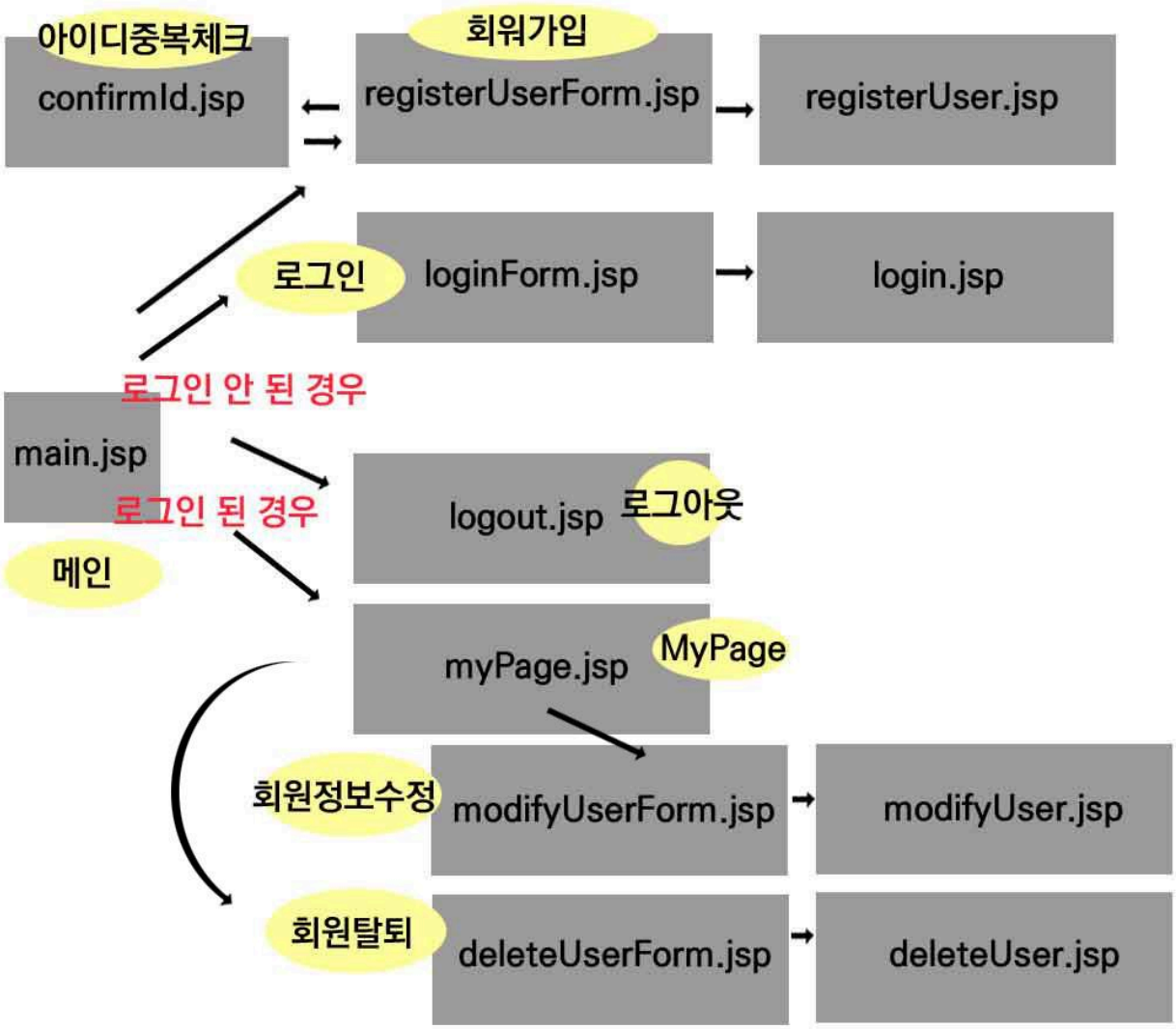
(5) 코딩 예

```
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:comp/env");
DataSource ds = (DataSource)envCtx.lookup("jdbc/java");
Connection conn = ds.getConnection();
또는
Context initCtx = new InitialContext();
DataSource ds = (DatSource) initCtx.lookup("java:comp/env/ jdbc/java ");
Connection conn = ds.getConnection();
```

