



CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

JSP: JAVA SERVER PAGES

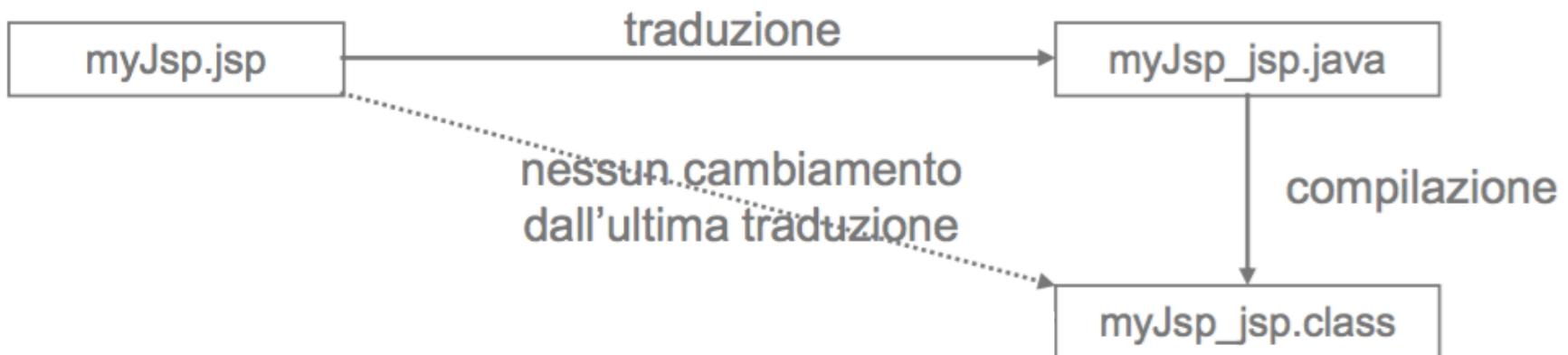
Docente: prof. Simone Romano
a.a. 2024-2025

Java Server Pages

- Le JSP sono uno dei due componenti di base della tecnologia Jakarta EE (precedentemente nota come 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*

