Java Server Pages

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Java Server Pages

- Tecnologia che utilizza template data, elementi custom, linguaggi e oggetti Java server-side per comporre pagine dinamiche
- Tipicamente i dati statici o template data sono rappresentati in
  - HTML
  - XML

Cosa è una JSP

- Documento in forma testuale in grado di restituire sia contenuti statici che contenuti dinamici ai client
- Nelle JSP sia i contenuti statici che i contenuti dinamici possono essere gestiti in modo semplice ed intermezzati gli uni agli altri
  - Contenuti statici: HTML, XML, Testo
  - Contenuti dinamici: codice Java, proprietà dei JavaBean, …

Primo Esempio di JSP

```
<html>
<body>Orario<br>
Sono le <%= new java.util.Date() %>
</body>
</html>
```

La pagina ha la struttura di una pagina statica, ma data e ora vengono generate dinamicamente

Servlet e JSP

- Nella servlet la logica per la generazione del documento HTML è implementata completamente in Java
  - È quindi tedioso e prono ad errori il processo di generazione delle pagine
  - È scomodo aggiornare la struttura delle pagine quando necessario
- Le JSP nascono per facilitare l'aggiornamento e l'authoring delle pagine
  - I web designer possono produrre pagine senza la necessità di conoscere i dettagli della logica server side
  - La generazione di codice dinamico è implementata sfruttando un linguaggio di scripting. A default si utilizza Java
  - Il codice delle JSP viene compilato in Servlet

Vantaggi derivanti dall'adozione delle JSP

- I contenuti sono separati dalla logica di presentazione
- Lo sviluppo delle applicazioni è semplificata rispetto alla programmazione tramite servlet
- Il deployment delle JSP è automatico. Le JSP vengono ricompilate "automaticamente" quando sono apportate modifiche alla pagina
- L'authoring delle pagine è particolarmente semplice
- Poiché vengono compilate in servlet anche le JSP sono portabili

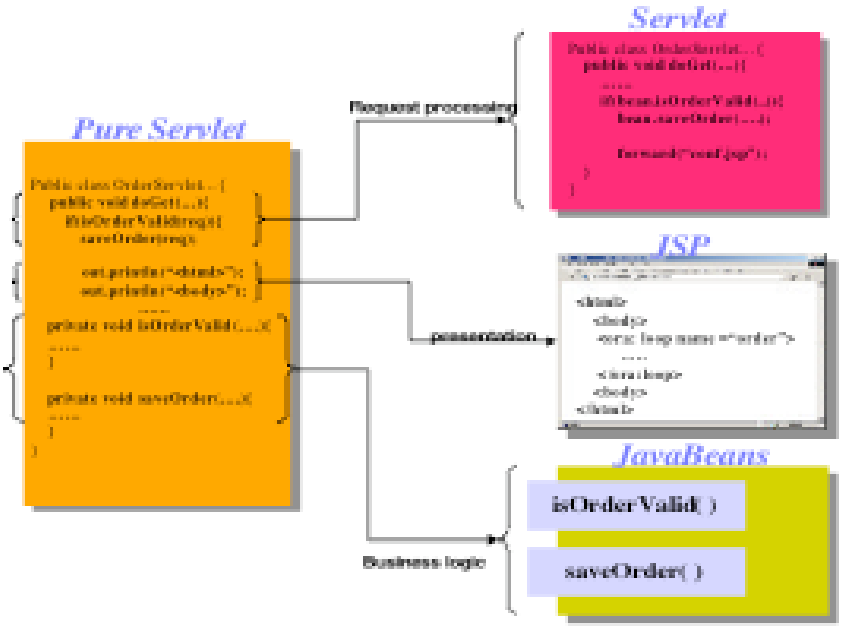
I limiti delle Servlet

- Le servlet forniscono un adeguato supporto per la generazione di pagine dinamiche ma hanno diversi svantaggi
  - Per generare le pagine (anche gli elementi statici) è scrivere del codice
  - Ogni modifica alla pagina richiede la ricompilazione e l'installazione della nuova release della servlet
  - La compilazione e l'installazione può essere tediosa

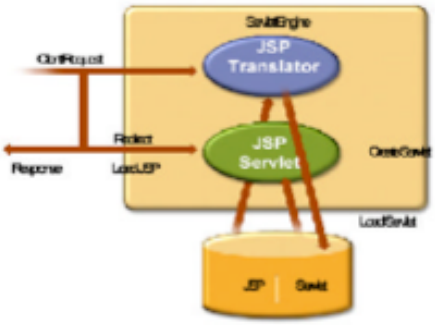
Servlet o JSP?