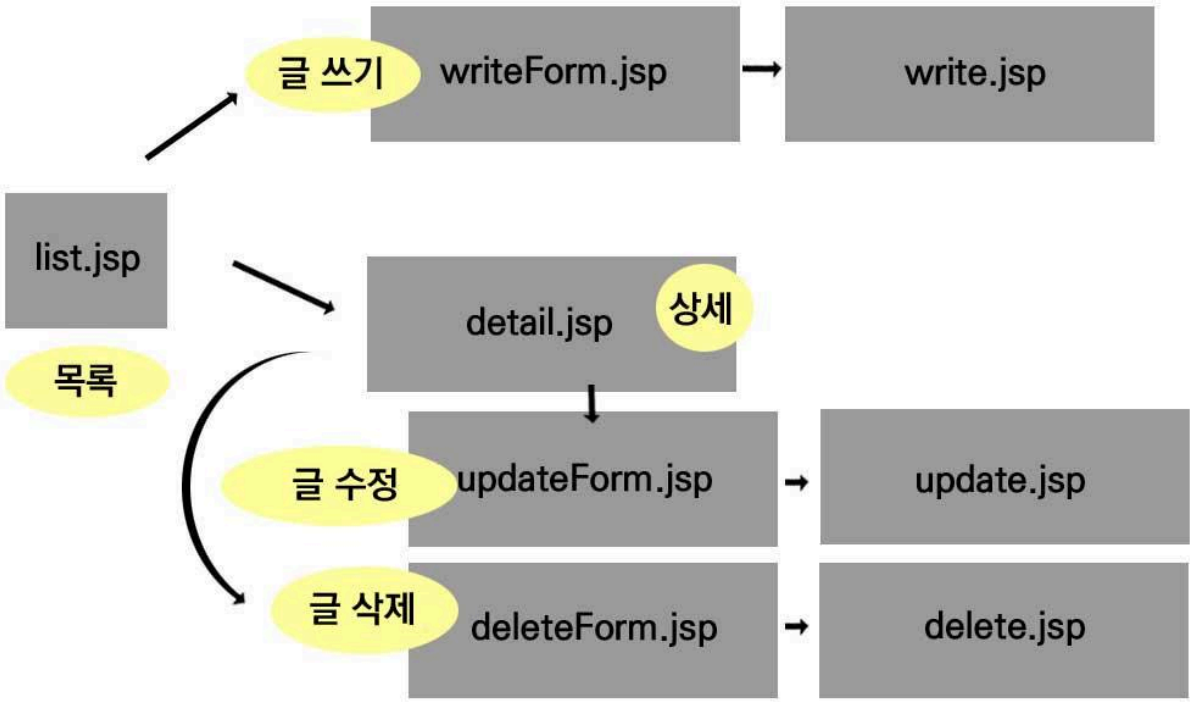
5) 모델1 방식으로 DB 연동하기



회원관리


아이디중복체크
confirmId.jsp
회원가입
registerUserForm.jsp
registerUser.jsp
로그인
loginForm.jsp
login.jsp
로그인 안 된 경우
main.jsp
로그인 된 경우
메인
logout.jsp
로그아웃
myPage.jsp
MyPage
회원정보수정
modifyUserForm.jsp
modifyUser.jsp
회원탈퇴
deleteUserForm.jsp
deleteUser.jsp

게시판



7. EL & JSTL

1) EL

(1) EL 특징

- JSP의 네가지 기본 객체가 제공하는 영역의 속성 사용
- 집합 객체에 대한 접근 방법 제공
- 수치 연산, 관계 연산, 논리 연산자 제공
- 자바 클래스 메서드 호출 기능 제공
- 표현 언어만의 기본 객체 제공

(2) EL 표기법

${expr}

(3) EL의 연산자

- 수치 연산자 : +, - , *, / 또는 div, % 또는 mod
- 비교 연산자 : == 또는 eq, != 또는 ne, < 또는 lt, > 또는 gt, <= 또는 le, >= 또는 ge
- 논리 연산자 : && 또는 and, || 또는 or, ! 또는 not
- empty 연산자

1) 값이 null이면 true 반환
2) 값이 빈 문자열(“”)이면 true 반환
3) 값이 길이가 0인 배열이면 true 반환
3) 값이 빈 Map이면 true를 반환
4) 값이 빈 Collection이면 true를 반환
5) 이외의 경우에는 false를 반환

