JSP
(Java Server Page)

1. 스크립트 요소(Scripting elements)

스크립트 요소는 선언부(declarations), 스크립트릿(scriptlet), 표현식(expression)으로 나뉜다.

선언부는 JSP page에서 사용할 변수와 method를 정의하는 부분으로서, 나머지 두 개의 스크립트 요소인 스크립트릿과 표현식에서 선언부에 정의되어 있는 변수와 mehtod를 사용할 수 있다.

스크립트릿은 JSP page를 요청할 때 마다 매번 실행되는 코드 영역으로서 스크립트 언어를 사용하여 필요한 처리를 할 때 사용된다.

표현식은 스크립트릿과 마찬가지로 client(web browser)가 JSP page를 요청할 때 마다 매번 실행되지만, 스크립트릿과 달리 표현식의 결과는 자동으로 JSP page의 출력에 삽입된다.

➢ Page Directive

속 성	값	기 본 값	예 제
language	스크립트 언어 이름	"java"	language="java"
contentType	MIME 타입, charset	"text/html; charset=ISO-8859-1"	contentType="text/html; charset=EUC-KR" contentType="text/html"
info	문자열		info="JSP page"
import	class or package name		import="java.io.*" import="java.util.*, java.sql.*"
session	boolean	"true"	session="false"
buffer	buffer size or false	"8kb"	buffer="16kb" buffer="none"
autoFlush	boolean	"true"	autoFlush="false"
isThreadSafe	boolean	"true"	isThreadSafe="true"
errorPage	local URL		errorPage="error/e500.jsp"
isErrorPage	boolean	"false"	isErrorPage="true"
Extends	class name		extends=""
pageEncoding	page 의 char-encoding	"ISO-8859-1"	pageEncoding="EUC-KR"

jdk에서 지원하는 인코딩 타입 목록

http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html

JSP Directive

1. page directive

1-1. language 속성 : 기본 속성은 java로 되어있기 때문에 따로 지정하지 않아도 된다.

```
<%@ page language="java" %>
```

1-2. contentType 속성 : JSP page가 생성하는 문서의 MIME Type을 나타내는데 사용된다. charset의 기본값은 «text/html ;charset=ISO-8859-1"이다.

```
<%@ page contentType="text/html(plain, xml)" %>
<%@ page contentType="text/html(plain, xml);charset=euc-kr" %>
```

일반적으로 JSP page가 HTML문서를 동적으로 생성하는데 사용되지만 일반 텍스트 문서(text/plain)와 XML 문서(text/xml)를 동적으로 생성할 때도 사용된다.

charset을 올바르게 지정하지 않는 경우 ➔ 한글 출력에 문제점 발생

```
<%@ page contentType= "text/html ;charset=jp" %>
<html>
  <head><title>잘못된 인코딩 </title></head>
  <body>
  글자가 깨진다.
  </body>
</html>
```

1-3. info 속성 : page를 설명해 주는 문자열을 표현.(값의 내용이나 길이에 제한 없다. 거의 사용X)

```
<%@ page info="JSP page 입니다." %>
```

1-4. import 속성 : JSP page에서 package 이름을 명시하지 않고 class를 참조하려 할 때 사용.

```
<%@ page import=«java.io.*" %>
<%@ page import=«java.net.*, java.util.*" %>
```

Page directive 중에서 import 속성만 다중으로 지정이 가능하다. 또, 하나의 import 속성에서 ,(comma)를 이용하여 여러 package, class를 지정할 수 있다.

1-5. session 속성 : JSP page가 session 기본객체를 사용할 지의 여부를 나타낸다. 기본값은 true이며, true일 경우는 JSP page는 session관리에 참여하게 된다.

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

1-6. extends 속성 : JSP page에서 servlet class로 변환된 class가 상속 받을 상위 class를 지정할 때 사용한다.

```
<%@ page extends="com.taglib.myJsp" %>
```

실제로, extends 속성은 거의 사용되지 않는다. 기본적으로 JSP Container는 자신에게 가장 알맞은 JSP Servlet class를 사용하기 때문이다.

1-7. buffer 속성 : JSP page가 출력하는 내용을 임시로 저장하는 버퍼와 관련된 사항 지정.
JSP page는 javax.servlet.jsp.JspWriter class의 객체인 out 기본 객체를 통해서 내용을 출력한다.
out 기본 객체가 사용하는 버퍼의 크기를 지정하는 속성이다. 기본값은 8kb이다.

```
<%@ page buffer="16kb" %>
<%@ page buffer="none" %> → 버퍼를 사용하지 않을 경우.
```

버퍼가 가득 찰 경우, autoFlush 속성에 지정된 값에 따라 버퍼에 내용이 처리된다.

*** buffer 속성을 사용하는 이유**

① 1000글자를 출력하는 경우, 한 글자씩 출력하는 것보다 1000글자를 버퍼와 같은 임시 저장 공간에 모아 놓았다가 한꺼번에 출력하는 것이 더 좋은 성능을 보인다.

② <jsp:forward> action tag를 사용하여 JSP page의 제어를 이동하거나 JSP page 내에서 예외가 발생하여 에러 JSP page를 보여주고 싶은 경우, 즉 JSP page가 출력 결과를 버퍼에 일단 저장한다는 것은 그 출력 결과가 웹 브라우저에 전송되지 않았다는 것을 의미한다. → 웹 브라우저는 서버로부터 어떠한 정보도 응답 받지 않았기 때문에 모든 버퍼의 내용을 지우고, 다시 버퍼에 새로 출력할 내용을 저장해도 상관이 없다.

JSP page page가 출력하는 내용의 크기보다 버퍼의 크기가 작다면, <jsp:forward> action tag나 예외 JSP page와 같은 JSP의 장점을 사용하지 못할 수도 있게 된다. 버퍼의 크기지정은 중요하다.

