# Servlet & JSP

# 웹 서버 프로그래밍

( Java Enterprise Edition Base Web Programming)

작성자 : 김 정현

# 1. 웹 프로그래밍 기초

## 1.1 World Wide Web

World Wide Web(World Wide Web, WWW, W3)은 인터넷에 연결된 컴퓨터들을 통해 사람들이 정보를 공유할 수 있는 전 세계적인 정보 공간을 말한다.
인터넷상의 정보를 하이퍼텍스트 방식과 멀티미디어 환경에서 검색할 수 있게 해주는 정보검색 시스템이다.

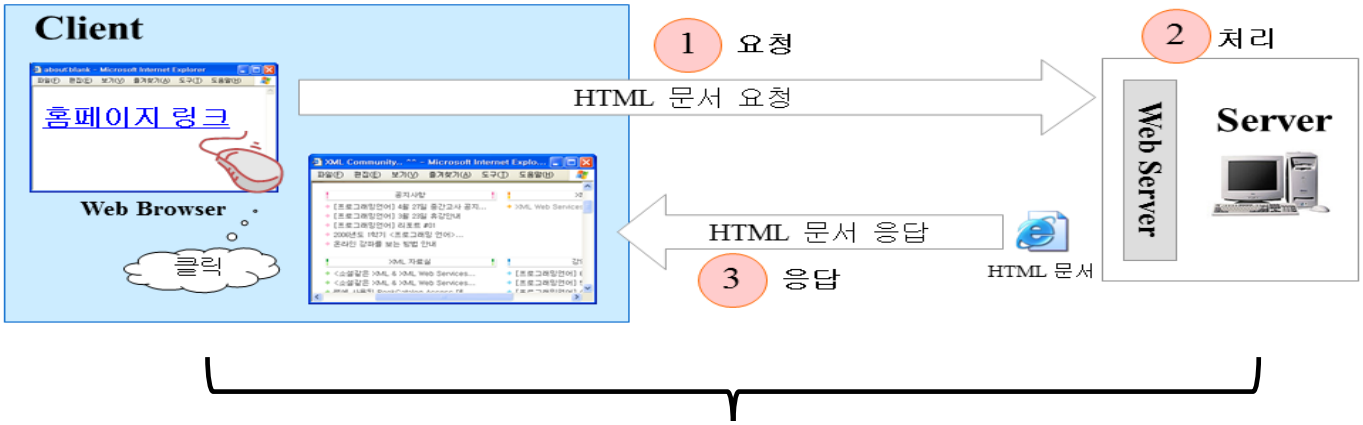
1989년 3월 유럽 입자 물리 연구소(CERN)의 소프트웨어 공학자인 **팀 버너스 리**의 제안으로 시작되어 연구, 개발되었다. 원래는 세계의 여러 대학과 연구기관에서 일하는 물리학자들 상호간의 신속한 정보교환과 공동연구를 위해 고안되었다.

World Wide Web에 관련된 기술은 World Wide Web 컨소시엄(W3C)이 개발하고 있다. W3C는 HTML, HTTP 등의 표준화를 진행하고 있으며, 최근엔 시맨틱 웹에 관련된 표준을 제정하고 있다.

W3C(World Wide Web Consortium)는 World Wide Web을 위한 표준을 개발하고 장려하는 조직으로 팀 버너스 리를 중심으로 1994년 10월에 설립되었다. W3C는 회원기구, 정직원, 공공기관이 협력하여 Web 표준을 개발하는 국제 컨소시엄이다. W3C의 설립취지는 웹의 지속적인 성장을 도모하는 프로토콜과 가이드라인을 개발하여 World Wide Web의 모든 잠재력을 이끌어 내는 것이다.

## 1.2 웹의 처리 구조

웹 통신에 사용되는 표준 통신 프로토콜은 HTTP(HyperText Transfer Protocol) 이다.




HTTP는 웹상에서 정보를 주고 **언트와 웹 서버 사이에 이루어지는 요청과 응답(request/response)에 대한 프로토콜**로서 클라이언트인 웹 브라우저가 HTTP를 통하여 서버로부터 웹 페이지나 그림 정보를 요청하면, 서버는 이 요청에 응답하여 필요한 정보를 해당 사용자에게 전달하게 된다.

HTTP 프로토콜은 **Connection Oriented와 Stateless 방식**으로 동작하는 프로토콜로서 신뢰성 있는 통신을 하면서도 처리 효율이라는 부분을 강조하였으므로 인터넷 환경에 가장 적합한 통신구조로 인정 받았다.

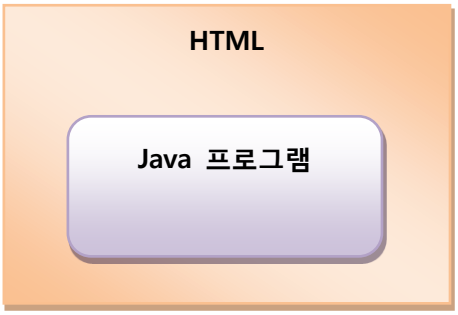
[ HTTP 1.1 에서 지원되는 요청 방식들 ]

| 요청방식 | 설 명 |
|---|---|
| GET | URI에 지정된 파일을 얻고자 할 때 사용되는 요청 방식으로 디폴트 방식이다. name=value로 구성되는 간단한 데이터(Query 문자열)를 URI 뒤에 추가하여 전달하면서 요청하고자 하는 경우에도 사용된다. http://localhost:8080/test.jsp?productid=00001 |
| HEAD | GET과 동일하나 바디 내용은 받지 않고 HTTP 헤더 정보만 받는다. |
| POST | 원하는 방식으로 인코딩 된 데이터를 요청 바디에 포함하여 전송하면서 파일을 요청하고자 하는 경우 사용된다. Query 문자열 전달시의 GET 방식을 보완한 요청 방식이다. |
| OPTIONS | 요청 URI에 대하여 허용되는 통신 옵션을 알고자 할 때 사용된다. |
| DELETE | 서버에서 요청 URI에 지정된 자원을 지울 수 있다. |
| PUT | 파일을 업로드할 때 사용된다. 그러나 일반적으로 사용되는 파일 업로드는 POST 방식을 사용하고 있고, PUT은 아직 많이 사용되지 않는다. |

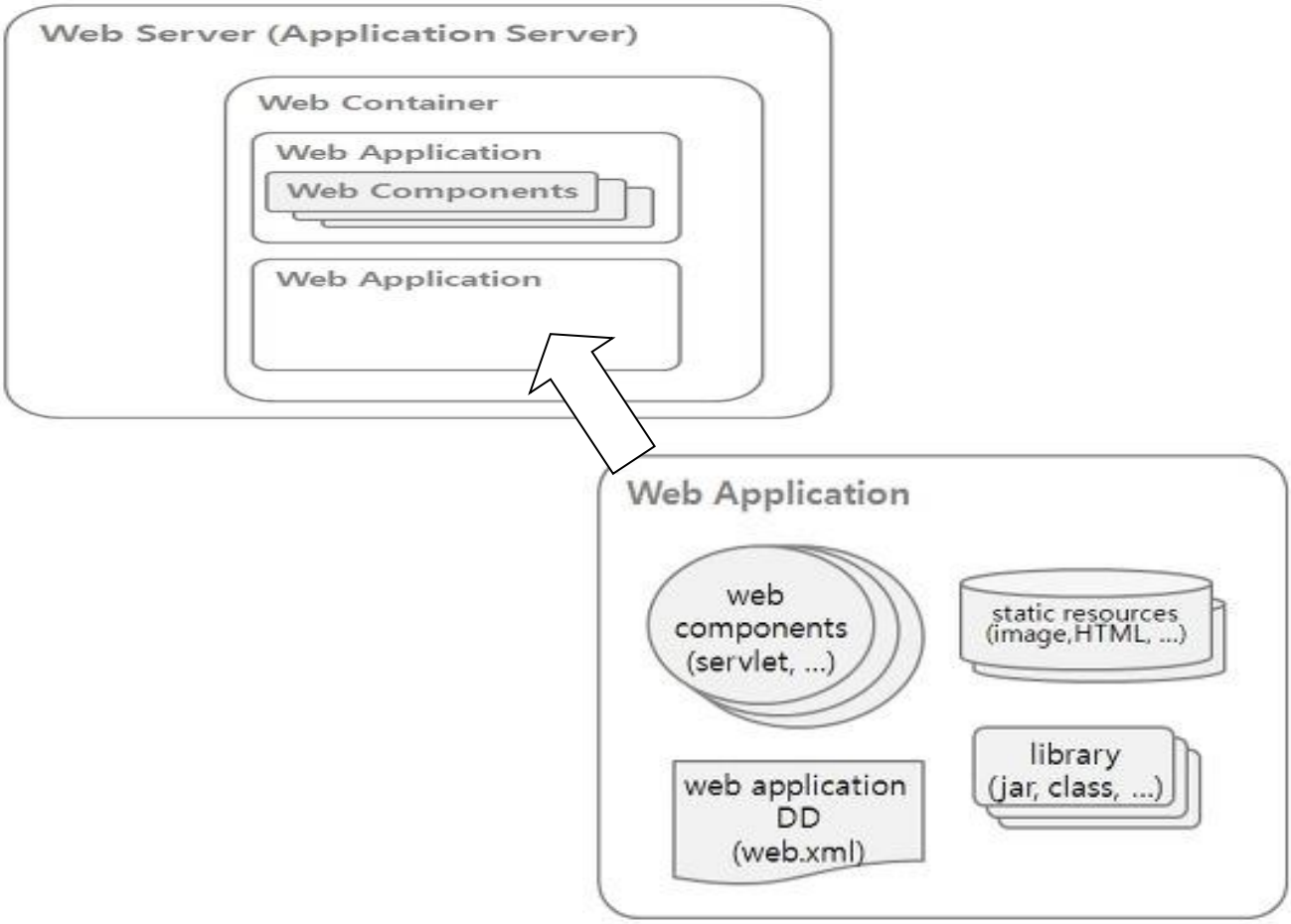
## 1.3 Java EE 기반의 웹 어플리케이션

Servlet 과 JSP 는 Java 의 Enterprise Edition 에 속하는 웹 어플리케이션 기술로서 웹 클라이언트의 요청에 의해 웹 서버에서 수행되고 그 수행 결과가 클라이언트에 응답되는 기술로서 Servlet 기술은 프로그래밍 성격이 강하며 JSP 는 프리젠테이션 성격이 강하다

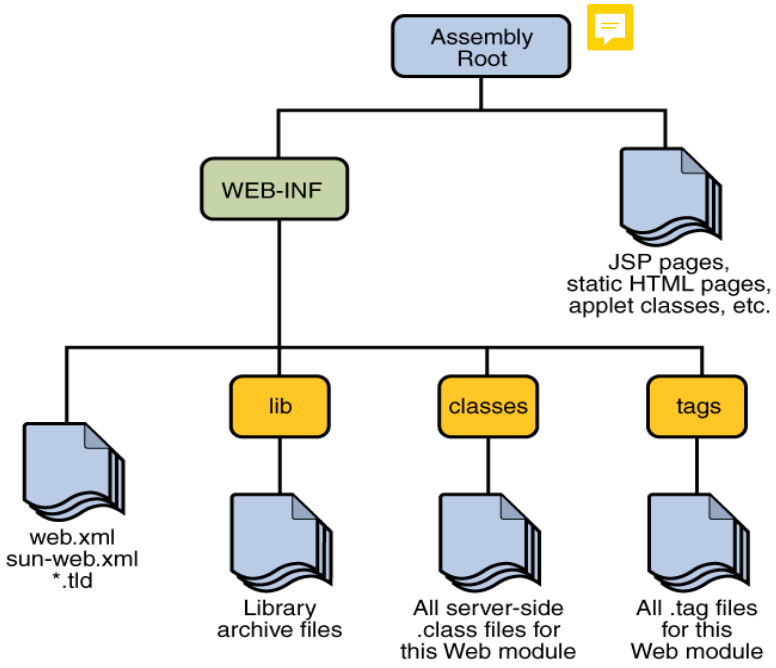
Servlet 은 상속 구문과 메서드 오버라이딩 구문을 적용한 Java 로 구현된 프로그램으로서 수행결과를 HTML 로 응답하도록 구현하는 기술이며 JSP 는 HTML 문서 안에 JSP 태그와 동적인 처리를 담당하는 Java 코드를 삽입하여 구현하는 기술이다.


Java EE 환경에서 Servlet 과 JSP 는 웹 컨테이너(엔진이라고도 함)에 의해 관리되고 수행되는 웹 컴포넌트로서 여러 웹 컴포넌트들이 모여 하나의 웹 어플리케이션을 구성하게 된다.

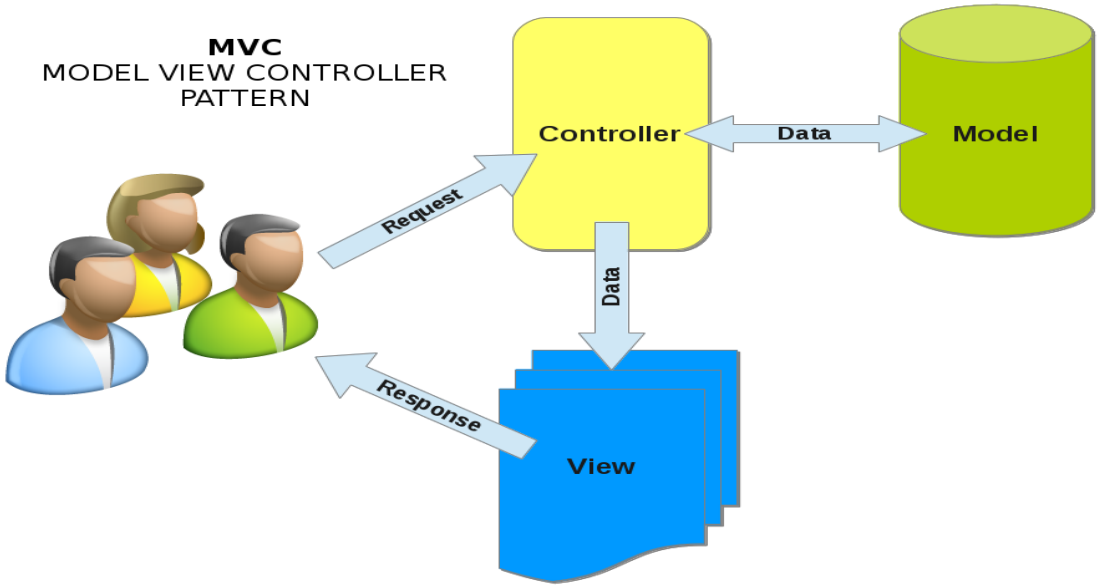


[ 웹 어플리케이션의 디렉토리 구조 ]

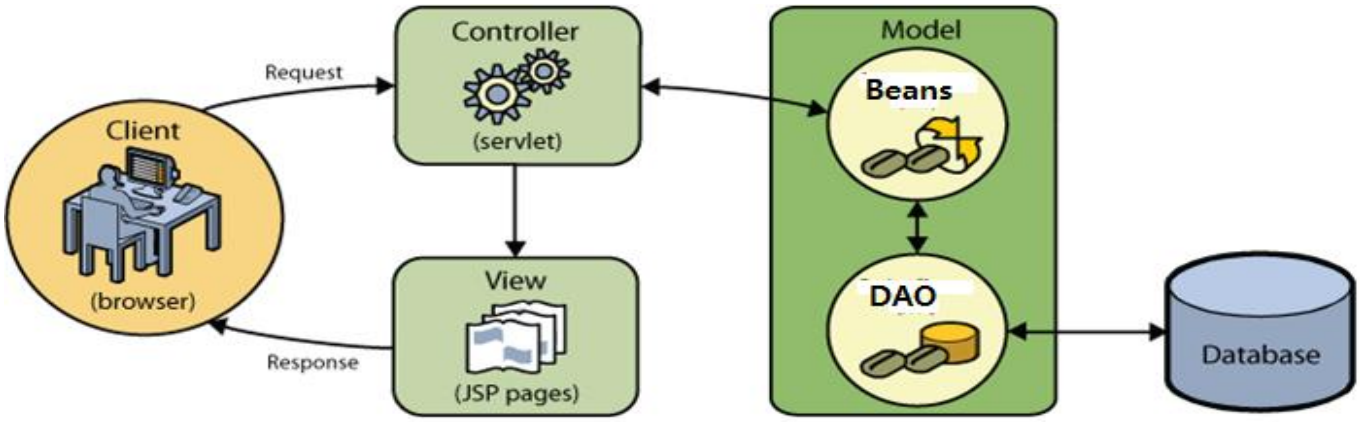



## 1.4 MVC(Model-View-Controller) 패턴

모델-뷰-컨트롤러(Model–View–Controller, MVC)는 소프트웨어 공학에서 사용되는 아키텍처 패턴으로서 이 패턴을 성공적으로 사용하면, 사용자 인터페이스로부터 비즈니스 로직을 분리하여 어플리케이션의 시각적 요소나 그 이면에서 실행되는 비즈니스 로직을 서로 영향 없이 쉽게 고칠 수 있는 어플리케이션을 만들 수 있다. MVC에서 **모델**은 어플리케이션의 정보(데이터)를 담당하며, **뷰**는 텍스트, 체크박스 항목 등과 같은 사용자 인터페이스 요소를 담당하고, **컨트롤러**는 데이터와 비즈니스 로직 사이의 상호동작 관리하며 그리고 어플리케이션의 기능을 담당한다.