(4) EL의 기본객체

기본객체	설명
pageContext	JSP의 page 기본 객체와 동일
pageScope	pageContext 기본 객체에 저장된 속성의 <속성,값>매핑을 저장한 Map 객체
requestScope	request 기본 객체에 저장된 속성의 <속성,값> 매핑을 저장한 Map 객체
sessionScope	session 기본 객체에 저장된 속성의 <속성,값> 매핑을 저장한 Map객체
applicationScope	application 기본 객체에 저장된 속성의 <속성,값>매핑을 저장한 Map객체
param	요청 파라미터의 <파라미터이름,값> 매핑을 저장한 Map객체
paramValues	요청 정보의 <파라미터이름,값 배열> 매핑을 저장한 Map 객체
header	요청 정보의 <헤더이름,값> 매핑을 저장한 Map 객체
headerValues	요청 정보의 <헤더이름,값 배열> 매핑을 저장한 Map 객체
cookie	<쿠키 이름, Cookie> 매핑을 저장한 Map 객체
initParam	초기화 파라미터의 <이름,값> 매핑을 저장한 Map 객체

2) JSTL

- core 라이브러리

jsp 페이지에 core 라이브러리를 사용할 수 있도록 taglib 디렉티브 명시
<%@ taglib prefix="c" uri=http://java.sun.com/jsp/jstl/core %>
prefix : 접두사
uri : core 라이브러리가 존재하는 위치

(1) **<c:set var="변수명" value="설정값" target="객체" property="값" scope="범위" />**

- 지정된 변수에 값을 할당하는 태그

target은 값을 설정하고자 하는 객체를 명시한다. target에 객체를 명시하면 명시된 객체에 값을 셋팅할 수 있다.
property는 taregt에 명시된 객체의 프로퍼티를 의미하며 명시된 프로퍼티에 value에 명시한 설정값을 셋팅합니다.
scope은 값을 셋팅한 범위를 의미(page,request,session,application), 생략하면 scope="page"

(2) **<c:remove var="변수명" scope="범위" />**
- 설정된 속성을 제거하는 태그
scope은 값이 셋팅된 범위를 의미(page,request,session,application), 생략하면 scope="page"

(3) **<c:out value="출력값" default="기본값" escapeXml="true/false" />**
- 지정된 값을 출력하는 태그

value 에 명시한 값을 출력하며 value 가 null 일경우 default에 기술한 값을 출력
escapeXml는 기본값이 true이며 true라고 지정하면 <과 같은 특수문자를 < 엔티티 레퍼런스 형태로 변환한다.즉, escapeXml를 true라고 하면 HTML 태그를 인정하지 않는다는 의미. false라고 지정하면 태그를 인정함.

(4) **<c:if test="조건" var="변수명" scope="범위" />**
- 조건이 true이면 수행문을 수행하는 태그

test에서 조건을 체크해서 true 또는 false가 되면 var에 지정된 변수의 그 값이 담김
scope은 var에 지정한 변수가 셋팅되는 범위를 의미(page,request,session,application), 생략하면 scope="page"

(5)
```
<c:choose>
    <c:when test="조건"></c:when>
    <c:when test="조건"></c:when>
    <c:otherwise></c:otherwise>
</c:choose>
```
- 여러개의 when 태그에 명시한 조건을 체크하고 조건에 만족하는 수행문을 수행, 만족하는 조건이 없으면 otherwise에 명시한 수행문을 수행

(6) **<c:forEach items="객체명" begin="시작 인덱스" end="끝 인덱스" step="증감식" var="변수명" varStatus="상태변수" />**
- 수행문을 반복해서 수행

items : 속성에 인덱스가 존재하는 객체를 지정하여 반복수행할 때 사용
begin : 수행문을 반복시킬 시작 인덱스 지정

end : 수행문의 반복이 끝날 인덱스 지정
step : 증감식 지정
var : 현재 반복하고 있는 값이 저장되는 변수지정
varStatus : 반복 상태를 지정하는 변수 지정

예) varStatus 의 사용예

```
<c:forEach var="list" items="itemList" varStatus="status">
        ${status.index} <-  0부터 시작하는 인덱스 표시
        ${status.count} <- 1부터 시작하여 1씩 증가
        ${status.first}  <- forEach 반복의 처음 일 경우 true
        ${status.last}  <- forEach 반복의 마지막 일 경우 true
</c:forEach>
```