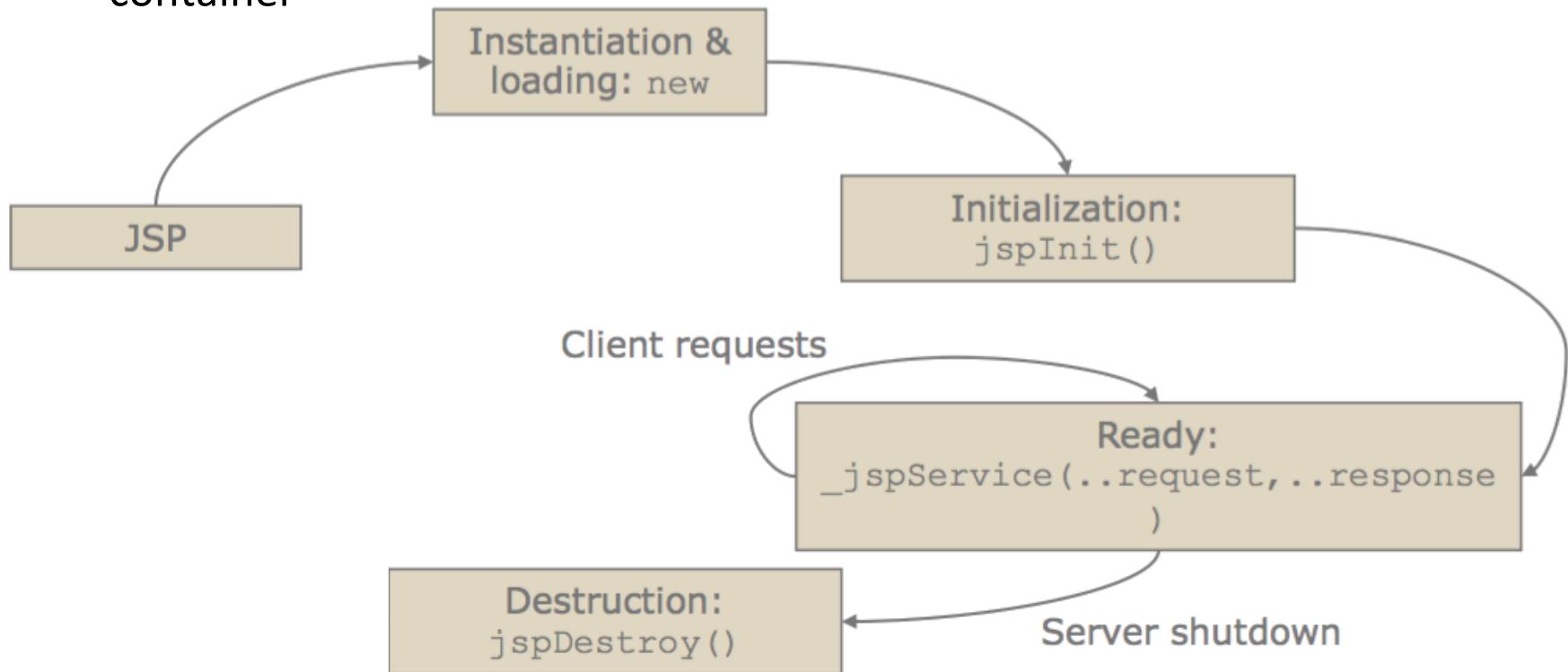
JspServlet

- Quando arriva una richiesta ad una JSP, un particolare Servlet (in Tomcat si chiama **JspServlet**) si occupa di
 1. tradurre la JSP in una Servlet
 2. compilare la Servlet risultante in una classe
- Questi due passi vengono eseguiti solo se 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 `println()`)
 - 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 (decl.jsp in jsp.zip)

```
1 <%@ page language="java" import="java.util.Date, java.text.SimpleDateFormat"
2   contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4<html>
5<head>
6 <title>JSP Declarations</title>
7 </head>
8<body>
9<%!
10    String name = "Paolo Rossi";
11    double[] prices = {1.5, 76.8, 21.5};
12
13    double getTotal() {
14        double total = 0.0;
15        for(int i=0; i <prices.length; i++){
16            total += prices[i];
17        }
18        return total;
19    }
20
21    String formattedDate(Date today) {
22        SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy HH.mm.ss");
23        return formatter.format(today);
24    }
25%>
26<h3>JSP Declarations</h3>
27<p>Sig. <%=name %>, </p>
28<p>l'ammontare del suo acquisto &egrave;: <%=getTotal() %> euro.</p>
29<p>La data di oggi &egrave;: <%=formattedDate(new Date()) %></p>
30</body>
31</html>
```

Esempio 2 (expr.jsp in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4<html>
5<head>
6 <title>JSP Expressions</title>
7 </head>
8<body>
9 <h3>JSP Expressions</h3>
10<ul>
11     <li>Current time: <%= new java.util.Date() %>
12     <li>Server: <%=application.getServerInfo() %>
13     <li>Application name: <%=application.getContextPath() %>
14     <li>Session Id: <%=session.getId() %>
15     <li>The <code>testParam</code> form parameter:<%=request.getParameter("testParam") %>
16 </ul>
17 </body>
18 </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/expr.jsp>
- <http://myHost/myWebApp/expr.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 in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   info="simple jsp examples" pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <title>JSP Scriptlet</title>
7 </head>
8 <body>
9 <h3>JSP Scriptlet</h3>
10 <ul>
11   <li>Current time: <%= new java.util.Date() %>
12   <li>Server: <%=application.getServerInfo() %>
13   <li>Application name: <%=application.getContextPath() %>
14   <li>Session Id: <%=session.getId() %>
15   <% String param = request.getParameter("testParam");
16   if(param != null) { %
17     <li>The <code>testParam</code> form parameter:
18       <%=param %>
19     <% } else { %
20     <li>No form parameter
21     <% } %
22     <li>Info: <%=this.getServletInfo() %>
23 </ul>
24 </body>
25 </html>
```

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.*, jackarta.servlet.*, jackarta.servlet.jsp.*, jackarta.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**, solo un thread alla volta può accedere alla pagina
- **info="text"** testo di commento
 - Può essere letto con il metodo **getServletInfo()** ereditato da GenericServlet
 - Es: **<%= this.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 (callerror.jsp in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"
3     errorPage="error.jsp"%>
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <title>JSP Call Error Page</title>
8 </head>
9 <body>
10 <h3>JSP Call Error Page</h3>
11 <% String param = request.getParameter("testParam");
12 if(param.equals("PW")) { %>
13     The <code>testParam</code> form parameter: <%=param %>
14 <% } else { %>
15     No form parameter
16 <% } %>
17 </body>
18 </html>
```

Uncaught exception

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

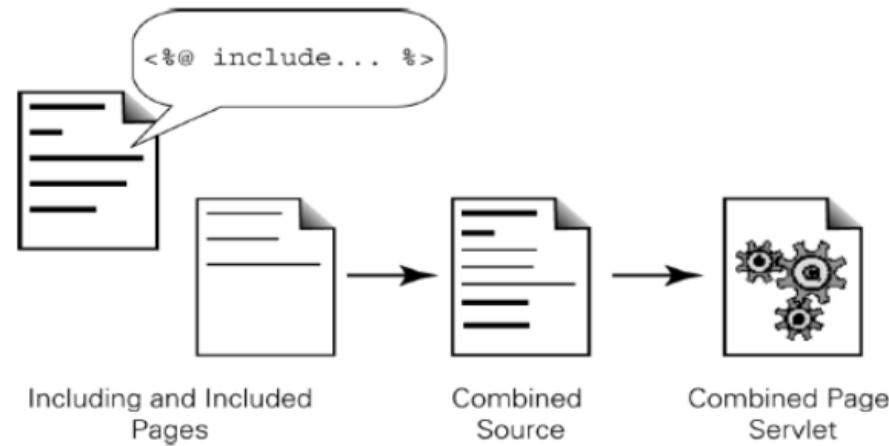
Esempio 4b (error.jsp in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8" isErrorPage="true"%>
3 <!DOCTYPE html>
4<html>
5<head>
6 <title>JSP Error</title>
7 </head>
8<body>
9 <h3> Error</h3>
10 <% if(exception != null) { %>
11 <p>An exception was raised: <%= exception.toString() %></p>
12 <p>Exception message is: <%= exception.getMessage() %></p>
13 <br>
14<%
15     StackTraceElement[] st = exception.getStackTrace();
16     for(StackTraceElement e: st){
17         out.println(e.toString());
18     }
19 }
20 %>
21 </body>
22 </html>
```



La direttiva include

- Sintassi: `<%@ include file = "localURL"%>`
- Serve ad includere il contenuto del file specificato (sia un HTML file che un'altra pagina JSP)
 - È possibile nidificare un numero qualsiasi di inclusioni



- 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 in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"  
2     pageEncoding="UTF-8"%>  
3 <!DOCTYPE html>  
4<html>  
5<head>  
6 <title>JSP Include</title>  
7 </head>  
8<body>  
9 <%@ include file="header.jsp" %>  
10 <br>  
11 Contact Us at: we@studytonight.com<br>  
12 <br>  
13 <%@ include file="footer.jsp" %>  
14  
15 </body>  
16 </html>
```

