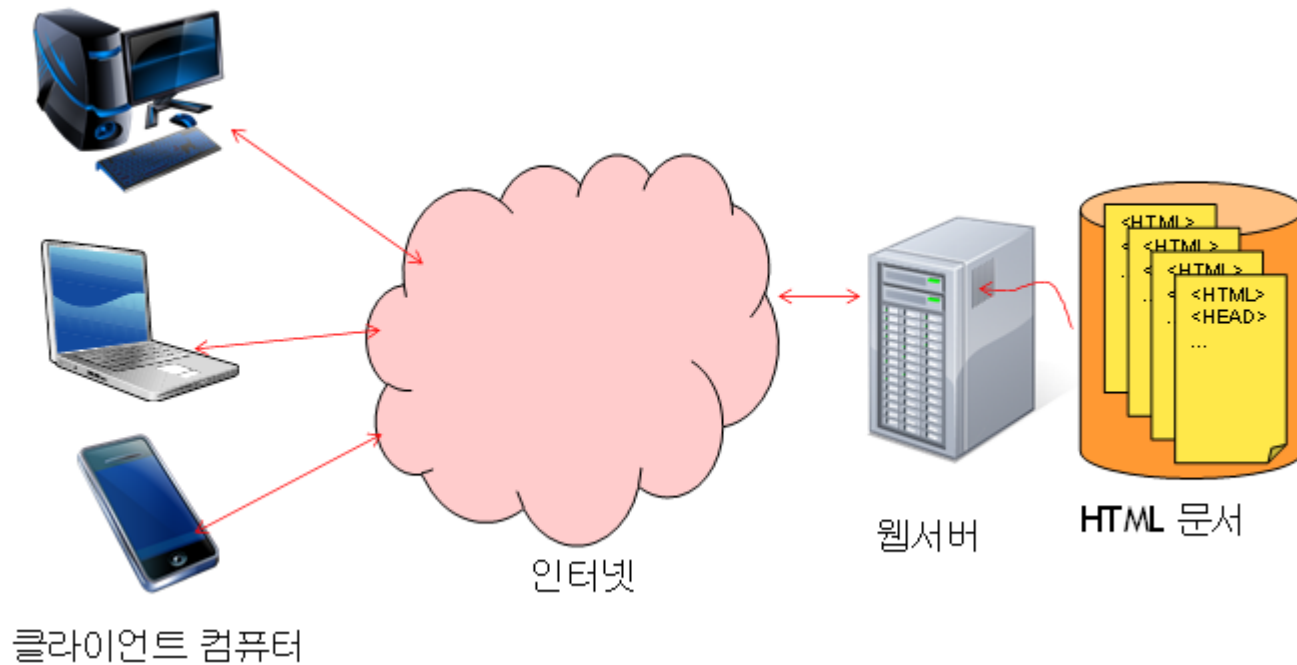


HTTP와 WWW(웹)

- WWW(World Wide Web): 세계를 뒤덮는 거미줄
- 초기 인터넷에서는 텔넷, FTP, 전자 메일, 유즈넷 등의 문자 위주 서비스
- WWW은 인터넷을 사용하기 쉽도록 하이퍼 텍스트와 그림을 통하여 모든 서비스를 이용할 수 있도록 만든 것

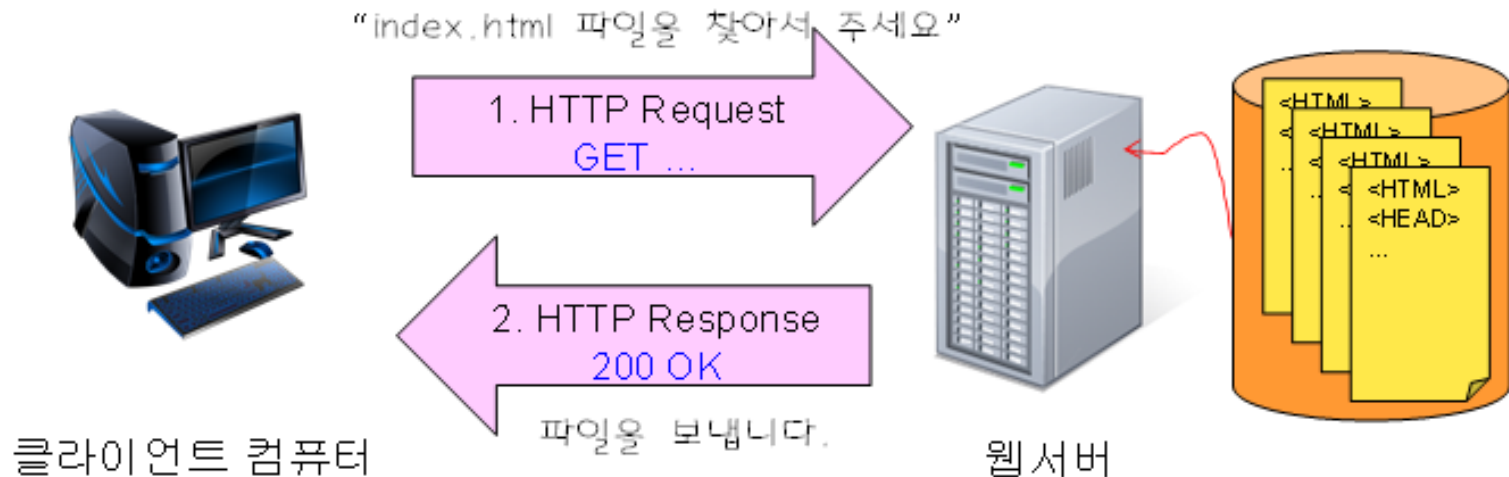


웹의 동작원리



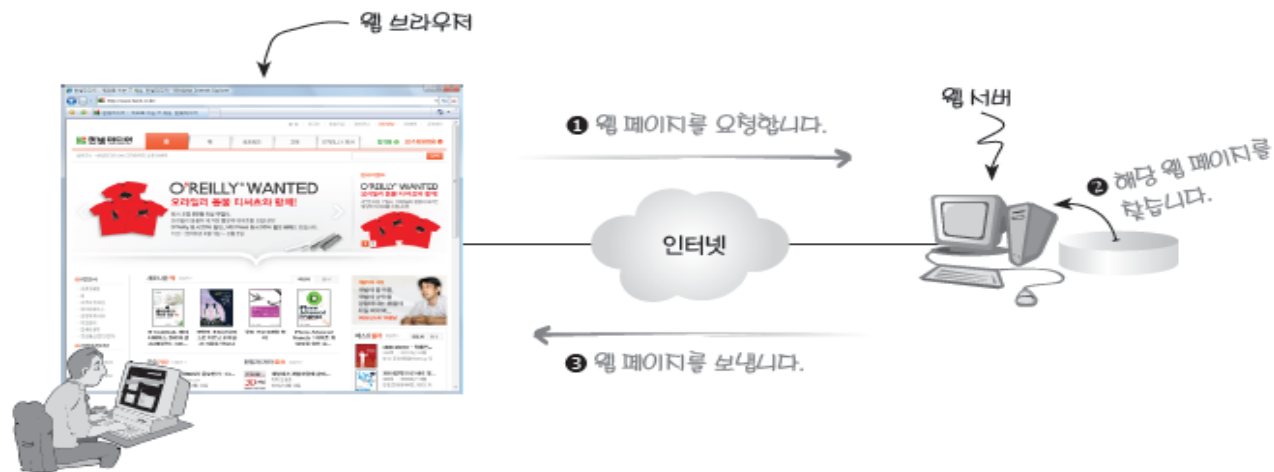
웹 서버와 웹 클라이언트

- 웹 클라이언트는 웹 서버로 파일을 요청하는 컴퓨터
- 웹 서버는 웹 클라이언트의 요청을 받아 처리하고 결과를 응답하는 컴퓨터
- 웹의 기본 프로토콜 : HTTP
 - 특정한 파일을 요청하는 HTTP Request
 - 찾은 파일을 돌려주는 HTTP Response

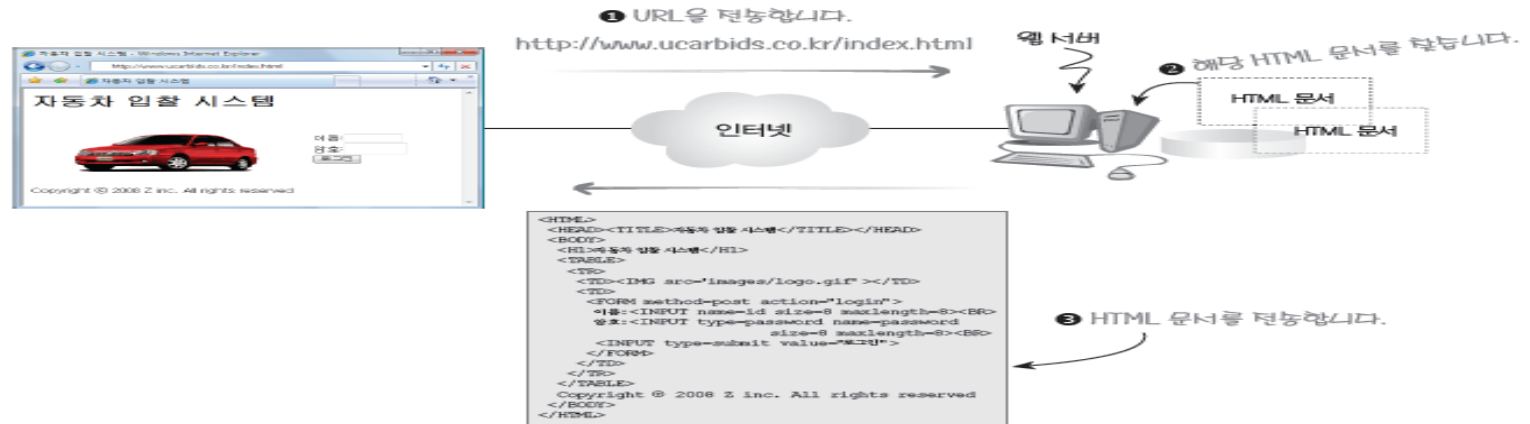


웹 서버와 웹 클라이언트

- 웹 클라이언트는 웹 브라우저를 이용하여 URL 형태로 웹 서버에 요청
- 웹 서버는 웹 브라우저로부터 **URL**을 받아서 그에 해당하는 **HTML** 문서를 찾아서 웹 브라우저로 전송



웹 서버와 웹 클라이언트



HTML 문서 형태로 전송되는 웹 페이지

- HTML문서는 순수하게 텍스트로만 이루어지며, **<HTML>**, **</HTML>**, **<BODY>**, **</BODY>**, **<H1>**, **</H1>**과 같이 꺾쇠괄호로 묶여진 부분을 태그(tag) 또는 마크업(markup)이라고 한다.
- 태그는 웹 브라우저 상에 그대로 표시되는 것이 아니라 그 밖의 부분이 웹 브라우저 상에 어떻게 표시될지 지시하는 역할을 한다.

웹 브라우저

- HTML 문서를 읽어서 눈에 보이는 웹 페이지를 만든다.

웹브라우저는 **HTML** 문서를 해석하여
화면에 웹페이지를 표시한다.

```
<!DOCTYPE html>
<html>
<head>
  <title>...</title>
</head>
<body>
  ...
</body>
</html>
```

HTML 문서



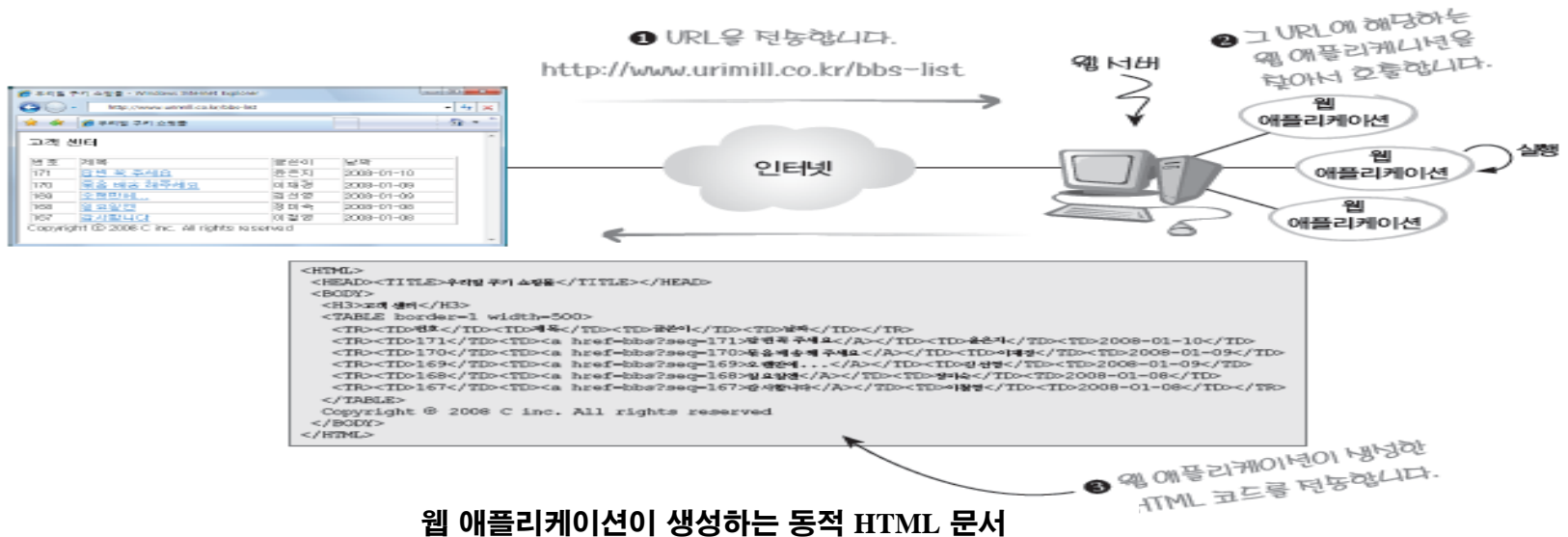
화면

웹 어플리케이션

- 웹 서버는 **HTML** 문서 파일을 찾아서 보내주거나 **HTML** 문서를 생성하는 프로그램을 호출, 실행 결과를 보내주는 일도 한다.
- 이 때 호출 되는 프로그램을 “웹 어플리케이션”이라고 한다.
- 웹 어플리케이션이 생성하는 **HTML** 문서를 동적 **HTML** 문서라 한다.
- 웹 서버 쪽에 파일 형태로 저장되어 있는 **HTML**문서를 정적 **HTML** 문서라고 한다.

