# chapter 4. JSP 내장 객체

- JSP에서 사용자가 객체를 생성하지 않아도,  이미 생성되어 사용할 수 있는 객체를 의미한다.
- JSP가 서블릿 파일로 변환되어 실행 될 때 서블릿 컨테이너가 자동으로 생성 해 준다.

## 내장 객체 종류

| 내장 객체 | 반환값 타입 | 설명 |
|---|---|---|
| *request* | javax.servlet.http.httpServletRequest 또는 javax.servlet.ServletRequest | 웹 브라우저의 요청 정보를 저장하고 있는 객체 |
| *response* | javax.servlet.http.httpServletResponse 또는 javax.servlet.ServletResponse | 웹 브라우저의 요청에 대한 응답 정보를 저장하는 객체 |
| *out* | javax.servlet.jsp.JspWriter | JSP 페이지의 출력할 내용을 가지고 있는 출력 스트림 객체 |
| *session* | javax.servlet.http.HttpSession | 하나의 웹 브라우저 내에서 정보를 유지하기 위한 세션 정보를 저장하고 있는 객체 |
| *application* | javax.servlet.ServletContext | 웹 애플리케이션 Context의 정보를 담고 있는 객체 |
| *pageContext* | javax.servlet.jsp.PageContext | JSP 페이지에 대한 정보를 저장하고 있는 객체 |
| *page* | java.lang.Object | JSP 페이지를 구현한 자바 클래스 객체 |
| *config* | javax.servlet.ServletConfig | JSP 페이지에 대한 설정 정보를 담고 있는 객체 |
| *exception* | java.lang.Throwable | JSP 페이지에서 예외가 발생한 경우 사용하는 객체 |

*※ 자주 사용되는 내장 객체는 굵은 글씨로 표기*

### (1) out 내장 객체 : 클라이언트로 html형태로 출력하기 위한 스트림 객체

```
// 보통 서블릿에서 우리는 이렇게 객체를 얻어 쓴다
PrintWriter out = response.getWriter();
```

## (2) request 내장 객체 : 클라이언트로 부터 요청(request) 될 때의 모든 정보를 담고 있는 객체이다.

사용자가 form에 입력한 요구 사항을 얻어낼 수 있도록하는 메소드 제공

| 메소드 | 설명 |
| --- | --- |
| *String getParameter(name)* | 파라미터 변수 name에 저장된 값을 얻어내는 메소드<br>해당 변수명이 없으면 null 리턴<br>단독값을 입력하는 text, select, radio 등에 사용 |
| *String[] getParameterValues(name)* | 파라미터 변수 name에 저장된 모든 값을 얻어내는 메소드<br>변수값은 String 배열로 리턴<br>다중값을 입력하는 checkbox 등에 사용 |
| *Enumeration getPrameterNames()* | 요청에 의해 넘어오는 모든 파라미터 변수를 java.util.Enumeration 타입으로 리턴<br>변수가 가진 객체들을 저장하기 위해 Enumeration 컬렉션 사용 |

request 객체는 요청된 파라미터 값 외에도
웹 브라우저와 웹 서버의 정보도 가져올 수 있음

| 메소드 | 설명 |
|---|---|
| *String getProtocol()* | 웹 서버로 요청 시 사용 중인 프로토콜 리턴 |
| *String getServerName()* | 서버의 도메인 이름 리턴 |
| *String getMethod()* | 요청에 사용된 요청 방식(GET, POST 등) 리턴 |
| *String getQueryString()* | 요청에 사용된 QueryString 리턴 |
| *String getRequestURL()* | 요청에 사용된 URL 주소 리턴 |
| *String getRequestURI()* | 요청에 사용된 URL로부터 URI값 리턴 |
| *String getRemoteHost()* | 웹 서버로 정보를 요청한 웹 브라우저의 host 이름 리턴 |