header.jsp

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

footer.jsp

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

HEADER

Contact Us at: we@studytonight.com

FOOTER

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

Esempio 6 (tags.jsp in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6     <title>JSP Tags</title>
7 </head>
8 <body>
9 <%@ include file="header.jsp" %>
10 <h3>JSP Tags</h3>
11 <%!
12     long fact(long n) {
13         if(n == 1) return 1;
14         return n*fact(n-1);
15     }
16 %>
17 <% String xStr = request.getParameter("num");
18 if(xStr != null) {
19     try {
20         long x = Long.parseLong(xStr); %>
21         Fattoriale di <%= x%>: <%= fact(x)%>
22
23     } catch (NumberFormatException e) {}>
24     Il paraemtro <b>num</b> non contiene un valore intero.
25 <%
26     }
27 }
28 %>
29 <%@ include file="footer.jsp" %>
30 </body>
31 </html>
```

*Se n è
negativo?*

tags.jsp
tags.jsp with param (num = 3)
tags.jsp with param (num = 'a')

Direttiva taglib

- Le JSP permettono di definire **tag custom** oltre a quelli predefiniti
- Una taglib è una libreria di tag custom (ovvero una collezione di tag custom), realizzata mediante una classe Java
- Sintassi: **<%@ taglib uri="tagLibraryURI" prefix="tagPrefix"%>**
 - **uri:** serve a specificare l'URI che identifica in modo univoco la libreria di tag. Questo URI in genere punta alla posizione del file Tag Library Descriptor (TLD)
 - **prefix:** Prefisso utilizzato per identificare i tag all'interno della libreria importata. Questo prefisso viene utilizzato nel codice JSP quando si richiamano i tag personalizzati dalla libreria.
- **Occorre aggiungere il file jar della libreria di tag al classpath (aggiungerli alla cartella lib in WEB-INF)!**

Libreria di tag JSTL

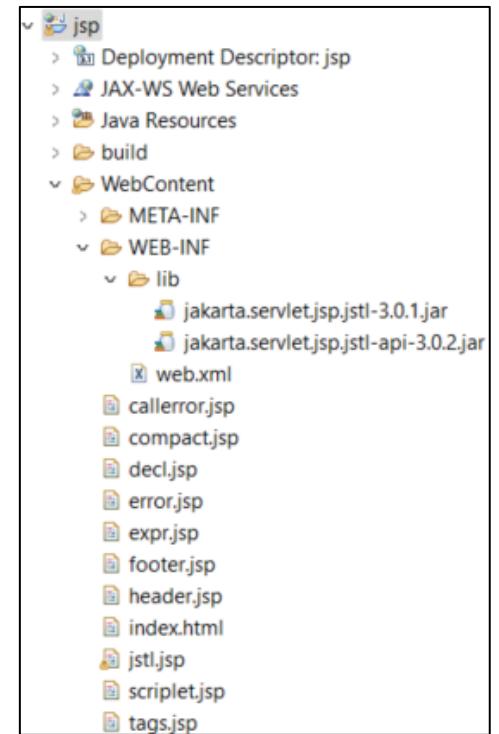
È una raccolta di tag custom che consente, tra le altre cose, di controllare di flusso condizionale ed iterativo all'interno di una pagina JSP

Occorre usare la seguente direttiva:

- `<%@ taglib prefix="c" uri="jakarta.tags.core" %>`

Inoltre, occorre scaricare ed aggiungere al classpath le seguenti librerie:

- `jakarta.servlet.jsp.jstl-api-x.x.x.jar`
(<https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api>)
- `jakarta.servlet.jsp.jstl-x.x.x.jar`
(<https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl>)



Esempio 7 (jstl.jsp in jsp.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="jakarta.tags.core" %>
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <meta charset="UTF-8">
8     <title>JSTL Tags</title>
9 </head>
10 <body>
11 <c:forEach var="i" begin="1" end="5">
12     <c:if test = "${i == 2 || i == 3}">
13         Item <c:out value="${i}" /><br>
14     </c:if>
15 </c:forEach>
16 </body>
17 </html>
```

Item 2

Item 3

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	java.lang.Object
config	jakarta.servlet.ServletConfig
request	jakarta.servlet.http.HttpServletRequest
response	jakarta.servlet.http.HttpServletResponse
out	jakarta.servlet.jsp.JspWriter
session	jakarta.servlet.http.HttpSession
application	jakarta.servlet.http.ServletContext
pageContext	jakarta.servlet.jsp.PageContext
exception	java.lang.Throwable