1-8. autoFlush 속성 : true가 기본값으로 버퍼가 다 차게 되면, 버퍼가 가득 찬 순간 버퍼에 있던 내용은 웹 브라우저에 전송되고 버퍼는 자동으로 비워진다. 반면, false일 경우에는 버퍼가 꽉 차게 되면 버퍼에 있는 내용을 웹 브라우저에 보내는 대신에 엉뚱한 page를 보여준다.

```
<%@ page autoFlush="false" %>
```

autoFlush 속성 사용시 주의할 내용 : buffer의 속성이 none일 경우 autoFlush 속성의 값을 false로 지정할 수 없다.

```
<%@ page buffer="1k" %>
<%@ page autoFlush="false" %> → true로 바꾸고도 실행
<html>
<head><title>버퍼 오버 플로우 테스트</title></head>
<body>
<% for (int i = 0 ; i < 1000 ; i++) { %> <%= i %> <% } %>
</body>
</html>
```

1-9. isThreadSafe 속성 : JSP page가 여러 웹 브라우저의 요청에 대해서 안전하게 응답할 수 있는지의 여부를 나타낼 때 사용. 즉, 동시에 여러 개의 thread가 JSP page에 접근할 때 JSP page내부에서 사용하는 값들이 thread에 안전하게 처리되는가의 여부를 나타낸다.
기본값은 true이며, 이 값은 JSP page가 multy thread에 안전하게 작성된 것을 나타낸다.

이 속성값에 따라서 웹 어플리케이션의 전체적인 성능에 영향을 미칠 수 있다. 예를 들어, isThreadSafe 속성의 값이 true일 경우 JSP Container는 JSP page를 변환한 servlet class의 instance를 오직 한 개만 생성한다. 그 이유는, thread에 안전하게 JSP page를 작성했기 때문에 그 JSP page를 변환한 servlet class 역시 multy thread 환경에서 안전하기 때문이다.
반면, JSP page가 multy thread 환경에서 thread에 안전하지 않아서 isThreadSafe 속성의 값을 false로 지정하게 되면 여러 개의 웹 브라우저가 동시에 이 JSP page에 접근한다고 할 경우 JSP Container에 따라 JSP page를 변환한 servlet class의 instance를 접속한 client의 개수만큼 생성(→ 동시 접속자가 증가할 경우 그 수만큼 instance를 생성하게 되어 자원의 낭비를 초래하거나, 메모리 관리를 위한 Garbage Collector가 자주 수행되는 경우가 발생)하거나 또는 한번에 하나의 요청만을 처리(→ 한명의 client에 대한 서비스가 끝날 동안 다른 client는 대기상태가 되어 있어야 하므로 전체적인 web application의 응답 시간을 길어지게 만드는 요인이 된다.)한다.

threadUnSafe.jsp

```
<%!
    int count = 0;
%>
<html>
<head>
<title>쓰레드에 안전하지 않은 JSP</title>
</head>
<body>
<%
    count ++;
%>
현재 count의 값: <%= count %>
</body>
</html>
```

➔ 동시에 똑 같은 client가 접속했을 경우 똑 같은 수의 count가 출력된다. (이유는 하나의 웹 브라우저의 요청을 처리하는 thread가 count++을 실행하고 <%= count %>를 처리하기 이전에 다른 웹 브라우저의 요청을 처리하는 thread가 count++을 실행할 수 있기 때문이다.

threadSafe.jsp

```
<%!
    int count = 0;
    Object syncObj = new Object();
%>
<html>
<head>
<title>쓰레드에 안전한 JSP</title>
</head>
<body>
<%
    synchronized(syncObj) {
        count ++;
%>
현재 count의 값: <%= count %>
<%
    }
%>
</body>
</html>
```

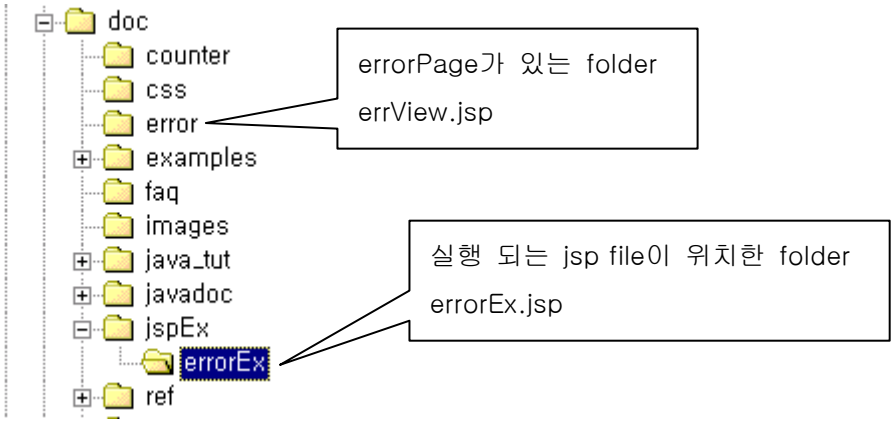
➔ synchronized block처리를 이용한 threadSafe 예제

synchronized block을 사용하여 JSP page 내에서 발생할 수 있는 동기화 문제를 해결할 수 있지만, 내부적으로 동기화 처리가 많이 발생하기 때문에 전체적인 응답속도가 길어지는 문제점을 야기할 수 있다. 또한, session 객체와 application 기본 객체를 사용하여 동기화를 할 경우 전체 application의 처리 속도가 급격히 느려질 수도 있다. 따라서 synchronized block은 필요한 곳에서만 사용해야 한다.

1-10. errorPage 속성 : JSP page에서 try{ . . . }catch를 이용하여 예외를 처리할 수 있다. JSP page 내에서 예외가 발생 했을 때 보여지는 JSP page를 errorPage라 한다.

```
<%@ page errorPage="errorPage`s local URL" %>
```

erroPage의 경로는 상대경로와 절대경로 모두다 사용이 가능하다.