웹 어플리케이션

- 웹 애플리케이션은 CGI, PHP, ASP, ASP#NET, SERVLET, JSP 등의 기술을 이용.
- SERVLET, JSP는 자바 기반의 웹 어플리케이션 기술



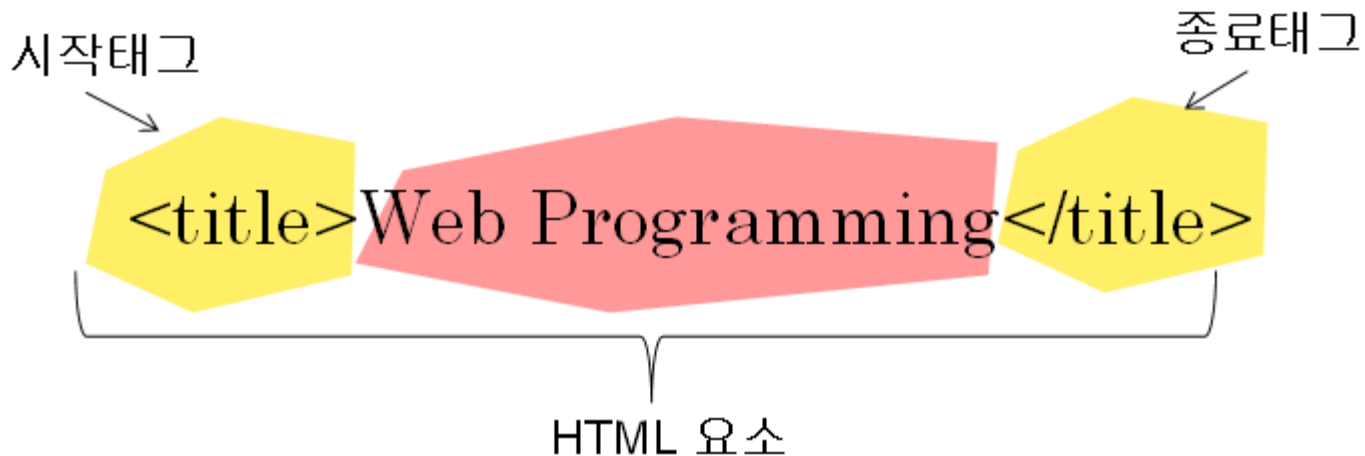
HTML5+CSS3+Javascript

- 웹 페이지의 내용은 HTML5로 작성
- 웹 페이지의 스타일은 CSS3로 지정
- 웹 페이지의 상호작용은 자바스크립트로 작성



HTML과 HTTP

- HTML(Hyper Text Markup Language)은 웹 페이지를 기술하기 위한 마크업(markup) 언어
- 마크업 언어는 텍스트에 태그를 붙여서 텍스트가 문서의 어디에 해당하는지를 기술한 것



HTML 버전

Version	Year
HTML	1991
HTML+	1993
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML 1.0	2000
HTML5	2012
XHTML5	2013

HTML5

- HTML5는 HTML의 새로운 표준
 - 완전한 CSS3 지원
 - 비디오와 오디오 지원
 - 2D/3D 그래픽 지원
 - 로컬 저장소 지원
 - 로컬 SQL 데이터베이스 지원
 - 웹 애플리케이션 지원



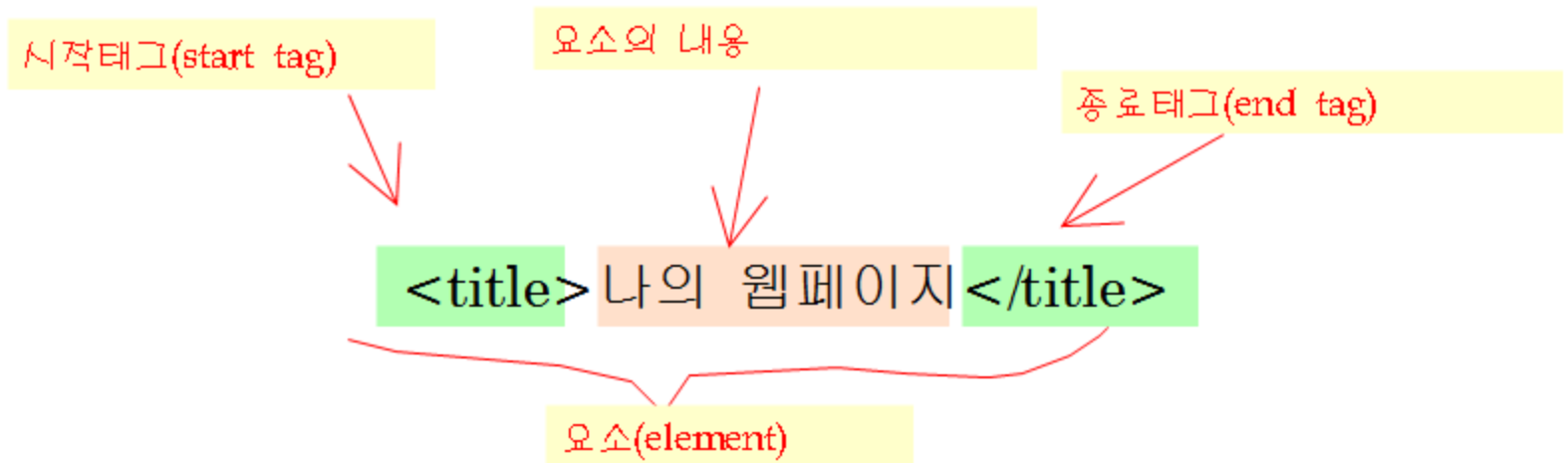
HTML 문서의 기본 구조

```
<!DOCTYPE html>
<html>
|
|   <head>
|   <title>나의 웹 페이지</title>
|   </head>
|
|   <body>
|   <p>Hello Web Programming World!</p>
|   </body>
|
</html>
```



요소(element)

- 시작태그와 종료태그로 이루어진 문서의 구성 요소
- 요소 = (시작 태그 + 콘텐츠 + 종료 태그)



속성

- 속성은 요소에 대한 추가적인 정보를 제공
- 속성은 항상 시작태그에 이름="값" 형태로 기술된다.

속성의 이름

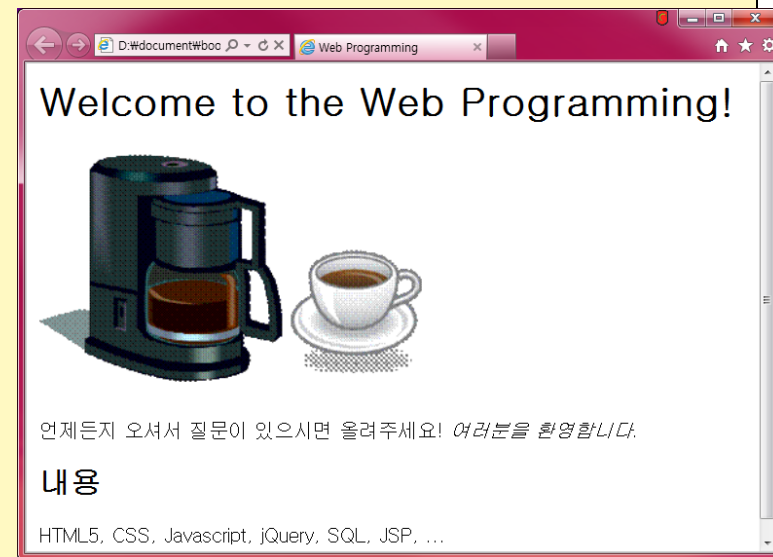
속성의 값

`W3C컨소시엄`

속성(attribute)

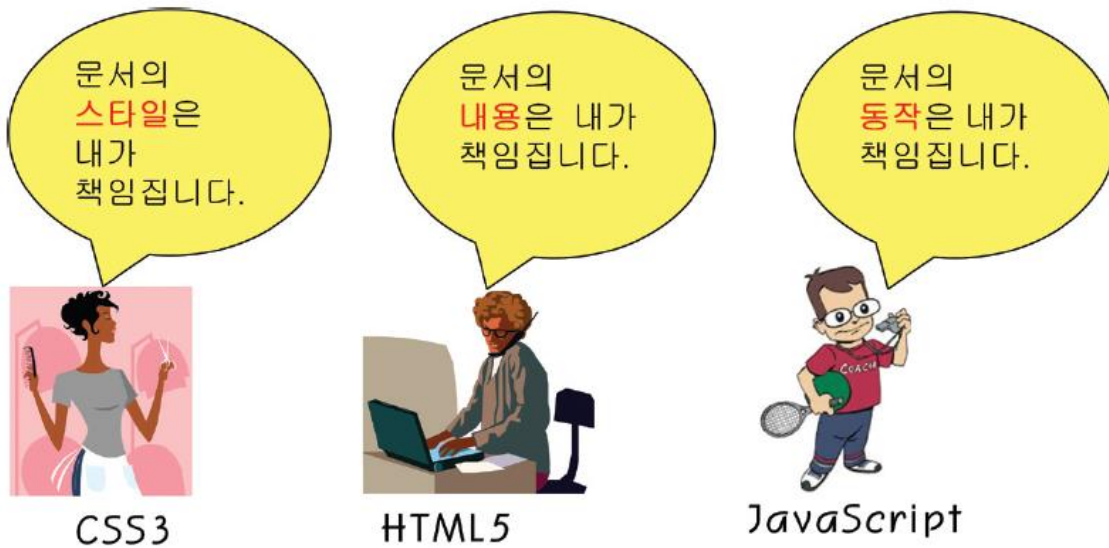
HTML 맛보기

```
<html>
<head>
  <title>Web Programming</title>
</head>
<body>
  <h1>Welcome to the Web Programming!</h1>
  
  <p>
    언제든지 오셔서 질문이 있으시면 올려주세요!
    <em>여러분을 환영합니다</em>.
  </p>
  <h2>내용</h2>
  <p>
    HTML5, CSS, Javascript, jQuery, SQL, JSP, ...
  </p>
</body>
</html>
```