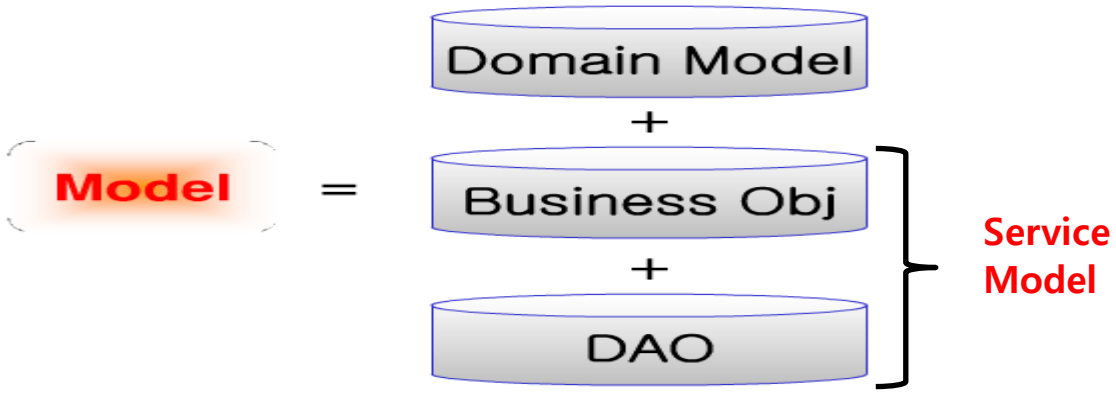


컨트롤러는 Servlet 객체로 구현하고 뷰는 JSP 로 그리고 모델은 VO, DTO, DAO 등의 Java 객체로 구현한다.




[ 모델 ]

다양한 비즈니스 로직(SERVICE, BIZ), DB 연동 로직(DAO) 그리고 처리 결과를 저장(VO, DTO)하는 기능을 지원하는 Java 객체로서 도메인 모델과 서비스 모델로 구성된다.

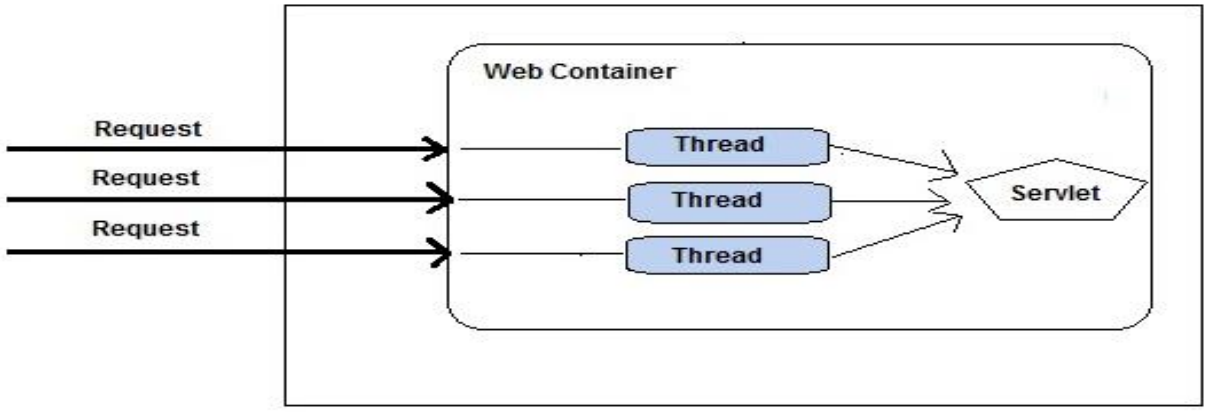


## 2. Servlet 프로그래밍

Java Servlet(Java Servlet)은 Java 를 사용하여 웹 페이지를 동적으로 생성하는 서버측 프로그램으로서 Java EE 사양의 일부분이다. 이 기술을 사용하여 쇼핑몰이나 이러닝 그리고 온라인 뱅킹 등의 다양한 웹 시스템이 구축되어 왔다.

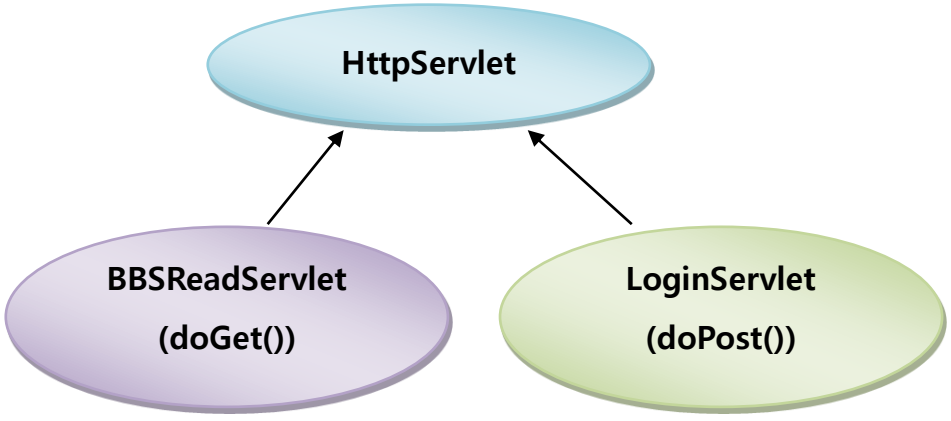

이 기술은 1998년에 발표된 기술로서 이 기술 이전에 CGI 라는 기술이 사용되고 있었다. CGI는 요청이 있을 때마다 새로운 프로세스가 생성되어 응답하는 데 비해, Java Servlet은 요청마다 프로세스보다 가벼운 스레드 기반으로 응답하므로 보다 가볍게 클라이언트 요청 처리할 수 있다. 또한, Java Servlet은 Java 로 구현되므로 다양한 플랫폼에서 동작 가능하다.

또한 웹  클라이언트로부터의 수행 요청으로 생성된 Servlet의 객체는 수행이 종료되어 응답된 후에도 객체 상태를 계속 유지하면서 다음 요청에 대하여 바로 수행될 수 있는 상태를 유지한다. 또한 하나의 Servlet을 여러 클라이언트가 동시 요청했을 때 하나의 Servlet 객체를 공유하여 다중 스레드 기반에서 처리되므로 응답 성능을 향상시킬 수 있다.




### ▣ Servlet 의 구현 방법

Servlet 은 HttpServlet 이라는 클래스를 상속하여 구현하며 어떠한 요청 방식을 지원하는 Servlet 인가에 따라 doGet() 또는 doPost() 메서드를 오버라이딩하여 구현한다.



▣ Servlet의 등록과 매핑

대부분의 웹 자원들은 파일의 확장자로 파일의 종류를 구분하지만 Servlet 의 경우에는 불가능하다. Java 프로그램으로 구현되는 Servlet 는 컴파일을 통해서 .class 라는 확장자를 갖는 실행 파일이 되는데 웹에서 .class 라는 실행 파일을 갖는 파일의 요청은 Servlet 보다 먼저 소개된 기술인 Applet 에서 이미 사용되고 있기 때문이다.

그러므로 Servlet 클래스의 경우에는 서버에서 Servlet 프로그램으로 인식되어 처리되도록 등록과 매핑이라는 설정을 web.xml 이라는 디스크립터 파일에 작성해주어야 한다. web.xml은 웹 어플리케이션에 대한 다양한 정보를 설정하는 파일로서 디스크립터 파일이라고 하며 다음과 같이 구성으로 WEB-INF 폴더에 만들어야 한다.

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

<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
   <servlet>
      <servlet-name>HS</servlet-name>
      <servlet-class>mypkg.HelloServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>HS</servlet-name>
      <url-pattern>/hello</url-pattern>
   </servlet-mapping>
</web-app>
```

- 초기 파라미터 전달

```
<servlet>
   <servlet-name>ServletName</servlet-name>
   <servlet-class>ServletClassFile</servlet-class>
   <init-param>
      <param-name>initParam1</param-name>
      <param-value>initParam1Value</param-value>
   </init-param>
   <init-param>
```

```
        <param-name>initParam2</param-name>
        <param-value>initParam2Value</param-value>
    </init-param>
</servlet>
```

[ Servlet 3.0 부터 추가된 Annotation ]

Servlet 3.0 에서는 web.xml 에 작성되던 Servlet 등록과 매핑, 초기 파라미터 설정, 리스너나 필터 등록과 같은 내용들을 소스 내에서 Annotation 구문으로 대신할 수 있는 구현 방법을 지원하고 있다. 즉, Annotation 을 사용하면 web.xml 에 일일이 설정 태그를 작성해주지 않아도 된다. 다음은 Servlet 3.0에서 지원되는 Annoation 리스트이다.

**@WebServlet: Servlet 프로그램을 등록과 매핑을 정의한다.**
**@WebInitParam: Servlet 프로그램에 전달할 초기 파라미터를 정의한다.**
**@WebListener: 리스너를 정의한다.**
**@WebFilter: 필터를 정의한다.**
**@MultipartConfig: Servlet 프로그램에서 다중 파티션으로 전달되는 파일 업로드를 처리할 수 있음을 정의한다.**

- 정의 예

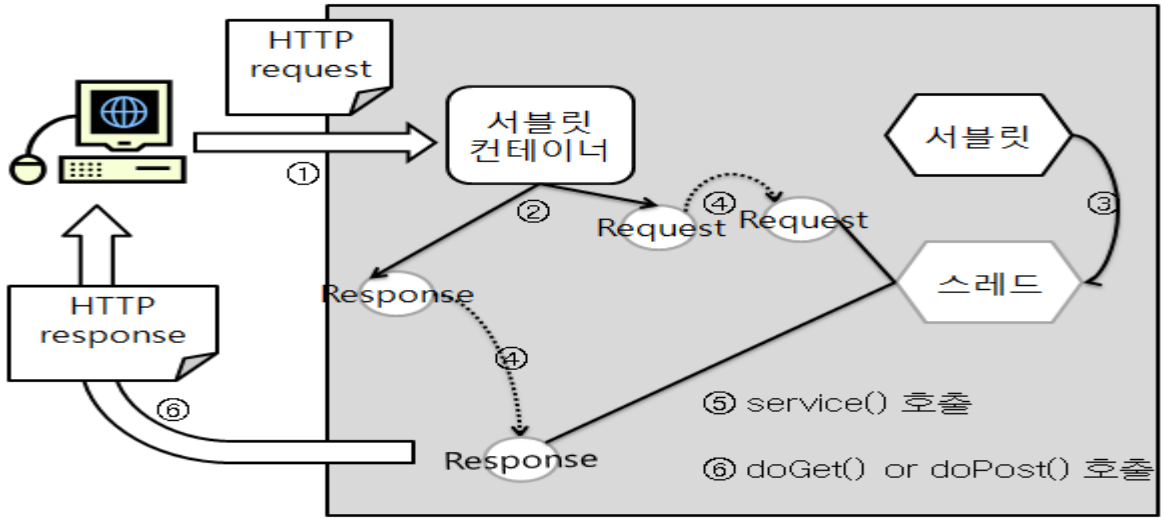
```
@WebServlet("/hello")
public class Hello1Servlet extends HttpServlet { ..... }


@WebServlet(urlPatterns = {"/hello1", "/hello2"})
public class Hello2Servlet extends HttpServlet { ..... }


@WebServlet(name = "HelloServletExample", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet { ..... }


