



CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

JSP: JAVA SERVER PAGES

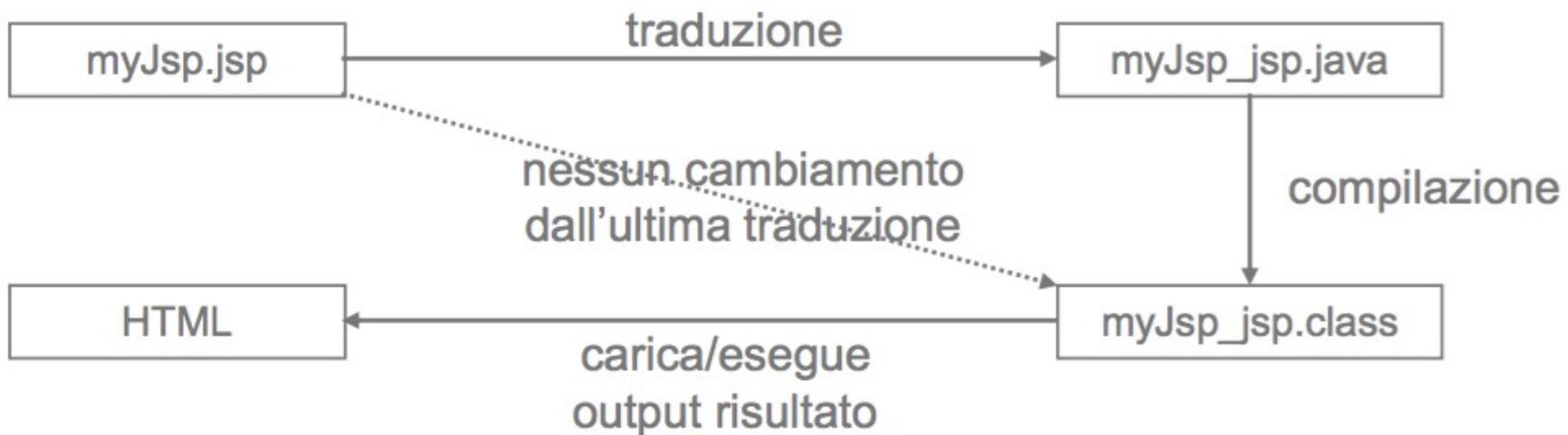
a.a. 2020-2021

Java Server Pages

- Le JSP sono uno dei due componenti di base della tecnologia J2EE, relativamente alla parte Web:
 - *template per la generazione di contenuto dinamico*
 - *estendono HTML con codice Java custom*
- Quando viene effettuata una richiesta a una JSP:
 - parte dell'HTML viene direttamente trascritta sullo stream di output
 - **Il codice Java viene eseguito sul server** per la generazione del contenuto HTML dinamico
 - la pagina HTML così formata (*parte statica + parte generata dinamicamente*) viene restituita al client
- Assimilabili ad un linguaggio di script (es. PHP, Perl, ...)
 - *In realtà vengono trasformate in Servlet dal container*

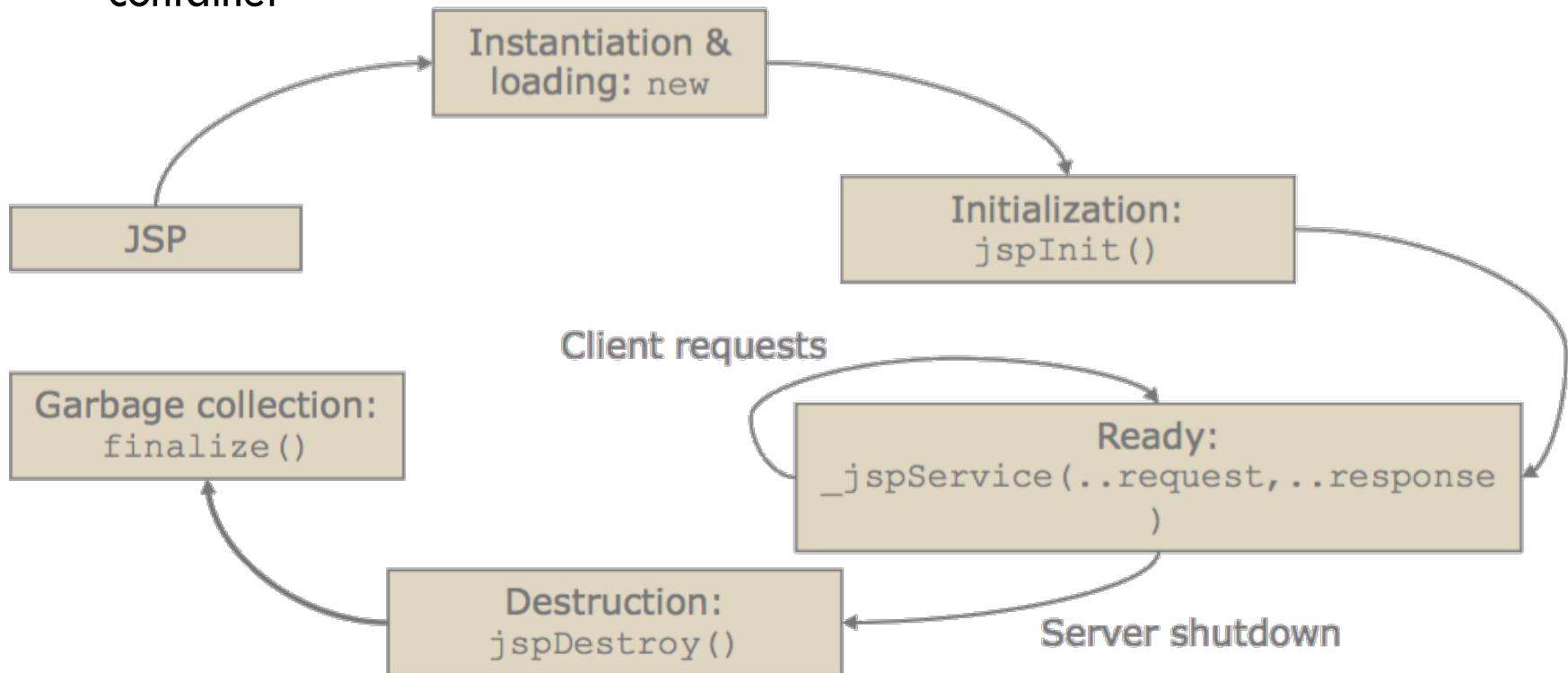
JspServlet

- Le richieste verso JSP sono gestite da una particolare Servlet (in Tomcat si chiama **JspServlet**) che effettua le seguenti operazioni:
 1. traduzione della JSP in una Servlet
 2. compilazione della Servlet risultante in una classe
 3. esecuzione della JSP (.class della Servlet)
- I primi due passi vengono eseguiti solo quando cambia il codice della JSP



Ciclo di vita delle JSP

- Dal momento che le JSP sono compilate in Servlet, il ciclo di vita delle JSP (dopo la compilazione) è controllato sempre dal medesimo Web container



Servlet e JSP: perché usare JSP?

- Nella Servlet la logica per la generazione del documento HTML è implementata completamente in Java
 - Il processo di generazione delle pagine è time-consuming, ripetitivo e soggetto a errori (sequenza di `print()`)
 - L'aggiornamento delle pagine è scomodo
- JSP nascono per facilitare la progettazione grafica e l'aggiornamento delle pagine
 - Si può separare agevolmente il lavoro fra **grafici** e **programmatori**
 - I Web designer possono produrre pagine senza dover conoscere i dettagli della logica server-side
 - La generazione di codice dinamico è implementata sfruttando il linguaggio Java

Servlet o JSP?