Oggetto page

- L'oggetto page rappresenta l'istanza corrente della Servlet
 - *Poco usato nella pratica poiché se serve accedere all'istanza della Servlet è più semplice usare il riferimento **this***

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:
 - 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

- Rappresenta la risposta che viene restituita al client
- È il parametro response passato al metodo service() della servlet
- Consente di inserire nella risposta diverse informazioni:
 - 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

- È uno stream di caratteri e rappresenta lo stream di output della pagina

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

Oggetto session

- Oggetto che rappresenta la sessione corrente per un utente
- L'attributo **session** della direttiva **page** deve essere **true** affinché 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 sul contesto (**ServletContext**) nel quale la JSP è eseguita
- 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
- Consente l'accesso a tutti gli oggetti impliciti (ad esempio richiesta, sessione, ...) tramite metodi getters **getRequest()**, **getSession()**, ...
- Consente il trasferimento del controllo ad altre pagine (forward e include)
 - Ad es:

```
<% pageContext.forward(<Relative URL Path>); %>
<% pageContext.include(<Relative URL Path>); %>
```
- È uno **scoped object**:
 - Consente di memorizzare, accedere, e rimuovere **attributi a livello di pagina**
 - Gli attributi a livello di pagina sono accessibili solo all'interno della pagina JSP fino a quando la pagina termina la propria esecuzione

Oggetto pageContext

- Utilizzando questo oggetto è possibile lavorare con gli attributi di qualsiasi scoped object tramite i metodi **findAttribute()**, **getAttribute ()**, **setAttribute ()** e **removeAttribute()**
 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**

Oggetto pageContext

- Esempi:

<code><% pageContext.findAttribute("mail"); %></code>	Cerca l'attributo nei contesti di pagina, richiesta, sessione e applicazione in ordine e lo restituisce (ritorna null se l'attributo non esiste)
<code><% pageContext.getAttribute("mail"); %></code>	Restituisce l'attributo dal contesto di pagina (ritorna null se l'attributo non esiste)
<code><% pageContext.getAttribute("mail", PageContext.APPLICATION_SCOPE); %></code>	Restituisce l'attributo dal contesto dell'applicazione (ritorna null se l'attributo non esiste)
<code><% pageContext.setAttribute("role", "manager"); %></code>	Salva l'attributo nel contesto di pagina
<code><% pageContext.setAttribute("role", "manager", PageContext.SESSION_SCOPE); %></code>	Salva l'attributo nel contesto nella sessione
<code><% pageContext.removeAttribute("mail"); %></code>	Elimina l'attributo da tutti i contesti
<code><% pageContext.removeAttribute("mail", PageContext.APPLICATION_SCOPE); %></code>	Elimina l'attributo dal contesto dell'applicazione

Oggetto exception

- Oggetto connesso alla gestione delle eccezioni
- 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 **isErrorPage** impostato a true*)
- Esempio:

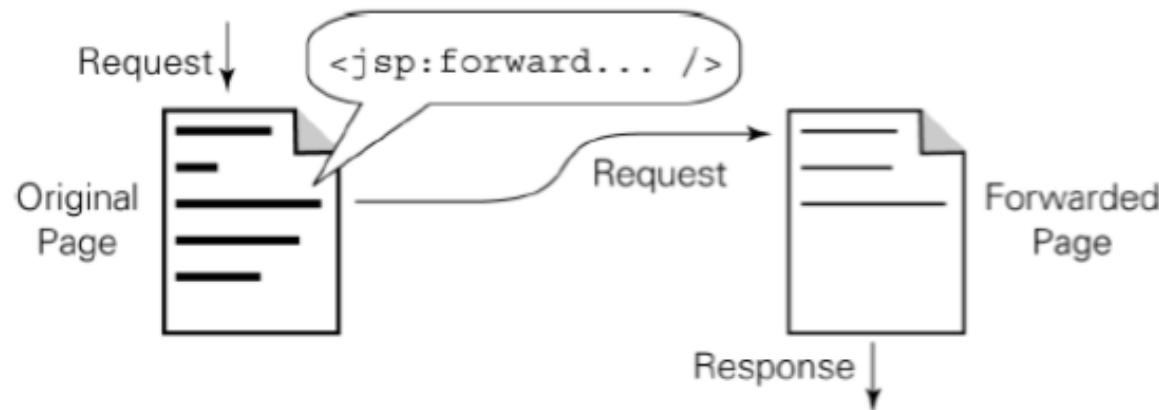
```
<%@ page isErrorPage="true" %>
<h1>Attenzione!</h1>
E' stato rilevato il seguente errore:<br/>
<b><%= exception %></b><br/>
<%
    exception.printStackTrace(out);
%>
```

Azioni

- Le azioni sono comandi JSP per gestire l'interazione con altre risorse gestite dal Servlet Container (pagine JSP, Servlet e file HTML) o con componenti **JavaBean**
- Sono espresse usando la sintassi XML
- Sono previsti **6 tipi di azioni** definite dai seguenti **tag**:
 - **forward**: effettua l'inoltro ad un'altra risorsa (JSP, pagina HTML o servlet)
 - **include**: include nella JSP il contenuto generato da un'altra risorsa (JSP, pagina HTML o servlet)
 - **useBean**: istanzia JavaBean e gli associa un identificativo
 - **getProperty**: ritorna la property indicata come oggetto
 - **setProperty**: imposta valore della property indicata per nome
 - **plugin**: genera contenuto per scaricare plug-in Java se necessario