- Le JSP non rendono inutili le servlet
- Le servlet forniscono agli sviluppatori delle web application 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. può essere conveniente lavorare con le servlet
  - Controlling e dispatching
- Le JSP rendono invece molto semplice presentare documenti HTML o XML all'utente
- Servlet e JSP sono perciò usate congiuntamente in applicazioni strutturate in accordo al modello Model-View-Controller.
  - JSP ricopre il ruolo di View
  - Servlet ricopre il ruolo di Controller

Servlet e JSP

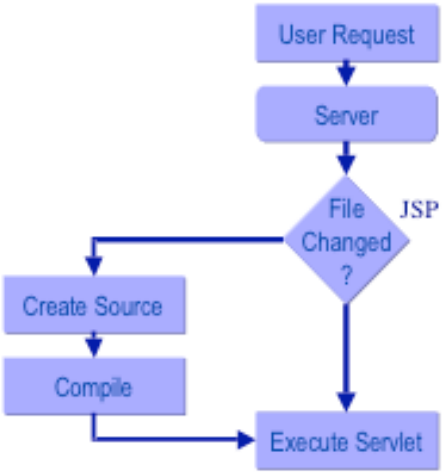


Ciclo di Vita delle JSP



- Quando un client manda una HTTP request ad una JSP la pagina viene compilata in una servlet
  - La compilazione avviene alla prima request
- Per alcuni web container la compilazione avviene a deployment time

Algoritmo per la Gestione delle JSP



Quando una HTTP Request è mappata su una JSP, questa viene gestita da una servlet fornita dal servlet container. Il servlet container prima controlla se la release della JSP a disposizione è più vecchia della servlet di cui dispone. In questo caso la JSP viene codificata in una servlet Java, che viene compilata in una classe

Fase di Codifica e Compilazione

- Le JSP vengono codificate in codice sorgente Java che viene compilato
- Codifica e compilazione sono a carico del JSP container (tipicamente alla prima request)
- I dati statici della pagina sono trasformati in codice che fornisce dati allo strem di uscita
- Vari elementi della JSP usati per la generazione di contenuto dinamico sono trattati come segue
  - Le direttive controllano come il Web Container codifica ed esegue le JSP
  - Gli elementi di scripting sono inseriti nel codice della servlet corrispondente alla JSP
  - Gli elementi nella forma <jsp:xxx /> sono convertiti in chiamate a componenti JavaBean

Ciclo di Vita della JSP

Se una istanza della servlet corrispondente alla JSP non esiste allora il JSP container la carica. Il JSP container poi istanzia la servlet e la inizializza invocando il metodo jspInit. Il container poi invoca il metodo jspService() passando gli oggetti Request e Response. Inifine, quando è necessario rimuovere la (servlet corrispondente alla) JSP il JSP container invoca jspDestroy()

Inizializzazione e finalizzazione di una JSP

- È possibile customizzare il processo di inizializzazione e di finalizzazione della JSP per consentire di
  - Leggere dati di configurazione persistenti
  - Inizializzare le risorse
  - Rilasciare risorse
  - …
- È possibile effettuare l'overriding di jspInit() e di jspDestroy()

Esempio

```
<%@ page import="database.*" %>
<%@ page errorPage="errorpage.jsp" %>
<%--  commento -%>
<%!
    private BookDBAO bookDBAO;
    public void jspInit() {     // ottiene un oggetto per l'accesso al DB
    bookDBAO = (BookDBAO)getServletContext().getAttribute("bookDB");
    if (bookDBAO == null)
          System.out.println("Couldn't get database.");
  }

    public void jspDestroy() {
          bookDBAO = null;  // rilascia l'oggetto
    }
%>
```