- Le **JSP** non rendono inutili le **Servlet**
 - Le Servlet forniscono agli sviluppatori delle applicazioni Web, un completo controllo dell'applicazione
- *Se si vogliono fornire contenuti differenziati a seconda di diversi parametri quali l'identità dell'utente, condizioni dipendenti dalla business logic, etc. è conveniente continuare a lavorare con le Servlet*
- Le JSP rendono viceversa molto semplice presentare documenti HTML o XML (o loro parti) all'utente; dominanti per la realizzazione di pagine dinamiche semplici e di uso frequente

Come funzionano le JSP

- Ogni volta che arriva una **request**, il server compone dinamicamente il contenuto della pagina
- Ogni volta che incontra un tag **<% ... %>**
 - valuta l'espressione Java contenuta al suo interno
 - inserisce al suo posto il risultato dell'espressione
- *Questo meccanismo permette di generare pagine dinamicamente*



Esempio: Hello world

- Consideriamo una JSP, denominata `helloWorld.jsp`, che realizza il classico esempio “Hello World!” in modo parametrico:

```
<html>
  <body>
    <% String visitor=request.getParameter("name");
      if (visitor == null) visitor = "World"; %>
    Hello, <%= visitor %>!
  </body>
</html>
```

<http://myHost/myWebApp/helloWord.jsp>

```
<html>
  <body>
    Hello, World!
  </body>
</html>
```

<http://myHost/myWebApp/helloWord.jsp?name=Mario>

```
<html>
  <body>
    Hello, Mario!
  </body>
</html>
```

Tag

- Le parti variabili della pagina sono contenute all'interno di **tag** speciali
- Sono possibili due tipi di sintassi per questi tag:
 1. **Scripting-oriented tag**
 2. **XML-Oriented tag**
- Le **scripting-oriented tag** sono definite da delimitatori entro cui è presente lo scripting (self-contained)
- Sono di quattro tipi:
 - <%! %> **Dichiarazione**
 - <%= %> **Espressione**
 - <% %> **Scriptlet**
 - <%@ %> **Direttiva**

XML-oriented tag

- XML-oriented tag seguono la sintassi XML
- Sono presenti XML tag equivalenti ai delimitatori visti nella pagina precedente
 - <jsp:declaration>declaration</jsp:declaration>
 - <jsp:expression>expression</jsp: expression>
 - <jsp:scriptlet>java_code</jsp:scriptlet>
 - <jsp:directive.dir_type dir_attribute />
- Nel seguito useremo scripting-oriented tag che sono più diffusi

Dichiarazioni

- Si usano i delimitatori `<%!` e `%>` per dichiarare variabili e metodi
- Variabili e metodi dichiarati possono poi essere referenziati in qualsiasi punto del codice JSP
- I metodi/variabili diventano metodi/variabili della Servlet quando la pagina viene tradotta

```
<%! String name = "Paolo Rossi";
   double[] prices = {1.5, 76.8, 21.5};

   double getTotal() {
       double total = 0.0;
       for (int i=0; i<prices.length; i++)
           total += prices[i];
       return total;
   }

%>
```

Espressioni

- Si usano i delimitatori `<%=` e `%>` per valutare espressioni Java
- Il risultato dell'espressione viene convertito in stringa inserito nella pagina al posto del tag

(continuando l'esempio della pagina precedente)

JSP

```
<p>Sig. <%=name%>,</p>
<p>l'ammontare del suo acquisto è: <%=getTotal()%> euro.</p>
<p>La data di oggi è: <%=new Date()%></p>
```



Pagina HTML risultante

```
<p>Sig. Paolo Rossi,</p>
<p>l'ammontare del suo acquisto è: 99.8 euro.</p>
<p>La data di oggi è: Tue Feb 20 11:23:02 2010</p>
```

Esempio 1 (declarations.jsp in jsps.zip)

```
<%@ page language="java" import="java.util.Date, java.text.SimpleDateFormat"
   contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Declarations</title>
</head>
<body>

<%! String name = "Paolo Rossi";
   double[] prices = {1.5, 76.8, 21.5};

   double getTotal() {
       double total = 0.0;
       for(int i=0; i <prices.length; i++)
           total += prices[i];
       return total;
   }

   String formattedDate(Date today) {
       SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy HH.mm.ss");
       return formatter.format(today);
   }
%>

<h3>JSP Declarations</h3>
<p>Sig. <%=name %>, </p>
<p>l'ammontare del suo acquisto &egrave;; <%=getTotal() %> euro.</p>
<p>La data di oggi &egrave;; <%=formattedDate(new Date()) %></p>
</body>
</html>
```

Esempio 2 (expressions.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Expressions</title>
</head>
<body>
<h3>JSP Expressions</h3>
<ul>
   <li>Current time: <%= new java.util.Date() %>
   <li>Server: <%=application.getServerInfo() %>
   <li>Application name: <%=application.getContextPath() %>
   <li>Session Id: <%=session.getId() %>
   <li>The <code>testParam</code> form parameter:
      <%=request.getParameter("testParam") %>
</ul>
</body>
</html>
```

JSP Expressions

- Current time: Sun May 01 18:43:21 CEST 2016
- Server: Apache Tomcat/8.0.32
- Application name: /jsp
- Session Id: 84F1ED2B7DD60439D6FD5BE749CF0043
- The `testParam` form parameter: null

- <http://myHost/myWebApp/expressions.jsp>
- <http://myHost/myWebApp/expressions.jsp?testParam=TSW>

Scriptlet

- Si usano `<%` e `%>` per aggiungere un frammento di codice Java eseguibile alla JSP (**scriptlet**)
- Lo **scriptlet** consente tipicamente di inserire logiche di controllo di flusso nella produzione della pagina
- La combinazione di tutti gli scriptlet in una determinata JSP deve definire un blocco logico completo di codice Java

`userIsLogged` è una variabile **boolean**

```
<% if (userIsLogged) { %>
    <h1>Benvenuto Sig. <%=name%></h1>
<% } else { %>
    <h1>Per accedere al sito devi fare il login</h1>
<% } %>
```

Esempio 3 (scriptlet.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   info="simple jsp examples" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Scriptlet</title>
</head>
<body>
<h3>JSP Scriptlet</h3>
<ul>
    <li>Current time: <%= new java.util.Date() %>
    <li>Server: <%=application.getServerInfo() %>
    <li>Application name: <%=application.getContextPath() %>
    <li>Session Id: <%=session.getId() %>
    <% String param = request.getParameter("testParam");
    if(param != null) { %>
        <li>The <code>testParam</code> form parameter:
            <%=request.getParameter("testParam") %>
    <% } else { %>
        <li>No form parameter
    <% } %>
        <li>Info: <%=this.getServletInfo() %>
    </ul>
</body>
</html>
```

oppure <%= param %>

Direttive

- Sono comandi JSP valutati a tempo di compilazione
- Le più importanti sono:
 - **page**: permette di importare package, dichiarare pagine d'errore, definire modello di esecuzione JSP relativamente alla concorrenza (ne discuteremo a breve), ecc.
 - **include**: include un altro documento
 - **taglib**: carica una libreria di custom tag implementate dallo sviluppatore
- *Non producono nessun output visibile*

```
<%@ page info="Esempio di direttive" %>
<%@ page language="java" import="java.net.*" %>
<%@ page import="java.util.List, java.util.ArrayList" %>
<%@ include file="myHeaderFile.html" %>
```

La direttiva page

- La direttiva **page** definisce una serie di attributi che si applicano all'intera pagina
- Sintassi:

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

Attributi di page

- **language="java"** linguaggio di scripting utilizzato nelle parti dinamiche, allo stato attuale l'unico valore ammesso è "java"
- **import="{package.class | package.*}, ..."** lista di package da importare
 - Gli import più comuni sono impliciti e non serve inserirli (`java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, `javax.servlet.http.*`)
 - La **virgola ","** è usata come separatore tra le dichiarazioni dei package
- **session="true | false"** indica se la pagina fa uso della sessione (altrimenti non si può usare **session**)
- **buffer="none | 8kb | sizekb"** dimensione in KB del buffer di uscita
- **autoFlush="true | false"** dice se il buffer viene svuotato automaticamente quando è pieno
 - Se il valore è **false** viene generata un'eccezione quando il buffer è pieno

Attributi di page (2)

- **isThreadSafe="true | false"** indica se il codice contenuto nella pagina è thread-safe
 - Se vale **false** le chiamate alla JSP vengono serializzate
- **info="text"** testo di commento
 - Può essere letto con il metodo **Servlet.getServletInfo()**
 - Es: `<%= this.getServletInfo()%>` oppure `<%= page.getServletInfo()%>`
- **errorPage="relativeURL"** indirizzo della pagina a cui vengono inviate le eccezioni
- **isErrorPage="true | false"** indica se JSP corrente è una pagina di errore
 - Si può utilizzare l'oggetto **exception** solo se l'attributo è **true**
- **contentType="mimeType [;charset=charSet]" | "text/html; charset=ISO-8859-1"** indica il tipo MIME e il codice di caratteri usato nella risposta

Esempio 4 (error.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8" isErrorPage="true"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Error</title>
</head>
<body>
<h3> Error</h3>
<% if(exception != null) { %>
<p>An exception was raised: <%= exception.toString() %></p>
<p>Exception message is: <%= exception.getMessage() %></p>
<br>
<%
      StackTraceElement[] st = exception.getStackTrace();
      for(StackTraceElement e: st){
          out.println(e.toString());
      }
    %
%>
</body>
</html>
```



Esempio 4b (call error.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"
   errorPage="error.jsp"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Call Error Page</title>
</head>
<body>
<h3>JSP Call Error Page</h3>
<% String param = request.getParameter("testParam");
   if(param.equals("PW")) { %>
      The <code>testParam</code> form parameter: <%=param %>
   <% } else { %>
      No form parameter
   <% } %>
</body>
</html>
```