Azioni: forward

- Sintassi: **<jsp:forward page="localURL" />**
- Consente di effettuare l'inoltro (ovvero cedere il controllo) ad un'altra risorsa (statica o dinamica)
- L'attributo **page** definisce l'URL della pagina a cui effettuare l'inoltro



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 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

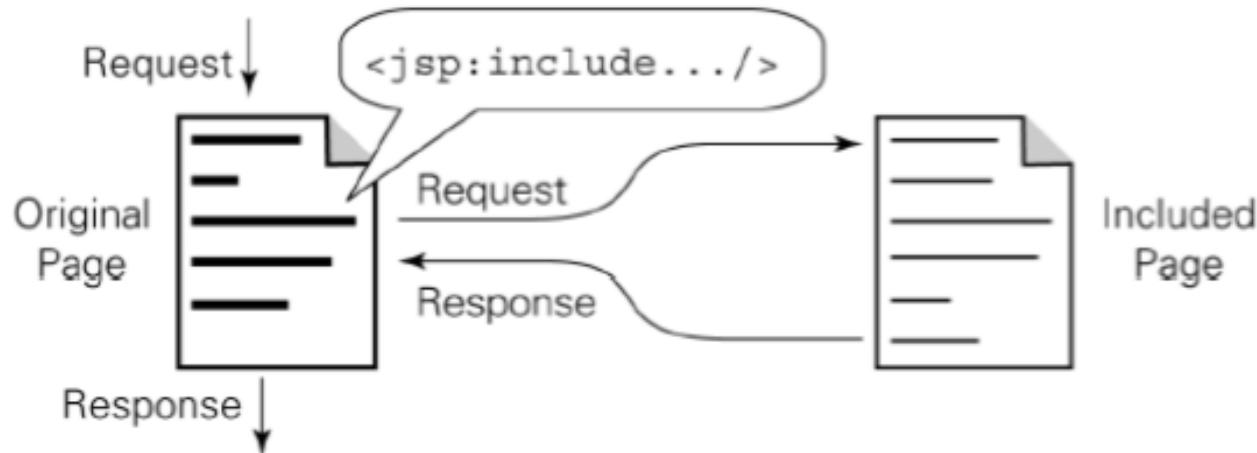
Azioni: include

- Sintassi: **<jsp:include page="localURL"/>**
- Consente di includere il contenuto generato da un'altra risorsa (statica o dinamica) all'interno dell'output della pagina corrente
 - Trasferisce temporaneamente controllo ad un'altra risorsa
 - L'attributo **page** definisce l'URL della pagina da includere
 - Gli oggetti **session** e **request** (ma non response) per la 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"/>
<jsp:param name="parName1" value="parValue1"/>
...
<jsp:param name="parNameN" value="parValueN"/>
</jsp:include>
```



JavaBeans

- I **JavaBeans**, o semplicemente **beans**, sono componenti software riutilizzabili che seguono una specifica convenzione:
 - Classe **public**
 - Ha un **costruttore** public di default (senza argomenti)
 - Espone proprietà, sotto forma di coppie di metodi di accesso (**accesso**) 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**
- Implementa l'interfaccia **Serializable**

Esempio (bean.zip)

```
package bean;

import java.io.Serializable; 

public class CurrentTimeBean implements Serializable {

    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 void setHours(int hours) {
        this.hours = hours;
    }

    public int getMinutes() {
        return minutes;
    }

    public void setMinutes(int minutes) {
        this.minutes = minutes;
    }
}
```

JSP e JavaBean

- Esistono dei tag di azione 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="bean.CurrentTimeBean" scope= "session" />
```



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

A blue arrow points upwards from the line 'myBean = new CurrentTimeBean();' in the JSP code towards the 'import' statement in the footer.

```
<%@ page import="bean.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
- Deve essere preceduto dal tag **<jsp:useBean>**

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>
```

Qual è l'ambito del Bean?

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
- Deve essere preceduto (o incluso) dal tag **<jsp:useBean>**
- Esempio:

```
<jsp:setProperty name="user"
    property="daysLeft" value="30"/>

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

Esempio di uso di bean

```
<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>
```

jsp:setProperty: outside vs inside jsp:useBean

- There are two basic ways to use the **jsp:setProperty** action tag:

1. You can use **jsp:setProperty** after, but outside of a **jsp:useBean** element, as given below:

```
<jsp:useBean id = "myName" ... />
...
<jsp:setProperty name = "myName" property = "someProperty" .../>
```

- In this case, the **jsp:setProperty** is executed regardless of whether a new bean was instantiated or an existing bean was found

2. You can use **jsp:setProperty** inside the body of a **jsp:useBean** element, as given below:

```
<jsp:useBean id = "myName" ... >
...
<jsp:setProperty name = "myName" property = "someProperty" .../>
</jsp:useBean>
```

- Here, the **jsp:setProperty** is executed only if a new object was instantiated, not if an existing one was found

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