절대 경로 표시 : <%@ page errorPage="/error/errView.jsp" %>
➔ resin의 경우 c:/resin/doc를 기준으로 잡혀있는데 이것은 resin의 root Context를 doc로 하고 있기 때문이다. Tomcat의 경우는 c:/tomcat/webapps/ROOT가 root Context이다.

상대 경로 표시 : <%@ page errorPage="../../error/errView.jsp" %>

1-11. isErrorPage 속성 : errorPage 속성이 예외가 발생할 때 보여줄 error page 지정할 때 사용한다면, isErrorPage 속성은 현재 page가 에러 page인지 아닌지의 여부를 지정할 때 사용한다.

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

기본값은 false이기 때문에, error page가 아닐 경우에는 특별히 이 속성을 지정할 필요가 없다.

zeroDivide1.jsp

```
<%@ page errorPage="divideException.jsp" %>
<html>
<head>
<title>고의로 0으로 나눔</title>
</head>
<body>
1 / 0 = <%= 1 / 0 %> → Exception 발생 code
</body>
</html>
```

➔ exception이 발생할 경우 divideException.jsp을 보여주게 된다.

divideException.jsp

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>나누기 예외 발생</title>
</head>
<body>
zeroDivide1.jsp 에서 다음과 같은
예외가 발생하였습니다.
<br>
- <%= exception.getMessage() %>
</body>
</html>
```

→ - /by zero 가 출력된다.

exception은 isErrorPage 속성의 값을 true로 지정했을 때 사용할 수 있는 기본 객체이다.
page directive의 errorPage 속성과 isErrorPage 속성을 사용하여 별도의 error page를 사용하여 예외를 처리할 경우의 장점은 현재 웹 사이트에서 사용되는 디자인을 사용하여 예외가 발생한 것을 표현할 수 있다.

1-12. pageEncoding 속성 : JSP 1.2규약에 새로 추가된 속성.
JSP page에서 사용하는 char encoding을 지정할 때 사용한다.

```
<%@ page pageEncoding="EUC-KR" %>
```

기본값은 ISO-8859-1을 사용하고 있다.

```
<%@ page pageEncoding="EUC-KR" %>
<%@ page contentType="text/html" %>
```

위의 두 가지 속성값을 모두 표현한 contentType 설정과 같은 효과를 얻는다.

```
<%@ page contentType="text/html ;charset=EUC-KR" %>
```

이 속성의 주의할 점은 JSP Container마다 이 속성을 처리하는 방식이 다르다. 실제 tomcat4.0의 경우 contentType의 속성값을 text/html ;charset=euc-kr로 지정했을 경우 한글문제가 발생하지 않지만, pageEncoding 속성의 값을 euc-kr로 지정하게 되면 한글이 깨지는 문제가 발생하게 된다.

2. include directive

다른 page의 내용을 현재 page에 포함하고자 할 때에 사용된다.
include directive는 file속성을 가지고 있다.

```
<%@ include file="포함할 page`s local URL" %>
```

간단한 사용 예

includeExample.jsp

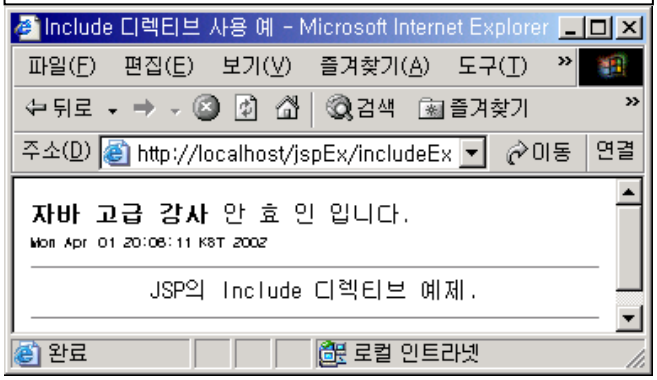
```
<html>
<head>
<title>Include 디렉티브 사용 예</title>
</head>
<body>

<%@ include file="header.jsp" %>

<hr size="1">
<center>
JSP의 Include 디렉티브 예제.
</center>
<hr size="1">
</body>
</html>
```

header.jsp

```
<b>자바 고급 강사 </b>안 효 인 입니다.<br>
<font size="1">
<%= new java.util.Date() %></font>
```



Include directive의 처리

JSP Container는 JSP page에 있는 include directive를 만나게 되면, file 속성에 지정된 file을 현재 위치에 삽입한 후 JSP page를 servlet file로 변환한다.

servlet file로 변환 되기 직전의 jsp page의 예

```
<html>
<head>
<title>Include 디렉티브 사용 예</title>
</head>
<body>
<b>자바 고급 강사 </b>안 효 인 입니다.<br>
<font size="1"><%= new java.util.Date() %></font>
<hr size="1">
<center>
JSP의 Include 디렉티브 예제.
</center>
<hr size="1">
</body>
</html>
```

Include directive의 file 속성에 표시된 값이 그대로 포함된 후 servlet으로 변환되기 때문에, 포함하는 JSP page와 포함될 문서를 하나의 JSP page로 봐도 무리가 없다.

Include directive 사용시 주의점

file 속성에 명시한 file이 변경 되더라도 그 변경된 결과가 반영되지 않는다. 이는 JSP Container가 JSP page가 변경된 경우에만 JSP page를 재컴파일하기 때문이다. 따라서, file 속성에 명시한 file이 변경될 경우 JSP page를 조금 수정해야만 변경된 결과를 확인 할 수 있다.

3. taglib directive

taglib directive는 JSP page에서 사용할 태그 라이브러리를 지정할 때 사용된다.
태그 라이브러리는 JSP의 기능을 확장하기 위해 사용되는 Custom Tag의 집합이다.