- <http://myHost/myWebApp/callerror.jsp> (genera un NullPointerException)
- <http://myHost/myWebApp/callerror.jsp?testParam=PW>

Uncaught exception

La direttiva include

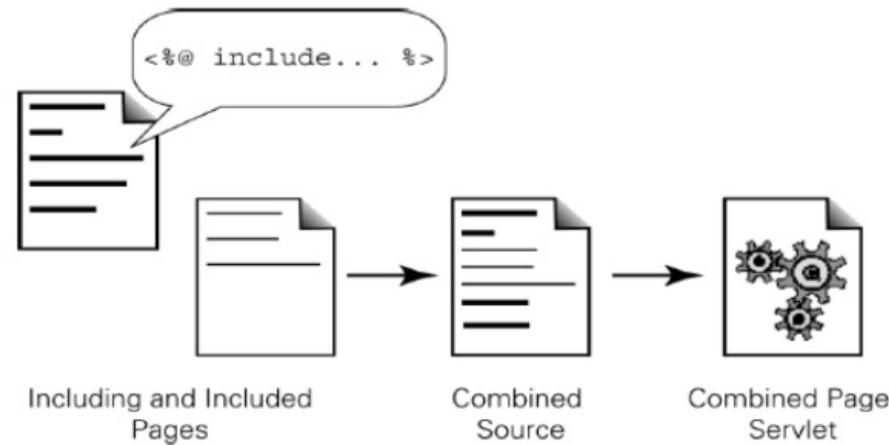
- Sintassi: `<%@ include file = "localURL"%>`
- Serve ad includere il contenuto del file specificato
 - È possibile nidificare un numero qualsiasi di inclusioni
 - L'inclusione viene fatta a tempo di compilazione: eventuali modifiche al file incluso non determinano una ricompilazione della pagina che lo include
- Esempio:

```
<%@ include file="/shared/copyright.html"%>
```

```
<html>
<body>
<%@ include file="header.jsp" %>
<br>
Contact Us at: we@studytonight.com
<br/>
<%@ include file="footer.jsp" %>
</body>
</html>
```

This says insert the complete content of header.jsp into this JSP page

This says insert the complete content of footer.jsp into this JSP page



Esempio 5 (compact.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Include</title>
</head>
<body>
<%@ include file="header.jsp" %>
<br>
Contact Us at: we@studytonight.com<br>
<br>
<%@ include file="footer.jsp" %>
</body>
</html>
```

header.jsp

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

footer.jsp

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

HEADER

Contact Us at: we@studytonight.com

FOOTER

The **<div>** tag defines a division or a section in an HTML document

Esempio (tutti i tag)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>JSP Tags</title>
</head>
<body>
<%@ include file="header.jsp" %>
<h3>JSP Tags</h3>
<%!
    long fact(long n) {
        if(n == 1) return 1;
        return n*fact(n-1);
    }
%>

<% String xStr = request.getParameter("num");
    if(xStr != null) {
        try {
            long x = Long.parseLong(xStr); %>
            Fattoriale: <%= x%>! = <%= fact(x)%>

        } catch (NumberFormatException e) {%
            Il paraemtro <b>num</b> non contiene un valore intero.
        }
    }
%>
<%@ include file="footer.jsp" %>
</body>
</html>
```

Se n è negativo?

<tags.jsp>

[tags.jsp with param \(num = 3\)](tags.jsp?num=3)

[tags.jsp with param \(num = 'a'\)](tags.jsp?num=a)

Direttiva taglib

- Le JSP permettono di definire **tag custom** oltre a quelli predefiniti
- Una taglib è una collezione di questi tag non standard, realizzata mediante una classe Java
- Sintassi: **<%@ uri="tagLibraryURI" prefix="tagPrefix"%>**

```
<%@ uri="http://java.sun.com/jsp/jstl/core" taglib prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:forEach var="i" begin="1" end="5">
    Item <c:out value="${i}" /><br>
</c:forEach>
</body>
</html>
```

Item 1
Item 2
Item 3
Item 4
Item 5

Built-in objects (con scope differenziati)

- Le specifiche JSP definiscono **9 oggetti built-in** (o impliciti) utilizzabili senza dover creare istanze
- Rappresentano utili riferimenti ai corrispondenti oggetti Java veri e propri presenti nella tecnologia Servlet

Oggetto	Classe/Interfaccia
page	javax.servlet.jsp.HttpJspPage
config	javax.servlet.ServletConfig
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
pageContext	javax.servlet.jsp.PageContext
exception	Java.lang.Throwable

Oggetto page

- L'oggetto page rappresenta l'istanza corrente della Servlet
 - Ha come tipo l'interfaccia HTTPJspPage che discende da JSP page, la quale a sua volta estende Servlet
 - Può quindi essere utilizzato per accedere a tutti i metodi definiti nelle Servlet

JSP

```
<%@ page info="Esempio di uso page." %>
<p>Page info:
    <%=page.getServletInfo() %>
</p>
```

Pagina HTML



```
<p>Page info: Esempio di uso di page</p>
```

Oggetto config

- Contiene la configurazione della Servlet (parametri di inizializzazione)
- *Poco usato in pratica in quanto in generale nelle JSP sono poco usati i parametri di inizializzazione*
- Metodi di config:
 - **getInitParameterNames()** restituisce tutti i nomi dei parametri di inizializzazione
 - **getInitParameter(name)** restituisce il valore del parametro passato per nome

Oggetto request

- Rappresenta la richiesta alla pagina JSP
- È il parametro `request` passato al metodo `service()` della servlet
- Consente l'accesso a tutte le informazioni relative alla richiesta HTTP:
 - indirizzo di provenienza, URL, headers, cookie, parametri, ecc.

```
<% String xStr = request.getParameter("num");
try
{
    long x = Long.parseLong(xStr); %>
    Fattoriale: <%= x %>! = <%= fact(x) %>
<%}
catch (NumberFormatException e) { %>
Il parametro <b>num</b>non contiene un valore intero.
<%} %>
```

Oggetto response

- Oggetto legato all'I/O della pagina JSP
- Rappresenta la risposta che viene restituita al client
- Consente di inserire nella risposta diverse informazioni:
 - content type ed encoding
 - eventuali header di risposta
 - URL Rewriting
 - i cookie

```
<%response.setDateHeader("Expires", 0);
response.setHeader("Pragma", "no-cache");
if (request.getProtocol().equals("HTTP/1.1"))
{
    response.setHeader("Cache-Control", "no-cache");
}
%>
```

Oggetto out

- Oggetto legato all'I/O della pagina JSP
- È uno stream di caratteri e rappresenta lo stream di output della pagina
- Esempio:

```
<p>Conto delle uova
<%
    int count = 0;
    while (carton.hasNext())
    {
        count++;
        out.print(".");
    }
%>
<br/>
Ci sono <%= count %> uova.
</p>
```

Metodi dell'oggetto out

- **isAutoFlush()** dice se output buffer è stato impostato in modalità autoFlush o meno
- **getBufferSize()** restituisce dimensioni del buffer
- **getRemaining()** indica quanti byte liberi ci sono nel buffer
- **clearBuffer()** ripulisce il buffer
- **flush()** forza l'emissione del contenuto del buffer
- **close()** esegue il flush e chiude lo stream

Oggetto session

- Oggetto che fornisce informazioni sul contesto di esecuzione della JSP
- Rappresenta la sessione corrente per un utente
- L'attributo **session** della direttiva **page** deve essere **true** affinchè la JSP partecipi alla sessione

```
<% UserLogin userData = new UserLogin(name, password);
   session.setAttribute("login", userData);
%>
<%UserLogin userData=(UserLogin)session.getAttribute("login");
  if (userData.isGroupMember("admin")) {
    session.setMaxInactiveInterval(60*60*8);
  } else {
    session.setMaxInactiveInterval(60*15);
}
%>
```

Oggetto application

- Oggetto che fornisce informazioni su contesto di esecuzione della JSP
(è **ServletContext**)
- Rappresenta la Web application a cui JSP appartiene
- Consente di interagire con l'ambiente di esecuzione:
 - fornisce la versione di JSP Container
 - garantisce l'accesso a risorse server-side
 - permette accesso ai parametri di inizializzazione relativi all'applicazione
 - consente di gestire gli attributi di un'applicazione

Oggetto pageContext

- Oggetto che fornisce informazioni sull'intero contesto di esecuzione della pagina JSP
- Rappresenta l'insieme degli oggetti impliciti di una JSP
 - Consente accesso a tutti gli oggetti impliciti e ai loro attributi
 - Consente trasferimento del controllo ad altre pagine
- Utilizzando questo oggetto è possibile lavorare con gli attributi (e.g., find, get, set, remove) in qualsiasi livello
 1. JSP Page – Scope: **PAGE_CONTEXT** (*default*)
 2. HTTP Request – Scope: **REQUEST_CONTEXT**
 3. HTTP Session – Scope: **SESSION_CONTEXT**
 4. Application Level – Scope: **APPLICATION_CONTEXT**
- Es:

```
<% pageContext.setAttribute("role", "manager", PageContext.SESSION_SCOPE); %>
<% pageContext.getAttribute("mail", PageContext.APPLICATION_SCOPE); %>
<% pageContext.include(<Relative URL Path>); %>
```