예) forEach를 사용할 때 List의 크기 구하기
<%@ taglib prefix="fn" uri=http://java.sun.com/jsp/jstl/functions %> <- 페이지 상단에 명시
${fn:length(itemList)}

(7) **<c:forTokens items="객체명" delims="구분자" begin="시작 인덱스" end="끝 인덱스" step="증감식" var="변수명" varStatus="상태변수" />**
- 문자열을 구분자로 잘라내어 출력

items : 잘라내고자 하는 문자열이 담겨있는 객체
delims : 구분자 지정
begin : 시작 인덱스
end : 끝 인덱스
step : 증감식
var : 잘라진 문자열이 담기는 변수
varStatus : 반복 상태지정 변수

(8) **<c:catch var="변수명" />**
- 예외 발생시 예외 처리 태그

예외가 발생할 수 있는 수행문 앞 뒤에 catch 태그를 배치하고 예외가 발생하면
var에 지정한 변수명에 예외 문구를 저장한다. out 태그 또는 el 를 통해 예외 문구를 호출해 출력할 수 있다.

(9) **<c:import url="URL" var="변수명" scope="범위" varReader="변수명" context="context" charEncoding="인코딩" />**
- 지정한 url 페이지의 내용을 읽어와 출력

(10) **<c:redirect url="URL" context="context" />**

- 지정한 url로 redirect

(11) **<c:url var="변수명" scope="범위" value="값" context="context" />**
- url 생성

(12) **<c:param name="파라미터명" value="값" />**
- 파라미터로 전달하고 싶은 값을 name에 기술된 파라미터명과 value에 명시한 값의 쌍으로 전송

-JSTL i18n capable formatting 라이브러리의 국제화 태그를 이용해 국제화 지원

jsp 페이지에 i18n capable formatting 라이브러리를 사용할 수 있도록 taglib 디렉티브 명시
<%@ taglib prefix="fmt" uri=http://java.sun.com/jsp/jstl/fmt %>
prefix : 접두사
uri : i18n capable formatting 라이브러리가 존재하는 위치

(1) **<fmt:bundle basename="properties파일경로및 파일명" prefix="prefix">....</fmt:bundle>**
- 국제화를 적용을 위해 properties 파일경로 및 파일을 읽어와서 언어 적용

basename : 사용할 언어별 데이터가 key와 value의 쌍으로 작성된 properties 파일의 경로및 파일명 지정
prefix : bundle 태그 내에서 message 태그를 통해 value를 읽어올 때 key속성 앞에 접두사 지정

(2) **<fmt:message key="메시지의 key값" bundle="setBundle 태그를 통해 로딩한 번들을 얽어올 때 사용함" var="변수명" scope="범위" />**
- 국제화를 적용한 메시지의 key에 대한 value를 호출

key : 메시지의 key값을 통해 value 호출
var : 변수명을 지정하면 message태그를 출력기능을 상실하고 변수를 out 태그 또는 el를 통해 호출 하여 value를 출력해야 함

(3) **<fmt:setBundle basename="properties파일경로및 파일명" var="메시지를 저장할 변수명" scope="범위" />**
- 페이지 전체에서 사용할 번들을 지정

국제화 처리 예)
1. 각 페이지에서 읽어갈 key와 value가 저장될 properties 파일 작성

message.properties <- default 언어가 사용될 파일
message_ko.properties <- 한글이 사용될 파일

message_en.properties <- 영어가 사용될 파일

2. properties 파일이 들어갈 폴더 생성
WEB-INF/classes/폴더 생성(bundle) <- 배포시 (Dynamic Web Project를 만들어 이클립스에서 작업중이라면 Java Resources:src 에 폴더를 만듬, 자동적으로 classes 이하에 폴더가 복사됨)

3. 생성한 폴더에 message.properties, message_ko.properties 파일 등을 만들고 내용 입력
message_ko.properties 예

member_admin_title=회원관리
member_admin_name=관리자
member_admin_email=admin@test.com
member_admin_phone= 관리자 연락처는 {0}입니다. <- {0}은 message 태그 호출시 param 태그로 전달되는 데이터를 받음

4-1. properties 파일의 내용을 읽어갈 jsp 파일 작성(bundle 태그 이용시)
bundle 태그를 사용하면 지정된 영역내에 단일한 언어를 적용함