```
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>
```

Funziona nonostante la classe non implementi l'interfaccia Serializable!

Trying to display the property of the property

Without standard actions (using scripting)

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

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

With standard actions (no scripting)

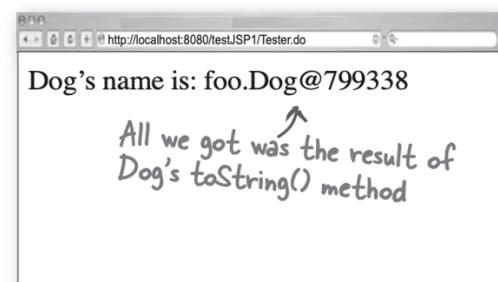
```
<html><body>  
<jsp:useBean id="person" class="foo.Person" scope="request" />  
Dog's name is: <jsp:getProperty name="person" property="dog" />  
</body></html>
```

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)

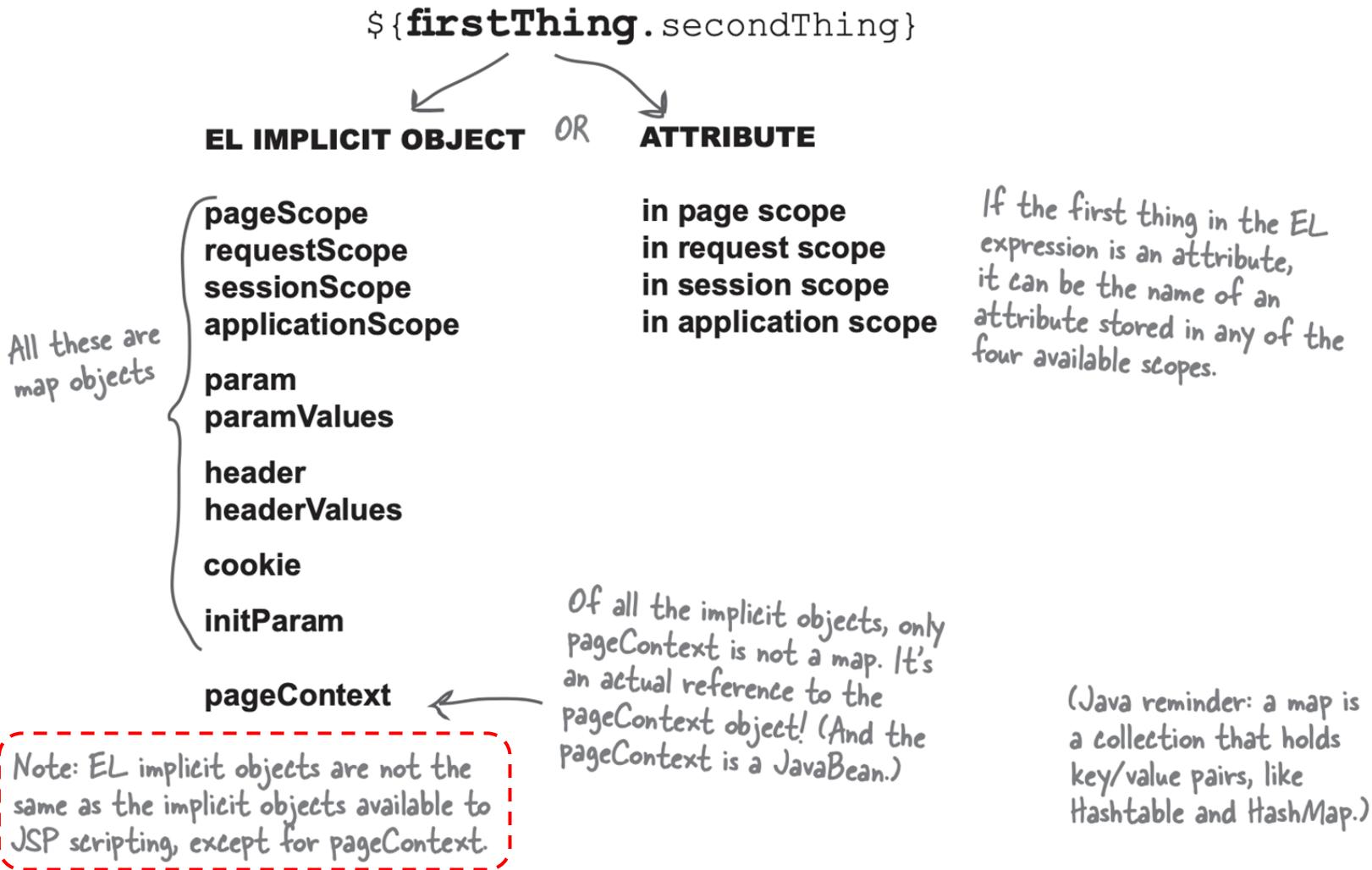
- The syntax and range of the language are simple

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

`$ {person.name}`

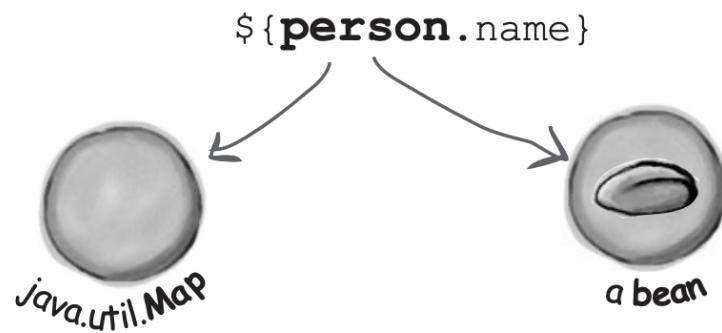
The first named variable in the expression is either an implicit object or an attribute.

Deconstructing the JSP Expression Language (EL) (2)

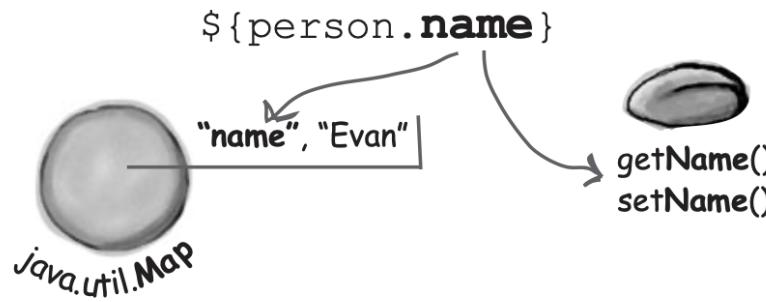


Using the dot (.) operator

- 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"] }
```