Oggetto exception

- Oggetto connesso alla gestione degli errori
- Rappresenta l'eccezione che non viene gestita da nessun blocco try...catch
- Non è automaticamente disponibile in tutte le pagine ma solo nelle Error Page (quelle dichiarate con l'attributo **errorPage** impostato a **true**)
- Esempio:

```
<%@ page isErrorPage="true" %>
<h1>Attenzione!</h1>
E' stato rilevato il seguente errore:<br/>
<b><%= exception %></b><br/>
<%
    exception.printStackTrace(out) ;
%>
```

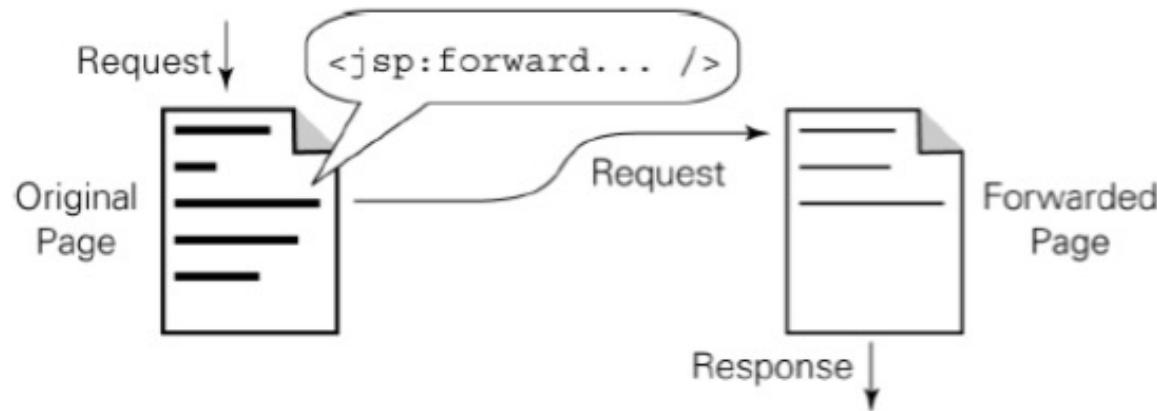
Azioni

- Le azioni sono comandi JSP tipicamente per gestire l'interazione con altre pagine JSP, Servlet, o componenti **JavaBean**; **sono espresse usando sintassi XML**
- Sono previsti **6 tipi di azioni** definite dai seguenti **tag**:
 - **useBean**: istanzia JavaBean e gli associa un identificativo
 - **getProperty**: ritorna la property indicata come oggetto
 - **setProperty**: imposta valore della property indicata per nome
 - **include**: include nella JSP il contenuto generato dinamicamente da un'altra pagina locale
 - **forward**: cede il controllo ad un'altra JSP o Servlet
 - **plugin**: genera contenuto per scaricare plug-in Java se necessario

```
<html>
  <body>
    <jsp:useBean id="myBean" class="it.unibo.deis.my.HelloBean"/>
    <jsp:setProperty name="myBean" property="nameProp" param="value"/>
    Hello, <jsp:getProperty name="myBean" property="nameProp"/>!
  </body>
</html>
```

Azioni: forward

- Sintassi: **<jsp:forward page="localURL" />**
- Consente trasferimento del controllo dalla pagina JSP corrente ad un'altra pagina sul server locale
- L'attributo **page** definisce l'URL della pagina a cui trasferire il controllo
- La request viene completamente trasferita in modo trasparente per il client



Azioni: forward

- È possibile generare dinamicamente l'attributo page
`<jsp:forward page='<%" + message + statusCode + ".html%">'>`
- Oggetti **request**, **response** e **session** della pagina d'arrivo sono gli stessi della pagina chiamante, ma viene istanziato un nuovo oggetto **pageContext**
- *Attenzione: forward è possibile soltanto se non è stato emesso alcun output*
- È possibile aggiungere parametri all'oggetto request della pagina chiamata utilizzando il tag `<jsp:param>`

```
<jsp:forward page="localURL">
  <jsp:param name="parName1" value="parValue1"/>
  ...
  <jsp:param name="parNameN" value="parValueN"/>
</jsp:forward>
```

localURL?parName1=parValue1&parName2=parValue2

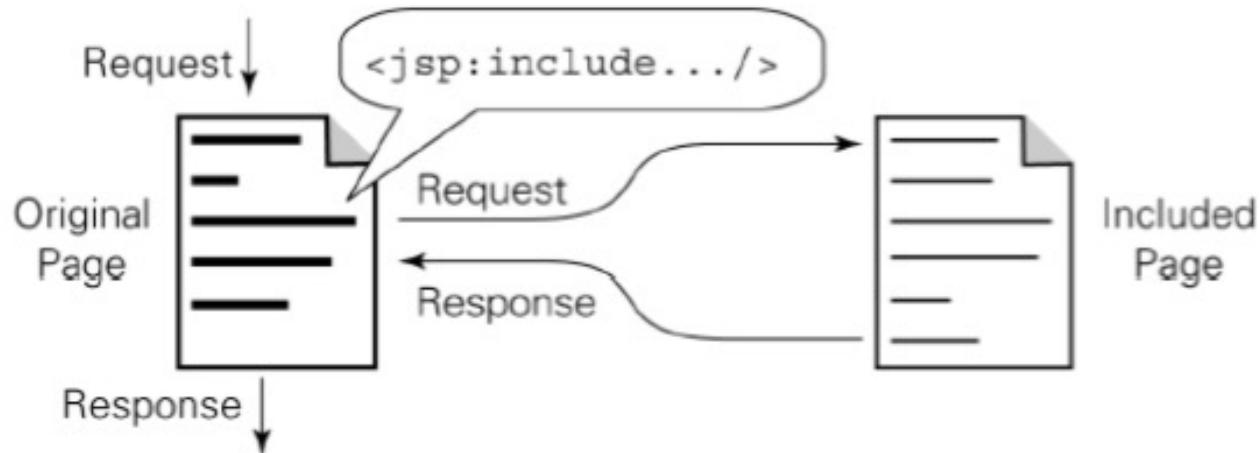
Azioni: include

- Sintassi: **<jsp:include page="localURL" flush="true" />**
- Consente di includere il contenuto generato dinamicamente da un'altra pagina locale all'interno dell'output della pagina corrente
 - Trasferisce temporaneamente controllo ad un'altra pagina
 - L'attributo **page** definisce l'URL della pagina da includere
 - L'attributo **flush** stabilisce se sul buffer della pagina corrente debba essere eseguito il flush prima di effettuare l'inclusione
 - Gli oggetti **session** e **request** (ma non response) per pagina da includere sono gli stessi della pagina chiamante, ma viene istanziato un nuovo contesto di pagina **pageContext**

Azioni: include (2)

- È possibile aggiungere parametri all'oggetto request della pagina inclusa utilizzando il tag **<jsp:param>**

```
<jsp:include page="localURL" flush="true">
  <jsp:param name="parName1" value="parValue1"/>
  ...
  <jsp:param name="parNameN" value="parValueN"/>
</jsp:include> localURL?parName1=parValue1&parName2=parValue2
```



JSP e modello a componenti

- Scriptlet ed espressioni consentono uno sviluppo centrato sulla pagina
 - Questo modello non consente una forte separazione tra logica applicativa e presentazione dei contenuti
 - Applicazioni complesse necessitano di maggiore modularità ed estensibilità, tramite una architettura a più livelli
- A tal fine, JSP consentono anche sviluppo basato su un modello a componenti
- Il modello a componenti:
 - Consente di avere una maggiore separazione fra logica dell'applicazione e contenuti
 - Permette di costruire architetture molto più articolate
 - *Fate mente locale su quanto avete già visto in altri corsi o nelle precedenti lezioni sui componenti...*

JavaBeans

- JavaBeans è il modello di “base” per componenti Java, il più semplice...
- Un **JavaBean**, o semplicemente **bean**, non è altro che una classe Java dotata di alcune caratteristiche particolari:
 - Classe **public**
 - Ha un **costruttore** public di default (senza argomenti)
 - Espone proprietà, sotto forma di coppie di metodi di accesso (**accessors**) costruiti secondo una convenzione standard per i nomi dei metodi (get... set...)
 - La proprietà **prop** è definita da due metodi **getProp()** e **setProp()**
 - Il tipo del parametro di **setProp(...)** e del valore di ritorno di ... **getProp()** devono essere uguali e rappresentano il tipo della proprietà (può essere un tipo primitivo o una qualunque classe Java)
 - Es.: **void setLength(int value)** e **int getLength()** identificano proprietà **length** di tipo **int**
 - ...