Primo Esempio di JSP

- Ogni documento HTML può diventare una JSP semplicemente cambiandone l'estensione da htm o html a jsp
- Il punto di forza delle JSP è però quello di consentire la possibilità di incapsulare codice Java-like nel documento
- Esempio

Ogni volta che ricarichiamo la pagina il tempo visualizzato è quello corrente

```
<HTML>
<BODY>
Hello!  The time is now <%= new java.util.Date() %>
</BODY>
</HTML>
```

Come Funzionano le JSP

- Ogni volta che arriva una request il server compone dinamicamente il contenuto della pagina e valuta l'espressione Java compresa fra **<%=** e **%>**
- Questo meccanismo permette di presentare all'utente pagine in cui contenuto è generato dinamicamente

Le Scriptlets

- Lo sviluppo di web application richiede strumenti sintattici più complessi di quelli visti sin ora
- Le JSP consentono di scrivere interi blocchi di codice Java
- Una scriptlet contiene codice Java che viene eseguito ogni volta che la JSP viene invocata

```
<HTML>
<BODY>
<% System.out.println( "Evaluating date now" );
   java.util.Date date = new java.util.Date();%>
Hello!  The time is now <%= date %>
</BODY>
</HTML>
```

Log del server

Si noti che la variabile date è disponibile nel seguito della JSP

Le Scriptlets

- Di per se le scriplet non generano codice HTML ed è necessario utilizzare l'oggetto **out** di classe **javax.servlet.jsp.JspWriter**. Questo oggetto è predefinito per le scriptlets e non deve essere istanziato esplicitamente (oggetto implicito)

```
<HTML>
<BODY>
<%
   System.out.println( "Evaluating date now" );
   java.util.Date date = new java.util.Date();
%>
Hello!  The time is now
<%
   // This scriptlet generates HTML output
   out.println( String.valueOf( date ));
%>
</BODY>
</HTML>
```


Request

- Analogamente alle servlet anche per le JSP possiamo beneficiare dell'oggetto predefinito request di classe javax.servlet.http.HttpServletRequest
- Request risponde ai medesimi requisiti del suo corrispondente nelle servlet. Esempio:

```
<HTML>
<BODY>
<%java.util.Date date = new java.util.Date();%>
Hello!  The time is now
<%out.println( date );
   out.println( "<BR>Your machine's address is " );
   out.println(request.getRemoteHost());%>
</BODY>
</HTML>
```

Response

- Analogo al corrispondente nelle servlet e con le medesime funzioni
- FARE ESEMPIO

Scriptlet e HTML

- Utilizzare out per la generazione del documento HTML non è vantaggioso
- Si tende perciò ad adottare un approccio differente utilizzando la scriptlet congiutamente all'HTML
- Esempio generazione di una tavola composta di numeri da 1 ad n

```
…
<TABLE BORDER=2>
<%
   for ( int i = 0; i < n; i++ ) {
      %>
      <TR>
      <TD>Number</TD>
      <TD><%= i+1 %></TD>
      </TR>
      <%
   }
%>
</TABLE>
```

Scriptlet e HTML

- La tecnica di frammezzare HTML e Scriplet è applicabile per le diverse strutture di controllo

```
<%
   if ( hello ) {
      %>
      <P>Hello, world
      <%
   } else {
      %>
      <P>Goodbye, world
      <%
   }
%>
```


Java Server Pages: Direttive

- Definiscono informazioni particolari per la pagina:
  - Definiscono il linguaggio di scripting utilizzato
  - Includono il contenuto di altre pagine
  - Definiscono i parametri di utilizzo di una Tag Library
  - etc.
- Non producono nessun output visibile
- Generano side effects che cambiano il modo in cui il JSP container processa la pagina

Java Server Pages: Page Directive

<%@ page attribute1="value1" attribute2="value2" attribute3=… %>

```
<%@ page info="This is a valid set of page directives." %>
<%@ page language="java" import="java.net.*" %>
<%@ page import="java.util.List, java.util.ArrayList" %>
```

```
<%@ page info="This is an invalid page directive" session="false“
          buffer="16k" autoFlush="false" session="false" %>
```

```
<%@ page info="This is not a valid set of page directives." %>
<%@ page extends="com.taglib.wdjsp.MyJspPage“
          info="Use my superclass." %>
```