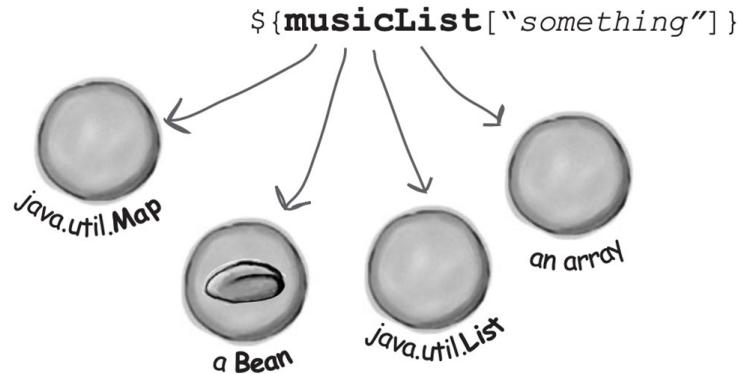
**Is the same
as this:**

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

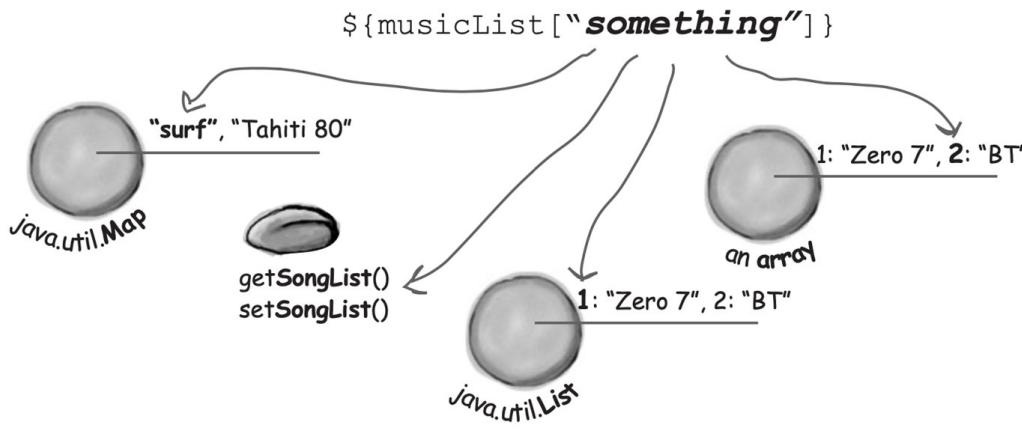


The [] gives you more options...

- 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.



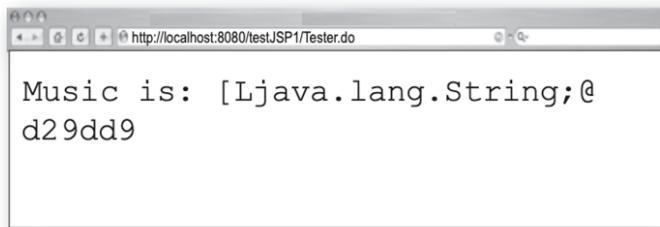
In a Servlet

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

Using [] operator with an array

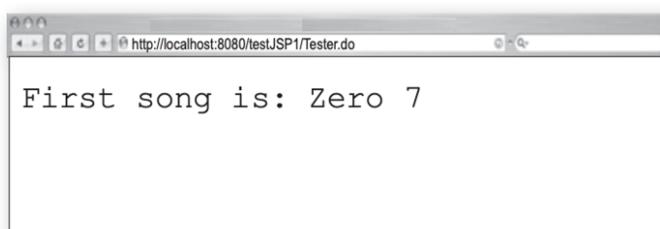
In a JSP

Music is: \${musicList}



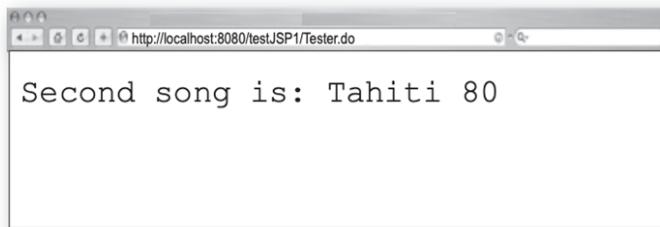
Makes sense... calls
toString() on the array.