JavaBeans (2)

- Se definiamo solo il metodo `get` avremo una proprietà in sola lettura (`read-only`)
- Le proprietà di tipo **boolean** seguono una regola leggermente diversa: metodo di lettura ha la forma `isProp()`
- Es: la proprietà `empty` sarà rappresentata dalla **coppia**
void setEmpty(boolean value) e **boolean isEmpty()**
- Ci sono anche proprietà indicizzate per rappresentare collezioni di valori
- Es: `String getItem(int index)` e `setItem(int index, String value)` definiscono la proprietà indicizzata `String item[]`
- Espone eventi con metodi di registrazione che seguono regole precise
 - e.g., `listener`, lancio di eventi

Esempio

- Creiamo un **bean** che espone due proprietà in sola lettura (ore e minuti) e ci dà l'ora corrente

```
import java.util.*  
public class CurrentTimeBean  
{  
    private int hours;  
    private int minutes;  
    public CurrentTimeBean()  
    {  
        Calendar now = Calendar.getInstance();  
        this.hours = now.get(Calendar.HOUR_OF_DAY);  
        this.minutes = now.get(Calendar.MINUTE);  
    }  
    public int getHours()  
    { return hours; }  
    public int getMinutes()  
    { return minutes; }  
}
```

Esempio (2)

```
package bean;

public class StudentBean implements java.io.Serializable {
    private String id = null;
    private String firstName = null;
    private String lastName = null;
    private int age = 0;
    private boolean inCourse = false;

    private String[] exams = null;

    public StudentBean() {
        this.exams = new String[30];
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public boolean isInCourse() {
        return inCourse;
    }

    public void setInCourse(boolean inCourse) {
        this.inCourse = inCourse;
    }

    public String getExam(int index) {
        return exams[index];
    }

    public void setExam(int index, String exam) {
        this.exams[index] = exam;
    }
}
```

JSP e JavaBean (bean.zip)

- JSP prevedono una serie di tag per agganciare un bean e utilizzare le sue proprietà all'interno della pagina
- Tre tipi:
 - Tag per **creare** un riferimento al bean (creazione di un'istanza)
 - Tag per **impostare** il valore delle proprietà del bean
 - Tag per **leggere** il valore delle proprietà del bean e inserirlo nel flusso della pagina

Tag jsp:useBean

- Sintassi: <jsp:useBean id="**beanName**" class="**class**"
 scope="page | request | session | application"/>
- Inizializza e crea il riferimento al bean (se non esiste)
- Gli attributi principali sono:
 - **id** è il nome con cui l'istanza del bean verrà indicata nel resto della pagina
 - **class** è classe Java che definisce il bean
 - **scope** definisce ambito di accessibilità e tempo di vita dell'oggetto (default = page)

```
<jsp:useBean id="time" class="CurrentTimeBean" scope="session" />
```



```
<% CurrentTimeBean myBean = (CurrentTimeBean) session.getAttribute("time");  
if(myBean == null) {  
    myBean = new CurrentTimeBean();  
    session.setAttribute("time", myBean);  
}  
%>
```

```
<%@ page import="CurrentTimeBean" %>
```

Tag jsp:getProperty

- Sintassi:**<jsp:getProperty name="beanName" property="propName"/>**
- Consente l'accesso alle proprietà del bean
- Produce come output il valore della proprietà del bean
- Il tag non ha mai body e ha solo 2 attributi:
 - **name**: nome del bean a cui si fa riferimento
 - **property**: nome della proprietà di cui si vuole leggere il valore

getProperty exploits the **PageContext#findAttribute()** method that scans in respectively the **page**, **request**, **session** and **application** scopes until the first **non-null** attribute value is found for a given attribute key

Esempio: uso di CurrentTimeBean

JSP

```
<jsp:useBean id="time" class="CurrentTimeBean"/>
<html>
<body>
    <p>Sono le ore
        <jsp:getProperty name="time" property="hours"/> e
        <jsp:getProperty name="time" property="minutes"/> minuti.
    </p>
</body>
</html>
```

Output HTML

```
<html>
<body>
    <p>Sono le ore
        12 e 18 minuti.</p>
    </body>
</html>
```

Tag `jsp:setProperty`

- Sintassi: `<jsp:setProperty name="beanName" property="propName" value="propValue"/>`
- Consente di modificare il valore delle proprietà del bean
- Esempio:

```
<jsp:setProperty name="user"  
    property="daysLeft" value="30"/>  
  
<jsp:setProperty name="user"  
    property="daysLeft" value="<% =15*2 %>" />
```

setProperty exploits the `PageContext#findAttribute()` method to find attributes

Esempio di uso di bean (UseBean.zip)

```
<jsp:useBean id="user" class="RegisteredUser" scope="session"/>

<jsp:useBean id="news" class="NewsReports" scope="request">
  <jsp:setProperty name="news" property="category" value="fin."/>
  <jsp:setProperty name="news" property="maxItems" value="5"/>
</jsp:useBean>

<html>
  <body>
    <p>Bentornato
      <jsp:getProperty name="user" property="fullName"/>,
      la tua ultima visita è stata il
      <jsp:getProperty name="user" property="lastVisitDate"/>.
    </p>
    <p>
      Ci sono <jsp:getProperty name="news" property="newItems"/>
      nuove notizie da leggere.</p>
  </body>
</html>
```

Comparazione tra tag e codice Java

```
<jsp:useBean id="time" class="CurrentTimeBean" scope="request"/>
```



```
<% CurrentTimeBean myBean = (CurrentTimeBean) request.getAttribute("time");
if(myBean == null) {
    myBean = new CurrentTimeBean();
    request.setAttribute("time", myBean);
} %>
```

```
<jsp:getProperty name="time" property="hours" />
```



```
<% CurrentTimeBean myBean = (CurrentTimeBean) pageContext.findAttribute("time");
if(myBean != null) { %>
    <%= myBean.getHours() %>
<% } %>
```

Comparazione tra tag e codice Java (2)

- Aggiungendo I metodi **void setHours(int hours)** e **void setMinutes(int minutes)** alla classe **CurrentTimeBean**

```
<jsp:useBean id="time" class="CurrentTimeBean">
    <jsp:setProperty name="time" property="hours" value="12"/>
    <jsp:setProperty name="time" property="minutes" value="0"/>
</jsp:useBean>
    =
<% CurrentTimeBean myBean = (CurrentTimeBean) pageContext.findAttribute("time");
if(myBean == null) {
    myBean = new CurrentTimeBean();
    pageContext.setAttribute("time", myBean);
}
myBean.setHours(12);
myBean.setMinutes(0);
%>
```

Comparazione tra tag e codice Java (3)

```
<jsp:setProperty name="time" property="hours" value="12"/>
```

...

```
<jsp:setProperty name="time" property="minutes" value="0"/>
```



```
<% CurrentTimeBean myBean1 = (CurrentTimeBean) pageContext.findAttribute("time");  
if(myBean1 != null) {
```

```
    myBean1.setHours(12);  
}
```

```
%>
```

...

```
<% CurrentTimeBean myBean2 = (CurrentTimeBean) pageContext.findAttribute("time");  
if(myBean2 != null) {
```

```
    myBean2.setMinutes(0);  
}
```

```
%>
```

Proprietà indicizzate

- *I tag per JavaBean non supportano proprietà indicizzate*
- Però un bean è un normale oggetto Java: è quindi possibile accedere a variabili e metodi
- Esempio:

```
public class weatherForecasts {  
    String[] forecasts;  
  
    public weatherForecasts() {  
        this.forecasts = new String[100];  
        // Set forecasts...  
    }  
  
    public String getForecasts(int index) {  
        if(index >= 0 && index < forecasts.length) {  
            return forecasts[index];  
        }  
        return null;  
    }  
}
```

```
<jsp:useBean id="weather" class="weatherForecasts"/>  
  
<p><b>Previsioni per domani:</b>:  
    <%= weather.getForecasts(0) %>  
    </p>  
<p><b>Resto della settimana:</b>  
<ul>  
    <% for (int index=1; index < 5; index++) { %>  
        <li><%= weather.getForecasts(index) %></li>  
        <% } %>  
</ul>  
</p>
```

Trying to display the property of the property

Without standard actions (using scripting)

```
<html><body>  
  <%= ((foo.Person) request.getAttribute("person")).getDog().getName() %>  
</body></html>
```

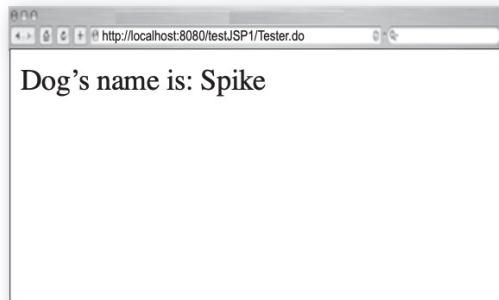
This works perfectly... but
we had to use scripting.