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:bundle basename="bundle.message">
<html>
<head>
   <title><fmt:message key="member_admin_title"/></title>
</head>
<body>
  <fmt:message key="member_admin_name"/>
  <br>
   <fmt:message key="member_admin_phone">
      <fmt:param value="${phone}" />
   </fmt:message>
</body>
</html>
</fmt:bundle>
```

4-2. properties 파일의 내용을 읽어갈 jsp 파일 작성(setBundle 태그 이용시)
setBunde 태그를 사용할 경우 setBundle 태그에 변수를 지정해서 페이지내에서 적용할 bundle를 선택적으로 사용할 수 있음

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:setBundle var="msg" basename="bundle.message"/>
```

```
<html>
<head>
  <title><fmt:message bundle="${msg}" key="member_admin_title"/></title>
</head>
<body>
<fmt:message bundle="${msg}" key="member_admin_name"/>
<br>
 <fmt:message bundle="${msg}" key="member_admin_phone">
    <fmt:param value="${phone}" />
 </fmt:message>
</body>
</html>
```
---

(4) **<fmt:requestEncoding value="인코딩값" />**
- request.setCharacterEncoding()과 같이 전송된 데이터의 인코딩 처리

(fmt:formatNumber5) **<fmt:setLocale value="로케일값" variant="브라우저 스펙" scope="범위" />**
- 로케일 지정
value : 로케일 값 지정 ex) ko, en

(6) **<fmt:bundle basename="properties파일경로및 파일명" prefix="prefix">....</fmt:bundle>**
- 국제화를 적용을 위해 properties 파일경로 및 파일을 읽어와서 언어 적용

basename : 사용할 언어별 데이터가 key와 value의 쌍으로 작성된 properties 파일의 경로및 파일명 지정
prefix : bundle 태그 내에서 message 태그를 통해 value를 읽어올 때 key속성 앞에 접두사 지정

(7) **<fmt:message key="메시지의 key값" bundle="setBundle 태그를 통해 로딩한 번들을 얽어올 때 사용함" var="변수명" scope="범위" />**
- 국제화를 적용한 메시지의 key에 대한 value를 호출

key : 메시지의 key값을 통해 value 호출
var : 변수명을 지정하면 message태그를 출력기능을 상실하고 변수를 out 태그 또는 el를 통해 호출 하여 value를 출력해야 함

(8) **<fmt:setBundle basename="properties파일경로및 파일명" var="메시지를 저장할 변수명" scope="범위" />**
- 페이지 전체에서 사용할 번들을 지정

(9) <fmt:formatNumber value="Number로 형식화할 수치 데이터"
type="숫자, 통화, 퍼센트 중 하나{number|currency|percent}"
pattern="사용자 지정 패턴"
currencyCode="통화코드지정"
currencySymbol="통화기호"
groupingUsed="{true|false}출력시 그룹 분리 기호(,)
포함여부"
maxIntegerDigits="출력시 integer 최대 자릿수 지정"
minIntegerDigits="출력시 integer 최소 자릿수 지정"
maxFractionDigits="출력시 소수점 이하 최대 자릿수 지정"
minFractionDigits="출력시 소수점 이하 최소 자릿수 지정"
var="변수"
scope="범위" />
- 수치 데이터를 숫자, 통화, 퍼센트로 변환

(10) <fmt:parseNumber value="Number로 파싱할 수 있는 수치 데이터"
type="숫자, 통화, 퍼센트 중 하나{number|currency|percent}"
pattern="사용자 지정 패턴"
parseLocale="로케일지정"
integerOnly="integer로만 파싱할지 지정"
var="변수"
scope="범위" />
- 문자열에서 숫자로 파싱

(11) <fmt:formatDate value="형식화할 날짜와 시간 데이터"
type="{time|date|both}"
deteStyle="{short|full}"
timeStyle="{short|full}"
pattern="사용자 지정 패턴"
timeZone="타임존 지정"
var="변수"
scope="범위" />
- 날짜에 형식 지정

(12) <fmt:parseDate value="파싱할 날짜 데이터"
type="{time|date|both}"
dateStyle="{short|full}"
timeStyle="{short|full}"
pattern="사용자 지정 패턴"
timeZone="타임존 지정"
parseLocale="로케일 지정"
var="변수"
scope="범위" />