@WebServlet(
    name = "HelloServletExample",
    urlPatterns = {"/hello"},
```

```
    initParams = {
        @WebInitParam(name = "param1", value = "value1"),
        @WebInitParam(name = "param2", value = "value2")}
)
public class HelloServlet extends HttpServlet { ..... }
```

## ▣ 요청 및 응답 객체 생성

웹 클라이언트로부터 Servlet 수행 요청이 전달되면, Servlet 컨테이너는 클라이언트로부터 전달된 요청 정보를 가지고 HttpServletRequest 객체와 HttpServletResponse 객체를 생성하여 Servlet 의 doGet() 또는 doPost() 메서드 호출시 아규먼트로 전달한다.

HttpServletRequest 객체는 클라이언트에서 전달되는 다양한 요청 정보를 Servlet 에 전달하는 기능으로 사용되며, HttpServletResponse 객체는 클라이언트로의 응답에 사용되는 객체이다.




- HttpServletRequest 객체 : 웹 클라이런트에서 전송되는 요청 정보 추출

  getHeader(name), getHeaders(name), getHeaderNames(),
  getContentLength(), getContentType(), getCookies()
  getRequestURI(), getQueryString(), getProtocol(), getMethod()

- HttpServletResponse 객체 : 웹 클라이언트로의 응답 처리

  setStatusCode(int statuscode), sendError(int code, String message),
  sendRedirect(url), setHeader(String headerName, String headerValue)
  setContentType(String mimeType), setContentLength(int length)

## ▣ Servlet 객체 생성과 객체 해제

HttpServletRequest 와 HttpServletResponse 객체를 생성한 후, Servlet 컨테이너는 요청된 Servlet의 객체가 생성된 상태인지 검사하게 된다. 만약 Servlet 객체가 생성되어 있는 상태라면 바로 수행시키고, 객체가 생성된 상태가 아니라면 해당 Servlet의 클래스 파일을 찾아서 로딩한 후 객체를 생성하여 수행시킨다. 또한 생성된 Servlet 객체는 서버 종료 시까지 또는 웹 어플리케이션이 리로드될 때까지 객체 상태를 계속 유지한다.

**Servlet을 객체 생성하여 수행시킬 때 처리 흐름**

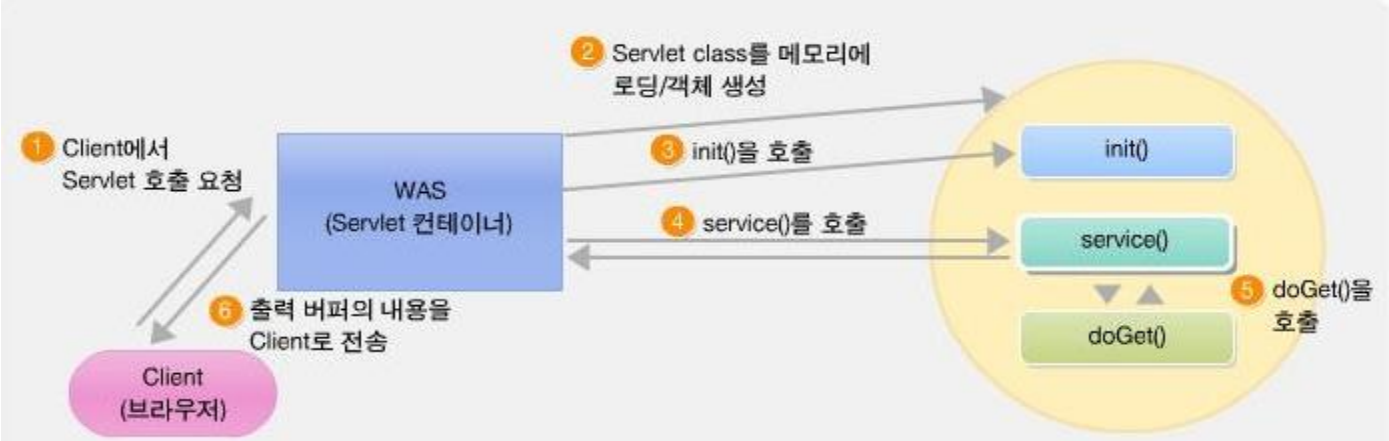
- Servlet 클래스를 로딩하여 객체 생성
- Servlet 객체의 init() 메서드 호출
- Servlet 객체의 service() 메서드 호출

**이미 생성된 Servlet 객체를 수행시킬 때 처리 흐름**

- Servlet 객체의 service() 메서드 호출

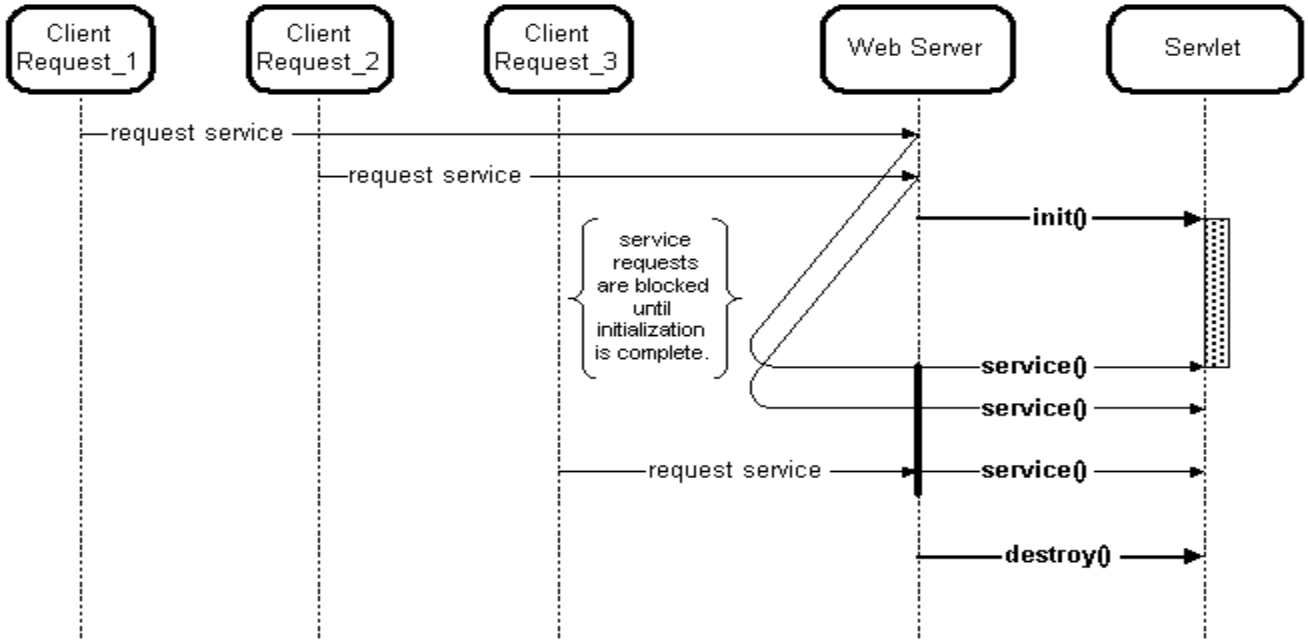
**Servlet 객체가 메모리에서 해제될 때의 처리 흐름**

- Servlet 객체의 destroy() 메서드 호출




[최초 요청인 경우의 Servlet의 자세한 수행 흐름]

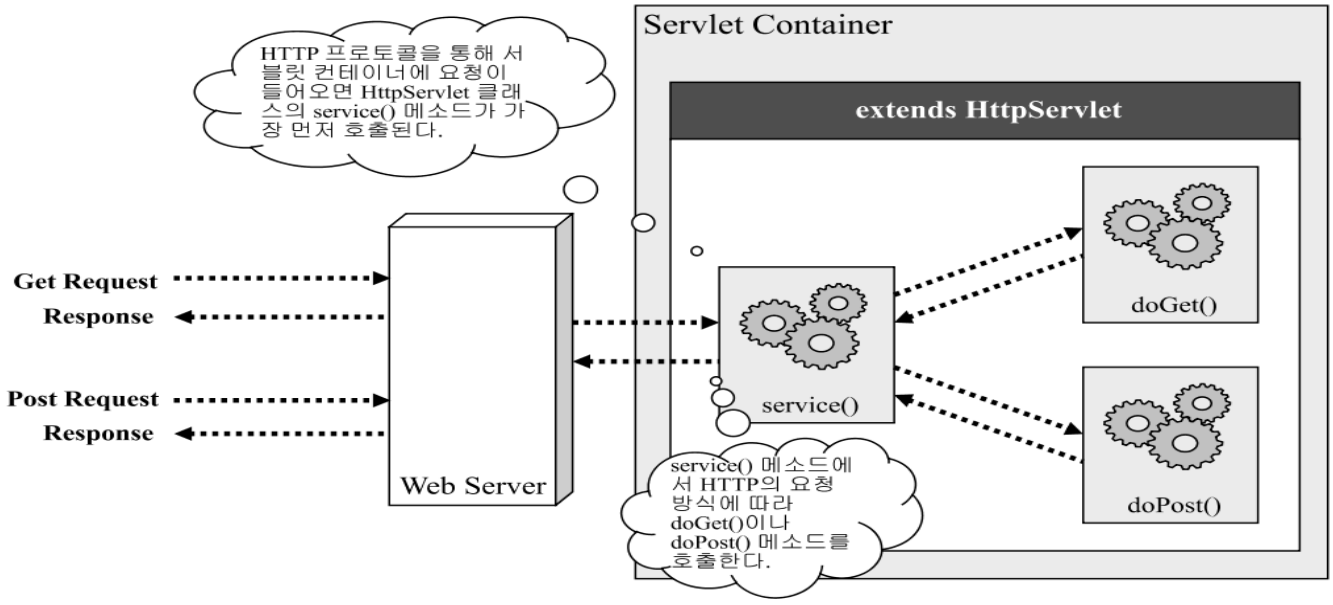

[두 번째 이후의 요청인 경우의 Servlet의 자세한 수행 흐름]




## ▣ Servlet의 실행

Servlet은 HttpServlet 클래스를 상속받아 어떠한 요청 방식을 지원하는 Servlet인가에 따라서 doGet() 또는 doPost() 메소드를 재정의하여 구현한다. 웹 클라이언트로부터 Servlet이 요청된 방식에 따라서 doGet() 또는 doPost() 메소드가 Servlet 컨테이너에 의해 호출되어 Servlet의 기능을 처리하게 된다.

- 하이퍼링크 텍스트(<A>태그)를 클릭하여 요청한다. → GET 방식
- URL 을 주소필드에 입력하여 직접 요청한다. → GET 방식
- <IMG>태그로 요청한다. → GET 방식
- <FORM>태그로 요청한다. → method 속성의 값에 따라서 GET 방식 또는 POST 방식
- <IFRAME> 태그로 요청한다. → GET 방식



## ▣ <FORM> 태그

HTML 문서에서 사용자의 입력을 서버로 전달하는 기능은 <FORM> 태그를 이용해 구현된다. 다음은 <FORM> 태그에서 지원하는 속성이다.

- action : 사용자의 입력 데이터를 처리할 CGI 프로그램의 URL 주소를 지정한다.
- method : 사용자 데이터를 넘겨주는 방식을 지정하는데, GET과 POST 방식 두 가지다.
  GET은 입력 내용을 요청 URI 뒤에 붙여서 보내고, POST는 요청 바디에 담아서 보낸다.
- enctype : 서버로 보내지는 데이터의 형식을 지정하며 종류는 세 가지다.

1. application/x-www-form-urlencoded
디폴트 값이다. 서버로 전송되기 전에 url-encode 된다는 뜻이다.

2. mutipart/form-data
파일 받으면서 설정해준게 이 값인데 이미지나 파일을 서버로 전송할 경우 이 방식을 사용
한다.

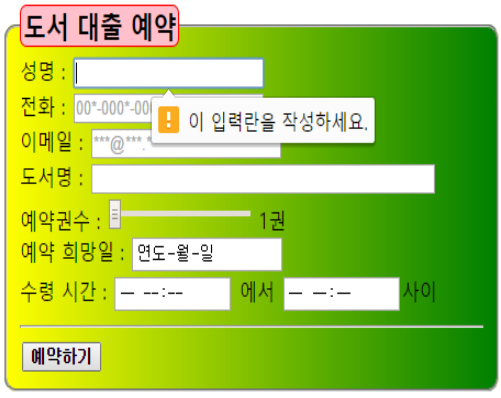
3. text/plain
인코딩을 하지 않은 문자 그대로의 상태를 전송한다는 의미이다.

웹 브라우저 화면에 사용자 정보 입력 형식을 표시할 때는 <input> 태그, <textarea> 태그 그리고 <select> 태그를 이용한다. 다음은 <input> 태그에서 지원하는 주요 속성이다.

- text : 텍스트 입력
- password : 암호 입력
- checkbox : 체크 박스
- radio : 라디오 버튼
- file : 서버로 업로드할 파일 선택
- submit : 입력 데이터 전송
- reset : 입력 데이터 취소
- hidden : 서버로 전달할 name=value 쌍의 데이터 정의


HTML5 에서는 <input> 태그에 다음과 같은 속성들을 추가하여 지원한다.

- <input type="email">
  이메일 주소 입력시 사용
  서버로 전송시 이메일 형식 자동 체크
- <input type="url">




웹 사이트 주소 입력시 사용

- <input type="number">

  숫자를 스핀 박스를 이용해서 입력가능

  min : 최소값, max, : 최대값, step : 간격, value : 초기값

- <input type="range">

  슬라이드 막대를 숫자 선택

  min : 최소값, max, : 최대값, step : 간격, value : 초기값으로 생략시 중간에 위치.

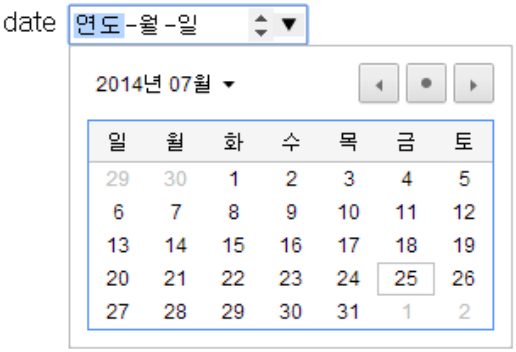
- <input type="search">

  검색 상자 삽입

  검색어 입력하면 오른쪽에 x가 표시됨

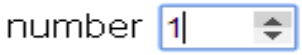
- <input type="date">, <input type="month">, <input type="week">, <input type="time">

  달력에서 날짜를 선택하거나 스핀 박스에서 시간을 선택

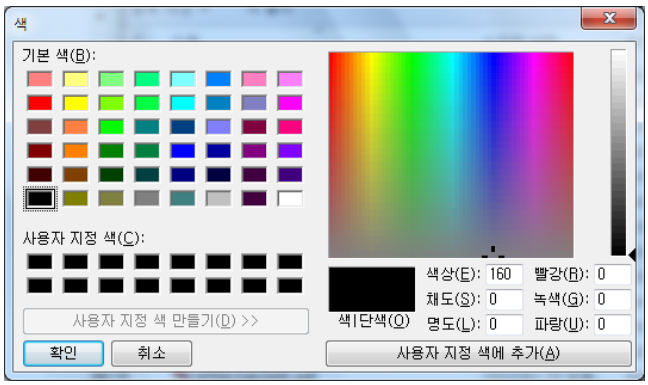



- <input type="color">

  색상 선택 상자 표시








## ▣ Query 문자열(요청 파라미터)

Query 문자열이란 웹 클라이언트에서 웹 서버에 요청을 보낼 때 추가로 전달하는 name 과 value 로 구성되는 문자열로서 요청 파라미터라고도 한다. 주로 다음과 같은 형식으로 전달되는데 전달방식은 GET 방식과 POST 방식이 있다.

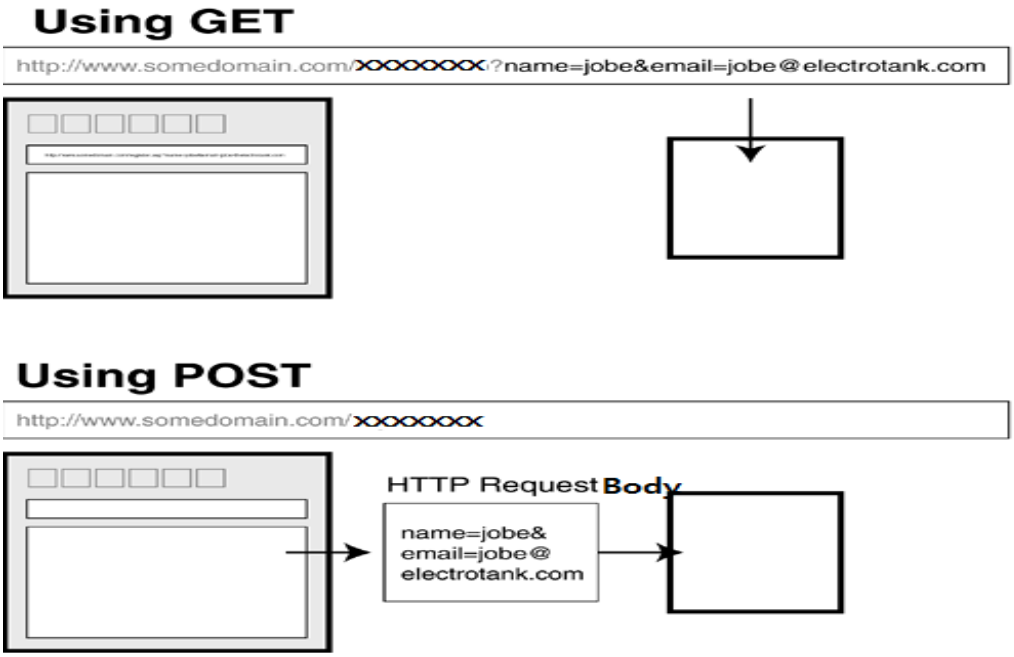
**name1=value1&name2=value2&name3=value3**

[ GET 방식 ]

- 전달되는 Query 문자열의 길이에 제한이 있고 내용이 브라우저의 주소 필드에 보여진다.
- <FORM> 태그를 사용해도 되고 요청 URL 에 ? 기호와 함께 직접 Query 문자열을 붙여서 전달하는 것도 가능하다.

[ POST 방식 ]

- 전달되는 Query 문자열의 길이에 제한 없고 내용이 브라우저의 주소 필드에 보여지지 않는다.
- 전달 내용이 요청 바디에 담겨져서 전달된다. <FORM> 태그로만 요청 가능하다.




- Query 문자열의 추출

name 으로 하나의 value 값이 전달될 때

**String address = request.getParameter("address");**

name 으로 여러 개의 value 값들이 전달될 때

**String hobby[ ] = request.getParameterValues("hobby");**

- Query 문자열 추출시의 한글 문제

Servlet 에서 위의 메서드들을 사용하여 Query 문자열을 추출할 때 한글 깨짐 현상이 발생한다. 해결 방법은 요청 방식에 따라 다르다.

GET 방식 – 서버에 따라 다르다.

Tomcat 8.0 : 한글 문제가 발생하지 않는다.

Tomcat 7.0 이하 : 환경 파일인 server.xml 파일에서 protocol="HTTP/1.1" 속성을 포함하고 있는 <Connector> 태그를 찾아 URIEncoding="utf-8" 속성을 추가한다.

POST 방식 – Query 문자열을 추출하기 전에 HttpServletRequest 에서 제공되는 **setCharacterEncoding("utf-8")** 을 호출한다.

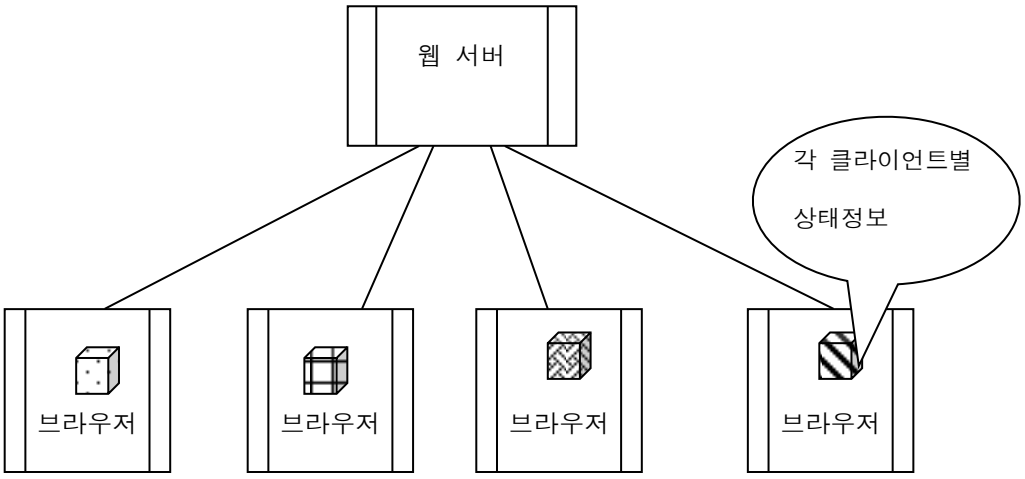
## ▣ 상태정보 유지 기술

웹 브라우저에서 웹 서버에 정보를 요청할 때 이전 접속시의 결과물(상태정보)를 일정시간 동안 유지하는 것을 상태정보 유지라고 한다. 상태정보 유지 방법은 여러 가지가 있으며

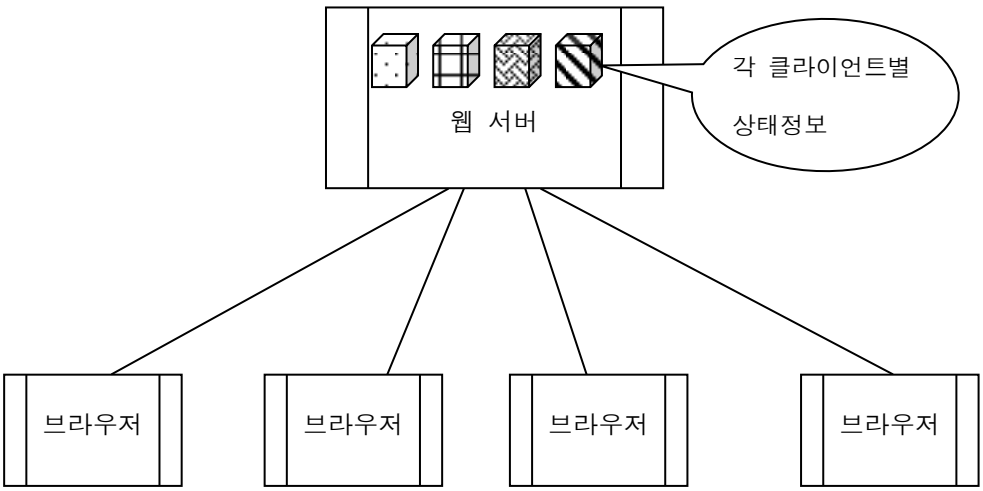
| Cookie 를 이용한 방법 | HttpSession 객체를 이용한 방법 | URL 문자열 뒤에 추가하는 방법 | <form> 태그의 hidden 타입을 사용하는 방법 |
|---|---|---|---|

다음 그림들은 상태 정보를 클라이언트에 저장하는 방식과 서버에 저장하는 방식을 소개하고 있다.

**[ 클라이언트 저장 ]**




**[ 서버 저장 ]**



[ HttpSession 객체를 이용한 상태정보 유지 ]

HttpSession 객체를 이용하는 상태 정보 유지는 다음과 같은 특징을 지원한다.

- 상태 정보는 객체로 만들어서 서버에 보관한다.
- 상태 정보가 유지되는 최대 시간은 요청을 보내온 브라우저가 기동되어 있는 동안이다.
- 구현 방법

(1) HttpSession 객체를 생성하거나 추출한다.
(2) HttpSession 객체에 상태정보를 보관할 객체를 등록한다. (한번만 등록하면 된다.)
(3) HttpSession 객체에 등록되어 있는 상태정보 객체의 참조 값을 얻어 사용한다.(읽기, 변경)
(4) HttpSession 객체에 등록되어 있는 상태정보 객체가 더 이상 필요 없으면 삭제 가능하다.