With standard actions (no scripting)

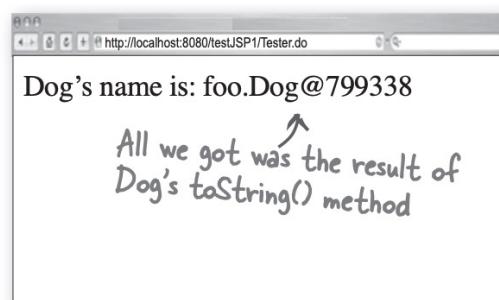
```
<html><body>  
  <jsp:useBean id="person" class="foo.Person" scope="request" />  
  Dog's name is: <jsp:getProperty name="person" property="dog" />  
</body></html>
```

But what's the
value of "dog"?

What we WANT



What we GOT



You can't say: property="dog.name"

The JSP Expression Language (EL)

- JSP 2.0 introduced a shorthand language for evaluating and outputting the values of Java objects that are stored in standard locations

JSP code **without scripting, using EL**

```
<html><body>  
  Dog's name is: ${person.dog.name}  
</body></html>
```

This is it! We didn't even declare
what person means... it just knows.

**EL makes it easy
to print nested
properties... in other
words, properties of
properties!**

This:

```
 ${person.dog.name}
```

Replaces this:

```
<%= ((foo.Person) request.getAttribute("person")).getDog().getName() %>
```

Deconstructing the JSP Expression Language (EL)

- The syntax and range of the language are simple

EL expressions are ALWAYS within curly braces, and prefixed with the dollar sign

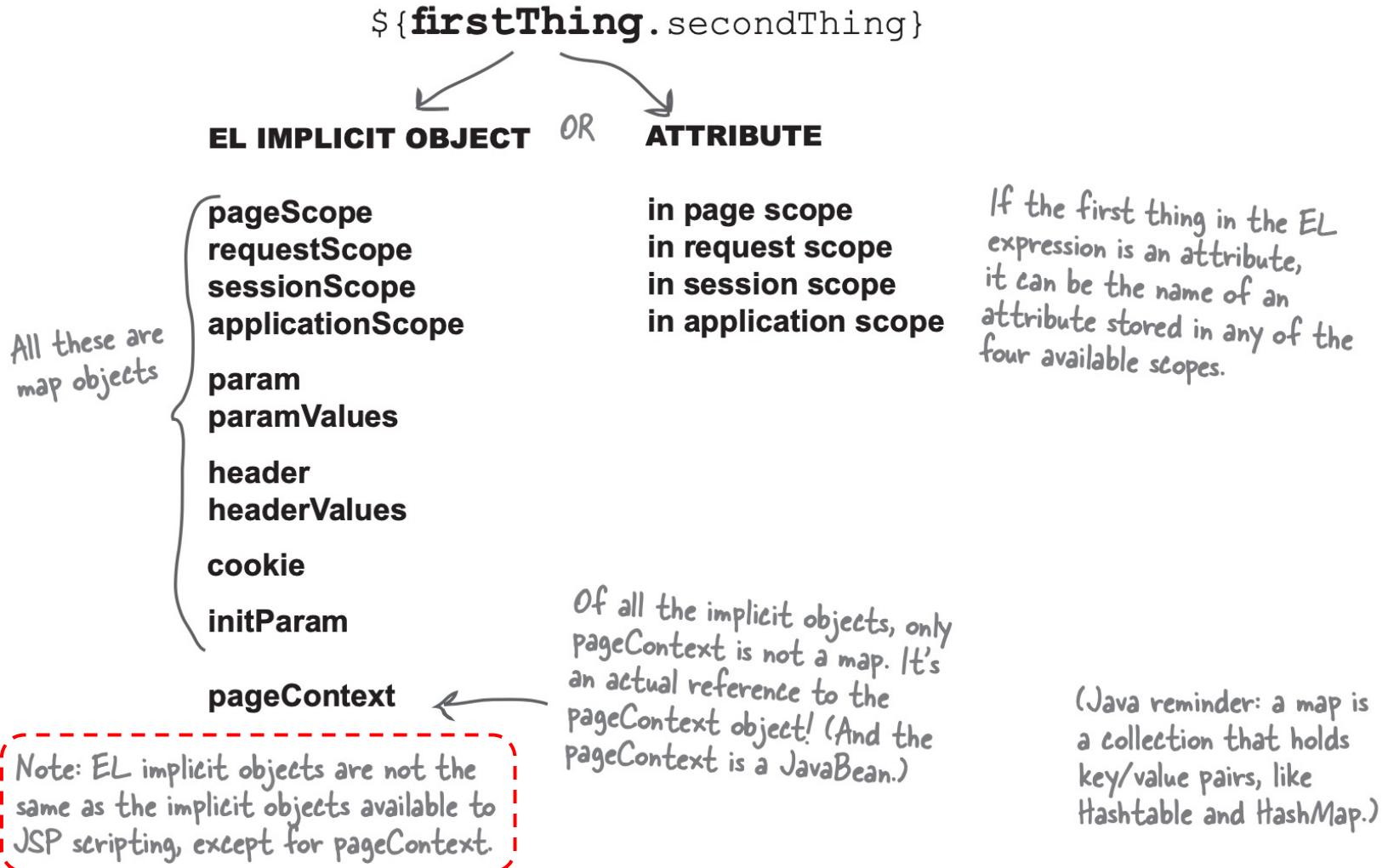
`$ {person.name}`

The first named variable in the expression is either an implicit object or an attribute.

```
<UL>
  <LI>Name: ${expression1}
  <LI>Address: ${expression2}
</UL>
<jsp:include page="${expression3}" />
<jsp:include page="${expr1}blah${expr2}" />
```

If you want \${ to appear in the page output, use \\$\{ in the JSP page

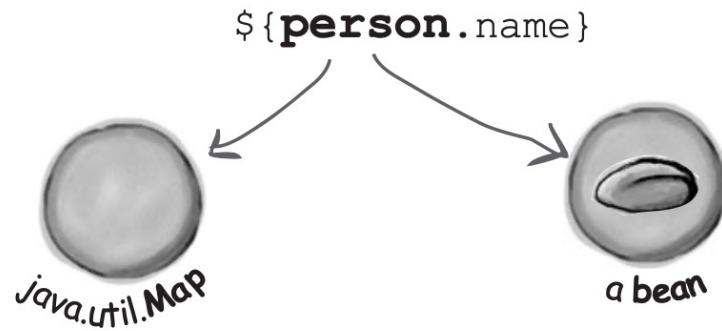
Deconstructing the JSP Expression Language (EL) (2)



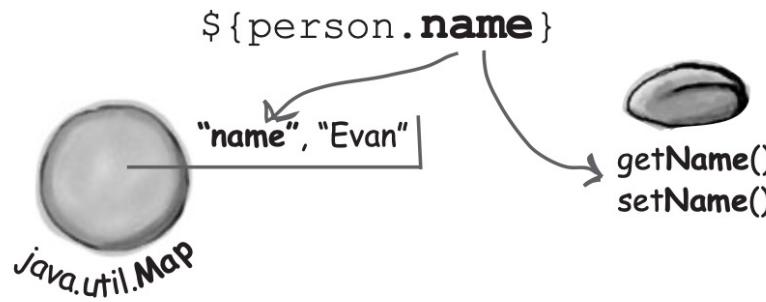
Using the dot (.) operator

- The thing to the *right* of the dot is either a map *key* (if the first variable is a map) or a bean *property* if the first variable is an attribute that's a

JavaBean ① **If the expression has a variable followed by a dot, the left-hand variable MUST be a Map or a bean.**



② **The thing to the right of the dot MUST be a Map key or a bean property.**



The [] operator is like the dot only way better

- The [] operator is a lot more powerful and flexible...

That doesn't look
better. That just
looks like more work,
adding brackets and
quotes...