Java Server Pages: Page Directive

Attribute	Value	Default	Examples
info	Text string	None	info="Registration form."
language	Scripting language name	"java"	language="java"
contentType	MIME type, character set	See first example	contentType="text/html; charset=ISO-8859-1" contentType="text/xml"
extends	Class name	None	extends="com.taglib.wdjsp.MyJspPage"
import	Class and/or package names	None	import="java.net.URL" import="java.util.*, java.text.*"
session	Boolean flag	"true"	session="true"
buffer	Buffer size, or false	"8kb"	buffer="12kb" buffer="false"
autoFlush	Boolean flag	"true"	autoFlush="false"
isThreadSafe	Boolean flag	"true"	isThreadSafe="true"
errorPage	Local URL	None	errorPage="results/failed.jsp"
isErrorPage	Boolean flag	"false"	isErrorPage="false"

Esempio

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%
   System.out.println( "Evaluating date now" );
   Date date = new Date();
%>
Hello!  The time is now <%= date %>
</BODY>
</HTML>
```

Per includere vari package posso usare la notazione

<%@ page import="java.util.*,java.text.*" %>

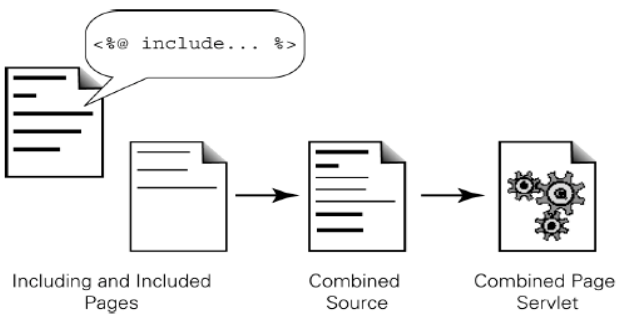
Java Server Pages: Include Directive

<%@ include file="localURL" %>

- Serve ad includere il contenuto del file specificato
- E' possibile nidificare un numero qualsiasi di inclusioni
- Esempi:

<%@ include file="includes/navigation.jsp" %>

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



Esempio

```
<HTML>
<BODY>
Going to include hello.jsp...<BR>
<%@ include file="hello.jsp" %>
</BODY>
</HTML>
```


Java Server Pages: Taglib Directive

Questa direttiva **specifica** al **JSP container** che **la pagina richiede** una o più **tag library**. Una tag library è una **collezione** di **tag custom** che **estendono** le **funzionalità delle JSP**. Quando si dichiara che la pagina dipende da una specifica tag library allora **tutti i custom tag** che questa **implementa** sono **resi disponibili alla pagina**.

<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>

- Esempi:

```
<%@ taglib uri="/EncomTags" prefix="mcp" %>

<mcp:endProgram/>
```

Java Server Pages: Scripting-Oriented Tag

Questi sono esempi di scripting tag JSP

Ogni tag è auto contenuto e delimitato da **<% %>**

- Sono disponibili altri delimitatori di apertura: **<%!**, **<%=**, **<%@**
- Esempi:

<%! double radius = 7.5; **%>** **→ codice dello script**

<%= 2 * Math.PI * radius **%> → valutazione di una espressione**

```
<% if (radius > 10.0) {
        out.println("Exceeds recommended maximum");
} %>
```

<%@ include file="copyright.html" %> → direttiva

Java Server Pages: XML-Based Tag

Le JSP hanno una seconda classe di tag che seguono le norme sintattiche e le convenzioni di XML.

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

<jsp:useBean id="login“
        class="com.taglib.wdjsp.fundamentals.UserBean">
  <jsp:setProperty name="login" property="group" value="admin"/>
</jsp:useBean>
```