| 메소드 | 설명 |
|---|---|
| *String getRemoteAddr() : String* | 웹 서버로 정보를 요청한 웹 브라우저의 ip 주소 리턴 |
| *String getServerPort()* | 웹 서버로 요청시 서버의 port 번호 리턴 |
| *String getContextPath() : String* | 해당 JSP페이지가 속한 웹 애플리케이션의 콘텍스트 경로 리턴<br>콘텍스트 경로는 웹 애플리케이션의 루트 경로 |
| *String getHeader(name)* | 웹 서버로 요청 시 HTTP 요청 header 이름인 name에 해당하는 속성값 리턴 |
| *Enumeration getHeaderNames()* | HTTP 요청 header에 있는 모든 헤더 이름 리턴 |


```
request.setCharacterEncoding(String env);  // env로 인코딩한다
request.getRequestDispatcher(String url); // url로 데이터를 이동시켜주는 RequestDispatcher 객체 반환
```

## (3) response 내장 객체 : (request 했던)클라이언트로 응답하는 모든 정보를 담고 있는 객체

웹 클라이언트로 보내는 사용자 응답과 관련된 기능을 제공
-request 와 반대되는 개념
-사용자 요청 (request)를 처리하고 응답을 다른 페이지로 전달하는 등의 기능을 제공
-javax.servlet.http.HttpServletResponse 객체에 대한 참조 변수

response 주요 메소드

| 메서드 | 설명 |
|---|---|
| **setContentType(type)** | 문자열 형태의 type 에 지정된 MINE TYPE 으로 contentType 을 설정한다. |
| setHeader(name, value) | 문자열 name 의 이름으로 문자열 value의 값을 헤더로 세팅한다. |
| setDateHeader(name, date) | 문자열 name 의 이름으로 date 에 설정된 밀리세컨드 시간 값을 헤더에 설정한다. |
| sendError(status, msg) | 오류 코드를 세팅하고 메시지를 보낸다. |
| **sendRedirect(url)** | 클라이언트 요청을 다른 페이지로 보낸다. |

(예제) 아래의 예제는 로그인 성공하면 mainTest.jsp로 페이지가 이동 되는 예제를 구현한 것이다.

! 페이지 이동 시킬 때 쿼리스트링을 이용하면 좀 더 다이나믹하게 처리 할 수 있다.

```
// mainTest.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<script>
function getParameter(paramName) {
```

```
    // 쿼리스트링에서 넘겨받은 paramName을 찾아 그 변수의 값을 return 한다.
    // 만약 쿼리스트링에 paramName이 없다면 null을 return 한다.

    let returnVal = null;
    let url = location.href;

    if (url.indexOf("?") != -1) {
      // 쿼리스트링이 있을때
      let queryStr = url.split("?")[1];
      console.log(queryStr);
      let tmpAr = queryStr.split("&");
      for (let tmp of tmpAr) {
        if (tmp.split("=")[0] == paramName) {
          returnVal = tmp.split("=")[1];
          break; // 해당 반복문 블럭 빠져나감
        }
      }
    }

    return returnVal;
}


window.onload = function() {
  let status = getParameter("status");
  if (status === "loginSuccess") {
    alert("로그인 성공! 회원님 방가~");
  }
}


</script>
</head>
<body>
  <h1>mainTest.jsp</h1>

  <a href="loginTest1.jsp">로그인 하러 가기</a>
</body>
</html>

// ---------------- loginTest1.jsp ----------------------------------------------
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <form action="sendRedirect.do" method="post">
    <div>
      아이디 : <input type="text" name="userId" />
    </div>
    <div>
```

```
      패스워드 : <input type="password" name="userPwd" />
    </div>
    <div>
      <input type="reset" value="취소" />
      <input type="submit" value="로그인" />
    </div>
  </form>
</body>
</html>

// ---------------------------- SendRedirectServlet.java ----------------------
package com.jspbasic;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class SendRedirectServlet
 */
@WebServlet("/sendRedirect.do")
public class SendRedirectServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;



  protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    // 아이디 : abcd123
    // 비밀번호 : a1b2c3!
    //
    // 아이디와 비밀번호가 일치하면 mainTest.jsp로 이동시키자

    response.setContentType("text/html; charset=utf-8");
    PrintWriter out =  response.getWriter();

    String userId = request.getParameter("userId");
    String password = request.getParameter("userPwd");

    if(userId.equals("abcd123") && password.equals("a1b2c3!")) {
      // 아래의 js 코드는 효과 없다 (왜? response.sendRedirect("mainTest.jsp"); 코드가 우선 실행됨)
//      out.print("<script>");
//      out.print("alert('로그인성공!');");
//      out.print("</script>");

      // response.sendRedirect(String url) : url 페이지로 이동
      // 쿼리스트링을 달고 갈 수 있다.
      response.sendRedirect("mainTest.jsp?status=loginSuccess");
    }
```

```
  }

}
```