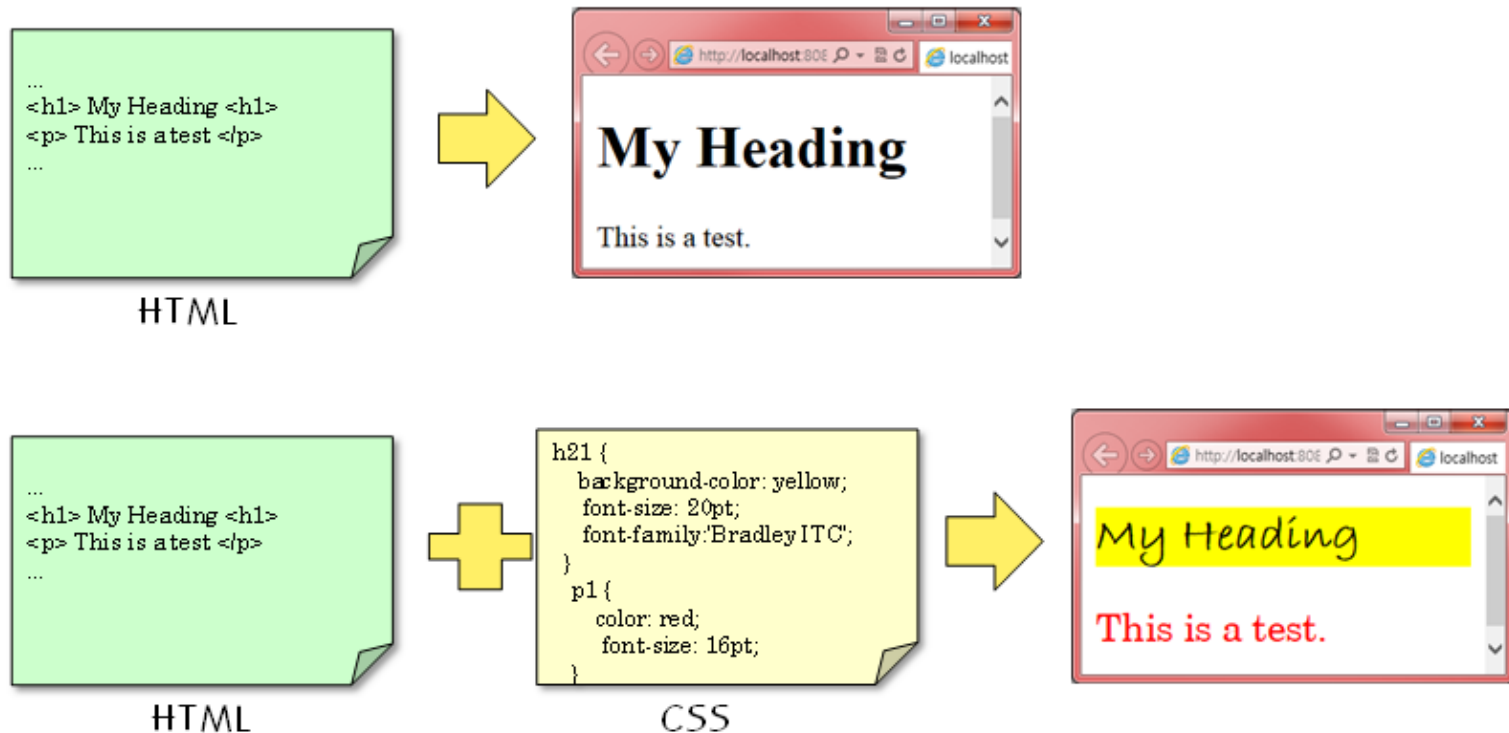
CSS

- 문서의 구조-> HTML
- 문서의 스타일 -> ?



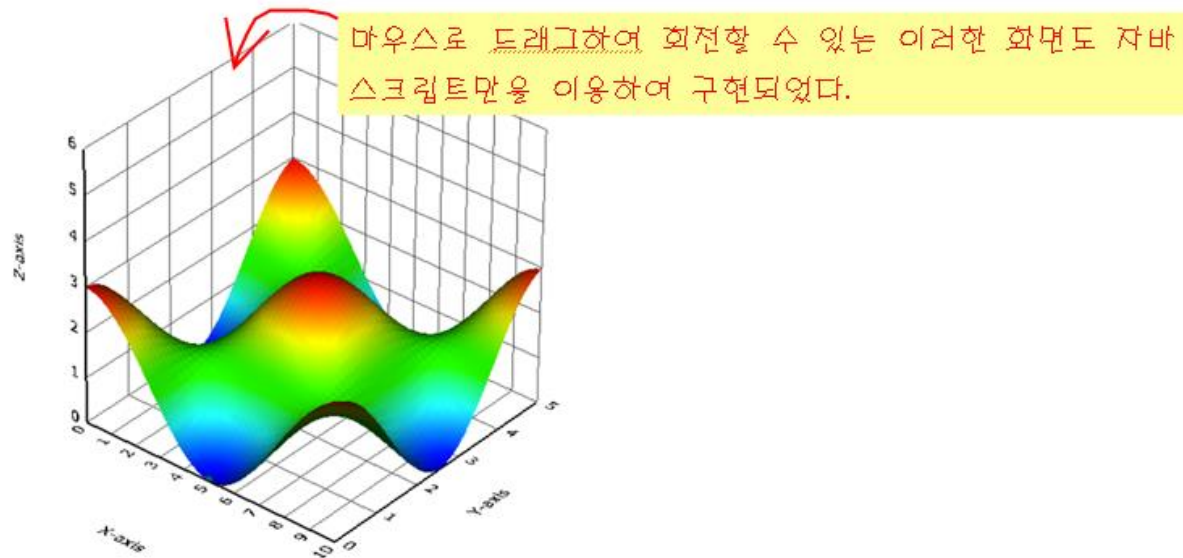
CSS

- CSS(Cascading Style Sheets): 문서의 스타일을 지정한다.



자바 스크립트

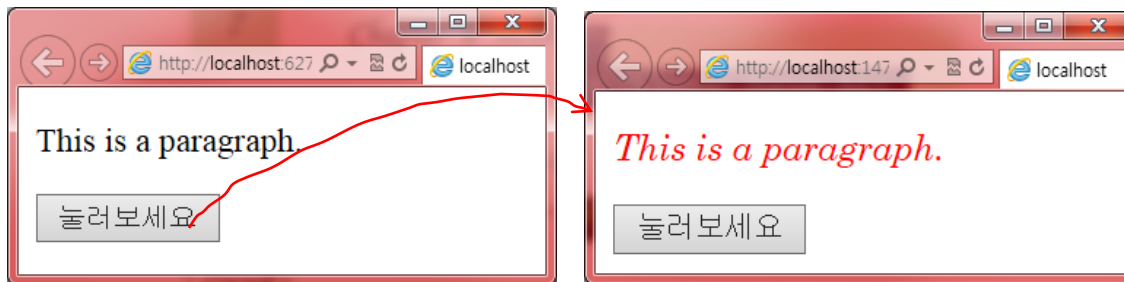
- 자바스크립트(javascript): 동적인 웹 페이지를 작성하기 위하여 사용되는 언어
- 웹의 표준 프로그래밍 언어
- 모든 웹브라우저들은 자바스크립트를 지원



자바스크립트

```
<!DOCTYPE html>
<html>

<body>
<p id="p1">This is a paragraph.</p>
<script>
  function changeStyle() {
    document.getElementById("p1").style.color = "red";
    document.getElementById("p1").style.fontFamily = "Century Schoolbook";
    document.getElementById("p1").style.fontStyle = "italic";
  }
</script>
  <input type="button" onclick="changeStyle()" value="눌러보세요" />
</body>
</html>
```



jQuery

- **jQuery**- 일종의 자바 스크립트 라이브러리
- jQuery를 사용하면 자바 스크립트 프로그래밍의 양을 상당히 줄일 수 있다.
- jQuery는 배우기 쉽다.
- 무료이다.

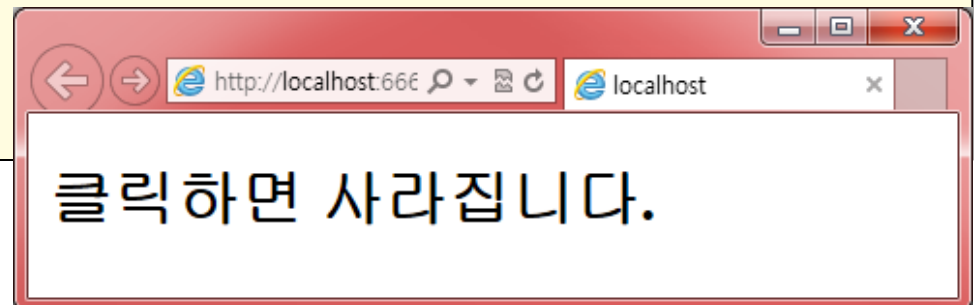


자바스크립트	jQuery
<code>document.getElementById("myPara").innerHTML = "안녕하세요?"</code>	<code>\$("#myPara").html("안녕하세요?");</code>

id가 myPara인 요소를 찾아서 내용을 변경한다.

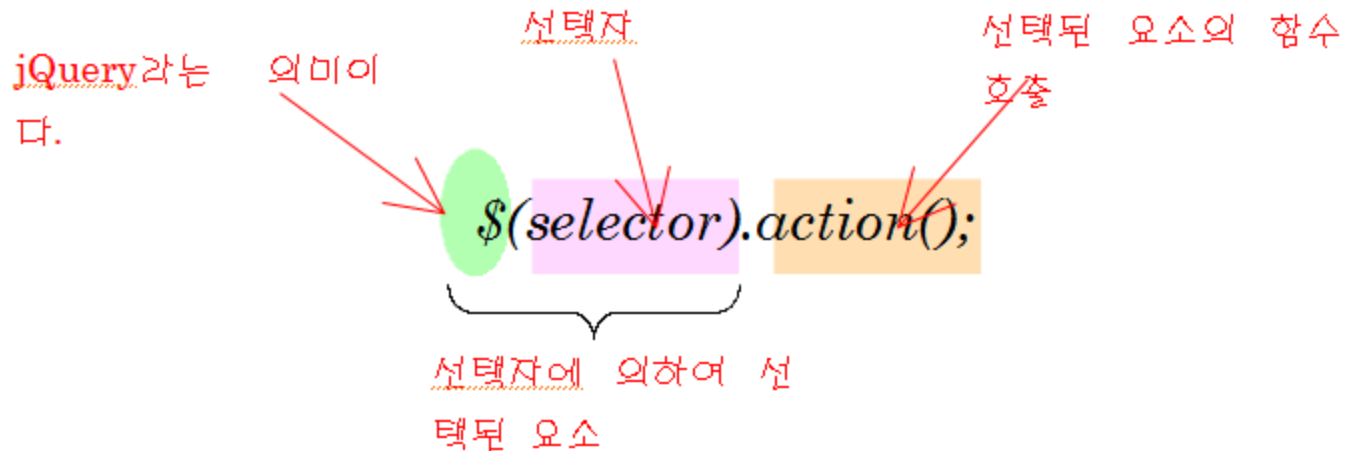
jQuery

```
<!DOCTYPE html>
<html>
<head>
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
</script>
<script>
    $(document).ready(function () {
        $("h2").click(function () {
            $(this).hide();
        });
    });
</script>
</head>
<body>
<h2>클릭하면 사라집니다.</h2>
</body>
</html>
```



jQuery 문장의 구조

- \$(...) 안에 선택자를 넣어서 원하는 요소를 선택하고, 선택된 요소에 대하여 여러 가지 조작을 한다.



- `$("p").show()` - 모든 `<p>` 요소들을 찾아서 화면에 표시한다.
- `$(".group1").slideup()` - `class=group1`인 요소를 슬라이드업 방식으로 표시한다.
- `$("#id9").hide()` - `id=id9`인 요소를 화면에서 감춘다.

SERVLET

- 자바 클래스 기반의 웹 어플리케이션 기술
- 계산 결과를 웹 브라우저로 출력하는 코드

```
public class HundredServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        int total = 0;  
        for (int cnt = 1; cnt < 101; cnt++)  
            total += cnt;  
        PrintWriter out = response.getWriter();  
        out.println( "<HTML> ");  
        out.println( "<HEAD><TITLE>Hundred Servlet</TITLE></HEAD> ");  
        out.println( "<BODY> ");  
        out.printf( "1 + 2 + 3 + ... + 100 = %d ", total);  
        out.println( "</BODY> ");  
        out.println( "</HTML> ");  
    }  
}
```

계산 결과를 HTML로 만들어서
웹 브라우저로 출력하는 명령문

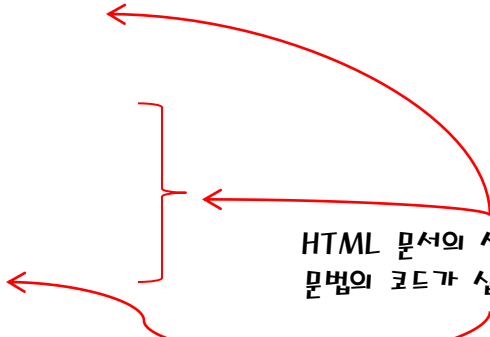
JSP

- 자바 기반의 웹 애플리케이션을 구현 기술.
- **JSP** 페이지는 **HTML** 문서의 사이에 **JSP** 문법의 코드가 삽입되는 형태로 작성된다.

```
<%@page contentType= "text/html; charset=euc-kr"%>
<HTML>
  <HEAD><TITLE>1부터 100까지의 합</TITLE></HEAD>
  <BODY>
    <%
      int total = 0;
      for (int cnt = 1; cnt <= 100; cnt++)
        total += cnt;

      %>
    1부터 100까지 더한 값은? <%= total %>
  </BODY>
</HTML>
```

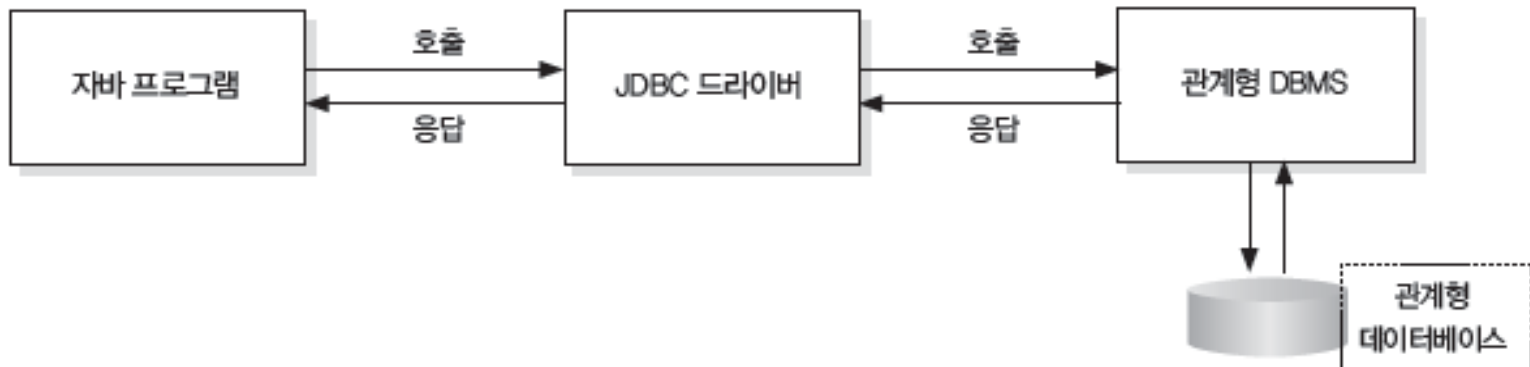
HTML 문서의 사이사이에 JSP 문법의 코드가 삽입됩니다



- **JSP** 페이지에 있는 **HTML** 코드는 웹 브라우저로 그대로 전송되지만, **JSP** 문법의 코드는 웹 어플리케이션 서버 쪽에서 실행되고 그 결과만 웹 브라우저로 전송된다.