```
<%@ taglib uri="태그 라이브러리 URI" prefix="이름 공간 구분자" %>
```

uri 속성은 TLD(Tag Library Descrptor) file의 위치를 나타내며, prefix 속성은 page에 잇는 라이브러리의 모든 태그 앞에 붙게 되는 XML의 이름 공간(namespace) 구분자를 지정한다.
다음 예의 taglib directive는 local URL/EncodeTag를 통해서 참조할 수 있는 TLD file이 제공하는 태그 라이브러리를 읽어 들인다.

```
<%@ taglib uri="/EncodeTag" prefix="antag" %>
```

지정한 uri 속성에서 지정한 TLD file이 존재하지 않거나, 혹은 태그 라이브러리에 존재하지 않는 태그를 사용할 경우, JSP Container는 Error를 발생한다.

JSP의 기본 객체

JSP Container는 개발자가 정의한 객체 뿐만 아니라 정의하지 않아도 사용할 수 있는 9개의 내부 객체를 제공한다. 이러한 객체를 기본 객체(implicit object)라 한다.

➢ JSP Container가 제공하는 기본 객체

객 체 명	Class or Interface	설 명
request	javax.servlet.ServletRequest	Parameter 를 포함한 요청 data
response	javax.servlet.Servlet.Response	응답 data
out	javax.servlet.jsp.JspWriter	출력 Stream 의 내용을 출력하는 객체
page	java.lang.Object	현재 요청을 처리하는 page 의 객체
config	javax.servlet.ServletConfig	JSP page 의 ServletConfig 객체
session	javax.servlet.HttpSession	Session 객체
application	javax.servlet.ServletContext	Servlet 의 config 객체로부터 구할 수 있는 Servlet Context
pageContext	javax.servlet.jsp.PageContext	현재 JSP page 의 page Context
exception	java.lang.Throwable	Error page 에서 사용되는 예외 객체

[표1. 기본 객체]

*** 기본 객체의 구분**

입출력 관련, servlet 관련, Context 관련, 예외 관련 기본 객체로 나뉜다.

위의 기본 객체 중에서 pageContext, request, session, application는 임의의 속성(attribute)값을 저장하고 읽을 수 있는 method를 제공한다. 속성값을 저장하고 읽을 수 있는 능력은 JSP page들 간에 또는 JSP page와 Servlet간에 정보를 주고 받을 수 있도록 해 준다.

M e t h o d	설 명
void setAttribute(key, value)	키 값이 key 인 객체에 value 를 저장한다.
Object getAttribute(key)	키 값이 key 인 객체를 읽어 들인다.
void removeAttribute(key)	키 값이 key 인 객체를 삭제한다.
Enumeration getAttributeNames()	관련된 모든 속성의 이름을 읽어 들인다.

[표2. 속성을 저장하고 읽을 때 사용되는 공통 method]

위의 표 중에서 value는 Object type을 갖는다. 따라서 모든 타입의 객체를 저장할 수 있으며, int, double, boolean등의 기본 타입을 저장하거나 읽어올 때에는 Integer, Double, Boolean과 같은 Wrapper class를 사용한다.

1. request, response, out 기본 객체

1-1. request 기본 객체 : client가 HTTP 기반의 요청을 할 경우 client가 요청한 URL, header 내용, Cookie 그리고 요청과 관련된 parameter 등의 정보에 접근할 수 있는 method를 제공.
request는 javax.servlet.ServletRequest interface를 구현하고 있으며, HTTP protocol을 사용하는 경우 javax.servlet.http.HttpServletRequest를 구현해야 한다.
표2에 해당하는 method외에 request 객체는 요청 parameter를 읽는 method, HTTP Header와 관련된 method, 그 외 요청과 관련된 method를 제공한다.

㉠ parameter의 처리

Method	설 명
getParameterNames()	모든 parameter 의 이름을 구한다.
getParameter(String name)	**지정한 이름을 가진 parameter 중 첫 번째 parameter 를 구한다.**
getParameterValues(String name)	지정한 이름을 가진 parameter 의 모든 값을 String []로 구한다.

paratest.html
```
당신의 이름? <input type="text" name="name" size="25"><p>
다음 중 좋아하는 동물을 선택하세요.<br>
<input type="checkbox" name="favor" value="호랑이">호랑이
<input type="checkbox" name="favor" value="독수리">독수리
<input type="checkbox" name="favor" value="토끼">토끼
```

paraEx.jsp
```
Enumeration parameterNames = request.getParameterNames();
request.getParameter("name");
String[] favor = request.getParameterValues("favor");
```

㉡ 속성 처리 : [표2]를 이용한 속성처리

reqAttribute.jsp
```
<%@ page import = "java.util.Date" %>
<html>
<head><title>request 기본 객체에서 속성 사용</title></head>
<body>
<%
        Date value1 = (Date)request.getAttribute("currentDate");
%>
현재 request 기본 객체의 "currentDate" 속성 값: <%= value1 %>
<p>
<%
        request.setAttribute("currentDate", new Date());
        value1 = (Date)request.getAttribute("currentDate");
%>
값 설정 후, "currentDate" 속성 값: <%= value1 %>
<p>
<%
        request.removeAttribute("currentDate");
        value1 = (Date)request.getAttribute("currentDate");
%>
값 삭제 후, "currentDate" 속성 값: <%= value1 %>
</body>
</html>
```

㉢ Header 및 기타 요청 관련 method

Method	설 명
getHeaderNames()	요청과 관련된 모든 header 의 이름을 구한다.
getHeader(name)	이름이 name 인 header 의 값을 String 으로 구한다.
getHeaders(name)	이름이 name 인 모든 header 의 값을 String[]로 구한다.
getIntHeader(name)	이름이 name 인 header 의 값을 int 형으로 구한다.
getDateHeader(name)	이름이 name 인 header 의 값을 long 형으로 구한다.
getCookies()	요청과 관련된 모든 쿠키를 구한다.

[표3. Header 정보를 구할 수 있는 method]

Method	설 명
getMethod()	요청 방식이 GET 인지 POST 인지 구한다.
getRequestURI()	HTTP 요청 URL 에서 줄에 있는 Query 문자를 제외한 부분을 구한다.
getQueryString()	요청한 URL 다음에 오는 쿼리 문자열을 구한다.
getSession(flag)	요청과 관련된 세션 객체를 구한다. 만약 세션이 존재하지 않고 flag 가 true 이면 새로운 세션 객체를 생성한다.
getRequestDispatcher(path)	지정한 로컬 URL 에 대한 요청 디스패처(dispatcher)를 구한다.
getRemoteHost()	요청한 호스트의 완전한 이름을 구한다.
getRemoteAddr()	요청한 호스트의 네트워크 주소를 구한다.
getRemoteUser()	요청한 사용자의 이름이 존재할 경우 구한다.
getSession()	요청과 관련된 세션 객체를 구한다. 만약 존재하지 않으면 새로 생성한다.
getCharacterEncoding()	요청에서 사용된 인코딩을 구한다.
setCharacterEncoding(env)	요청에서 사용된 인코딩을 env 로 지정한다.

[표4. 기타 요청에 관한 method]