(예제) 아래의 예제는 서블릿에서 페이지를 이동 시키는 4가지 방법을 나열하였다. 2번, 3번은 반드시 알아둘 것.

```
package com.jspbasic;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/pageMove.do")
public class PageMovingServlet extends HttpServlet{

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {

    PrintWriter out = resp.getWriter();

    // 서블릿에서 페이지를 이동시키는 방법들

    // 1) js의 location.href 를 이용하는 방법
//    out.print("<script>");
//    out.print("location.href= 'mainTest.jsp?status=fail';");
//    out.print("</script>");

    // 2) response객체의 sendRedirect()를 이용하는 방법
//    resp.sendRedirect("mainTest.jsp?status=fail");

    // 3) RequestDispatcher 객체를 이용하여 이동하는 방법
    // URL 주소가 바뀌지 않는다
    // 데이터를 바인딩하여 데이터를 보낼때 사용
//    req.getRequestDispatcher("mainTest.jsp").forward(req, resp);

    // 4) meta 태그의 <meta http-equiv="refresh" content="시간(초)" url="url주소"> 이용하는 방법

//    out.print("<html>");
//    out.print("<head>");
//    out.print("<meta http-equiv='refresh' content='5; url=mainTest.jsp'>");
//    out.print("</head>");
```

```
//    out.print("</html>");
//
//    out.flush();
//    out.close();
  }


}
```

## (4) application 내장 객체 : 하나의 웹 어플리케이션을 관리하고 웹 어플리케이션 안에서의 자원을 담고 있는 객체.

[ application - javax.servlet.ServletContext 객체 ]

- 생성 / 소멸 : 컨테이너의 생명주기와 동일
- 사용 범위 : 프로젝트 내 모든 객체에서 접근 가능

어플리케이션의 내외부의 여러 환경 정보를 담고 있는 객체입니다. 어플리케이션 가장 상단에 위치한 객체쯤으로 생각하면 될 것 같고, 처음 컨테이너가 구동될 때 단 하나의 객체만 생성됩니다. 기본적으로 서버에 대한 정보나 서블릿에 대한 정보들을 가지고 있는데 어플리케이션(프로젝트 단위) 내 모든 JSP/서블릿들이 공유하는 객체다 보니 데이터를 편리하게 운송하는 수단으로 주로 쓰입니다.

| application 주요 메소드 | 설명 |
|---|---|
| void setAttribute(String name, Object o) | application 객체에 추가 객체를 저장 |
| Object getAttribute(String name) | application 객체에 추가된 객체를 가져옴 |
| void removeAttribute(String name) | application 객체에 특정 Attribute를 지움 |
| String getServerInfo() | 컨테이너 이름과 버전을 리턴 |
| void log(String msg) | 제공된 문자열을 서블릿 로그 파일에 기록 |

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <div>웹 어플리케이션의 컨텍스트 패스 : <%=application.getContextPath() %></div>
  <div>웹 어플리케이션의 실제 저장경로 : <%=application.getRealPath(request.getRequestURI()) %></div>