```
(1) HttpSession session = request.getSession();

(2) session.setAttribute("xxx", new Data());

(3) Data ref = (Data)session.getAttribute("xxx");

(4) session.removeAttribute("xxx");
```

- request.getSession() : HttpSession 객체를 추출하거나 새로이 생성한다.
  request.getSession(true) 와 동일하다. request.getSession(false) 는 HttpSession 객체를 추출하여 리턴하는데 없으면 null을 리턴한다.

- session.setAttribute("xxx", new Data()) : 보관하려는 정보를 객체로 만들어 HttpSession객체에 저장한다. "xxx" 라는 이름으로 객체의 참조 값을 보관한다 .

- session.getAttribute("xxx") : "xxx" 라는 이름으로 보관된 객체의 참조 값을 리턴한다.

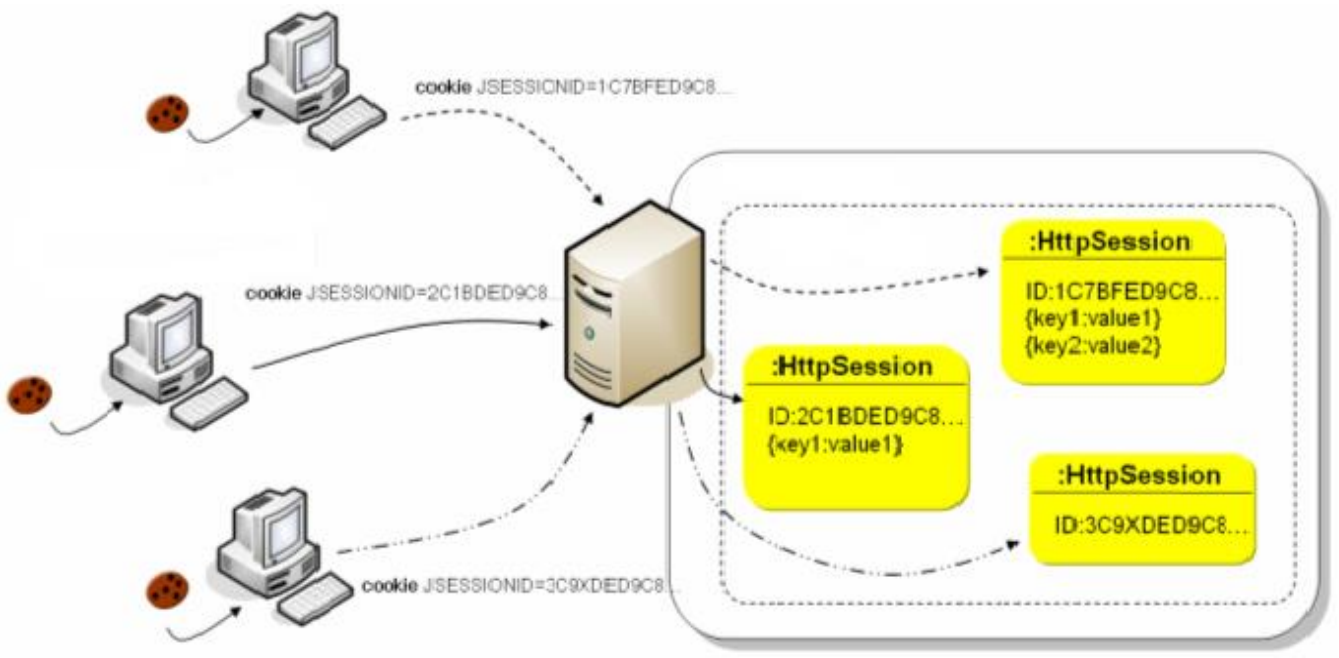
- session.removeAttribute("xxx") : "xxx" 라는 이름으로 보관된 객체의 참조 값을 삭제한다.

- session.invalidate() : HttpSession 객체를 강제로 삭제한다.

서버상에 생성되는 HttpSession 객체는 웹 클라이언트별로 하나씩 만들어진다. HttpSession 객체가 생성될 때 세션ID 가 하나 부여되며 이 세션ID 는 요청을 보내온 클라이언트의 브라우저에 쿠키 기술로 저장된다. 브라우저에 저장되는 세션ID에 대한 쿠키는 최대 유지 시간이 브라우저가 기동되어 있는 동안이다.

만일 브라우저가 재 기동 되어 세션ID 를 분실하게 되면 서버에 생성된 HttpSession 객체는 더

이상 사용 불가하게 된다. 뿐만 아니라 클라이언트로부터 일정시간 동안 요청이 없는 경우((Inactive Interval Time : 기본 30분)) 에도 서버에 생성된 HttpSession 객체는 더 이상 사용 불가하게 된다.




[ HttpSession 의 기타 주요 메서드 ]

public Enumeration getAttributeNames()

세션에 등록된 객체들의 이름을 열거한다.

public long getCreationTime()

1970. 1.1 GMT 부터 세션이 만들어졌을 때까지의 시간을 밀리초의 단위로 리턴한다.

public String getId()

세션에 지정된 세션 ID를 리턴한다.

public long getLastAccessedTime()

클라이언트 요청이 마지막으로 시도된 시간을 밀리초로 리턴한다.

public int getMaxInactiveInterval()

클라이언트의 요구가 없을 때 서버가 현재의 세션을 언제까지 유지할지를 초시간 단위로 리턴한다. 이때 디폴트 세션마감시간은 30분으로 지정되어 있다.

public boolean isNew()

서버측에서 새로운 세션을 생성한 경우에는 true를 리턴하고 기존의 세션이 유지되고 있는 경우라면 false를 리턴한다.

public void setMaxInactiveInterval(int seconds)

세션 유지 시간을 설정한다. 이 시간이 지나면 세션은 자동 종료(HttpSession객체 삭제)

된다.

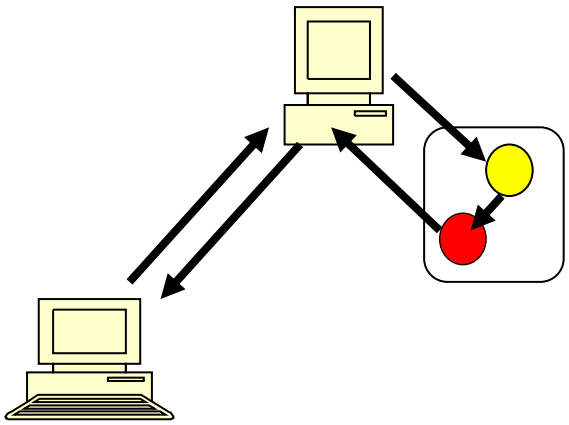
▣ 요청 재지정

요청재지정이란 클라이언트에서 요청한 페이지 대신 다른 페이지를 클라이언트가 보게 되는 기능으로서 redirect 방법과 forward 방법으로 나뉜다.

- redirect : HttpServletResponse 의 sendRedirect() 메서드를 사용한다.

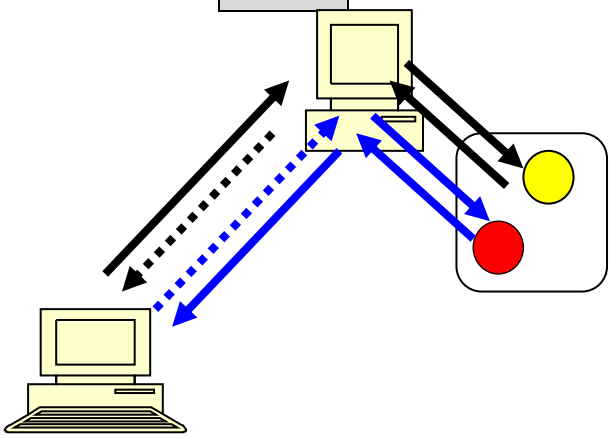
- forward : RequestDispatcher 의 forward() 메서드를 사용한다.

Forward



- 동일한 요청상에서 다른 자원에 요청을 넘겨서 대신 응답하게 함
- 동일한 서버의 동일 웹 어플리케이션에 존재하는 대상만 가능
- 브라우저의 주소필드의 URL 이 바뀌지 않음
- 두 자원이 HttpServletRequest 객체 공유

Redirect

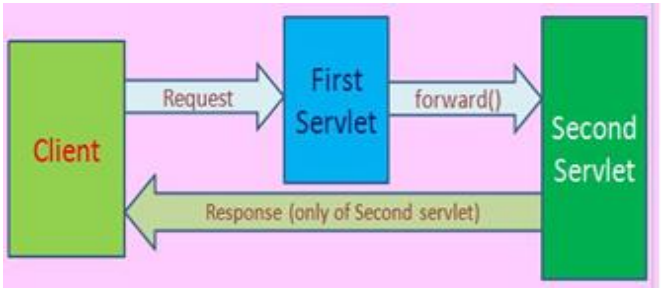


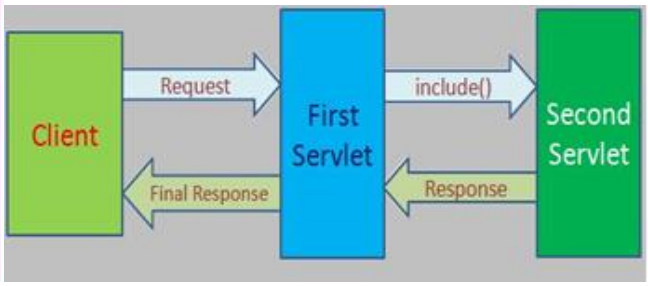
- 다른 자원을 다시 요청하여 응답하게 함
- Web 상의 모든 페이지로 요청재지정 가능
- 브라우저의 주소필드의 URL 이 바뀜
- 재지정 대상에 대한 요청 자체를 브라우저가 하게 됨
- 두 자원이 HttpServletRequest 객체를 공유하지 않음

**RequestDispatcher** 를 사용하는 요청 재지정 방법은 forward() 메서드 외에도 include() 메서드가 사용될 수 있다.

forward() : 요청 페이지 대신 다른 페이지가 대신 응답하게 한다.

include() : 요청 페이지 안에 다른 페이지의 처리 내용이 포함되어 같이 응답하게 된다.




## ▣ 객체 공유

객체의 스코프란 객체가 생성되어 유지되는 기간을 의미하며 **Page Scope, Request Scope, Session Scope 그리고 Application Scope** 로 구성된다.

Page Scope : Servlet 또는 JSP가 수행되는 동안만 유효한 객체가 된다.

Request Scope : Web 클라이언트로 부터의 요청이 끝날 때까지 유효한 객체가 된다.

Session Scope : 요청을 보내온 Web 클라이언트가 기동되어 있는 동안 유효한 객체가 된다.

Application Scope : 서버가 기동되어 있는 동안 유효한 객체가 된다.

Request Scope → HttpServletRequest 객체에 객체를 보관한다.

Session Scope → HttpSession 객체에 객체를 보관한다.

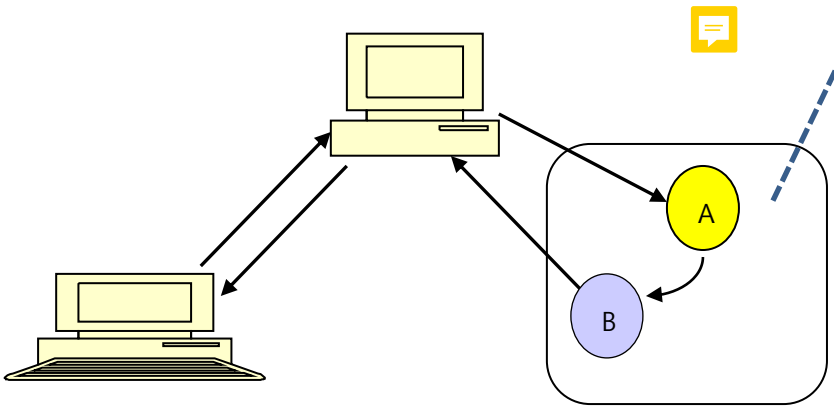
Application Scope → ServletContext 객체에 객체를 보관한다.

HttpServletRequest, HttpSession 그리고 ServletContext 는 모두 객체를 저장하는 방으로 사용하는 것이 가능하며 다음과 같은 객체의 저장, 추출, 삭제 기능의 메서드들을 지원한다.