```
<html>
<head><title>요청과 관련된 method</title></head>
<body>
<h1>요청 관련 정보</h1><font size="4">
요청 방식: <%= request.getMethod() %><br>
요청 URI: <%= request.getRequestURI() %><br>
요청 프로토콜: <%= request.getProtocol() %><br>
Servlet 경로: <%= request.getServletPath() %><br>
요청 쿼리 스트링: <%= request.getQueryString() %><br>
서버 이름: <%= request.getServerName() %><br>
서버 포트: <%= request.getServerPort() %><br>
클라이언트 접속 주소: <%= request.getRemoteAddr() %><br>
클라이언트 접속 호스트: <%= request.getRemoteHost() %><br>
인증 스키마: <%= request.getAuthType() %> <br>
로케일: <%= request.getLocale() %><hr>
사용하는 브라우저: <%= request.getHeader("User-Agent") %><hr>
</font>
</body>
</html>
```

reqMethod.jsp

1-2. response 기본 객체 : JSP page를 처리한 결과를 사용자에게 보낼 때 사용되는 응답을 나타냄. Header, Cookie, 응답과 관련된 정보를 설정.

㉠ Header 및 응답의 컨텐트 타입 설정과 관련된 method

Method	설 명
addCookie(Cookie)	응답에 지정한 cookie 를 저장한다.
containsHeader(header)	이름이 header 인 Header 를 포함하고 잇는 지 검사한다.
setHeader(name, value)	이름이 name 인 header 의 값을 value 로 지정한다.
setIntHeader(name, value)	이름이 name 인 header 의 값을 int 형 값인 value 로 지정한다.
setDateHeader(name, date)	이름이 name 인 header 의 값을 long 형 값인 date 로 지정한다.
addHeader(name, value)	이름이 name 이고 값이 String 형 value 인 header 를 추가한다.
addIntHeader(name, value)	이름이 name 이고 값이 int 형 value 인 header 를 추가한다.
addDateHeader(name, date)	이름이 name 이고 값이 long 형 date 인 header 를 추가한다.

사용 예 : 웹 브라우저가 cache를 사용하지 않도록 설정

```
<%
      response.setHeader("Pragma", "no-cache");
      if(request.getProtocol().equals("HTTP/1.1")){
          response.setHeader("Cache-Control", "no-cache");
      }
%>
```

㉡ URL 재작성과 관련된 method

웹 브라우저는 대부분 cookie를 기본적으로 지원하기 때문에 encodeRedirectURL(url) method와 encodeURL(name)는 잘 사용되지 않는다.
sendRedirect(url) method는 page를 자동적으로 이동시켜야 하는 경우에 많이 사용된다.

```
<%
        String val = request.getParameter("val");
        if (val == null || val.equals("")) {
                response.sendRedirect("http://localhost:8080/");
        } else if (val.equals("a")) {
                response.sendRedirect("http://localhost:8080/requestMethod.jsp");
        } else {
%>
<html>
<head><title>오류</title></head>
<body>
잘못된 파라미터를 입력하셨습니다.<p>
파라미터 "val"의 값은 없거나 "a"이어야 합니다.
</body>
</html>
<%
        }
%>
```

1-3. out 기본 객체 : JSP page의 결과를 client에 전송해주는 출력 Stream을 나타낸다. JSP page가 client에게 보내는 모든 정보는 out객체를 통해서 전달된다.
JSP page의 스크립트릿에서 직접 어떤 내용을 출력하고자 할 때 사용된다.

- out을 이용한 출력과 <%= %>를 이용한 출력의 비교

1. out.print()를 이용한 출력.

```
<%!
        java.util.Random rand = new java.util.Random();
%>
<html>
<head>
<title>out 기본 객체 사용</title>
</head>
<body>
<%
        int num = rand.nextInt(10); // 0 부터 9사이의 임의 정수 생성
        out.println("첫번째 임의의 정수: "+num);
        out.println(",");
        num = rand.nextInt(10);
        out.println("두번째 임의의 정수: "+num);
%>
</body>
</html>
```

2. <%= %>를 이용한 출력.