JDBC

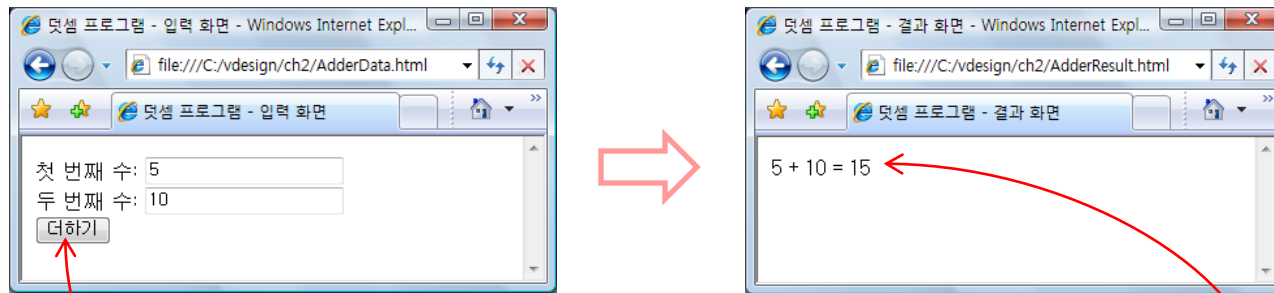
- 데이터베이스(**database**)는 파일과 마찬가지로 보조기억장치에 데이터를 저장하는 수단이다.
- **JDBC**는 자바의 관계형 데이터베이스 이용 기술
- 자바 프로그램에서 **관계형 데이터베이스**를 사용하려면 **JDBC** 드라이버 필요.



HTTP 요청과 응답

❖ 웹 브라우저로부터 데이터를 입력받는 서블릿 클래스

- 왼쪽 웹 페이지를 통해 두 수를 입력받은 후 그 둘을 합한 결과를 오른쪽 웹 페이지를 통해 보여주는 웹 애플리케이션이다.



[그림 2-21] 두 수의 합을 구하는 웹 애플리케이션의 화면 설계

① 두 수를 입력하고
더하기 버튼을 누르면

② 두 수의 합을 보여주는
웹 페이지가 나타난다.

- 둘 이상의 웹 페이지로 구성되는 웹 애플리케이션을 개발할 때는 먼저 화면 설계를 하고 다음에 각 화면의 **URL**을 정하고, 코딩 작업에 들어가는 것이 좋다.

HTTP 요청과 응답

❖ 웹 브라우저로부터 데이터를 입력받는 서블릿 클래스

- 각 화면의 **URL**은 다음과 같이 정한다.

`http://localhost:8080/brain/AdderInput.html`

← (그림 2-21) 왼쪽 화면 URL

`http://localhost:8080/brain/adder`

← (그림 2-21) 오른쪽 화면 URL

- 왼쪽 화면은 **<FORM>** 엘리먼트를 사용해서 구현한다.

[예제2-3] 두 개의 수를 입력받는 HTML 문서

```
<HTML>
  <HEAD>
    <META http-equiv= "Content-Type" content= "text/html; charset=euc-kr" >
    <TITLE>덧셈 프로그램 - 입력 화면</TITLE>
  </HEAD>
  <BODY>
    <FORM ACTION=/brain/adder>
      첫 번째 수: <INPUT TYPE=TEXT NAME=NUM1><BR>
      두 번째 수: <INPUT TYPE=TEXT NAME=NUM2><BR>
      <INPUT TYPE=SUBMIT VALUE= '더하기' >
    </FORM>
  </BODY>
</HTML>
```

HTTP 요청과 응답

- ❖ 웹 브라우저로부터 데이터를 입력받는 서블릿 클래스
 - 오른쪽 화면을 구현하는 서블릿 클래스는 [예제 2-3]을 통해 입력된 두 수를 받아서 합을 계산한 후 **HTML** 문서로 만들어서 출력해야 한다.
 - **<FORM>** 엘리먼트를 통해 입력된 데이터는 **doGet, doPost** 메서드의 첫 번째 파라미터인 **HttpServletRequest** 타입의 파라미터에 대해 **getParameter** 메서드를 호출해서 가져올 수 있다.
 - 각 **<INPUT>** 서브엘리먼트를 통해 입력된 데이터를 가져오기 위해서는 다음과 같은 메서드를 호출해야 한다.

```
String str = request.getParameter( "NUM1 ");
```

<INPUT> 엘리먼트의 **NAME** 애트리뷰트 값

- 이 메서드가 리턴하는 값은 수치 타입이 아니라 문자열 타입이다.