```
public void setAttribute(String key, Object value)

public Object getAttribute(String key)

public void removeAttribute(String key)
```

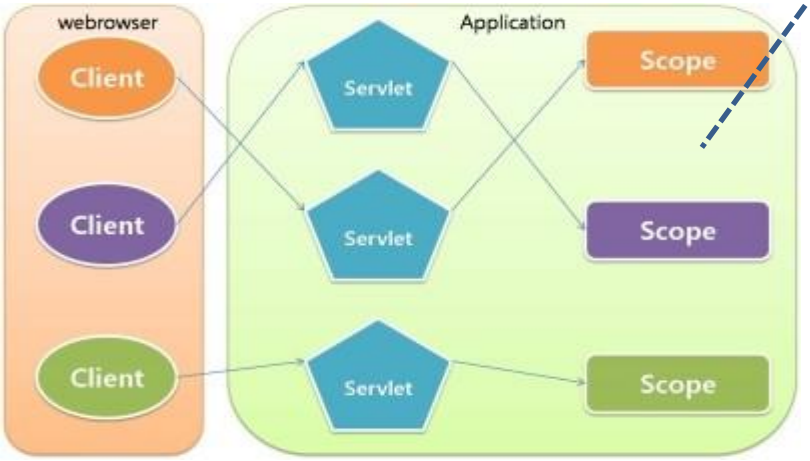
[ Servlet 과 JSP 에서 사용되는 Java 객체의 scope ]

[ 요청 동안의 객체 공유 ]



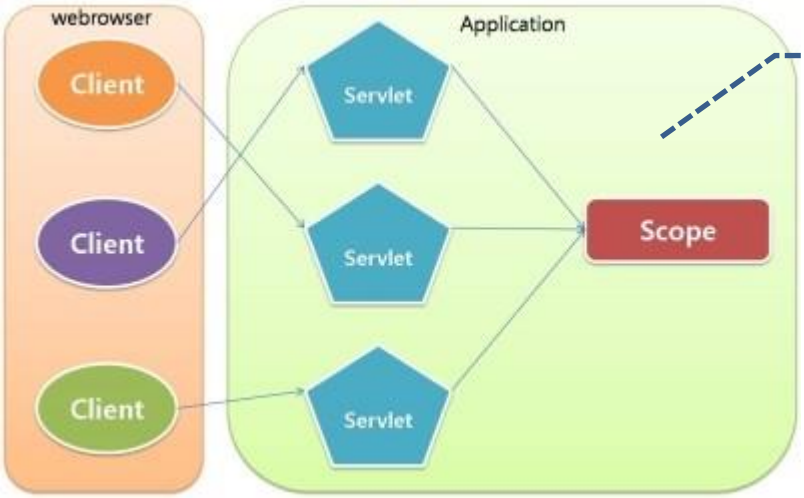

A와 B가 forward 또는 include 관계에 있는 경우 A 가 생성하는 객체를 HttpServletRequest 객체에 보관하면 B 에서 추출할 수 있다. 요청이 끝나면 사라진다.

[ 세션이 유지되는 동안 각 클라이언트별 객체 공유 ]




각 클라이언트별로 서버에 하나씩 만들어지는 HttpSession 객체에 객체를 보관하면 세션이 유지되는 한 계속해서 클라이언트별로 이 객체를 꺼내서 사용할 수 있다.

[ 서버가 기동되어 있는 동안 모든 클라이언트에 의한 객체 공유 ]




서버에 등록되는 Web Application 당 하나씩 만들어지는 ServletContext 객체에 객체를 보관하면 서버 종료시까지 이 객체를 꺼내서 사용할 수 있다. 이 객체는 모든 클라이언트에 의해 공유된다.

## 3. JSP 프로그래밍

JSP(JavaServer Pages)는 웹 페이지의 컨텐트를 구현하는 HTML 파일 내에 서버상에서 동적으로 처리하려는 부분을 적당한 JSP 태그와 Java 코드를 삽입하여 구현하는 기술이다. SUN 에서 1998 년 발표된 Servlet 기술의 성공에 힘을 받고 ASP 나 PHP 에 대응될 수 있는 웹 서버 스크립트 기술을 만든 것이 바로 JSP 이다. JSP 기술은 1999년에 발표되었다.

JSP는 실행 시에 Java Servlet 으로 변환된 후 실행되므로 Servlet 이라고 할 수도 있다. 하지만 Servlet 과는 달리 HTML 문서에 스크립트 방식으로 작성되므로 Servlet 보다 구현하기 쉬우며 웹 페이지의 디자인과 연계하여 개발하기에 편리하다. 웹 페이지로 표현하고자 하는 HTML에서 동적인 처리 내용을 포함하고자 하는 부분에 만 JSP 태그와 Java 코드를 사용하여 구현한다.

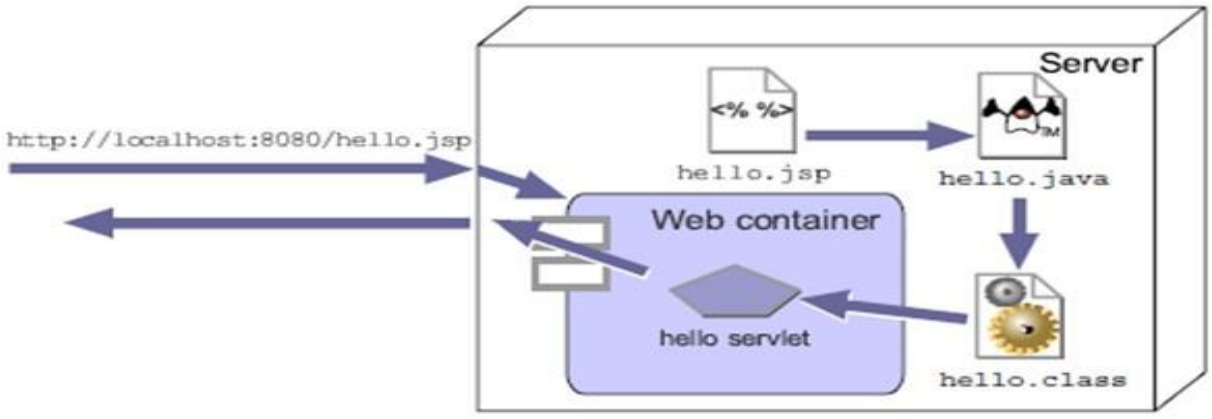
JSTL 등의 JSP 태그 라이브러리를 사용하는 경우에는 Java 코딩없이 태그만으로 구현하는 것도 가능하므로 개발생산성을 높일 수 있다.

▣ JSP 구현 시 알고 있어야 하는 내용

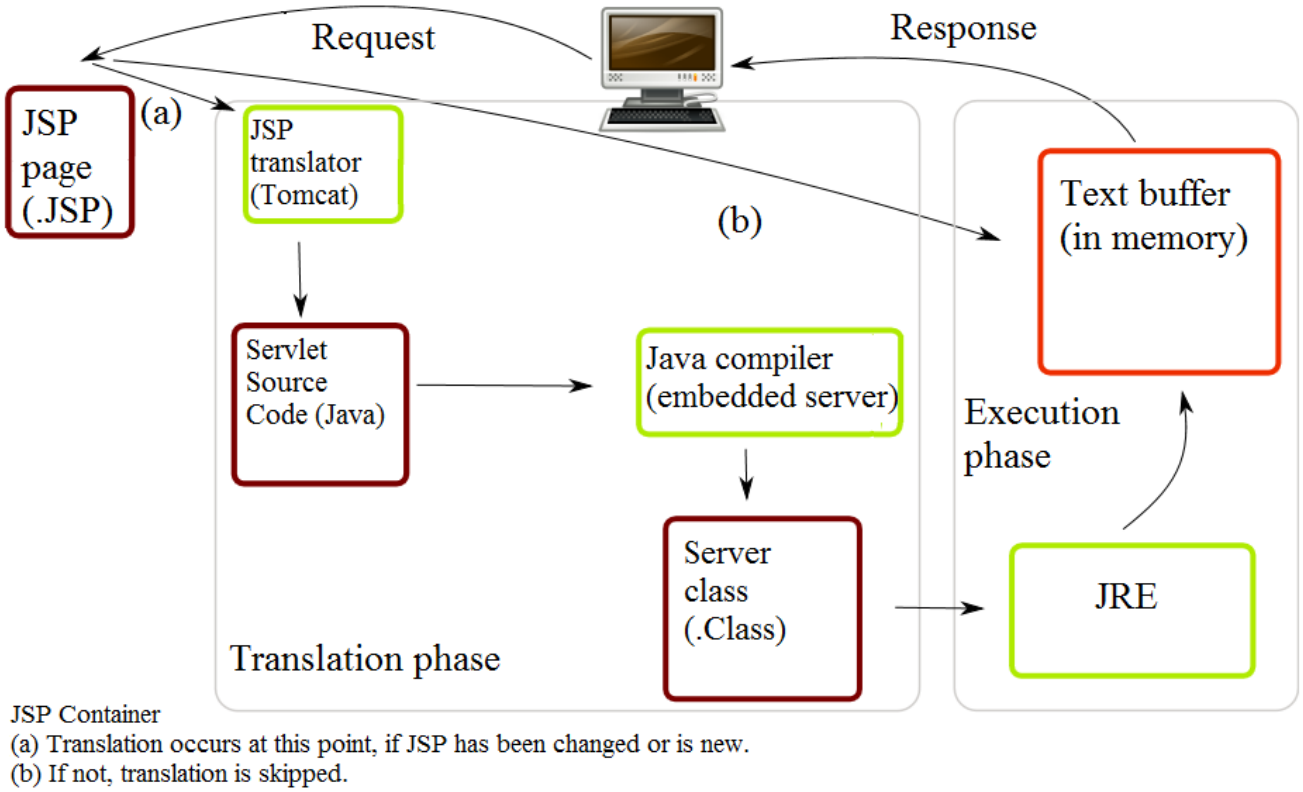
- Servlet 프로그래밍에서 학습한 모든 내용
- JSP의 스크립트 태그
- JSP의 액션 태그
- JSP의 내장 객체

▣ JSP의 처리구조

웹 클라이언트에서 JSP의 실행을 요청하면 서버의 JSP 컨테이너(컨버터)에 의해 Servlet 소스코드로 변환되고 컴파일된 후 실행 가능한 Servlet 클래스가 된다. 이 때 부터는 Servlet 컨테이너에 의해 객체 생성되어 실행되며 실행 흐름과 특성은 Servlet 과 동일하다. JSP 가 Servlet 소스코드로 변환되는 것은 JSP 가 작성 또는 수정된 후 최초 요청 시에만 처리된다.



[ JSP 의 상세 처리 과정 ]




▣ JSP 태그의 종류

<%-- --%> 주석문 태그
<%@ %> 지시자 태그
<% %> 스크립트릿(수행문) 태그
<%! %> 선언문(변수선언, 메서드정의) 태그
<%= %> 표현식

스크립팅 태그

<jsp:include page="..." />
<jsp:forward page="..." />
<jsp:useBean id="..." class="..." scope="..." />
<jsp:setProperty name="..." property="..." />
<jsp:getProperty name="..." property="..." />
<jsp:plugin ..... />
<jsp:param name="..." value="..." />

액션 태그

[ 지시자 태그 ]

JSP 를 Servlet 으로 변환할 때 지시하고자 하는 내용을 정의하는 태그이다. 지시할 사항에 따라서 다양한 지시
자 태그를 지원한다.

- page 지시자
  <%@page {attr = value ..} %>
- include 지시자
  <%@include {attr = value ..} %>
- taglib 지시자
  <%@taglib {attr = value ..} %>
- tag 지시자
  <%@tag {attr = value ..} %>
- variable 지시자
  <%@variable {attr = value ..} %>
- attribute 지시자
  <%@attribute {attr = value ..} %>

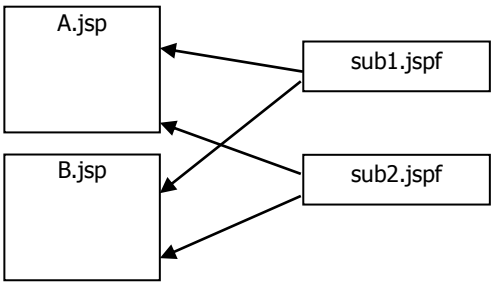
- page 지시자 태그

page 지시자는 컨테이너가 참조하는 다양한 정보들 중에서 JSP 페이지에 종속적인 설정 정보들을 알려주기 위한 수단으로 사용된다. page 지시자를 사용하면 해당 JSP 페이지가 어떤 문서(text, xml등)를 생성하는지, 어떤 Java 클래스를 사용하는지, 세션에 참여하는지, 출력 버퍼의 존재 여부와 같이 JSP 페이지를 실행 하는데 기본이 되는 정보들을 지정해 줄 수 있다.

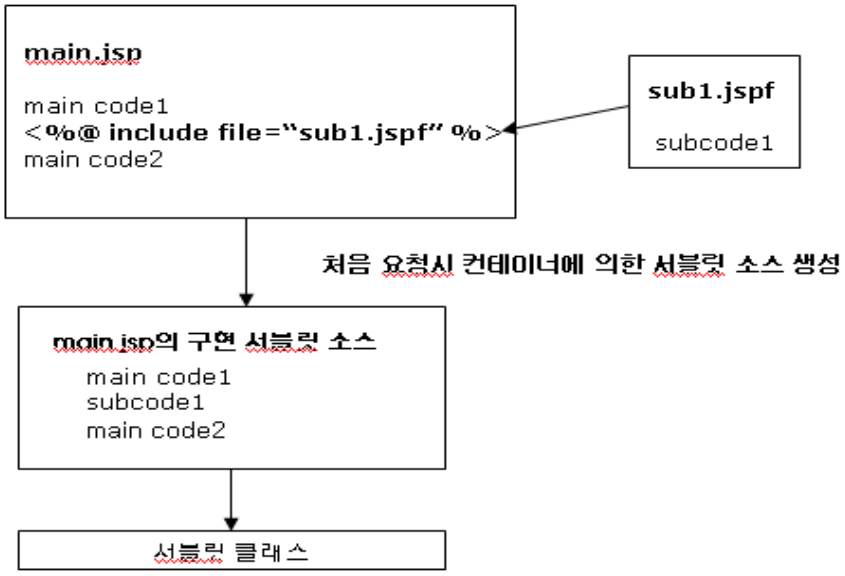
```
<%@ page [ language="java" ]
[ extends="package.class" ]
[ import="{package.class | package.*}, ..." ]
[ session="true|false" ]
[ buffer="none|8kb|sizekb" ]
[ autoFlush="true|false" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType [ ; charset=characterSet ]" |
        "text/html ; charset=ISO-8859-1"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

- include 지시자 태그

하나의 JSP 페이지는 여러 개의 JSP 파일로 구성될 수 있다. 일정한 Java 코드나 정적인 데이터(HTML, Text등)가 여러 페이지에서 반복된다면 반복되는 부분을 다른 파일로 저장하고 필요한 JSP 페이지에서 불러 쓰는 것이 훨씬 효과적인 것이다.