This:

```
$ {person["name"] }
```

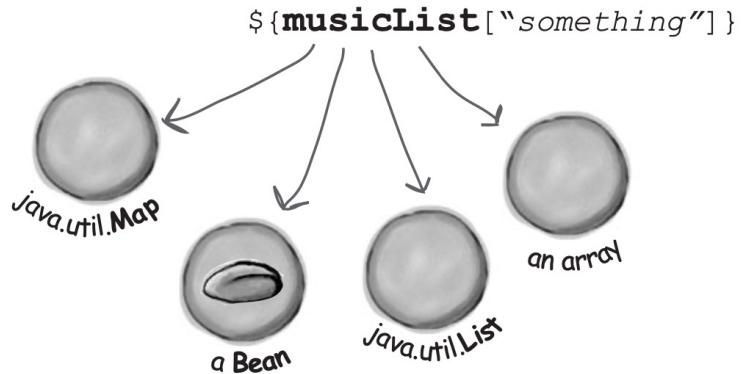
**Is the same
as this:**

```
$ {person.name}
```

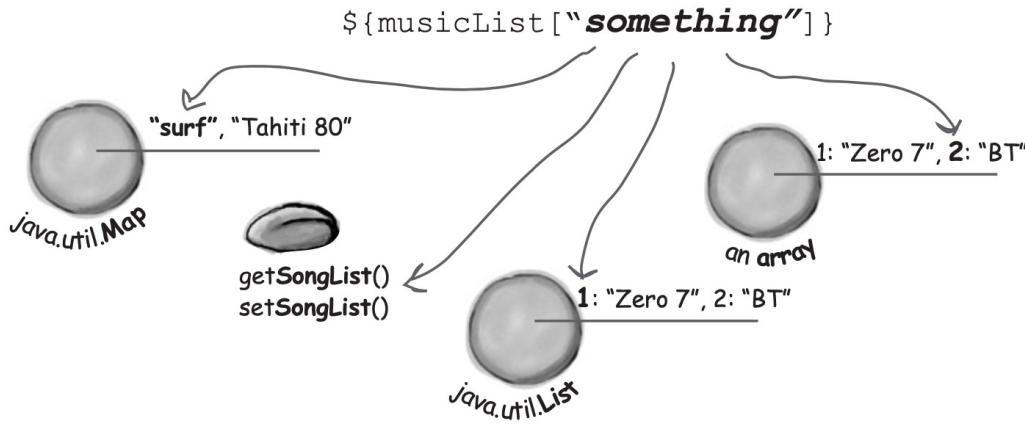


The [] gives you more options...

- ① If the expression has a variable followed by a bracket [], the left-hand variable can be a Map, a bean, a List, or an array.



- ② If the thing *inside* the brackets is a String literal (i.e., in quotes), it can be a Map key or a bean property, or an index into a List or array.



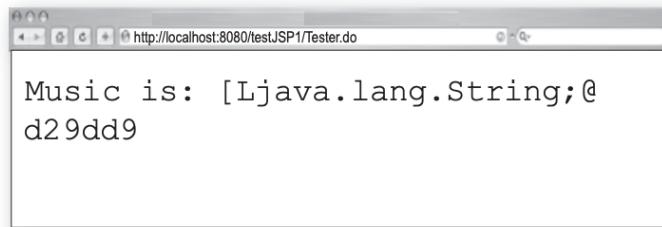
In a Servlet

```
String[] favoriteMusic = {"Zero 7", "Tahiti 80", "BT", "Frou Frou"};
request.setAttribute("musicList", favoriteMusic);
```

Using [] operator with an array

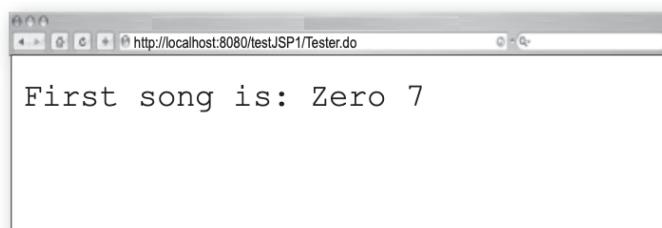
In a JSP

Music is: \${musicList}

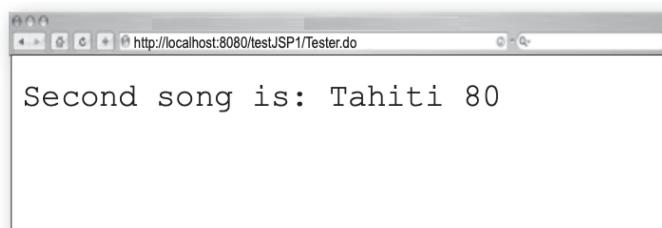


Makes sense... calls
toString() on the array.

First song is: \${musicList[0]} duh..



Second song is: \${musicList["1"]} WTF???



In a Servlet

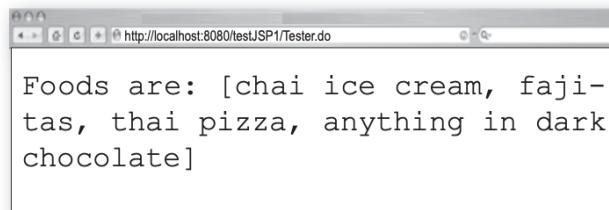
```
java.util.ArrayList favoriteFood = new java.util.ArrayList();
favoriteFood.add("chai ice cream");
favoriteFood.add("fajitas");
favoriteFood.add("thai pizza");
favoriteFood.add("anything in dark chocolate");
request.setAttribute("favoriteFood", favoriteFood);
```

A String index is coerced
to an int for arrays and Lists

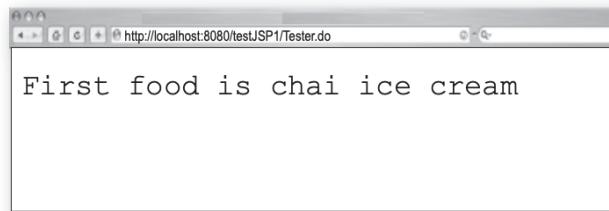
In a JSP

Foods are: \${favoriteFood}

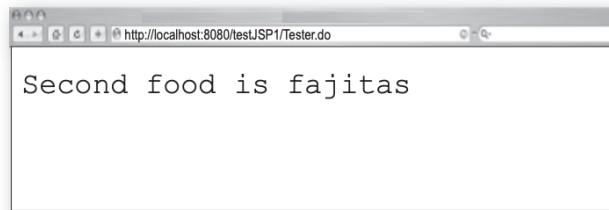
Obviously ArrayList has a nice overridden `toString()`.



First food is \${favoriteFood[0]}



Second food is \${favoriteFood["1"]}



right

Very, very weird, but OK...
if that's the way it works,
I'll have to get used to it.

For beans and Maps you can use either operator

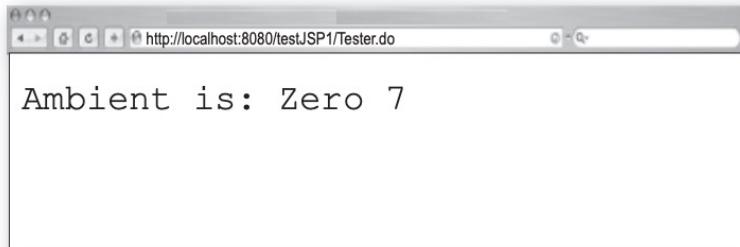
In a Servlet

```
java.util.Map musicMap = new java.util.HashMap();  
musicMap.put("Ambient", "Zero 7");  
musicMap.put("Surf", "Tahiti 80");  
musicMap.put("DJ", "BT");  
musicMap.put("Indie", "Travis");  
request.setAttribute("musicMap", musicMap);
```

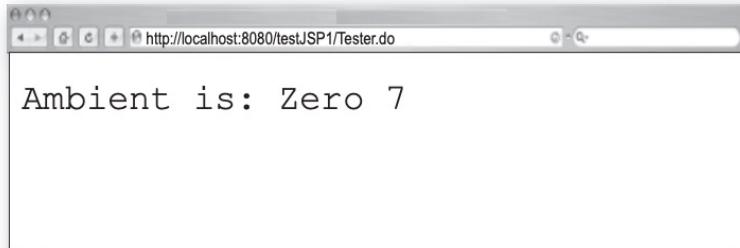
Make a Map, put some String keys and objects in it, then make it a request attribute.

In a JSP

Ambient is: \${musicMap.Ambient}



Ambient is: \${musicMap["Ambient"]}



Both expressions use Ambient as the key into a Map (since musicMap is a Map!).

If it's NOT a String literal, it's evaluated

- If there are no quotes inside the brackets:

Music is: \${musicMap [Ambient] }



Without quotes around Ambient, this does NOT work!! Since there's no bound attribute named "Ambient", the result comes back null..

Find an attribute named “Ambient”.

Use the VALUE of that attribute as the key into the Map, or return null.

If it's NOT a String literal, it's evaluated (2)

In a servlet

```
java.util.Map musicMap = new java.util.HashMap();
musicMap.put("Ambient", "Zero 7");
musicMap.put("Surf", "Tahiti 80");
musicMap.put("DJ", "BT");
musicMap.put("Indie", "Frou Frou");

request.setAttribute("musicMap", musicMap);

request.setAttribute("Genre", "Ambient");
```

This **DOES** work in a JSP

Music is \${musicMap[**Genre**] }  evaluates to Music is \${musicMap["**Ambient**"] } 

because there **IS** a request attribute named "Genre" with a value of "Ambient", and "Ambient" is a key into musicMap.