HTTP 요청과 응답

❖ 웹 브라우저로부터 데이터를 입력받는 서블릿 클래스

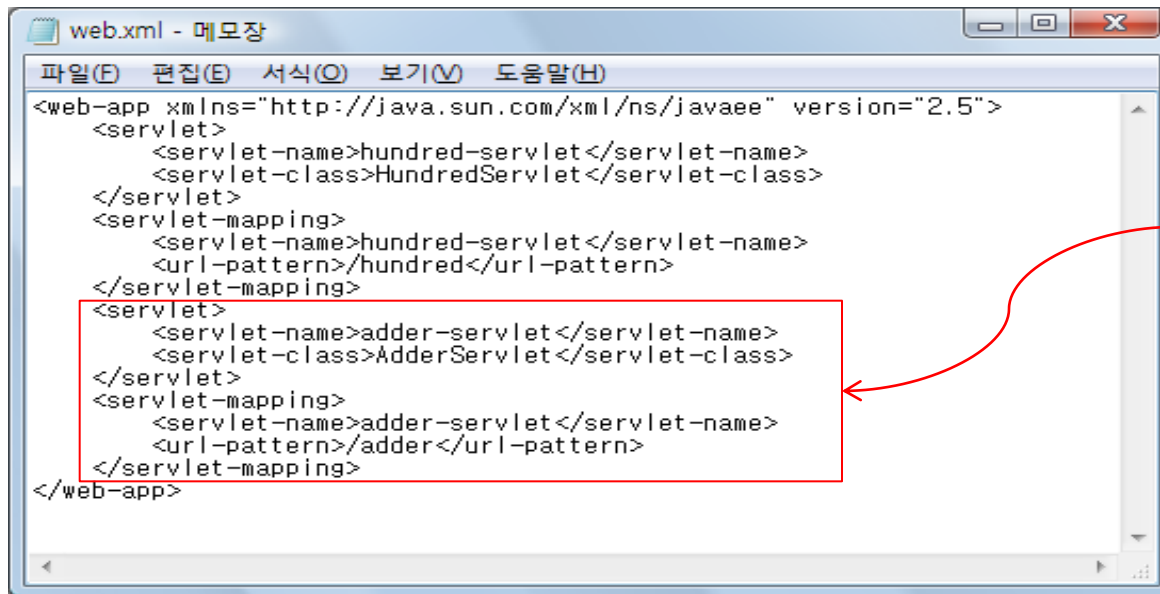
[예제2-4] 두 수의 합을 구하는 서블릿 클래스

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class AdderServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        String str1 = request.getParameter( "NUM1 " );
        String str2 = request.getParameter( "NUM2 " );
        int num1 = Integer.parseInt(str1);
        int num2 = Integer.parseInt(str2);
        int sum = num1 + num2;
        response.setContentType( "text/html;charset=euc-kr " );
        PrintWriter out = response.getWriter();
        out.println( "<HTML> " );
        out.println( "<HEAD><TITLE>덧셈 프로그램 - 결과 화면</TITLE></HEAD> " );
        out.println( "<BODY> " );
        out.printf( "%d + %d = %d ", num1, num2, sum );
        out.println( "</BODY> " );
        out.println( "</HTML> " );
    }
}
```

HTTP 요청과 응답

- ❖ 웹 브라우저로부터 데이터를 입력받는 서블릿 클래스
 - 서블릿 클래스 컴파일 후 그 결과를 **brain** 웹 애플리케이션 디렉터리의 **WEB-INF\classes** 서브디렉터리에 저장한다.
 - **WEB-INF** 디렉터리에 있는 **web.xml** 파일을 열어서 다음과 같이 서블릿 클래스를 등록한다.

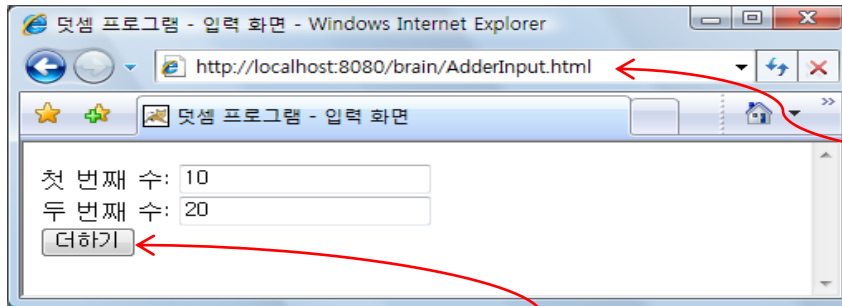


(예제2-4) 서블릿 클래스를
등록하는 코드

HTTP 요청과 응답

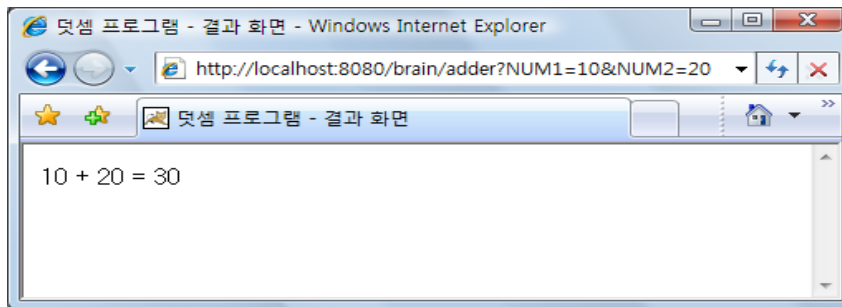
❖ 웹 브라우저로부터 데이터를 입력받는 서블릿 클래스

■ 두 수의 합을 구하는 웹 애플리케이션의 실행 방법은 다음과 같다.



① (예제 2-3)의 URL을 입력한다.

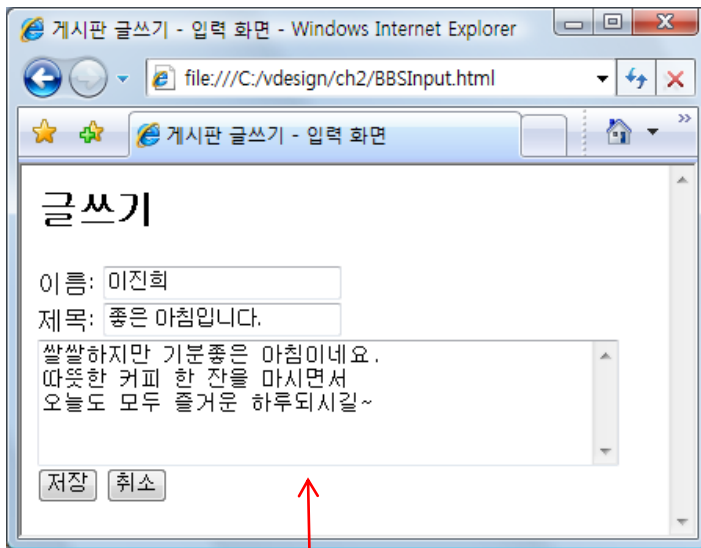
② 두 수를 입력하고 더하기 버튼을 누르면 결과 화면이 나온다.



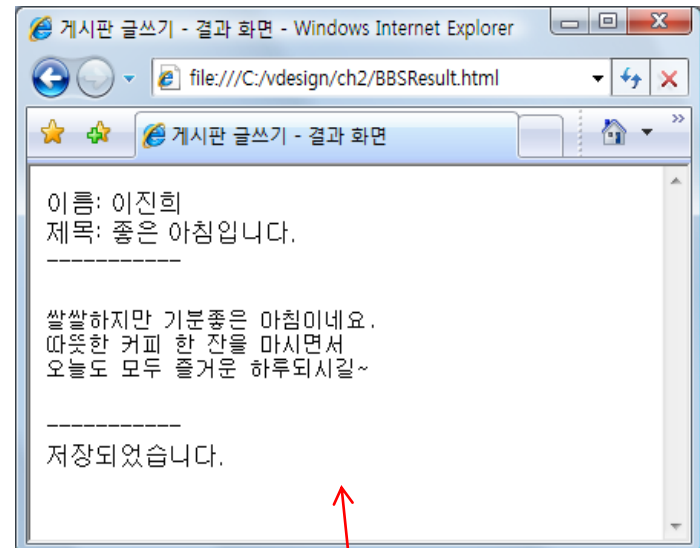
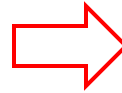
HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

- 웹 페이지를 통해 입력 받은 데이터를 웹 서버 쪽에 저장한 후에 또 다른 웹 페이지를 통해 저장된 결과를 보여주는 웹 애플리케이션이다.



① 이름, 제목, 내용을 입력하고 저장 버튼을 누르면



② 데이터가 웹 서버 쪽에 저장되고 결과 화면이 나타난다.

HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

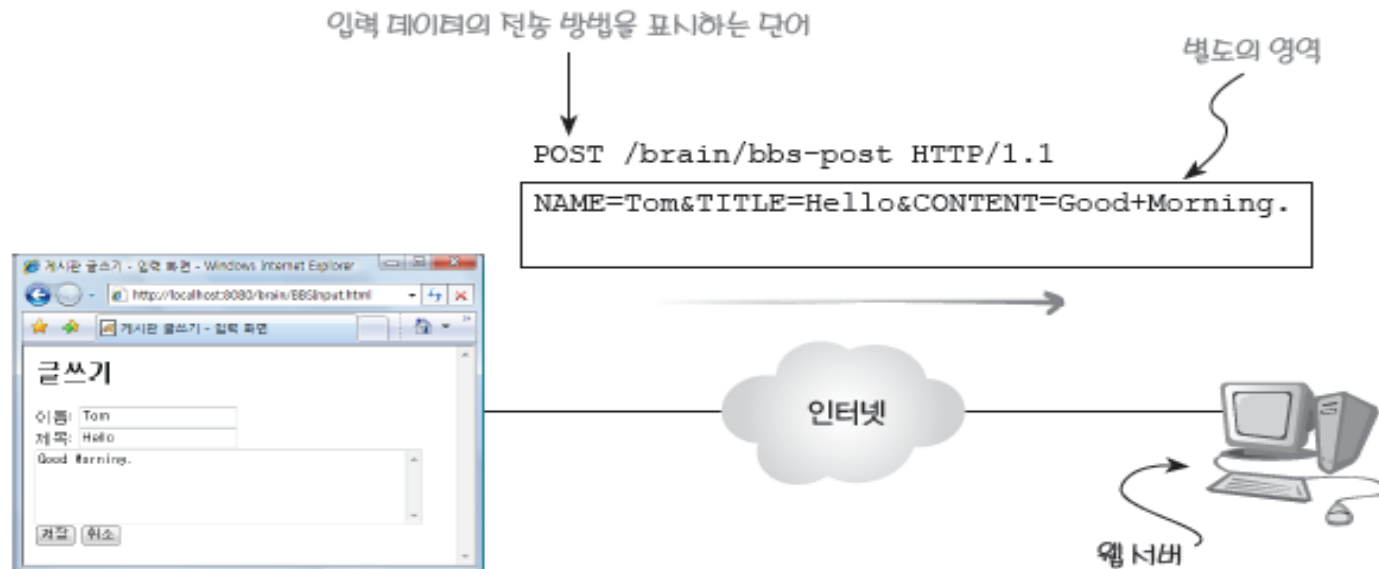
- 입력 데이터가 클 경우에는 **URL** 뒷부분의 데이터가 잘려나갈 수 있으므로 **URL**이 아닌 별도의 영역을 통해 입력 데이터를 전송해야 한다.
- **<FORM>** 엘리먼트의 시작 태그에 **METHOD**라는 애트리뷰트를 추가하고, 애트리뷰트 값으로 **POST**를 입력 데이터가 URL이 아닌 별도의 영역을 통해 전송되도록 만드는 METHOD 애트리뷰트 값

```
<FORM ACTION =/brain/bbs-post METHOD=POST>  
  이름 : <INPUT TYPE=TEXT NAME=WRITER>  
  제목 : <INPUT TYPE=TEXT NAME=TITLE>  
  <TEXTAREA NAME=CONTENT> </TEXTAREA>  
  <INPUT TYPE=SUBMIT VALUE= '저장' >  
  <INPUT TYPE=RESET VALUE= '취소' >  
</FORM>
```

HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

- <FORM> 엘리먼트를 통해 입력된 데이터는 **URL** 다음에 오는 별도의 영역을 통해 전송되며, **URL** 앞에는 **POST**라는 단어가 붙는다. 웹 서버는 이 단어를 보고 입력 데이터가 어디에 있는지 판단할 수 있다.



HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

- URL을 정한 후 URL에 해당하는 HTML 문서를 작성한다.

`http://localhost:8080/brain/BBSInput.html`



(그림 2-24) 왼쪽 화면 URL

`http://localhost:8080/brain/bbs-post`



(그림 2-24) 오른쪽 화면 URL

- [예제2-5] 게시판 글쓰기 기능의 데이터 입력을 위한 HTML 문서

```
<HTML>
  <HEAD>
    <META http-equiv= "Content-Type" content= "text/html; charset=euc-kr" >
    <TITLE>게시판 글쓰기 - 입력 화면</TITLE>
  </HEAD>
  <BODY>
    <H2>글쓰기</H2>
    <FORM ACTION=/brain/bbs-post METHOD=POST>
      이름: <INPUT TYPE=TEXT NAME=NAME><BR>
      제목: <INPUT TYPE=TEXT NAME=TITLE><BR>
      <TEXTAREA COLS=50 ROWS=5 NAME=CONTENT></TEXTAREA><BR>
      <INPUT TYPE=SUBMIT VALUE= '저장' >
      <INPUT TYPE=RESET VALUE= '취소' >
    </FORM>
  </BODY>
</HTML>
```

HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

■ 입력 데이터를 처리하는 서블릿 클래스의 작성 방법

- **doGet** 메서드를 선언하는 대신 **doPost** 메서드를 선언해야 한다. 웹 컨테이너는 **POST**라는 단어가 붙은 **URL**을 받으면 **doGet** 메서드가 아니라 **doPost** 메서드를 호출하기 때문이다
- **doPost** 메서드는 **doGet** 메서드와 마찬가지로 **public** 키워드를 붙여서 선언해야 하고, 파라미터 변수, 리턴 타입, 익셉션 타입도 **doGet** 메서드와 동일하다.

```
public class BBSPostServlet extends HttpServlet {  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
    }  
}
```

doGet 메서드와 리턴 타입, 파라미터 변수,
익셉션 타입이 동일합니다

HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

■ 두 번째 화면을 구현하는 서블릿 클래스

[예제2-6] 게시판 글쓰기 기능을 처리하는 서블릿 클래스 - 미완성

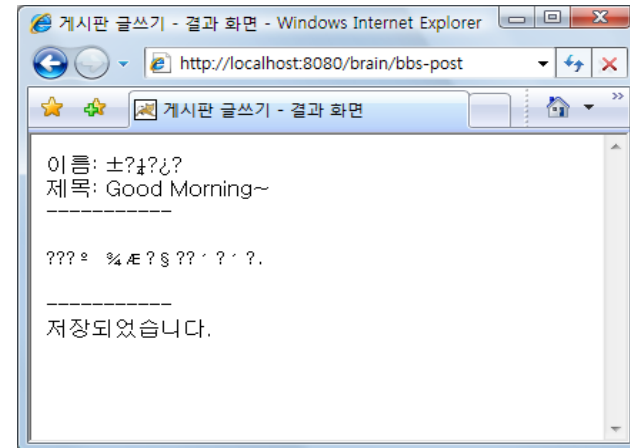
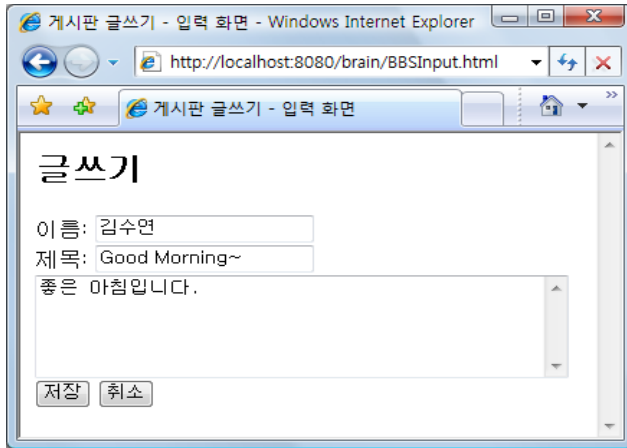
```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class BBSPostServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        String name = request.getParameter( "NAME " );
        String title = request.getParameter( "TITLE " );
        String content = request.getParameter( "CONTENT " );
        response.setContentType( "text/html;charset=euc-kr " );
        PrintWriter out = response.getWriter();
        out.println( "<HTML> " );
        out.println( "<HEAD><TITLE>게시판 글쓰기 - 결과 화면</TITLE></HEAD> " );
        out.println( "<BODY> " );
        out.printf( "이름: %s <BR> ", name );
        out.printf( "제목: %s <BR> ", title );
        out.println( "-----<BR> " );
        out.printf( "<PRE>%s</PRE> ", content );
        out.println( "-----<BR> " );
        out.println( "저장되었습니다. " );
        out.println( "</BODY> " );
        out.println( "</HTML> " );
    }
}
```

HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

- 위 예제는 다음과 같이 한글 데이터의 입력 처리가 제대로 되지 않는다.



- 문제 해결: **doPost** 메서드 안에서 한글 데이터를 올바르게 가져오려면 첫 번째 파라미터인 **HttpServletRequest** 파라미터에 대해 **setCharacterEncoding** 이라는 메서드를 호출해야 한다.

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

↑
한글 코드 이름

HTTP 요청 GET / POST 방식

❖ POST 메서드를 이용한 데이터 전송

- **setCharacterEncoding** 메서드는 **getParameter** 메서드보다 반드시 먼저 호출해야 한다.

[예제2-7] 게시판 글쓰기 기능을 처리하는 서블릿 클래스 - 완성

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class BBSPostServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        request.setCharacterEncoding( "euc-kr ");
        String name = request.getParameter( "NAME " );
        String title = request.getParameter( "TITLE " );
        String content = request.getParameter( "CONTENT " );
        response.setContentType("text/html;charset=euc-kr ");
        PrintWriter out = response.getWriter();
        out.println( "<HTML> ");
        out.println( "<HEAD><TITLE>게시판 글쓰기 - 결과 화면</TITLE></HEAD> ");
        out.println( "<BODY> ");
        out.printf( "이름: %s <BR> ", name);
        out.printf( "제목: %s <BR> ", title);
        out.println( "-----<BR> ");
        out.printf( "<PRE>%s</PRE> ", content);
        out.println( "-----<BR> ");
        out.println( "저장되었습니다. " );
        out.println( "</BODY> ");
        out.println( "</HTML> ");
    }
}
```


JSP 이해

- 자바 기반의 웹 애플리케이션을 구현 기술.
- **JSP** 페이지는 **HTML** 문서의 사이에 **JSP** 문법의 코드가 삽입되는 형태로 작성된다.

```
<%@page contentType= "text/html; charset=euc-kr"%>
<HTML>
  <HEAD><TITLE>1부터 100까지의 합</TITLE></HEAD>
  <BODY>
    <%
      int total = 0;
      for (int cnt = 1; cnt <= 100; cnt++)
        total += cnt;
    %>
    1부터 100까지 더한 값은? <%= total %>
  </BODY>
</HTML>
```

HTML 문서의 사이사이에 JSP 문법의 코드가 삽입됩니다

- **JSP** 페이지에 있는 **HTML** 코드는 웹 브라우저로 그대로 전송되지만, **JSP** 문법의 코드는 웹 어플리케이션 서버 쪽에서 실행되고 그 결과만 웹 브라우저로 전송된다.