[ include 지시자 태그의 처리 과정 ]




## [ 스크립팅 태그 ]

JSP 지시자는 JSP 컨테이너가 JSP 소스를 Servlet 으로 변환할 때 어떻게 페이지를 처리할 것인지에 영향을 끼치는 반면, 스크립팅 태그들은 JSP 페이지에 직접 Java 코드를 끼워 넣을 수 있게 한다. JSP 의 스크립팅 태그는 다음의 세가지 유형이 있다.

- 선언문 태그 : <%!  %>
  JSP 페이지 내부에서 사용할 멤버 변수나 메서드를 선언한다.
- 표현식 태그 : <%=  %>
  동적 데이터를 응답 정보에 추가하기 위한 식을 정의한다.
- 스크립트릿 태그 : <%  %>
  Java API를 이용하거나 기타 Java 를 이용한 소스 코드를 작성한다.

## [ 액션 태그 ]

정해진 기능을 지원하는 태그로서 JSP 에서는 다음의 6 개의 액션 태그가 사용된다.

<jsp:include>
JSP 페이지의 수행 결과 내에 다른 자원의 내용 또는 수행 결과를 포함한다.
<jsp:forward>
요청된 JSP 대신 다른 자원의 내용 또는 수행 결과를 대신 클라이언트로 응답한다.
<jsp:plugin>
JSP 의 수행 결과 안에 Applet 을 수행시키고 결과를 포함한다.
<jsp:useBean>
주어진 JavaBeans 클래스의 객체를 생성하거나 이미 생성된 객체를 추출한다.
<jsp:getProperty>
JavaBeans 객체의 프로퍼티 값을 추출한다.
<jsp:setProperty>
JavaBeans 객체의 프로퍼티에 값을 설정한다.

<jsp:forward page="{relativeURL |'${' Expression'}' | <%= expression %>}" />

<jsp:forward page="{relativeURL | '${' Expression'}' | <%= expression %>}" { >
[<jsp:param name="parameterName" value="{parameterValue | '${' Expression '}' | <%= expression %>}" }
/> ]+ </jsp:forward>}

<jsp:include page="{relativeURL | '${' Expression'}' | <%= expression %>}" [ flush="true|false" ] />

<jsp:useBean id="name" scope="page|request|session|application" class="className" />

<jsp:getProperty name="name" property="property">

<jsp:setProperty name="beanName" prop_expr />
prop_expr ::= property="*" |
property="propertyName" |
property="propertyName" param="parameterName" |

property=”propertyName” value=”propertyValue”
propertyValue ::= string

▣ JSP의 내장 객체

JSP는 표현식(expression)태그와 스크립트릿(scriptlets)태그에서 스크립트 코드를 심플하게 작성할 수 있게 내장 객체라는 것을 지원하고 있다. 내장 객체를 선언하고 초기화 하는 것은 JSP 컨테이너가 JSP 소스를 Servlet 소스 코드로 변환하는 과정에서 자동적으로 추가한다.

JSP의 스크립트 태그에서는 Java에서 제공하는 표준 API를 이용해서 객체를 생성하거나 활용할 수 있다. 또한 개발자가 만든 클래스도 JSP 객체 생성하여 사용할 수 있다. JSP 에서 이런 표준 API 및 클래스를 활용하는 방법은, 지시자를 사용해서 사용할 패키지를 import하고 해당 객체를 생성 혹은 참조하는 프로그램을 작성하는 것이다. JSP 에서 사용되는 객체의 클래스는 반드시 패키지화 되어야 한다. 내장 객체는 내부적으로 정의된 객체이므로 이런 과정을 필요로 하지 않으며 사용하고자 하는 객체의 내장 객체 변수만 알고 있으면 된다.

| 객체변수 | 클래스 및 인터페이스 | 설 명 |
|---|---|---|
| request | http.HttpServletRequest | 클라이언트에서 전송되는 다양한 요청 데이터 추출 |
| response | http.HttpServletResponse | 응답 시 필요한 기능을 제공 |
| pageContext | jsp.PageContext | 페이지가 처리되는 시점에서의 외부 환경 데이터 추출 |
| session | http.HttpSession | 클라이언트 별로 생성되는 HttpSession 객체 |
| application | ServletContext | application scope 객체 생성과 관리 |
| config | ServletConfig | Servlet 구성 데이터 추출 |
| out | jsp.JspWriter | 응답용 출력 스트림 |
| page | jsp.HttpJspPage | 페이지의 Servlet 인스턴스 |
| exception | java.lang.Throwable | 생성된 예외 객체를 참조 |

▣ EL(Expression Language)

특정 스코프 영역에 보관되어 있는 객체를 추출하여 이 객체의 값 또는 속성 값을 추출하여 표현하고 하는 경우 사용된다. 적절한 Java 코드와 함께 표현식 태그를 사용해도 되지만 JSP 가 추가로 지원하는 Expression Language 라는 구문으로 좀 더 간단하게 구현하는 것이 가능하다.

EL 은 $ 와 블록({ })으로 구현하는 것으로 표현하는 것과 관련된 연산자와 EL 만의 내장 객체를 사용할 수 있다. Query 문자열을 추출하여 표현하는 경우도 다음과 같이 스크립팅 태그를 사용하는 것보다 간단하게 구현한다.

```
<% out.println(request.getParameter("q")); %>

<%= request.getParameter("q") %>

${param.q} 또는 ${param["q"]}
```

- EL(Expression Language)의 내장 객체

pageContext - PageContext 객체
pageScope - page 스코프에 포함된 객체들
requestScope - request 스코프에 포함된 객체들
sessionScope - session 스코프에 포함된 객체들
applicationScope - application 스코프에 포함된 객체들
param - HTTP의 파라메터들
paramValues - 한 파라메터의 값들
header - 헤더 정보들
headerValues - 한 헤더의 값들
cookie – 쿠키들
initParam - 컨텍스트의 초기화 파라미터들

HttpSession 객체에 cart 라는 명칭으로 저장된 객체의 getApple() 을 호출하여 리턴된 결과를 표현하려면 다음과 같이 구현한다.

```
${ sessionScope.cart.apple } 또는 ${ cart.apple }
```

- EL 에서의 . 연산자

**변수명.xxx**

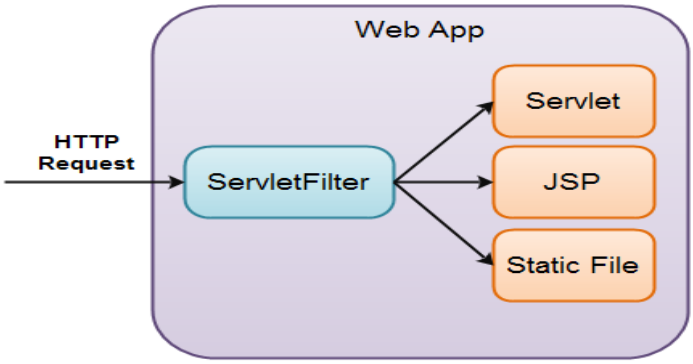
(1) 변수의 참조 대상이 일반 Java 객체이면 **getXxx()** 를 호출한 결과가 된다.

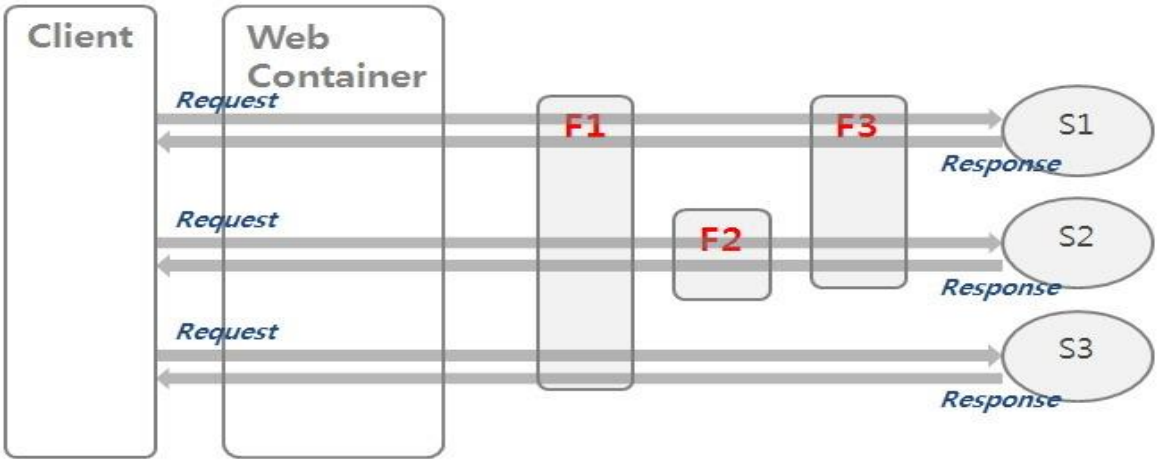
(2) 변수의 참조 대상이 Map 객체이면 **get("xxx")** 을 호출한 결과가 된다.

## ▣ Filter

Filter 란 웹 클라이언트에서 요청한 웹 자원들(Servlet 또는 JSP)이 수행되기 전 또는 후에 수행되는 객체로서 request 또는 response에 영향을 주거나 또는 특정 처리를 할 수 있다. Filter 의 응용 예로 인증, 로깅, 이미지 변환, 데이터 압축, 암호화, 스트림 토큰화, XML 변환 등이 있다.

웹 자원이 순서대로 하나 또는 두 개 이상의 Filter 들의 chain 에 의해 필터링 되도록 설정 할 수 있다.



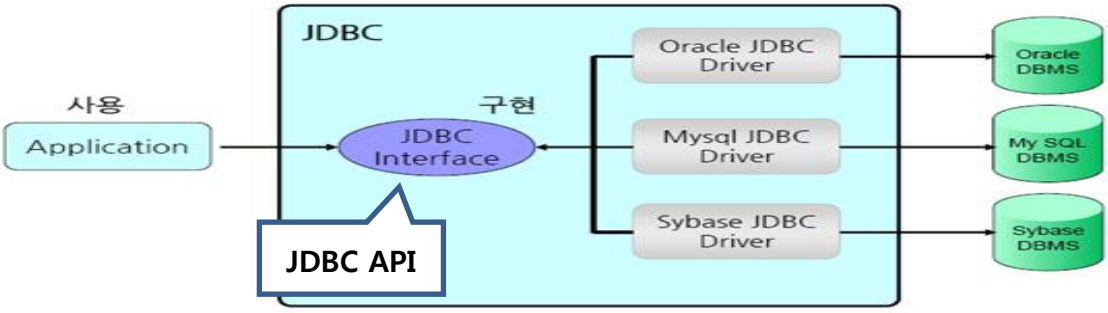




Filter 구현 시에는 javax.servlet.Filter 라는 인터페이스를 상속하여 init(), doFilter(), destroy() 를 오버라이딩 한다.

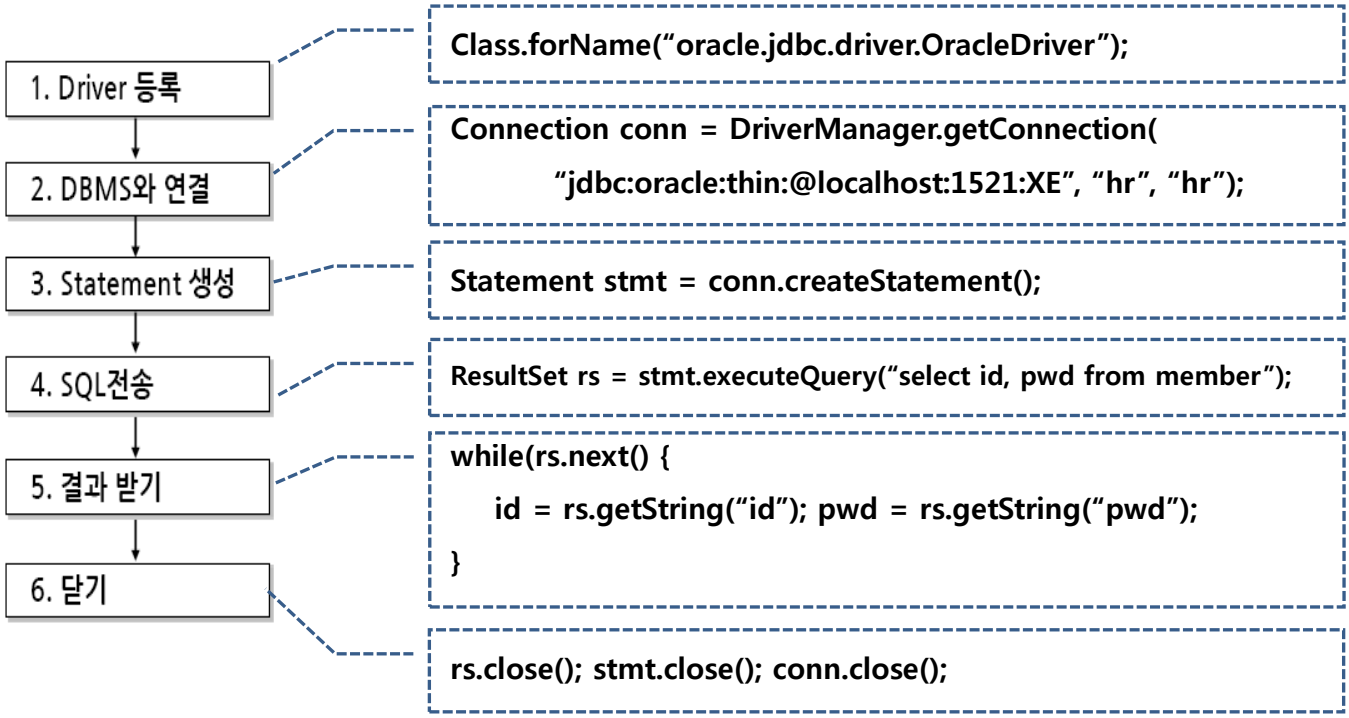
```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
                                                    throws IOException, ServletException {
    // 웹 자원의 수행 전에 처리할 기능
    chain.doFilter(req, res);
    // 웹 자원의 수행 후에 처리할 기능
}
```

## JDBC

JDBC(Java Database Connectivity)는 Java에서 데이터베이스에 접속하여 기능을 처리할 수 있도록 지원하는 Java API이다. JDBC는 데이터베이스에서 자료를 쿼리하거나 업데이트하는 방법을 제공한다. JDBC API 는 java.sql, javax.sql 에서 Java의 표준 API 로 제공되지만 JDBC 드라이버는 연동하려는 DB 서버에 알맞은 것을 추가로 준비해야 한다.