```
<%
        int num = rand.nextInt(10); // 0 부터 9사이의 임의 정수 생성
%>
첫번째 임의의 정수: <%= num %><br>
<%
        num = rand.nextInt(10);
%>
두번째 임의의 정수: <%= num %>
```

스크립트릿 내에서 out을 이용하여 내용을 출력하는 것은 가능한 한 자제해야 하며 JSP page의 장점을 해치지 않는 범위에서 out객체를 사용해야 한다.(out을 많이 사용하면 Servlet 소스코드에 근접하게 되고, 이는 JSP의 장점을 이용하지 못하는 결과를 초래한다.)

- 출력 버퍼와 관련된 method

<table>
<tr><th>Method</th><th colspan="2">설 명</th></tr>
<tr><td>isAutoFlush()</td><td colspan="2">이 method 의 리턴값은 page directive 의 autoFlush 속성에서 지정한 값에 따라 결정된다. 즉 버퍼가 다 찼을 때 자동으로 출력 버퍼를 flush 할 경우 true 를 return.</td></tr>
<tr><td>getBufferSize()</td><td>출력 버퍼의 크기를 바이트 단위로 return.</td><td rowspan="2">현 JSP page 에 알맞은 버퍼의 크기를 계산할 때 사용</td></tr>
<tr><td>getRemaining()</td><td>출력 버퍼의 남아 있는 크기를 return.</td></tr>
<tr><td>clearBuffer()</td><td colspan="2">출력 버퍼의 내용을 지운다.(버퍼에 쌓여 있는 내용을 출력하지 않고 삭제)</td></tr>
<tr><td>clear()</td><td colspan="2">출력 버퍼의 내용을 지우고, 만약 버퍼가 이미 flush 되었다면 에러를 발생한다.</td></tr>
<tr><td>newLine()</td><td colspan="2">출력 버퍼에서 줄 구분자를 출력한다.</td></tr>
<tr><td>close()</td><td colspan="2">출력 버퍼를 닫고, 나머지 내용을 stream 에 보낸다.</td></tr>
</table>

usedBuffer.jsp

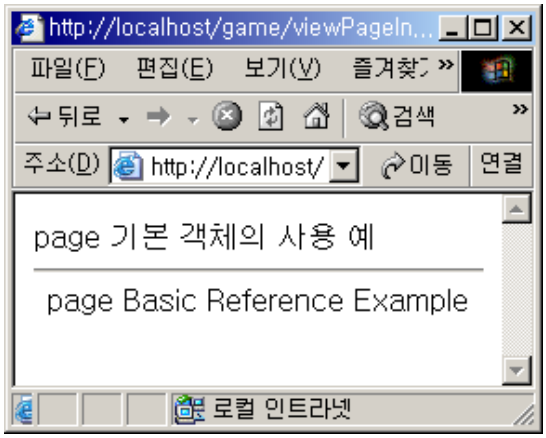
2. request, response, out 기본 객체

→ JSP 규약은 page기본 객체와 config 기본객체를 사용해서 JSPpage가 변환된 Servlet 클래스와 관련된 내용에 접근할 수 있도록 해 준다.

2-1. page 기본 객체 : JSP page를 변환한 Servlet class의 instance. 즉 JSP page자체를 나타낸다. JSP page를 변환한 모든 Servlet class들은 javax.servlet.jsp.JspPage interface를 구현해야 하며, Http 기반으로 할 경우 javax.servlet.jsp.HttpJspPage interface를 구현해야 한다. 이 말은 page 기본객체를 통해서 HttpJspPage interface가 정의한 method를 호출할 수 있다.

JSPpage는 스크립트언어를 자바로 사용하기 때문에 page 기본객체를 사용하지 않아도 Servlet 객체의 method를 호출할 수 있기 때문에 잘 사용하지 않는다. 단, 자바가 아닌 다른 언어를 스크립트 언어로 사용할 경우에는 반드시 사용해야만 한다.

```
<%@ page info = "page Basic Reference Example"
        contentType = "text/html;charset=EUC-KR"
%>
<html>
  <head><title></title></head>
  <body>
    page 기본 객체의 사용 예
    <hr>
    <center>
<%= ((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>
    </center>
  </body>
</html>
```



2-2. config 기본 객체 : JSP page를 변환한 Servlet을 초기화할 때 사용되는 초기화 파라미터를 저장하고있다.

Method	설 명
getInitParameterNames()	모든 초기화 파라미터의 이름을 return.
getInitParameter(name)	이름이 name 인 초기화 파라미터의 값을 return.
getServletName()	Servlet 의 이름을 return.
getServletContext()	ServletContext 객체를 return.

config 객체는 Servlet에서는 의미가 있을 지 몰라도, JSP내에서는 큰 의미를 갖지 못한다.

ConfigExam1.jsp

```
<html>
<head><title>Config 사용 예1</title></head>
<body>
Servlet 이름: <%= config.getServletName() %><br>
</body>
</html>
```

ConfigExam2.jsp

```
<html>
<head><title>Config 사용 예2</title></head>
<body>
Servlet 이름: <%= config.getServletName() %><br>
</body>
</html>
```