JSP의 내장 변수

- **JSP** 페이지의 내장 변수(**implicit variable**) : **JSP** 페이지 안에 선언을 하지 않고도 사용할 수 있는 변수

```
<%@page contentType= "text/html; charset=euc-kr"%>
<HTML>
  <HEAD><TITLE>정수를 순서대로</TITLE></HEAD>
  <BODY>
    <H3>정수를 순서대로</H3>
    <%
      String str = request.getParameter("MAX");
      int max = Integer.parseInt(str);
      for (int cnt = 1; cnt <= max; cnt++)
        out.println(cnt + "<BR>");
    %>
  </BODY>
</HTML>
```

내장 변수

내장 변수

[그림 3-15] JSP 페이지의 내장 변수의 예

- **request** 내장 변수는 서블릿 클래스의 **doGet**, **doPost** 메서드의 첫 번째 파라미터와 동일한 역할을 한다.
- **out** 내장 변수는 서블릿 클래스에서 **getWriter** 메서드를 호출해서 얻은

JSP의 내장 변수

- **JSP** 페이지 안에서 내장 변수를 사용할 수 있는 이유는 웹 컨테이너가 **JSP** 페이지를 서블릿 클래스로 변환할 때 자동으로 내장 변수를 선언하기 때문이다.

변수 이름	제공하는 기능/변수의 역할	변수 타입	다루는 장
J request	doGet, doPost 메서드의 첫 번째 파라미터와 동일한 역할	javax.servlet.http.HttpServletRequest	3장
response	doGet, doPost 메서드의 두 번째 파라미터와 동일한 역할	javax.servlet.http.HttpServletResponse	3장
out	웹 브라우저로 HTML 코드를 출력하는 기능	javax.servlet.jsp.JspWriter	3장
application	JSP 페이지가 속하는 웹 애플리케이션에 관련된 기능	javax.servlet.ServletContext	3장, 6장
config	JSP 페이지의 구성 정보를 가져오는 기능	javax.servlet.ServletConfig	6장
pageContext	JSP 페이지 범위 내에서 사용할 수 있는 데이터 저장 기능 등	javax.servlet.jsp.PageContext	7장
session	세션에 관련된 기능	javax.servlet.http.HttpSession	4장
page	JSP 페이지로부터 생성된 서블릿	java.lang.Object	다루지 않음
exception	익셉션 객체	java.lang.Throwable	5장

JDBC 이해

- 오라클 SQL
- SQL PLUS 열어서 system 계정으로 접속(초기 암호 인스톨시 설정)
- hr 계정 활성화
- alter user hr identified by hr account unlock;
- hr 계정으로 접속(암호 hr)
- create table 테이블명 (
- 컬럼명 타입(길이) 제약조건,,);
- drop table 테이블명 ;
- insert into 테이블명(컬럼리스트) values(값리스트);

JDBC 이해

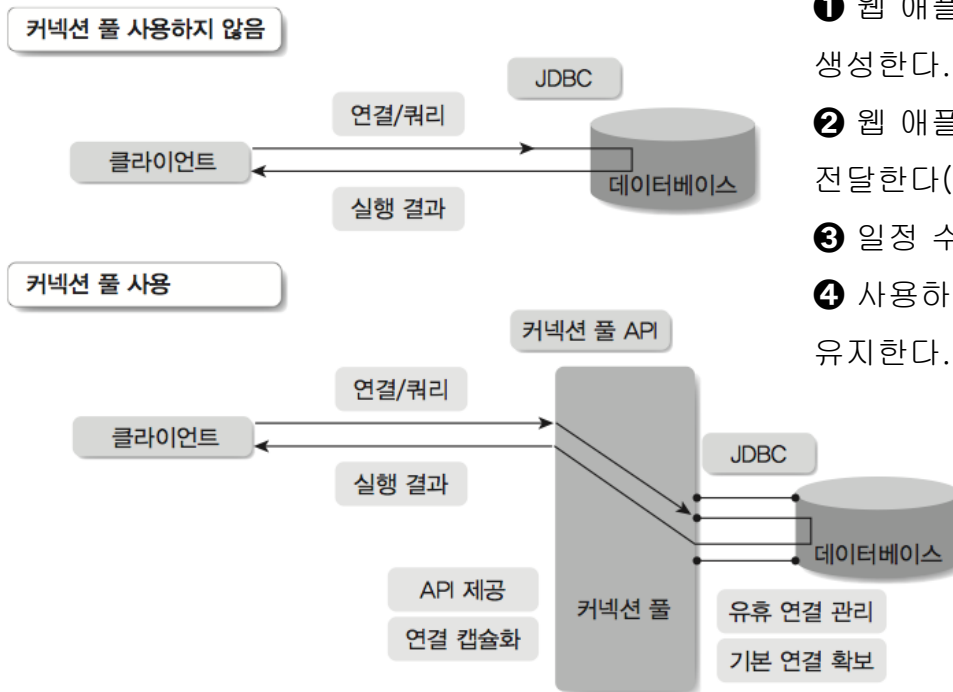
- oracle jdbc driver 등록하기
- 회원가입 정보를 데이터베이스에 저장하기
- 로그인시에 가입했던 정보 확인하여 로그인 허용하기

CONNECTION POOL

1. 데이터베이스 커넥션 풀이란

- 데이터베이스 커넥션 풀(Database Connection Pool)이란 애플리케이션에서 필요로 하는 시점에 커넥션을 만드는 것이 아니라 미리 일정한 수의 커넥션을 만들어놓고 필요한 시점에 애플리케이션에 제공하는 서비스 및 관리 체계를 말한다.

■ 커넥션 풀의 구조와 동작



- ① 웹 애플리케이션 서버가 시작될 때 일정 수의 커넥션을 미리 생성한다.
- ② 웹 애플리케이션 요청에 따라 생성된 커넥션 객체를 전달한다(JNDI 이용).
- ③ 일정 수 이상의 커넥션이 사용되면 새로운 커넥션을 만든다.
- ④ 사용하지 않는 커넥션은 종료하고 최소한의 기본 커넥션을 유지한다.

[그림 12-1] 커넥션 풀 동작 구조

CONNECTION POOL

■ 커넥션 풀 구현 유형

[표 12-1] 데이터베이스 커넥션 풀 구현 유형

유형	설명
직접 구현	개발자가 직접 <code>javax.sql.DataSource</code> 인터페이스를 구현하거나, 직접 새로운 형태의 커넥션 풀을 구현하는 방법이다. 학습을 위한 목적이 아니라면 일반적으로는 권장되지 않는다.
아파치 자카르타 DBCP API를 이용한 구현 (commons-dbcp)	아파치 그룹의 공개된 데이터베이스 커넥션 풀 API인 DBCP를 이용하는 방법이다. 프로그래밍에 자신이 있고 서블릿, 리스너 등의 활용에도 능숙하다면 이를 이용해 자신만의 커넥션 풀을 구성할 수 있다.
애플리케이션 서버 제공	애플리케이션에서 제공되는 커넥션 풀을 사용하는 방법이다. 최근의 웹 애플리케이션 서버들은 <code>javax.sql.DataSource</code> 인터페이스를 따르는 커넥션 풀을 제공하므로 호환에는 큰 문제가 없다. 이 경우 JNDI 네이밍 서비스(Naming Service)를 통해 커넥션 풀을 사용할 수 있다.
프레임워크 제공	애플리케이션 서버와는 별도로 스프링이나 스트러츠 같은 애플리케이션 프레임워크에서 제공하는 커넥션 풀을 사용하는 방법이다. 웹 애플리케이션 이외의 개발에도 사용할 수 있다는 점을 제외하고는 기본적으로 애플리케이션 서버가 제공하는 방식과 다르지 않으므로 개발 성격에 따라 프레임워크에서 제공되는 커넥션 풀을 사용하는 것도 괜찮은 방법이다. 자세한 내용은 해당 프레임워크 개발문서를 참고하도록 한다.

CONNECTION POOL

2. DBCP API를 통한 커넥션 풀 구현

■ DBCP API 설치하기

- 톰캣 7.0에서 부터는 자체적으로 DBCP 를 내하고 있어 아파치 DBCP(common-dbc) 를 따로 설치할 필요가 없다.

■ 톰캣에 DataSource 설정하기

- 톰캣 서버 설정 파일인 server.xml 혹은 context.xml 에서 설정한다.(context.xml 권장)
- DB 연결을 위한 정보와 커넥션풀 운영 관련 정보로 구성된다.
- **context.xml** 파일 수정 – 교재 p.514 참고

```
01 <Resource name="jdbc/mysql" auth="Container"
02     type="javax.sql.DataSource"
03     driverClassName="com.mysql.jdbc.Driver"
04     url="jdbc:mysql://127.0.0.1/jspdb"
05     username="jspbook" password="1234"
06     maxActive="5" maxIdle="3" maxWait="-1" />
```


CONNECTION POOL

[표 12-3] <Resource> 태그 속성

매개변수	설명
username	데이터베이스 접속을 위한 계정 이름이다.
password	데이터베이스 접속을 위한 계정 비밀번호다.
driverClassName	데이터베이스 JDBC 드라이버 클래스 이름이다(예를 들어, 오라클에서는 oracle.jdbc.OracleDriver로 사용한다).
url	데이터베이스 접속에 필요한 정보다(예를 들어, 오라클에서는 jdbc:oracle:thin:@localhost:1521로 사용한다).
maxActive	데이터베이스 최대 연결 개수다. 즉 클라이언트 요청에 따라 커넥션을 생성할 최대 개수다. maxActive 이상의 연결이 동시에 발생할 경우 우선순위에 따라 연결을 처리해야 하므로, 일부 클라이언트 연결에서는 지연이 발생할 수 있다. 0인 경우 시스템 성능 및 데이터베이스 서버 설정에 따라 무제한으로 연결할 수 있다.
maxIdle	데이터베이스 최대 유휴 연결 개수다. 즉, 클라이언트 사용이 없을 때에도 항상 유지할 연결의 최대 개수를 의미한다. -1인 경우 무제한으로 연결된다.

CONNECTION POOL

■ 주요 소스코드 분석

- 커넥션을 가져오는 부분을 제외하고는 기존 소스와 동일함.
- `new InitialContext()`로 Context 객체를 확보하고 JNDI 이름으로 DataSource 객체를 참조한다.
- JNDI 접근은 `java:/comp/env` 로 Context 객체에 접근한 다음 `web.xml`에 등록한 JNDI 이름인 `jdbc/mysql`로 DataSource 객체를 얻는다.
- Connection 객체는 DataSource의 `getConnection()` 메서드로 구하게 된다.

```
09  try{
10      Context initContext = new InitialContext();
11      Context envContext =
12          (Context)initContext.lookup("java:/ comp/env");
13      DataSource ds = (DataSource)envContext.lookup("jdbc/mysql");
14      // 커넥션 얻기
15      Conn = ds.getConnection();
```

FORWARD와 INCLUDE

❖ **forward** 메서드의 사용 방법

- **forward** 메서드는 **SERVLET**과 **JSP** 안에서 다른 **SERVLET**과 **JSP** 를 호출할 때 사용하는 메서드이다.
- 이 메서드는 **javax.servlet.RequestDispatcher** 인터페이스에 속하기 때문에 이 타입의 객체가 있어야 호출할 수 있다.

```
RequestDispatcher dispatcher = request.getRequestDispatcher( "Result.jsp ");
```

↑
호출할 JSP 페이지의 URL 경로명

- **forward** 메서드를 호출할 때에는 **request** 내장 변수와 **response** 내장 변수를 파라미터로 넘겨줘야 한다.

```
dispatcher.forward(request, response);
```

request 내장 변수

response 내장 변수

FORWARD와 INCLUDE

❖ **forward** 메서드의 사용 방법

- **JSP** 페이지로 데이터를 넘겨주려면 **request.setAttribute** 메서드를 호출해서 **request** 내장 변수 안에 데이터를 저장해 놓아야 한다.

```
request.setAttribute( "HEIGHT ", new Integer(178));
```

↑ ↑
데이터 이름 데이터 값

- 호출된 **JSP** 페이지 안에서 **request** 내장 변수 안의 데이터를 가져오려면 **request.getAttribute** 메서드를 호출하면 된다.

```
Integer height = (Integer) request.getAttribute( "HEIGHT ");
```

↑ ↑
캐스트 연산자 데이터 이름

FORWARD와 INCLUDE

❖ include 메서드의 사용 방법

- **include** 메서드를 통해 호출되는 **JSP** 페이지로 데이터를 넘겨주기 위해서는 **forward** 메서드의 경우와 마찬가지로 **request** 내장 변수에 대해 **setAttribute** 메서드와 **getAttribute** 메서드를 호출하면 된다.

```
request.setAttribute( "WEIGHT ", new Double(72.5));
```

↑
request 내장 변수에
데이터를 저장하는 메서드

```
Double weight = (Double) request.getAttribute( "WEIGHT ");
```