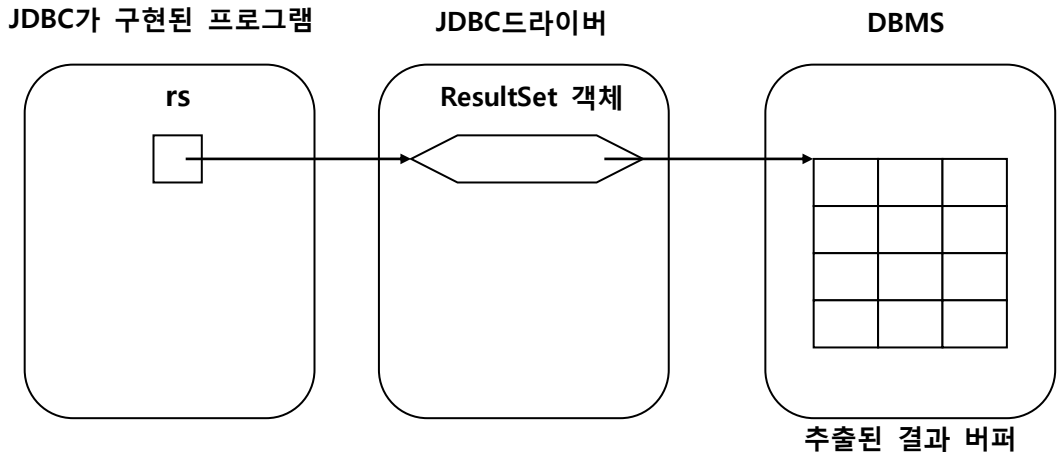

▣ JDBC 프로그래밍 절차



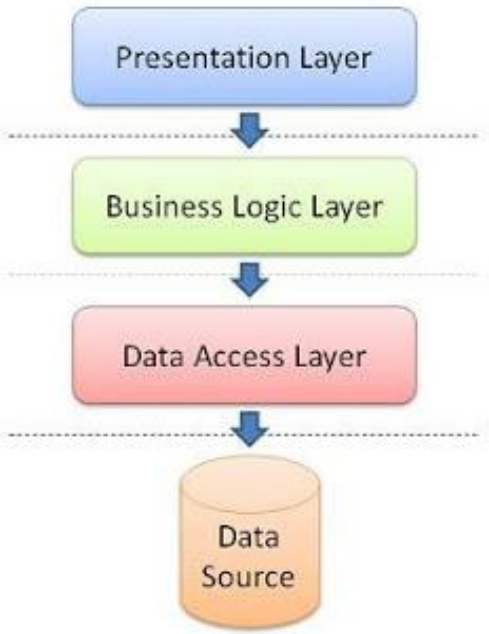

- Statement 와 PreparedStatement

```
Statement stmt = conn.createStatement();
stmt.executeUpdate("insert into account values('"+name+"','"+ passwd+"')");

PreparedStatement pstmt = conn.prepareStatement(
                                "insert into account values (?, ?)");
pstmt.setString(1, name);
pstmt.setString(2, passwd);
pstmt.executeUpdate();
```

- SELECT 수행 결과를 처리하는 ResultSet

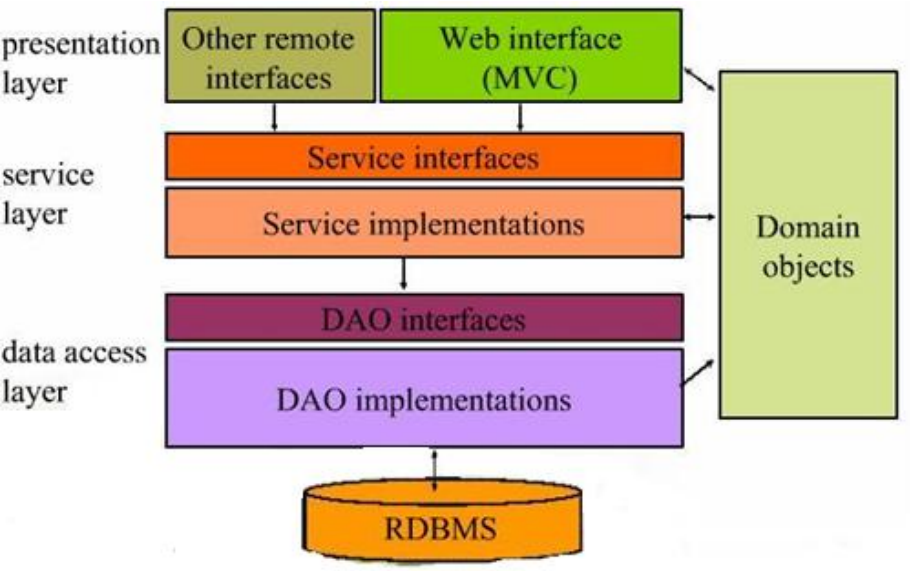



- DAL(Data Access Layer)의 구성




DB 연동 기능을 전담하는 객체들이 모여있는 계층으로 응용 프로그램과 DB 사이에 존재하여 DB 에 대한 세부 정보를 노출하지 않고 DB 연동에 대한 모든 역할을 처리한다.

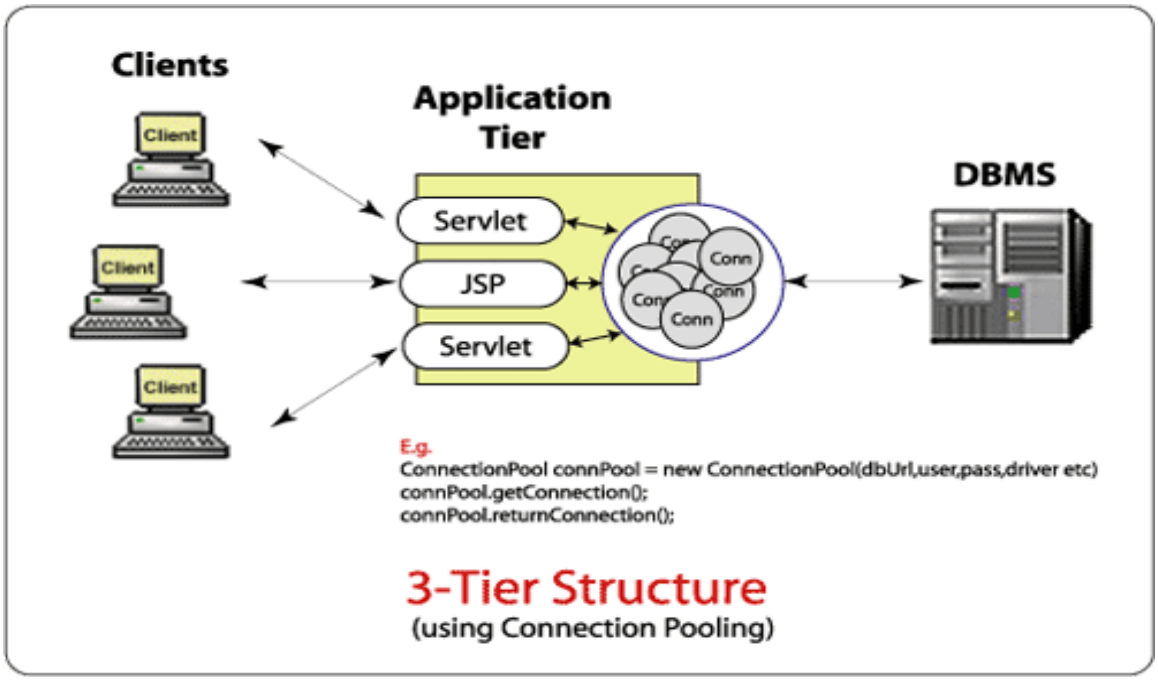
Data Access Layer 의 기능을 구현하는데 있어서 주로 DAO(Data Access Object)라고 불리우는 객체를 순수하게 JDBC 기술을 사용하여 구현할 수도 있으며 JPA, Spring JDBC, iBatis 그리고 Hibernate 와 같은 프레임워크 기술을 사용할 수도 있다.



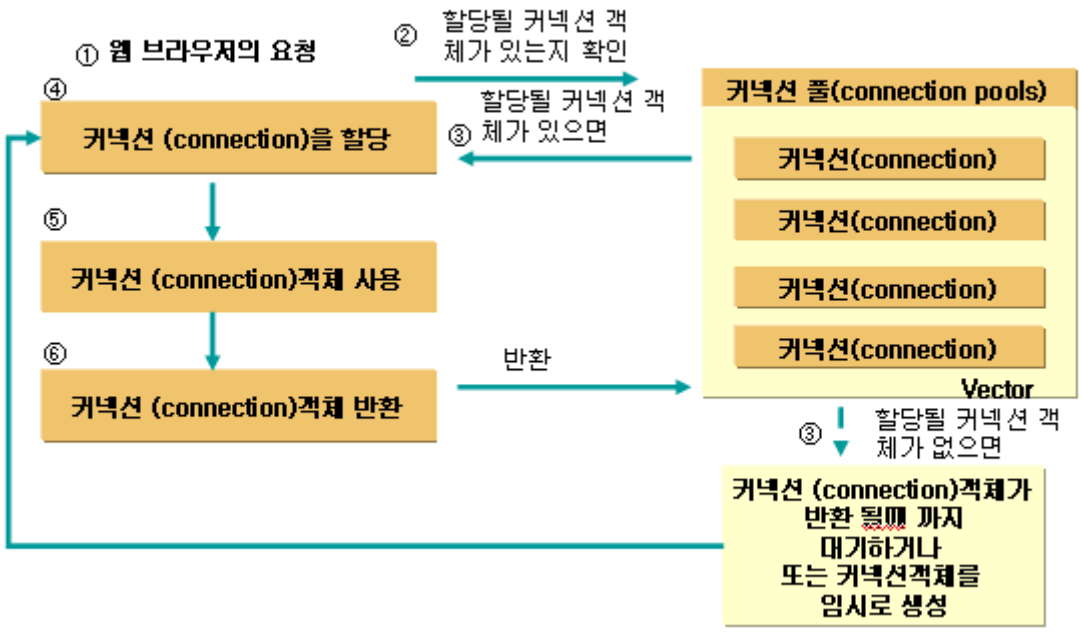

▣ Connection Pool

Connection Pool 이란 데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool)이라는 저장소에 저장해 두고 필요할 때에 커넥션을 풀에서 가져다 쓰고 다시 풀에 반환하는 구현 방법이다. 클라이언트로부터 요청이 올 때마다

DB 서버에 접속하고 Connection 객체를 생성하는 부분에서 발생하는 대기 시간을 줄여서 효율적으로 DB 연동을 수행할 수 있도록 지원하는 기술이므로 웹 서버 프로그램 구현 시 필수적으로 사용되는 기술이다.

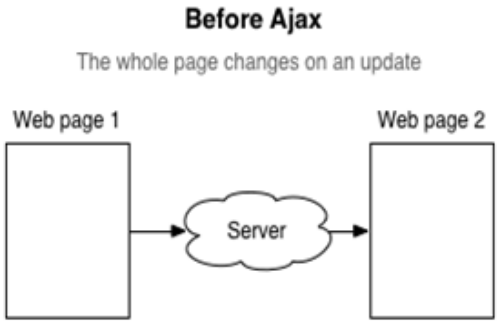


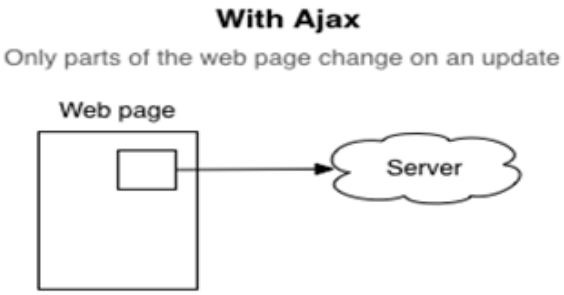
[ Connection Pool 의 처리 과정 ]




# AJAX

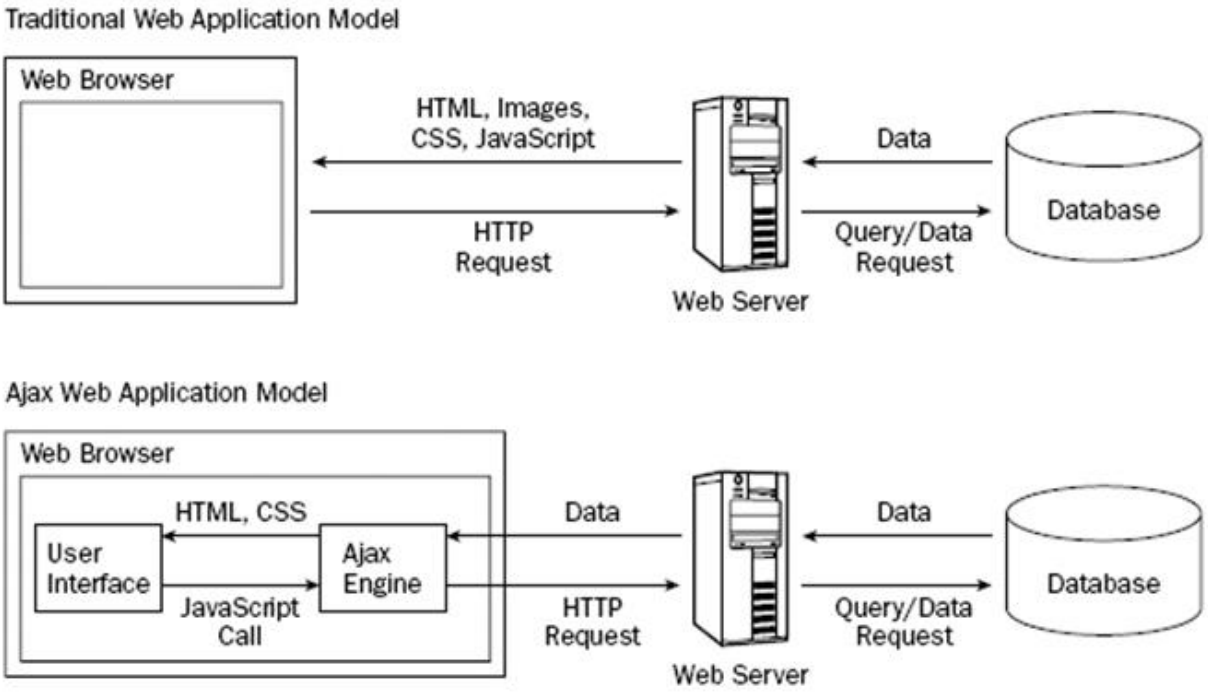
- AJAX = Asynchronous JavaScript and XML
  - 고전적인 웹의 통신 방법은 웹페이지의 일부분을 갱신하기 위해서는 페이지 전체를 다시 로드 해야 했다.
  - AJAX 의 핵심은 재로드(refresh 재갱신) 하지 않고 웹페이지의 일부만을 갱신하여 웹서버와 데이터를 교환하는 방법이다. 즉, 빠르게 동적 웹페이지를 생성하는 기술이다.