- 문자열에서 날짜로 파싱

(13) <fmt:setTimeZone value="타임존 지정"
var="변수"
scope="범위" />
- 타임존 지정

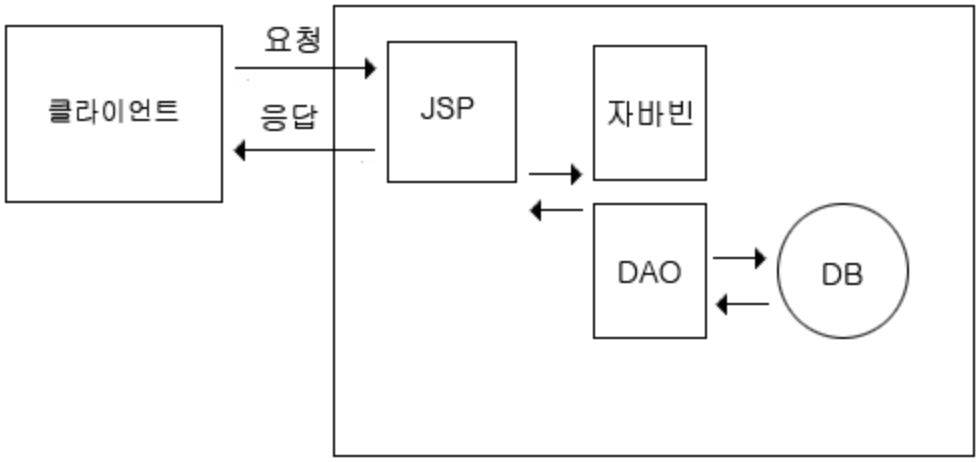
value : ex) Australia/Brisbane, America/New_York 등등

(14) **<fmt:timeZone value="">....</fmt:timeZone>**
- 타임존 지정

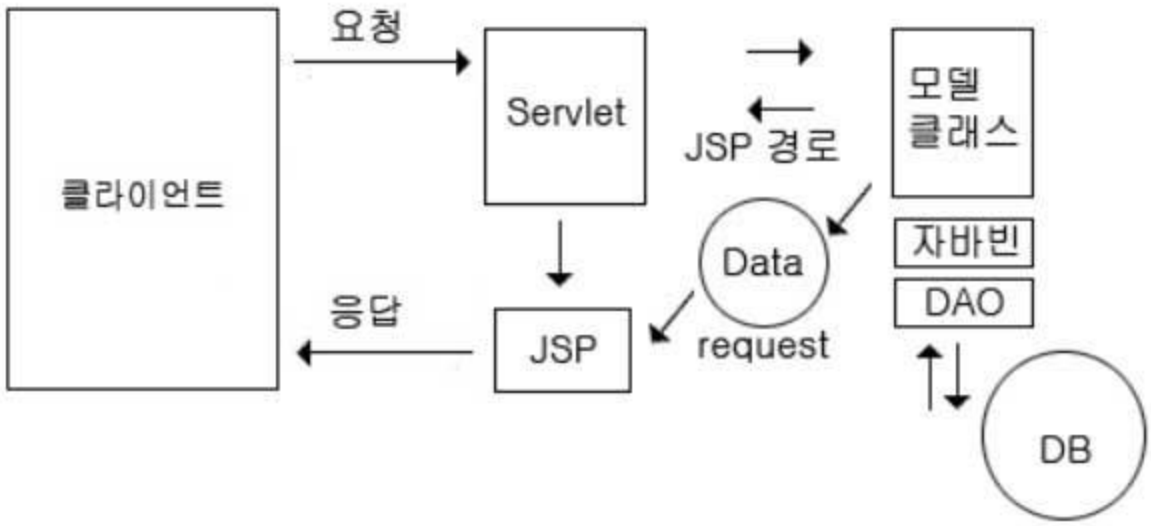
value : ex)GMT, KST, UTC

8. MVC

1) 모델 1 구조



2) 모델 2 구조(MVC)



3) MVC 패턴

모델	비즈니스 영역의 상태 정보를 처리한다.
뷰	비즈니스 영역에 대한 프레젠테이션 뷰(즉, 사용자가 보게 될 결과 화면)를 담당한다.
컨트롤러	사용자의 입력 및 흐름 제어를 담당한다.

특징

로직을 처리하는 모델과 결과 화면을 보여주는 뷰가 분리되 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중