Java Server Pages: Output Buffering

- **Limiti del protocollo HTTP:**
  - Il **Client** si **aspetta** di ricevere **tutto il response header prima** del **response body**: la **JSP** deve **effettuare** tutte le **modifiche** all'**header** (es: modifica di un cookie) **prima** di cominciare a **creare** il **body**
  - Una volta che il **web server** comincia a **servire** la **risposta non** può più **interrompere** il **processo**, altrimenti il browser mostra solo la frazione parziale che ha ricevuto: se ho iniziato a eseguire la JSP non posso più effettuare un forward ad un'altra JSP (se per esempio voglio gestire il verificarsi di eventuali errori)
- **Buffering dell'output:**
  - Quando si processa una JSP l'**output prodotto** non viene immediatamente restituito dal container, ma **viene bufferato** fino a quando la **JSP** non viene **completamente processata** e la **response** è **completa** sia nella parte header che nella parte body (è possibile effettuare redirect in qualsiasi momento)
  - Tutto questo con **il limite della dimensione del buffer**

Java Server Pages: Sessione

- Possibilità di **gestire efficientemente** i **cookie** (come per le servlet)
- Esiste un **oggetto session implicito** che può essere utilizzato per la sessione:
  - utilizza lo **stesso sistema delle servlet**
  - **sessione server side**
  - la sessione **scade dopo un determinato time-out**
- **Tutte le JSP** che partecipano alla gestione della sessione di interazione con un utente **possono memorizzare informazioni** nell'oggetto implicito **session**
- **Attenzione** alla **memoria** richiesta per la **gestione** della **sessione** degli utenti ed al **tempo** in cui questa è mantenuta
  - per il tempo della sessione vedi parametri del **JSP container** e metodi di **javax.servlet.http.HttpSession**

Java Server Pages: Script Tag

- **Consentono** di **inserire codice** in una **pagina JSP**
- Ci sono **tre tipi di Script Tag**:
  - **Dichiarazioni**: consentono di definire variabili e mteodi per una pagina.
  - **Espressioni**: la valutazione delle espressioni restituisce un valore che viene presentato all'utente nella pagina
  - **Scriptlet**: sezioni di codice che vengono eseguite ogni volta che la pagina viene processata.
- **Nessun tipo di Script Tag ha attributi**

Java Server Pages: Dichiarazioni

<%! declaration(s) %>

- Serve a **definire variabili** e **metodi** per una **JSP**. Metodi e variabili **possono essere referenziati** da altri **scripting element** nella **stessa pagina**
- E' possibile effettuare **dichiarazioni multiple**
- Le variabili definite diventano **variabili di istanza** della **servlet** generata

```
<%! private int x = 0, y = 0; private String units = "ft"; %>
```

- E' possibile dichiarare anche uno o più metodi

```
<%! public long fact (long x) {
      if (x == 0) return 1;
      else return x * fact(x-1);
    } %>
```

Java Server Pages: Dichiarazioni

Le variabili definite come dichiarazioni diventano variabili di istanza della servlet corrispondente alla JSP

<%! private int x = 0, y = 0; private String units = "ft"; %>

La direttiva sopra ha come effetto la creazione di tre istanze di variabile nella JSP. Esistono ovvimente problemi correlati alla **thread safety**. A seguito di più request è infatti **possibile** che **diversi thread** operino sulle **stesse variabili**.

Per creare **variabili con scope locale** alla **request** è opportuno **definirle all'interno di una scriplet**

Java Server Pages: Dichiarazioni

È possibile anche creare variabili statiche. Esempio

<%! static public int counter = 0; %>

Tutte le istanze della servlet associata alla JSP condividono questa variabile. Se il valore viene cambiato il nuovo valore è visibile a tutte le pagine

Java Server Pages: Espressioni

<%= expression %>

- **Scopo** principale delle espressioni e quello di **generare output** da presentare nelle pagine
- Il risultato dell'espressione viene valutato e visualizzato
- Esempi:

```
<%= Math.PI %>
<%= Math.PI * Math.pow(radius, 2) %>
<%= fact(12) %>
```

- Il risultato dell'espressione può essere di qualsiasi tipo che viene poi convertito in stringa per la visualizzazione
- **Le espressioni non sono terminate da un ";" perché non sono statement**
- E molto utilizzato con le espressioni l'operatore ternario condizionale:

```
test_expr ? true_expr : false_expr
```