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



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

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



In a Servlet

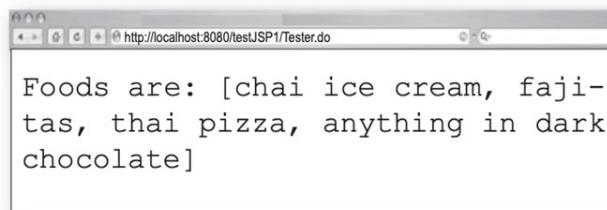
```
java.util.ArrayList favoriteFood = new java.util.ArrayList();
favoriteFood.add("chai ice cream");
favoriteFood.add("fajitas");
favoriteFood.add("thai pizza");
favoriteFood.add("anything in dark chocolate");
request.setAttribute("favoriteFood", favoriteFood);
```

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

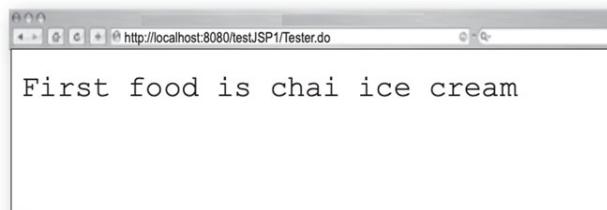
In a JSP

Foods are: \${favoriteFood}

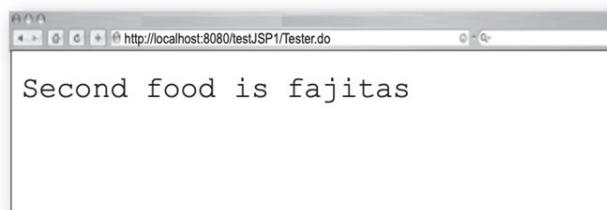
Obviously ArrayList has a nice overridden `toString()`.



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);
```

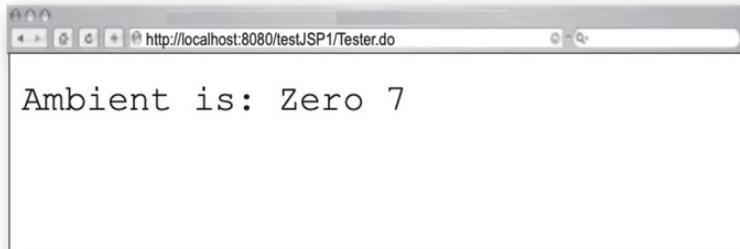
Make a Map, put some String keys and objects in it, then make it a request attribute.

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.

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)

A screenshot of a web browser window showing an HTML form. The URL is http://localhost:8080/testJSP1/TestBean.jsp. The form contains four input fields: 'Name' with value 'Fluffy', 'ID#' with value '423', 'First food' with value 'Sushi', and 'Second food' with value 'Macaroni & Cheese'. A 'Submit' button is at the bottom right.

The response

A screenshot of a web browser window showing the JSP response. The URL is http://localhost:8080/testJSP1/Tester.do. The page displays the submitted data: Request param name is: Fluffy, Request param empID is: 423, Request param food is: Sushi, First food request param: Sushi, Second food request param: Macaroni & Cheese, 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**

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**



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:

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**

true a boolean literal

false the OTHER boolean literal

null It means... null

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."

EL handles null values gracefully

\${7 + foo}	7	In arithmetic expressions, EL treats the unknown variable as "zero".
\${7 / foo}	Infinity	
\${7 - foo}	7	
\${7 % foo}	Exception is thrown	

\${7 < foo}	false	
\${7 == foo}	false	
\${foo == foo}	true	In logical expressions, EL treats the unknown variable as "false".
\${7 != foo}	true	
\${true and foo}	false	
\${true or foo}	true	
\${not foo}	true	

Esempio (bean2.zip)

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8" %>
3 <!DOCTYPE html>
4 <jsp:useBean id="time" class="bean.CurrentTimeBean" scope="session"/>
5<html>
6<head>
7 <title>Bean</title>
8 </head>
9<body>
10 <p> Sono le ore ${time.hours} e ${time.minutes}</p>
11 </body>
12 </html>
```

```
1 package bean;
2
3*import java.io.Serializable;□
5
6 public class CurrentTimeBean implements Serializable {
7
8     private int hours;
9     private int minutes;
10
11    public CurrentTimeBean() {
12        Calendar now = Calendar.getInstance();
13        this.hours = now.get(Calendar.HOUR_OF_DAY);
14        this.minutes = now.get(Calendar.MINUTE);
15    }
16
17    public int getHours() {
18        return hours;
19    }
20
21    public void setHours(int hours) {
22        this.hours = hours;
23    }
24
25    public int getMinutes() {
26        return minutes;
27    }
28
29    public void setMinutes(int minutes) {
30        this.minutes = minutes;
31    }
32 }
```