↑
request 내장 변수에 저장되어
있는 데이터를 가져오는 메서드

FORWARD와 INCLUDE

❖ include 메서드의 사용 방법

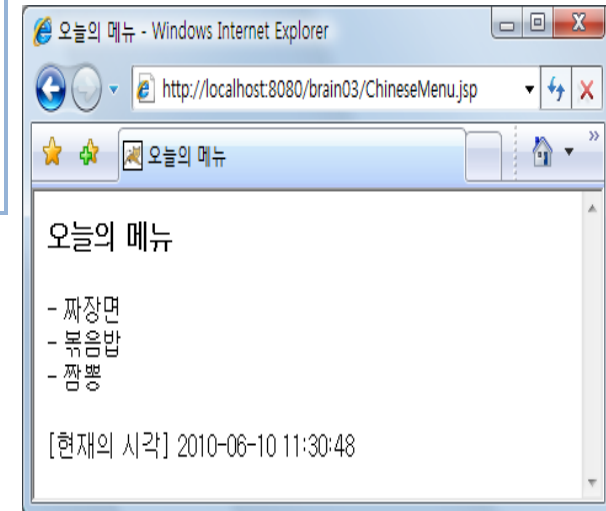
■ include 메서드의 사용 예를 보여주는 JSP 페이지

[예제3-20] include 메서드의 사용 예

```
<%@page contentType= "text/html; charset=euc-kr" %>
<HTML>
  <HEAD><TITLE>오늘의 메뉴</TITLE></HEAD>
  <BODY>
    <H3>오늘의 메뉴</H3>
    - 짜장면 <BR>
    - 볶음밥 <BR>
    - 짬뽕 <BR><BR>
    <%
      out.flush();
      RequestDispatcher dispatcher = request.getRequestDispatcher( "Now.jsp ");
      dispatcher.include(request, response);
    %>
  </BODY>
</HTML>
```

Now.jsp를 include합니다

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.util.*"%>
<% GregorianCalendar now = new GregorianCalendar(); %>
[현재의 시각] <%= String.format("%TF %TT ", now, now) %>
```



FORWARD와 INCLUDE

- ❖ JSP 내에서는 다음과 같은 JSP 태그를 이용하여 INCLUDE 가능하다.
 - **<jsp:include>**는 **JSP** 페이지에서 다른 웹 자원(**JSP** 페이지, **HTML** 문서 등)을 포함시키고자 할 때 사용하는 표준 액션이다.

```
<jsp:include page= "Copyright.html "/>
```

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

- 이 표준 액션에는 포함할 웹 자원의 **URL**을 지정하는 **page** 애트리뷰트를 써야 한다.

FORWARD와 INCLUDE

❖ <jsp:include> 표준 액션의 사용 방법

[예제 8-2] <jsp:include> 액션의 사용 예를 보여주는 예제 (2)

```
<%@page contentType= "text/html; charset=euc-kr" %>
<HTML>
  <HEAD><TITLE>당첨자 명단</TITLE></HEAD>
  <BODY>
    <H3>당첨자 명단</H3>
    14553 연흥부 <BR>
    63563 심청 <BR>
    73992 이몽룡 <BR><BR>
    <jsp:include page= "Now.jsp" />
  </BODY>
</HTML>
```

Now.jsp를 include합니다

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.util.*"%>
<%
  GregorianCalendar now = new GregorianCalendar();
  String date = String.format("%TY년 %Tm월 %Td일" , now, now, now);
  String time = String.format("%Tp %TR", now, now);
%>
[현재 시각] <%= date %> <%= time %>
```

시스템 시계로부터 현재 시각을 가져다가
YY년 MM월 DD일 포맷의 날짜와 AM/PM
hh:mm 포맷의 시각으로 편집합니다.

FORWARD와 INCLUDE

- JSP 내에서는 다음과 같은 JSP 태그를 이용하여 FORWARD가 가능하다.
- **<jsp:forward>**는 **JSP** 페이지에서 다른 **JSP** 페이지로 제어를 넘기고자 할 때 사용하는 표준 액션이다.

```
<jsp:forward page= "Next.jsp" />
```

Next.jsp로 실행의 제어를 넘기는 표준 액션

- **<jsp:include>**와 마찬가지로 **page** 애트리뷰트를 이용해서 해당 **JSP** 페이지의 **URL**을 지정해야 한다.

FORWARD와 INCLUDE

❖ <jsp:forward> 표준 액션의 사용 방법

[예제 8-3] <jsp:forward> 액션의 사용 예를 보여주는 예제

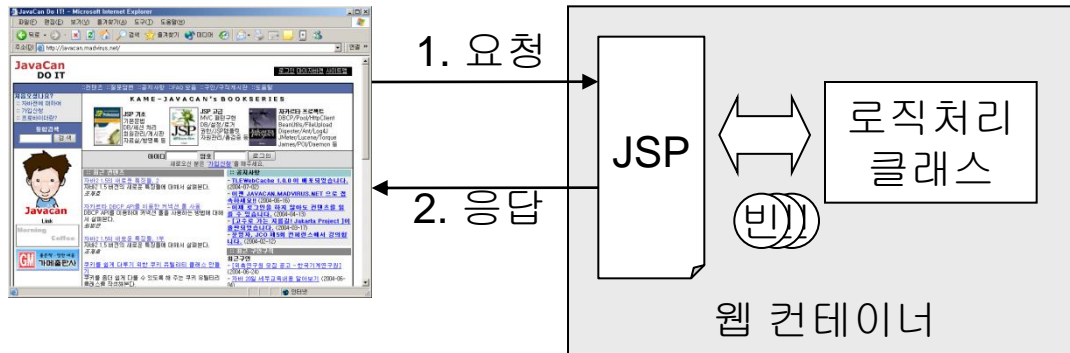
```
<%  
    int sum = 0;  
    for (int cnt = 1; cnt <= 100; cnt++)  
        sum += cnt;  
    request.setAttribute( "RESULT ", new Integer(sum));  
%>  
<jsp:forward page= "HundredResult.jsp " />
```

↓ 실행의 제어를 넘긴다.

```
<%@page contentType= "text/html; charset=euc-kr" %>  
<HTML>  
    <HEAD><TITLE>1부터 100까지의 합</TITLE></HEAD>  
    <BODY>  
        1부터 100까지 더한 결과는? ${RESULT}  
    </BODY>  
</HTML>
```

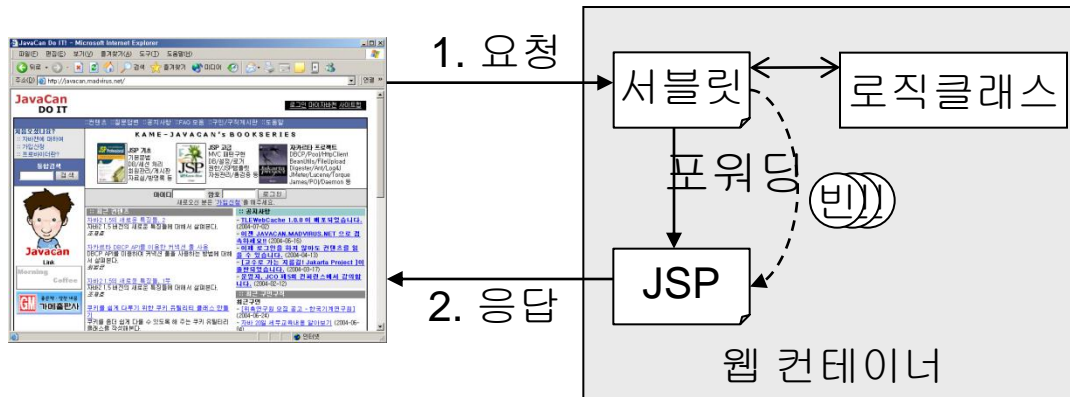
↖ 앞장에서 배운 익스프레션 언어를 이용해서
결과를 출력한다

모델1 구조와 모델2 구조



모델1 구조

- 비즈니스 로직을 JSP에서 실행
- 로직 코드와 UI 코드가 혼합
- 클라이언트 요청이 JSP로 전달

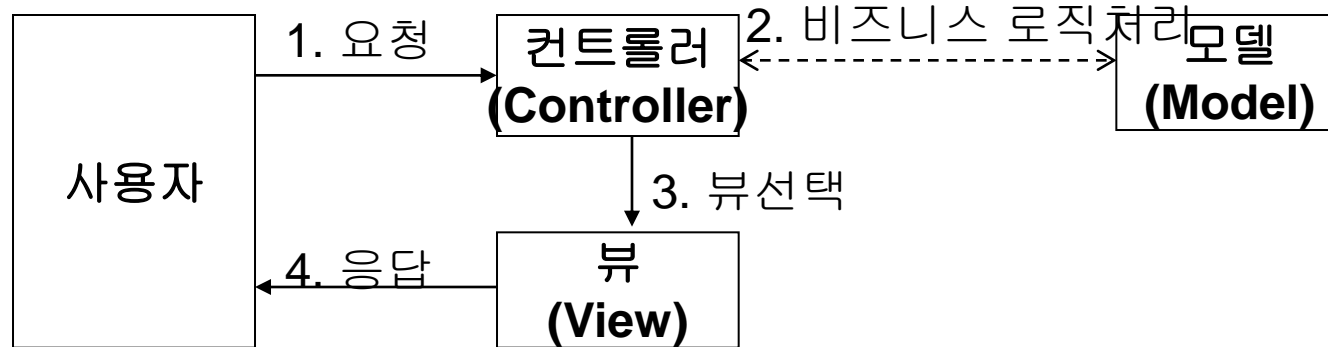


모델2 구조

- 서블릿에서 로직을 실행
- JSP에서 UI를 생성
- 클라이언트 요청이 서블릿으로 전달

Model-View-Controller 패턴

MVC 패턴의 구성



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

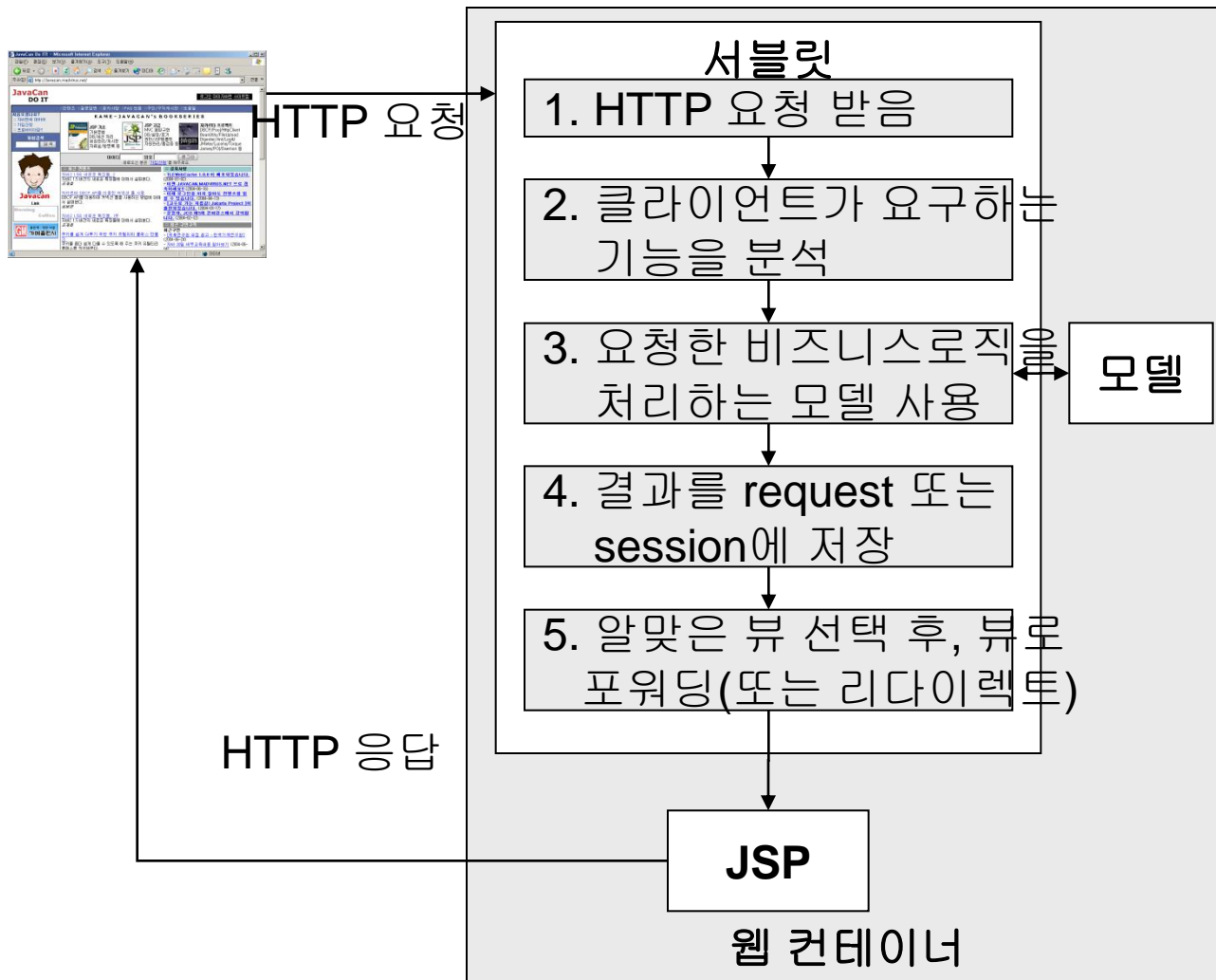
MVC 패턴의 핵심

- 비즈니스 로직을 처리하는 모델과 결과 화면을 보여주는 뷰가 분리되어 있다.
- 어플리케이션의 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중된다.