Zero 7

This does **NOT** work in a JSP (given the servlet code)

Music is \${musicMap["**Genre**"] }  doesn't change Music is \${musicMap["**Genre**"] }

because there **IS** no key in musicMap named "Genre".
With the quotes around it, the Container didn't try to evaluate it and just assumed it was a literal key name.

↑ null

This is a valid EL expression, but it doesn't do what we wanted.

You can use nested expressions inside the brackets In a servlet

```
java.util.Map musicMap = new java.util.HashMap();
musicMap.put("Ambient", "Zero 7");
musicMap.put("Surf", "Tahiti 80");
musicMap.put("DJ", "BT");
musicMap.put("Indie", "Frou Frou");
request.setAttribute("musicMap", musicMap);

String[] musicTypes = {"Ambient", "Surf", "DJ", "Indie"};
request.setAttribute("MusicType", musicTypes);
```

This DOES work in a JSP

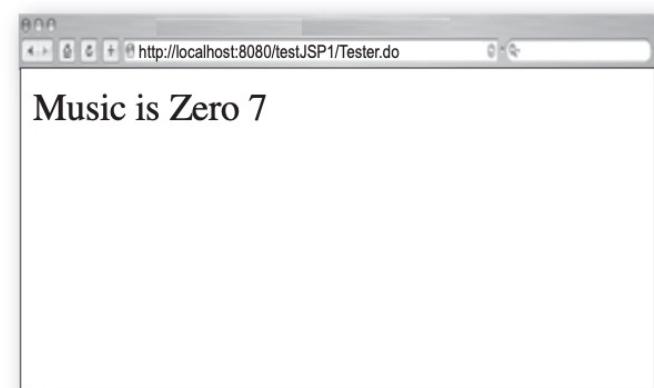
Music is \${musicMap[MusicType[0]]}



Music is \${musicMap["Ambient"]}



Music is **Zero 7**



You can't do \${foo.1}

This

`${musicMap.Ambient}` works

Is the same as this

`${musicMap["Ambient"]}` works

But this

`${musicList["1"]}`

CANNOT be turned into this

`${musicList.1}` NO! NO! NO!

In the HTML form

```
<form action="TestBean.jsp">  
    Name: <input type="text" name="name">  
    ID#: <input type="text" name="empID">  
  
    First food: <input type="text" name="food">  
    Second food: <input type="text" name="food"> ←  
  
    <input type="submit">  
</form>
```

The "name" and "empID" will each have a single value. But the "food" parameter could have two values, if the user fills in both fields before hitting the submit button...

Request parameters in EL

In the JSP

Request param name is: \${param.name}

Remember, param is just a Map of parameter names and values. The things to the right of the dot come from the names specified in the input fields of the form.

Request param empID is: \${param.empID}

Even though there might be multiple values for the "food" parameter, you can still use the single param implicit object, but you'll get only the first value.

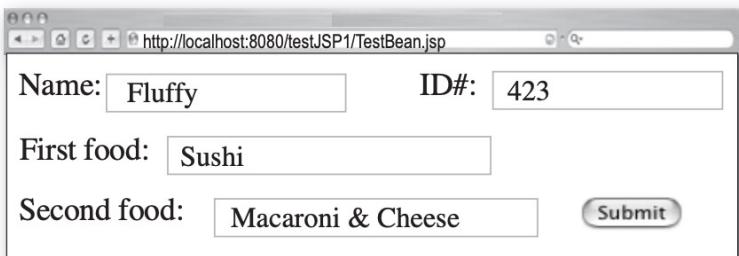
Request param food is: \${param.food}
 ←

First food request param: \${paramValues.food[0]}

Second food request param: \${paramValues.food[1]}

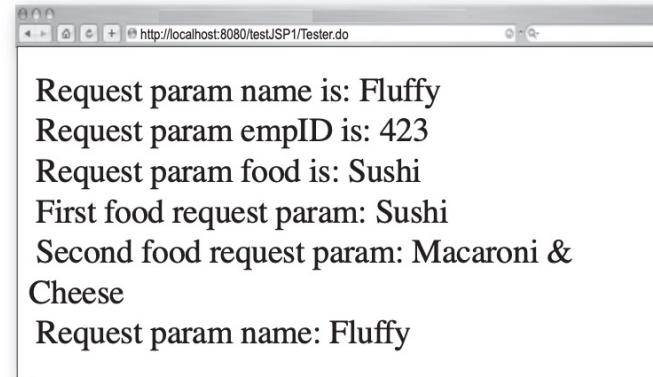
Request param name: \${paramValues.name[0]}

In the client's browser (client fills in the form and hits the submit button)



Name: Fluffy ID#: 423
First food: Sushi
Second food: Macaroni & Cheese
Submit

The response



Request param name is: Fluffy
Request param empID is: 423
Request param food is: Sushi
First food request param: Sushi
Request param name: Fluffy

Referencing Implicit Objects

- **pageContext:**
 - \${pageContext.session.id}
- **header and headerValues:**
 - \${header.Accept} or \${header["Accept"]}
 - \${header["Accept-Encoding"]}
 - \${headerValues.Accept[0]}
- **cookie:**
 - \${cookie.userCookie.value}
 - \${cookie["userCookie"].value}
- **initParam**
 - \${initParam.defaultColor}
- **pageScope, requestScope, sessionScope, and applicationScope**
 - \${requestScope.name}

Arithmetic (5)

Addition:

+

Subtraction:

-

Multiplication:

Division:

/ and div

Remainder:

% and mod

And a few other EL operators...

By the way... you CAN divide by zero in EL—you get INFINITY, not an error.

But you CANNOT use the Remainder operator against a zero—you'll get an exception.

Logical (3)

AND: **&& and and**

OR: **|| and or**

NOT: **! and not**

Relational (6)

Equals: **== and eq**

Not equals: **!= and ne**

Less than: **< and lt**

Greater than: **> and gt**

Less than or equal to: **<= and le**

Greater than or equal to: **>= and ge**



Don't use EL reserved words as identifiers!

Watch it!

You can already see 11 of them on this page—the alternate “words” for the relational, logical and some arithmetic operators. But there are a few more:

true	a boolean literal
false	the OTHER boolean literal
null	It means... null
instanceof	(this is reserved for “the future”)
empty	an operator to see if something is null or empty (eg. \${empty A}) returns true if A is null or empty

`${foo}`

`${foo[bar]}`

`${bar[foo]}`

`${foo.bar}`

Nothing prints out for these expressions. If you say "The value is: \${foo}." You'll just see "The value is."

`${7 + foo}`

7

In arithmetic expressions, EL treats the unknown variable as "zero".

`${7 / foo}`

Infinity

In arithmetic expressions, EL treats the unknown variable as "zero".

`${7 - foo}`

7

`${7 % foo}`

Exception is thrown

`${7 < foo}`

false

`${7 == foo}`

false

In logical expressions, EL treats the unknown variable as "false".

`${foo == foo}`

true

`${7 != foo}`

true

`${true and foo}`

false

`${true or foo}`

true

`${not foo}`

true

EL handles null values gracefully

Example exprlang.zip

Example (Sales.zip)

```
package bean;

public class SalesBean {
    private double q1, q2;

    public SalesBean() {}

    public SalesBean(double q1Sales, double q2Sales) {
        q1 = q1Sales;
        q2 = q2Sales;
    }

    public double getQ1() { return q1; }

    public void setQ1(double q1) { this.q1 = q1; }

    public double getQ2() { return q2; }

    public void setQ2(double q2) { this.q2 = q2; }

    public double getTotal() { return (q1 + q2); }
}
```

	Apples	Oranges
First Quarter	12.7	-4.2
Second Quarter	9.3	1.2
Total	22.0	-3.0

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" import="bean.SalesBean"%>

<jsp:useBean id="apples" class="bean.SalesBean">
    <jsp:setProperty name="apples" property="q1" value="12.7"/>
    <jsp:setProperty name="apples" property="q2" value="9.3"/>
</jsp:useBean>

<jsp:useBean id="oranges" class="bean.SalesBean">
    <jsp:setProperty name="oranges" property="q1" value="-4.2"/>
    <jsp:setProperty name="oranges" property="q2" value="1.2"/>
</jsp:useBean>

<!DOCTYPE html>
<html>
<body>
    <table BORDER=1>
        <tr><th><th>Apples<th>Oranges
        <tr><th>First Quarter<td>${apples.q1}<td>${oranges.q1}
        <tr><th>Second Quarter<td>${apples.q2}<td>${oranges.q2}
        <tr><th>Total
            <td bgcolor="${(apples.total < 0) ? "red" : "green"}">
                ${apples.total}
            <td bgcolor="${(oranges.total < 0) ? "red" : "green"}">
                ${oranges.total}
        </table>
    </body>
</html>
```