```
<servlet-mapping>
  <servlet-name>
      jsp
  </servlet-name>
  <url-pattern>
      *.jsp
  </url-pattern>
</servlet-mapping>
```

→ **web.xml**의 내용

엪 예제의 결과는 모두 **Servlet 이름 : jsp**로 같은 결과가 나온다.

이는 web.xml에 모든 변환되는 Servlet의 이름을 jsp로 해 놓았기 때문이다.

따라서 config객체도 큰 의미를 갖지 못한다.

3. session, application, pageContext 기본 객체

session 기본 객체는 현재 그 page를 요청한 사용자와 관련된 session 정보를 담고 있는 객체.
application 기본 객체는 page가 실행되고 있는 ServerSide Context를 제공.
pageContext 기본 객체는 JSP page자체의 context에 초점을 맞추고 있으며, 그 JSP page에서 사용 가능한 모든 다른 기본 객체와 속성을 프로그래밍을 통해서 접근할 수 있도록 해 준다.

3-1. session 기본 객체 : 각 client의 현재 session을 나타내며, session과 관련된 정보를 저장.
web application에서 지속적으로 사용하는 정보를 session에 저장한다.javax.servlet.http.HttpSession interface의 instance이다.

속성과 관련된 method == 8page 표2 참조

Session 관리를 위해 제공하는 method

Method	설 명
getId()	Session ID 를 return. Session ID 는 각각의 client 마다 고유한 값이 할당된다.
getCreationTime()	session 이 생성된 시간을 long 형으로 return.
getLastAccessedTime()	session 과 관련된 마지막 요청을 받은 시간을 long 형으로 return.
getMaxInactiveInterval()	두 요청 사이에 session 이 유지되기 위한 최대 시간을 초단위로 return.
setMaxInactiveInterval()	두 요청 사이에 session 이 유지되기 위한 최대 시간을 초단위로 지정.
isNew()	사용자의 브라우저가 session ID 를 아직 할당 받지 않았을 경우 true 를 return.
invalidate()	Session 을 버리고, 속성으로 저장된 모든 객체를 풀어준다.

Session은 주로 사용자와 관련된 정보를 여러 JSP page에서 사용할 수 있도록 하기 위해서 사용된다.
정보의 저장은 속성을 통해서 이루어진다.
session 기본객체는 하나의 web application에 속해 있는 모든 JSP/Servlet이 하나의 session 기본 객체를 공유한다.

setSessionAttr.jsp

```
<html>
<head><title>Session에 속성값 추가</title></head>
<body>
세션에 속성값을 추가합니다.<br>
속성 ("newAttr", "new Attr 속성의 값")<br>
<%
        session.setAttribute("newAttr", "new Attr 속성의 값");
%>
</body>
</html>
```

getSessionAttr.jsp

```
<html>
<head><title>Session으로부터 속성값 읽음</title></head>
<body>
<%  String attrValue = (String)session.getAttribute("newAttr");
%>
session 기본 객체의 "newAttr" 속성의 값:
<br>
<%= attrValue %>
</body>
</html>
```

JSP Container는 기본적으로 session의 time out시간을 30분으로 지정하고 있다.
Session의 time out시간을 변경하고자 할 경우 setMaxInactiveInterval()을 사용하면 된다.
ex) <% session.setMaxInactiveInterval(60*60*1); %> ➔ 1시간으로 지정.

JSP page에서 session과 관련된 기본 객체인 session을 사용하기 위해서는 page directive의 session 속성을 true로 지정해 주어야 한다.
<%@ page session = "true" %>
만약 위와 같이 true로 지정하지 않고 session 객체를 사용하게 되면 Servlet으로 변환하고 compile하는 과정에서 error가 발생한다.

3-2. application 기본 객체 : JSP page가 속해 있는 application or ServletContext를 나타낸다.
JSP page는 그 page의 URL에 따라서 여러 개의 application으로 분류 된다. 일반적으로 JSP container는 URL에 있는 첫 번째 directory의 이름을 application으로 사용한다.
ex) http://www.test.com/examples/cart.jsp과 http://www.test.com/examples/list.jsp의 두 JSP page가 있다고 할 경우, 이 두 page는 examples라는 application에 속하게 된다.
application 기본 객체는 javax.servlet.ServletContext interface의 instance이다.

Method	설 명
getServerInfo()	Servlet container 의 이름과 버전을 return.
getMajorVersion()	Servlet container 의 majorversion 을 return.
getMinorVersion()	Servlet container 의 minorversion 을 return.

viewContainerInfo.jsp

```
<html>
<head><title>application 기본 객체 사용</title></head>
<body>
JSP 콘테이너 이름: <%= application.getServerInfo() %>
<br>
JSP 콘테이너 메이저 버전: <%= application.getMajorVersion() %>
<br>
JSP 콘테이너 마이너 버전: <%= application.getMinorVersion() %>
</body>
</html>
```

Application 기본 객체는 하나의 application에 속해 잇는 모든 JSP page에서 공통으로 사용되는 정보를 저장하거나 읽을 때 유용하게 사용된다.
application 내에 있는 모든 JSP page와 servlet이 공통으로 사용할 수 있는 속성을 지정하기 위해서는 application 기본 객체의 setAttribute() method를 사용한다.

setApplicationAttr .jsp

```
<%@ page contentType = "text/html;charset=euc-kr" %>
<html>
<head><title>어플리케이션 속성 설정</title></head>
<body>
<%
        Integer rate = new Integer(98);
        application.setAttribute("commonInfo", rate);
%>
[<%= application.getServletContextName() %>] 콘텍스트의 "commonInfo" 속성 설정
</body>
</html>
```

getApplicationAttr.jsp