</body>
</html>
```

## (5) session 내장 객체 : (session의 사전적 의미: 연결된 상태)

## [ session - javax.servlet.http.HttpSession 객체 ]

- 클라이언트가 웹 서버에 접속하면 자동 생성, 웹 서버에서 떠날 때까지 소멸되지 않음(세션연결 시간 동안 아무 작업도 하지 않으면 세션객체 소멸)
- 웹서버에 접속한 클라이언트 마다 하나의 세션 객체가 만들어짐
- session 내장 객체의 특성을 이용해 로그인/로그아웃 기능을 구현한다.

| session 주요 메소드 | 설명 |
|---|---|
| void setAttribute(String name, Object value) | 파라미터 이름에 따른 객체를 추가 |
| Object getAttribute(String name) | 파라미터 이름에 따른 객체를 반환 |
| void removeAttribute(String name) | 파라미터 이름에 따른 객체를 삭제 |
| void invalidate() | 세션의 모든 삭제를 삭제 (세션 무효화) |
| Enumeration<String> getAttributeNames() | 등록된 모든 파라미터 이름을 반환 |
| String getID() | 해당 세션(브라우저)의 고유 식별자 정보 반환 |
| Boolean isNew() | 세션 객체가 최초로 생성되었는지 여부 반환 |
| void setMaxInactiveInterval(int interval) | 세션이 유지되는 유효시간을 설정 |
| int getMaxInactiveInterval() | 세션에 설정된 유효시간을 반환 |

- 세션 유효시간은 보통 30분으로 설정한다.(30분동안 아무 request도 없으면 그 session객체는 만료되고 다른 session객체로 갱신된다) setMaxInactiveInterval()메서드로 세션마다 유효시간을 설정할 수 있지만, 서버 관리자가 아닌 이상 사용하지 않을 것을 권고 한다.

### 세션의 유효시간 지정 방법

그렇다면 세션의 유효시간은 어떻게 지정할 수 있을까요? 먼저 배포서술자인 web.xml을 통해 지정 가능합니다. 웹어플리케이션 경로의 WEB-INF/web.xml 파일을 통해 설정이 가능합니다. 다음과 같이 **<session-config>** 태그의 **<session-timeout>** 를 사용하여 지정 가능하며 단위는 **분 단위**입니다.

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```



### 세션별로 유효시간을 지정하는 방법

만약 세션객체별로 유효시간을 지정하고 싶다면 세션객체의 setMaxInactiveInterval(int interval) 메서드를 통해 지정할 수 있습니다. 단 이 경우 유효시간 단위는 분 단위가 아닌 초(second) 단위입니다.

```
<%
    session.setMaxInactiveInterval(1800);
%>
```



- 서블릿 컨테이너는 **클라이언트로부터 최초 request가 들어올 때 session객체를 생성**한다. 이때 브라우저의 고유 식별자(sessionID)를 받아, 같은 식별자를 가진 session 객체가 있으면 해당 session객체를 가져 쓰고, 없으면 새로 만들어 준다. 또한 session 객체를 보관하고 있다가 유효시간이 끝나면 session 객체를 갱신 시킨다.

예제) 아이디 : abcd123,  비번 : 555566 이라고 할 때, 세션 객체의 특징을 이용하여 로그인 처리와 로그아웃 처리를 해보자

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <form action="sessionLogin.do" method="post">
    <div>
      아이디 : <input type="text" name="userId" />
    </div>
    <div>
      패스워드 : <input type="password" name="userPwd" />
    </div>
    <div>
```

```
      <input type="reset" value="취소" />
      <input type="submit" value="로그인" />
    </div>
  </form>
</body>
</html>
--------------------------------------------------------------------------------
package com.jspbasic;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/sessionLogin.do")
public class SessionLoginServlet extends HttpServlet {

  @Override
  protected void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {

    // 아이디 : abcd123
    // 비번 : 555566
    // 이라고 할때, 로그인 처리와 로그아웃 처리를 해보자

    String userId = req.getParameter("userId");
    String userPwd = req.getParameter("userPwd");

    if (userId.equals("abcd123") && userPwd.equals("555566")) { // 로그인 성공
      // 세션객체에 로그인 정보를 남겨야 함 -> 로그인정보를 세션객체에 바인딩
      HttpSession ses =req.getSession();  // request로 부터 세션 객체를 얻어옴
      System.out.println("세션 id : " + ses.getId());

      ses.setAttribute("loginMember", userId); // 바인딩

      resp.sendRedirect("mainTest.jsp?status=loginSuccess");  // mainTest.jsp로 이동

    }


  }

}

--------------------------------------------------------------------------------
package com.jspbasic;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * Servlet implementation class SessionLogoutServlet
 */
@WebServlet("/sessionLogout.do")
public class SessionLogoutServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public SessionLogoutServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    // 로그아웃 처리 -> session객체에 남아 있는 로그인 정보를 지운후 session객체 갱신
    HttpSession ses = request.getSession();

    ses.removeAttribute("loginMember"); // 로그인 정보 삭제

    ses.invalidate(); // session객체 무효화 -> 갱신

    response.sendRedirect("mainTest.jsp?status=logoutSuccess"); // mainTest.jsp로 이동


  }



}
```