MVC 패턴과 모델2 구조의 매핑

- 컨트롤러 = 서블릿
- 모델 = EJB 내지 비즈니스 로직 클래스, 자바빈
- 뷰 = JSP
- 사용자 = 웹브라우저, 휴대폰과 같은 디바이스

MVC: 컨트롤러 서블릿



● 과정1 - 웹 브라우저가 전송한 HTTP 요청을 받는다. 서블릿의 doGet() 메소드나 doPost() 메소드가 호출된다.

● 과정2 - 웹 브라우저가 어떤 기능을 요청했는지 분석한다. 예를 들어, 게시판 목록을 요청했는지, 글 쓰기를 요청했는지 알아낸다.

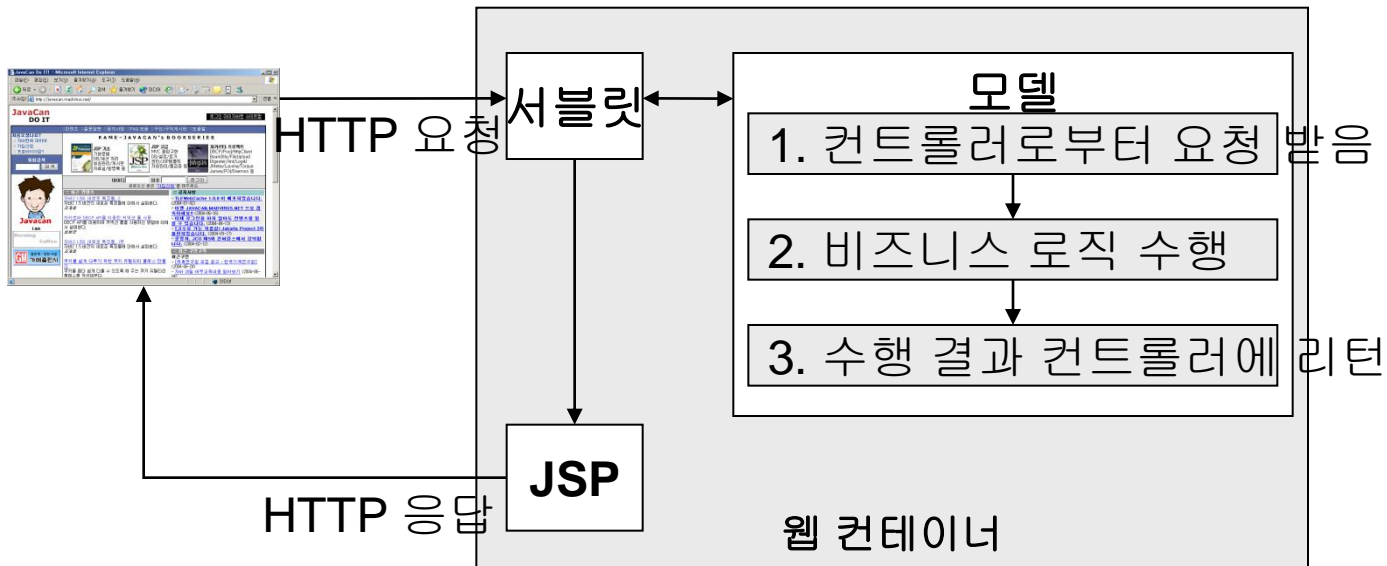
● 과정3 - 모델을 사용하여 요청한 기능을 수행한다.

● 과정4 - 모델로부터 전달받은 결과물을 알맞게 가공한 후, request 나 session 의 setAttribute() 메소드를 사용하여 결과값을 속성에 저장한다. 이렇게 저장된 결과값은 뷰인 JSP에서 사용된다.

● 과정5 - 웹 브라우저에 보여질 JSP를 선택한 후, 해당 JSP로 포워딩한다. 경우에 따라서 리다이렉트를 하기도 한다.

MVC: 컨트롤러와 모델의 연결

모델의 일반적 처리 순서



컨트롤러 서블릿의 기본 구현

Code

```
public class ControllerServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        processRequest(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        processRequest(request, response);
    }

    private void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        // 2단계, 요청 분석
        // request 객체로부터 사용자의 요청을 분석하는 코드
        ...

        // 3단계, 모델을 사용하여 요청한 기능을 수행한다.
        // 사용자에게 요청에 따라 알맞은 코드
        // 4단계, request나 session에 처리 결과를 저장
        request.setAttribute("result", responseObject); // 이런 형태의 코드
        ...
        // 5단계, RequestDispatcher를 사용하여 알맞은 뷰로 포워딩
        RequestDispatcher dispatcher =
            request.getRequestDispatcher("/view.jsp");
        dispatcher.forward(request, response);
    }
}
```

1단계, 요청받음
GET/POST 방식으로
전달받은 웹브라우저의
요청을 하나의 메소드인
`processRequest()`로
전달한다.

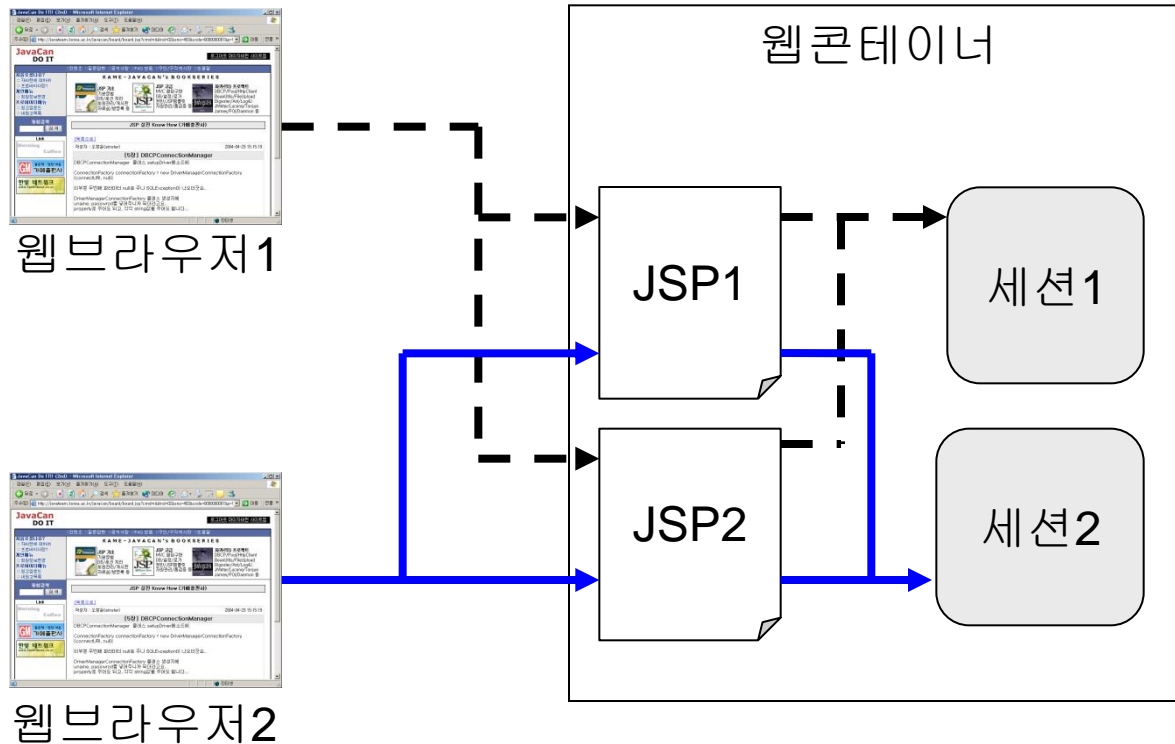
모델1 구조와 모델2 구조

모델	장점	단점
모델 1	<ul style="list-style-type: none">-배우기 쉬움-자바 언어를 몰라도 구현 가능-기능과 JSP의 직관적인 연결. 하나의 JSP가 하나의 기능과 연결	<ul style="list-style-type: none">-여전히 로직 코드와 뷰 코드가 혼합되어 JSP 코드가 복잡해짐-뷰 변경시 논리코드의 빈번한 복사 즉, 유지보수 작업이 불편함-통합 권한 인증 코드를 JSP 단위로 적용해야 함
모델 2	<ul style="list-style-type: none">-로직 코드와 뷰 코드의 분리에 따른 유지 보수의 편리함-컨트롤러 서블릿에서 집중적인 작업 처리 가능.(권한/인증 등)-확장의 용이함	<ul style="list-style-type: none">-자바 언어에 친숙하지 않으면 접근하기가 쉽지 않음-작업량이 많음(커맨드클래스+뷰JSP)-디버깅 작업의 어려움

세션(session)이란?

- HTTP 프로토콜은 STATELESS 프로토콜이다.
- 이전 클라이언트 정보를 응답한 이후에 서버에 클라이언트의 정보를 남기지 않는다.
- 이전 클라이언트 상태를 유지하려면 쿠키나 세션 기술이 필요하다.
- 쿠키는 클라이언트와의 연결 상태 정보를 클라이언트측에 저장한다.
- 세션은 클라이언트와의 연결 상태를 지속적으로 서버측에 저장한다.
- 쿠키와 달리 세션은 웹서버에서 실행되므로 자바 객체를 사용할 수 있다.

웹브라우저와 세션



- * 서로 다른 웹 브라우저는 각각 다른 세션을 사용하게 된다.
- * 웹브라우저가 전송하는 모든 요청은 세션을 공유하게 된다. 즉, 웹브라우저2의 요청은 세션2를 공유게 된다.
- * 세션 공유를 통해 특정 사용자와 관련된 정보를 지속적으로 저장할 수 있다.

세션의 생성 및 session 기본 객체

Code

```
<%@ page session="true" %>
```

session 속성의 기본값을 true이므로, 별도로 지정하지 않으면 세션은 자동으로 생성된다. 세션은 **session** 기본 객체를 통해서 구현되었다.

[session 기본 객체의 메소드]

메소드	리턴타입	설명
getId()	String	세션의 고유 ID를 구한다. (세션 ID라고 한다.)
getCreationTime()	long	세션이 생성된 시간을 구한다. 시간은 1970년 1월 1일 이후 흘러간 시간을 의미하며, 단위는 1/1000초이다.
getLastAccessedTime()	long	웹 브라우저가 가장 마지막에 세션에 접근한 시간을 구한다. 시간은 1970년 1월 1일 이후 흘러간 시간을 의미하며, 단위는 1/1000초이다.

★ 웹 브라우저마다 별도의 세션을 갖게 된다고 했는데, 이때 각각의 세션을 구분하기 위해서 세션마다 고유의 ID를 할당하며, 그 아이디를 세션 ID라고 한다.

세션의 속성 사용

- 한번 생성된 세션은 종료되기 전까지 지속적으로 유지되므로, 웹 어플리케이션을 실행하는 동안 지속적으로 사용해야 하는 데이터의 저장 장소로서 세션이 적당
- 세션에 값을 저장할 때는 `session.setAttribute()` 사용
- 저장된 값을 읽을 때는 `session.getAttribute()` 사용
- 세션은 웹 브라우저와 1대 1로 매핑되므로, 특정 사용자와 관련된 정보를 저장할 때 세션의 속성을 사용

세션의 종료

Code

```
<%@ page session="true" %>  
<%  
    session.invalidate();  
%>
```

세션의 타임아웃

지정한 시간동안 클라이언트로부터 연결이 없으면 세션은 자동 삭제된다.



세션 타임아웃 시간 지정

Code

web.xml 파일에 명시

```
<?xml version="1.0" encoding="euc-kr"?>
<web-app ...>
  ...
  <session-config>
    <session-timeout>50</session-timeout>
  </session-config>
</web-app>
```

분단위로 지정

또는 코드에서 직접 명시

```
<%@ page session="true" %>
<%
  session.setMaxInactiveInterval(3000);
%>
```

초단위로 지정

세션을 이용한 로그인/로그아웃

- 로그인 성공시 성공했음을 나타내는 표식을 생성/저장한다.
 - 세션, session의 "MEMBER" 속성에서 회원 아이디 저장
- 로그인 여부 체크
 - 세션, session의 "MEMBER" 속성이 존재하는지 검사
- 로그아웃을 할 경우 로그인 표식을 삭제
 - 세션, session.invalidate()로 세션 무효화