```
<%@ page contentType = "text/html;charset=euc-kr" %>
<html>
<head><title>어플리케이션 속성 읽음</title></head>
<body>
<%
        Integer rate = (Integer)application.getAttribute("commonInfo");
%>
[<%= application.getServletContextName() %>] 콘텍스트의 "commonInfo" 속성 값:
<%= rate %>
</body>
</html>
```

위의 두 JSP page를 tomcat/webapps/ROOT에 복사한 후 set → get 순으로 실행해 보고, → 98
getApplicationAttr.jsp를 tomcat/webapps/examples에다가 복사한 후 다시 실행해 본다. → null
이렇게 되는 이유는 하나는 ROOT에 하나는 examples라는 application에 속하기 때문에 발생한다.

Method	설 명
getServletContextName()	현재 JSP page(또는 servlet)가 속해 있는 application 의 이름을 return.
getMimeType(filename)	지정한 파일의 MIME type 을 return.
getResource(path)	지정한 URL 에 접근할 수 있는 객체를 return.
getResourceAsStream(path)	저정한 URL 로부터 내용을 읽어 들이는 입력 Stream 을 return.
getRealPath(path)	Local file system 의 path 를 local URL 로 변경한다.
getContext(path)	지정한 local URLdp 해당하는 application context 를 return.
getRequestdispatcher(path)	지정한 local URLdp 해당하는 요청 dispatcher 를 생성한다.
log(message)	log file 에 message 를 기록한다.
log(message, exception)	log file 에 message 와 예외 message 를 기록한다.

3-3. pageContext 기본 객체 : 현재 JSP page의 context를 나타내며, pageContext 기본 객체를 통해서 모든 다른 기본 객체에 접근할 수 있다.
ex) <% HttpSession httpSession = pageContext.getSession() ; %>
위의 경우 httpSession은 session 기본 객체와 완전히 동일한 객체이다.

pageContext는 javax.servlet.j네.PageContext class의 instance이다.

pageContext 기본 객체는 다른 page로 제어를 이동시키거나 다른 page를 포함할 수 있는 method를 지원한다.
그러나 제어 이동이나 page 포함은 action tag인 <jsp:forward>와 <jsp:include> 실제로 사용.
dispatcher와 관련된 pageContext의 method

Method	설 명
forward(path)	지정한 URL 로 forwarding 한다.
include(path)	지정한 local URL 을 처리한 결과를 출력 Stream 에 포함한다.

기본 객체에 접근할 수 있도록 해 주는 pageContext의 method

Method	설 명
getPage()	현재 page 의 servlet instance 를 return.(page 기본 객체)
getRequest()	Page 의 요청에 해당하는 객체를 return.(request 기본 객체)
getResponse()	Page 의 응답에 해당하는 객체를 return. (response 기본 객체)
getOut()	Page 의 출력 Stream 을 return.(out 기본 객체)
getSession()	현재 요청과 관련된 session 객체를 return.(session 기본 객체)
getServletConfig()	Servlet 의 설정 관련 객체를 return.(config 기본 객체)
getServletContext()	Page 의 servlet 이 실행되고 있는 context 를 return. (application 기본 객체)
getException()	Error page 일 경우, page 에 전달된 예외 객체를 return. (exception 기본 객체)

pageContext 기본 객체는 기본 속성을 지정하는 method뿐만 아니라 pageContext가 아닌 다른 영역(scope)에 있는 속성에 접근할 수 있다.
예를 들어 pageContext 기본 객체는 session 영역에 있는 속성(즉, session 기본 객체와 관련된 속성)에 접근이 가능하다. 이처럼 다른 영역에 있는 속성에 접근할 수 있는 것은 pageContext 기본객체가 다른 기본 객체에 접근할 수 있기 때문이다.

여러 영역의 속성에 접근할 수 있도록 해 주는 pageContext 기본 객체 method

Method	설 명
setAttribute(key, value, scope)	지정한 영역에 키가 key 이고 값이 value 인 속성을 저장.
getAttribute(key, scope)	지정한 영역에서 키 값이 key 인 속성을 읽어서 return.
removeAttribute(key, scope)	지정한 영역에서 키 값이 key 인 속성을 삭제.
findAttribute(name)	모든 영역에서 지정한 이름을 가진 속성을 찾는다.
getAttributesScope(name)	지정한 이름을 가진 속성이 저장된 영역을 return.
getAttributeNamesInScope(scope)	지정한 영역에 있는 모든 속성의 이름을 return.

위의 표 중에서 scope에 해당하는 상수 값.

상 수	설 명
PAGE_SCOPE	pageContext 객체에 저장된 속성에 대한 영역.
REQUEST_SCOPE	request 객체에 저장된 속성에 대한 영역.
SESSION_SCOPE	session 객체에 저장된 속성에 대한 영역.
APPLICATION_SCOPE	application 객체에 저장된 속성에 대한

ex)pageContext기본 객체를 사용하여 application 기본 객체에 저장되어 있는 모든 속성의 이름 구하기

```
<%@ page
        contentType = "text/html;charset=euc-kr"
        import = "javax.servlet.jsp.Pagecontext"
%>
.....................
<%
    Enumeration attributes =
         pageContext.getAttributeNamesInScope(PageContext.APPLICATION_SCOPE) ;
    while(attributes.hasMoreElements()){
%>
Application 속성 이름 : <%= attributes.nextElement() %><br>
<%
    }
%>
.............
```

영역은 그 속성이 어느 기간 동안 유지되는지를 결정한다.
pageContext 기본 객체에 저장된 속성의 경우, pageContext 객체가 나타내는 page를 처리하는 동안에만 유효하다.
request 기본 객체에 저장된 속성의 경우는, 요청의 처리가 끝날 때 까지 유효하다.(즉 forwarding을 해도 request 기본 객체에 저장된 속성은 변경되지 않는다.)
session 기본 객체에 저장된 속성은 사용자가 웹 서버와 상호 작용을 하는 한 지속적으로 츄지된다.
application 기본 객체에 저장된 속성은 JSP Container가 application의 page를 한 개 이상 메모리에 읽어 들이고 있는 한 지속된다. 즉, JSP Container가 실행되고 있는 동안 application과 관련된 속성이 유지된다.

4. exception 기본 객체

exception 기본 객체 : JSP page에서 발생하는 예외를 나타내며, error page에서 사용된다.
exception 기본 객체를 사용하기 위해서는 반드시 page directive에서 isErrorPage의 값을 true로 설정이 되어 있어야만 사용할 수 있다.
어떤 JSP page에서 에러가 발생할 경우, 그 page에서 지정한 error page에 보낼 예외 객체(java.lang.Throwable class의 instance)를 생성한다. 이 예외 객체가 바로 exception 기본 객체가 된다.

```
<%@ page isErrorPage = "true" %>
.................
<h2>에러 발생</h2>
에러 : <%= exception.toString() %>
.............
```

Exception은 java.lang.Throwable class를 상속 받은 class의 instance 이므로 getMessage(), toString(), printStackTrace()등의 method를 사용할 수 있다.