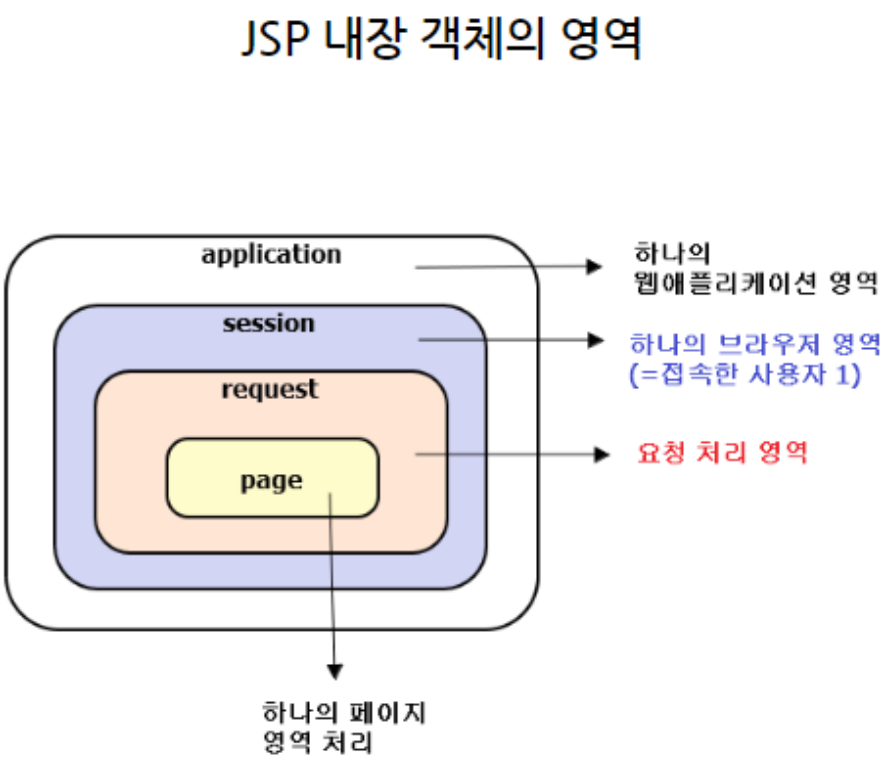
### (6) 내장 객체의 영역

- JSP 내장된 객체들의 유효기간.
- 해당 객체가 얼마 동안 살아 있느냐.

(해당 객체에 바인딩을 하게 되는데, 바인딩 한 정보가 어느 기간까지 유효 할 것인가?)