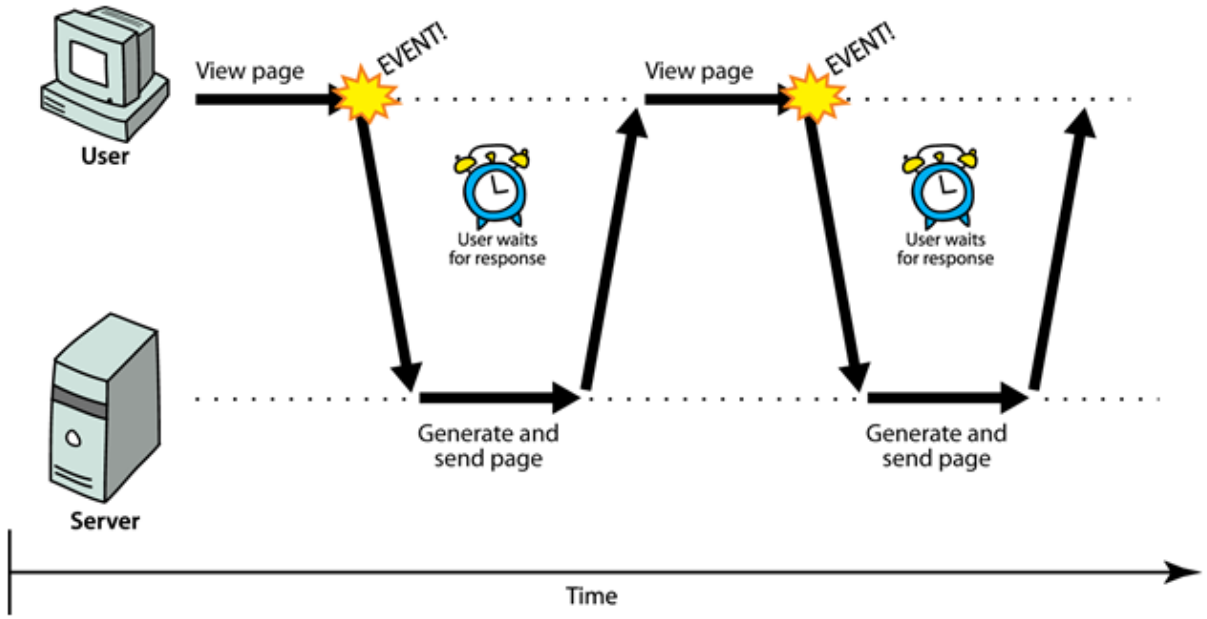

# AJAX

- 고전적 웹 통신과 AJAX 웹 통신



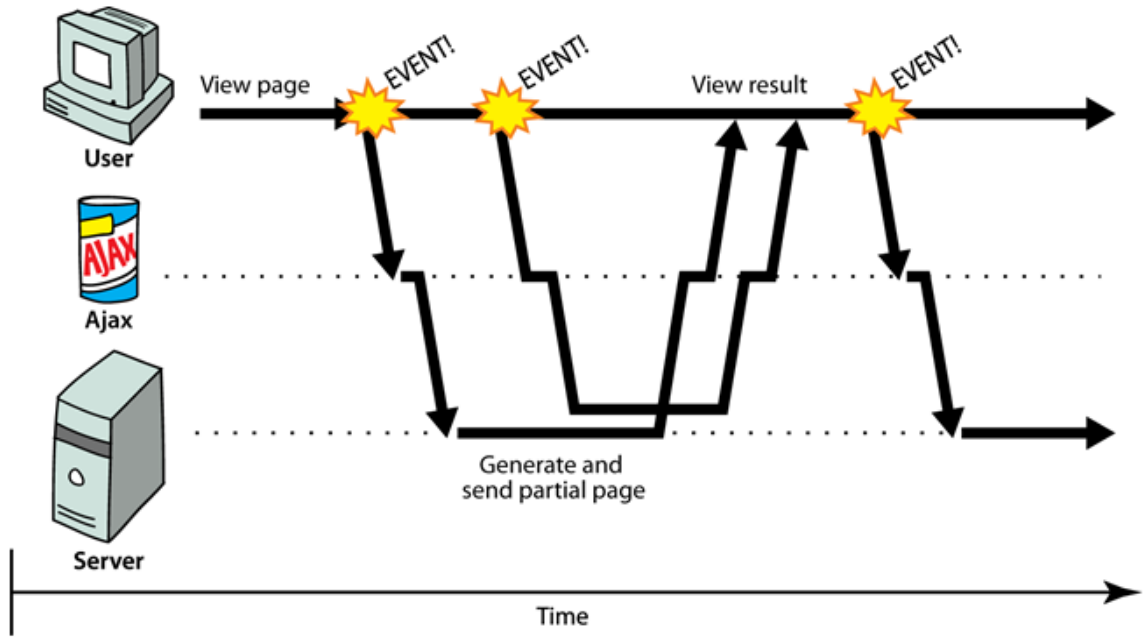

# AJAX

- 고전적 웹 통신(동기)



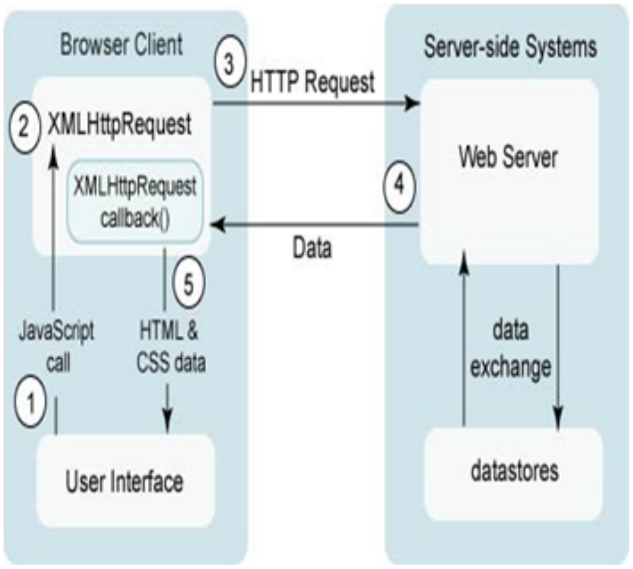
# AJAX

- AJAX 웹 통신(비동기)




# AJAX

- AJAX의 동작과정




① 이벤트 발생에 의해 이벤트핸들러 역할의 JavaScript 함수를 호출한다.

② 핸들러 함수에서 XMLHttpRequest 객체를 생성한다. 요청이 종료되었을 때 처리할 기능을 콜백함수로 만들어 등록한다.

③ XMLHttpRequest 객체를 통해 서버에 요청을 보낸다.

④ 요청을 받은 서버는 요청 결과를 적당한 데이터로 구성하여 응답한다.

⑤ XMLHttpRequest 객체에 의해 등록된 콜백함수를 호출하여 응답 결과를 현재 웹 페이지에 반영한다.

# AJAX

- **XMLHttpRequest 객체**
  - 서버 측과의 비동기 통신을 제어하는 것은 XMLHttpRequest 객체의 역할이다.
  - XMLHttpRequest 객체를 이용함으로써 지금까지 브라우저가 실행해 온 서버와의 통신 부분을 JavaScript가 제어할 수 있게 된다.
  - XMLHttpRequest 객체 생성 : **new XMLHttpRequest()**

| 분류 | 멤버 | 개요 |
|---|---|---|
| 프로퍼티 | **onreadystatechange** | 통신상태가변화된타이밍에호출되는이벤트핸들러 |
| | readyState | HTTP 통신상태를취득 |
| | status | HTTP Status코드를취득 |
| | statusText | HTTP Status의상세메시지를취득 |
| | responseText | 응답본체를 plaintext로취득 |
| | responseXML | 응답본체를 XML(XMLDocument 객체)로취득 |
| 메서드 | abort() | 현재의비동기통신을중단 |
| | getAllResponseHeaders() | 수신한모든 HTTP 응답헤더를취득 |
| | getResponseHeader(header) | 지정한 HTTP 응답헤더를취득 |
| | **open( ... )** | HTTP 요청을초기화 |
| | setRequestHeader(header, value) | 요청 시에송신하는헤더를추가 |
| | **send([body])** | HTTP 요청을송신(인수 body는요청본체) |

# AJAX

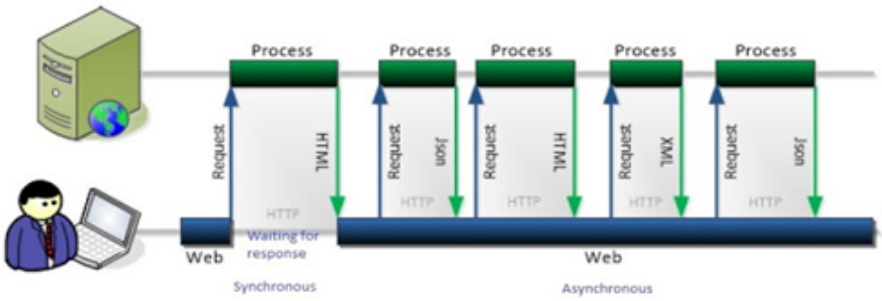
- **readyState 값**

| 반환값 | 개요 |
|---|---|
| 0 | 미초기화(open 메서드가호출되지않음) |
| 1 | 로드 중(open 메서드는호출됐지만, send 메서드는호출되지않았다) |
| 2 | 로드 완료(send 메서드는호출됐지만, 응답스테이터스/헤더는 미취득) |
| 3 | 일부 응답을 취득(응답스테이터스/헤더만취득, 본체는미취득) |
| 4 | 모든 응답데이터를 취득 완료 |

- **XMLHttpRequest 객체에서 제공되는 이벤트 관련 속성**
  - **onloadstart**
  - **onprogress**
  - **onabort**
  - **onerror**
  - **onload**
  - **ontimeout**
  - **onloadend**
  - **onreadystatechange**

# AJAX

- open()와 send() 메서드
  - open(HTTP 메서드, URL [, 비동기 모드 통신 여부])
    - HTTP 메서드 : 요청 방식(GET, POST, PUT, DELETE..)
    - URL : AJAX 로 요청하려는 서버의 대상 페이지
    - 비동기 모드 통신 여부 : true(비동기통신), false(동기통신)
  - send([요청 파라미터])
    - POST 의 경우 Query 문자열을 인수로 지정
    - ArrayBufferView, Blob, Document, DOMString, FormData, null 이 올 수 있다.




# AJAX

- Same Origin Policy(SOP)
  - 브라우저에서 보안상의 이슈로 동일 사이트의 자원(Resource)만 접근해야 한다는 제약이다.
  - AJAX는 이 제약에 영향을 받으므로 Origin 서버가 아니면 AJAX 로 요청한 컨텐츠를 수신할 수 없다.

- Cross Origin Resource Sharing(CORS)
  - 초기에는 Cross Domain이라고 하였다.(동일 도메인에서 포트만 다른 경우, 로컬 파일인 경우 등으로 인해 Origin이라는 으로 용어 통일됨)
  - Origin 이 아닌 다른 사이트의 자원을 접근하여 사용한다는 의미이다.
  - Open API 의 활성화와 공공 DB 의 활용에 의해서 CORS 의 중요성이 강조되고 있다.
  - HTTP Header에 CORS 와 관련된 항목을 추가한다.

```
response.addHeader("Access-Control-Allow-Origin", "*");
```

# jQuery 의 AJAX 지원 API

| Ajax 메소드 | 기능/예 |
|---|---|
| $.ajax( ) | 모든 Ajax 메소드의 기본이 되는 메소드<br>예) $.ajax({<br>url: 'service.php',<br>success: function(data) {<br>$('#area').html(data);<br>}<br>});</br> |
| $.get( ) | GET 방식의 ajax( ) 메소드<br>예) $.get('sample.html', function(data) {<br>$('#area').html(data);<br>});</br> |
| $.post( ) | POST 방식의 ajax( ) 메소드<br>예) $.post('sample.html', function(data) {<br>$('#area').html(data);<br>});</br> |
| $.getJSON( ) | JSON 형식으로 응답 받는 ajax( ) 메소드<br>예) $.getJSON('sample.json', function(data) {<br>$('#area').html(' ' + data.age + ' ');<br>});</br> |
| load( ) | 서버로부터 데이터를 받아서 일치하는 요소 안에 HTML을 추가<br>예) $('#area').load('sample.html', function( ) {<br>;<br>});</br> |
| $.getScript( ) | 자바스크립트 형식으로 응답 받는 ajax( ) 메소드<br>예) $.getScript('sample.js', function( ) {<br>;<br>});</br> |
| $.ajaxSetup( ) | ajax( ) 메소드의 선택 사항들에 대한 기본값 설정<br>예) $.ajaxSetup({<br>url: 'service.php'<br>});</br> |

# jQuery 의 AJAX 지원 API

| Ajax 메소드 | 기능 |
|---|---|
| ajaxStart( ) | 첫 번째 Ajax 요청이 시작될 때 호출되는 이벤트 메소드<br>예) $('#img1').ajaxStart(function( ){<br>$(this).show();<br>});</br> |
| ajaxStop( ) | 모든 Ajax 요청이 끝날 때 호출되는 이벤트 메소드<br>예) $('#img1').ajaxStop(function( ){<br>$(this).fadeOut(2000);<br>});</br> |
| ajaxSend( ) | 특정 Ajax 요청을 보내기 전에 호출되는 이벤트 메소드<br>예) $("#msg").ajaxSend(function(event, request, settings){<br>$(this).append(" " + settings.url + "페이지 요청 시작 ");<br>});</br> |
| ajaxSuccess( ) | 특정 Ajax 요청이 성공적으로 완료될 때마다 호출되는 이벤트 메소드<br>예) $("#msg").ajaxSuccess(function(event, request, settings){<br>$(this).append(" 요청 성공 ");<br>});</br> |
| ajaxError( ) | Ajax 요청들에 대한 오류 발생시 호출되는 이벤트 메소드<br>예) $("#msg").ajaxError(function(event, request, settings){<br>$(this).append(" " + settings.url + "페이지 요청 실패 ");<br>});</br> |
| ajaxComplete( ) | Ajax 요청들이 완료되면(성공/실패 관련 없이) 호출되는 이벤트 메소드<br>예) $("#msg").ajaxComplete(function(event,request, settings){<br>$(this).append(" 요청 완료 ");<br>});</br> |

# jQuery 의 AJAX 지원 API

- $.ajax( ) 메서드
  - 모든 AJAX 메서드가 내부적으로는 사용하는 기본 메서드
  - AJAX 요청을 기본적인 부분부터 직접 설정하고 제어할 수 있어 다른 AJAX 메서드로 할 수 없는 요청도 수행 가능
  - $.ajax() 메서드의 기본 형식

```
$.ajax( options ) ;
```

```
$.ajax({ url: URL주소 [,type: 요청방식] [,data: 요청내용] [,timeout: 응답제한시간] [,dataType: 응답데이터유형
] [,async: 비동기여부] [,success: 성공콜백함수] [,error: 실패콜백함수] });
```

- $.ajax( ) 메서드 선택 항목들(options)을 맵 형식으로 명세

| 선택항목 : 항목값 | 의미 |
|---|---|
| url : URL 주소 | 요청이 보내질(주로 서버)의 URL 주소(필수 항목, 기본값: 현재페이지)<br>예) "sample.php", "sample.html", "sample.xml" |
| type : 요청방식 | 요청을 위해 사용할 HTTP 메소드<br>예) "get"(기본값), "post" |
| data : 요청내용 | 서버로 전달되는 요청 내용(제이쿼리 객체맵이나 문자열) |
| timeout : 응답제한시간 | 요청 응답 제한 시간(밀리초)<br>예) 20000 |
| dataType : 응답데이터유형 | (서버로부터의) 반환될 응답 데이터의 형식<br>예) "xml", "html", "json", "jsonp", "script", "text" |
| Async : 논리값 | 요청이 비동기식으로 처리되는지 여부(기본값: true) |
| success : function(data) | 요청 성공 콜백함수(data: 서버 반환 값) |
| error : function( ) | 요청 실패 콜백함수 |

# jQuery 의 AJAX 지원 API

- $.getJSON( ) 메서드
  - GET 요청 방식으로 서버로부터 JSON 형식의 데이터를 요청

```
$.getJSON( url [, data] [,function(data)] ) ;
```

  - $.getJSON 메서드 입력인자

| 인자 | 의미 |
|---|---|
| url | 요청이 보내질(주로 서버)의 URL 주소(필수 항목, 기본값: 현재 페이지)<br>예) "sample.json" |
| data | 서버로 전달되는 요청 내용(제이쿼리 객체 맵이나 문자열) |
| function(data) | 요청 성공 콜백 함수 (data: 서버 반환 값) |

# jQuery 의 AJAX 지원 API

- $.load( ) 메서드
  - 서버로부터 데이터를 받아오는 가장 간단한 메서드로 많이 이용
  - 서버로부터 데이터를 받아 메서드를 실행하는 대상 엘리먼트에 직접 추가 -> 복잡한 선택 사항을 설정하지 않고도 빠르고 간단하게 웹 페이지의 동적 갱신이 가능
  - 요청이 성공하면 메서드가 실행되는 대상 엘리먼트 내용이 서버에서 응답 받은 HTML5 마크업 데이터로 대체

```
$.load( url [, data] [,function(data)] ) ;
```

  - $.load( ) 메서드 선택 항목

| 인자 | 의미 |
|---|---|
| url | 요청이 보내질(주로 서버)의 URL 주소(필수 항목, 기본값: 현재페이지)<br>예) "sample.html" |
| data | 서버로 전달되는 요청 내용(제이쿼리 객체맵이나 문자열) |
| function(data) | 요청 성공 콜백 함수(data: 서버 반환 값) |