- 영역은 총 4가지로 아래와 같다

| Scope | 설명 |
| --- | --- |
| page | 하나의 JSP 페이지를 처리 할 때 사용 되는 영역. (하나의 jsp page에서 만 유효) |
| request | 하나의 요청(request)를 처리 할 때 사용되는 영역. (request한곳과 request를 받은 페이지에서만 유효) |
| session | 하나의 브라우저와 관련된 영역. (최초 request가 되어 session객체가 생성되고, session객체가 갱신 될 때까지 유효) |
| application | 하나의 웹 어플리케이션(context)이 동작 하는 동안 유효 |




1) page영역

- 한 번의 클라이언트 요청에 대해 응답된 하나의 JSP 페이지를 범위로 갖는다.
- 클라이언트의 요청을 처리하는 JSP 페이지는 요청에 대해 새로운 page영역을 갖게 되며, 이 때 pageContext 객체를 할당 받는다(메모리에서 로딩됨)

2) request영역

- 하나의 jsp나 서블릿에서 다른 jsp나 서블릿으로 요청할 때 사용되는 영역

- 요청한 페이지와 요청을 받은 페이지 사이에 request 내장 객체에 정보를 저장 할 수 있다.
- 요청을 받은 페이지에서는 그 요청과 관련된 request 내장 객체는 사라진다.
- 요청을 할 때 마다 새로운 request 내장 객체가 생성되고, 매번 새로운 요청에 대해 새로운 request영역이 할당되는 것.

3) session영역

- 웹 브라우저를 닫기 전까지 페이지를 이동하더라도 사라지지 않는 영역.

4) application 영역

- 하나의 웹 어플리케이션과 관련된 전체 영역을 모두 포함.

(7) JSP 내장 객체에서 정보를 주고 받기 위한 메서드(바인딩 처리와 관련된 메서드)

| 메서드 | 설명 |
|---|---|
| void setAttribute(name, value) | name으로 value를 바인딩 |
| Object getAttribute(name) | name이름으로 바인딩된 객체를 Object 타입으로 반환 |
| Enumeration<String> getAttributeNames() | 모든 바인딩된 name을 반환 |
| void removeAttribute(name) | name으로 바인딩된 정보를 삭제 |

예제) 아래의 예제에서는 내장객체의 영역에 바인딩한 정보가 어디까지 살아 있는지 확인하는 예제이다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <h1>scope_1Page</h1>

  <%
    // 각 내장 객체의 영역에 정보 바인딩
```

```
    pageContext.setAttribute("name", "page data");
    request.setAttribute("name", "request data");
    session.setAttribute("name", "session data");
    application.setAttribute("name", "application data");
  %>

  <%
    // 각 내장 객체의 영역에 바인딩 된 정보 확인
    out.print("pageContext : " + pageContext.getAttribute("name") + "<br />");
    out.print("request : " + request.getAttribute("name") + "<br />");
    out.print("session : " + session.getAttribute("name") + "<br />");
    out.print("application : " + application.getAttribute("name") + "<br />");

    request.getRequestDispatcher("scope_2Page.jsp").forward(request, response); // 데이터 포워딩
  %>
</body>
</html>

--------------------------------------------------------------------------------
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <h1>scope_2Page</h1>


  <%
    // 각 내장 객체의 영역에 바인딩 된 정보 확인
    out.print("pageContext : " + pageContext.getAttribute("name") + "<br />");
    out.print("request : " + request.getAttribute("name") + "<br />");
    out.print("session : " + session.getAttribute("name") + "<br />");
    out.print("application : " + application.getAttribute("name") + "<br />");


  %>

  <a href="scope_3Page.jsp">세번째 페이지</a>
</body>
</html>
--------------------------------------------------------------------------------
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <h1>scope_3Page</h1>
```

```
  <%
    // 각 내장 객체의 영역에 바인딩 된 정보 확인
    out.print("pageContext : " + pageContext.getAttribute("name") + "<br />");
    out.print("request : " + request.getAttribute("name") + "<br />");
    out.print("session : " + session.getAttribute("name") + "<br />");
    out.print("application : " + application.getAttribute("name") + "<br />");


  %>


</body>
</html>
```