- Esempio

```
<%= (hours < 12) ? "AM" : "PM" %>
```


Java Server Pages: Scriptlets

<% scriptlet %>

- Possono contenere **qualsiasi statement** del linguaggio di **scripting** (per default il linguaggio di scripting è **Java**)
- E' possibile **lasciare aperto** un **blocco** di istruzioni utilizzando {
- Gli scriptlet di una pagina vengono **eseguiti ad ogni chiamata**
- Esempio:

```
<% GameGrid grid = GameGrid.getGameGrid();
   Recognizer r1 = new Recognizer(new Coordinates(grid, 0, 0));
   Recognizer r2 = new Recognizer(new Coordinates(grid,10,1));
   r1.findProgram("Flynn");
   r2.findProgram("Flynn"); %>
<%= r1.statusReport() %>
```

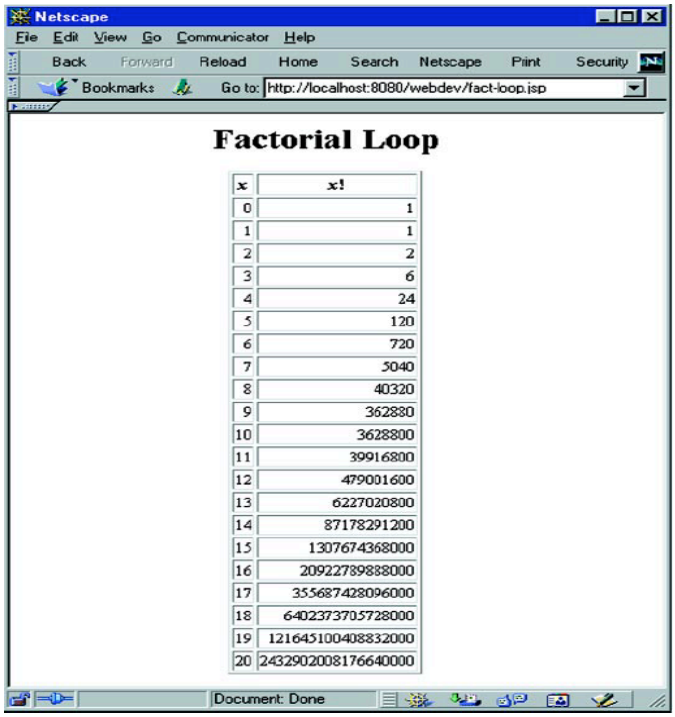
Java Server Pages: Condizioni

```
<% if (x < 0) { %>
  <p>
    Sorry, can't compute the factorial of a negative number.
  </p>
<% } else if (x > 20) { %>
  <p>
    Sorry, arguments greater than 20 cause an overflow error.
  </p>
<% } else { %>
  <p align=center><%= x %>! = <%= fact(x) %></p>
<% } %>
```


Java Server Pages: Iterazioni

```
<table>
  <tr>
    <th><i>x</i></th><th><i>x</i>! </th>
  </tr>
  <% for (long x = 0l; x <= 20l; ++x) { %>
    <tr>
      <td><%= x %></td><td><%= fact(x) %></td>
    </tr>
  <% } %>
</table>
```

Java Server Pages: Iterazioni



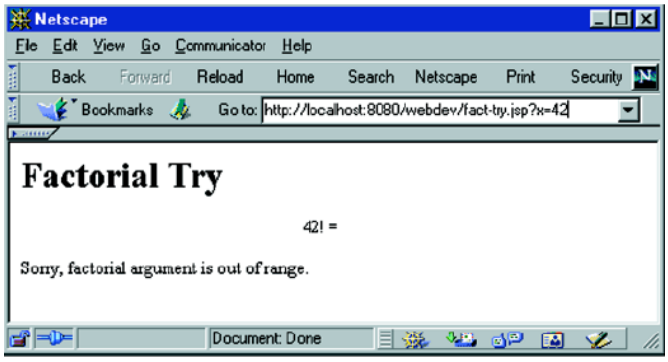
x	x!
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000
16	20922789888000
17	355687428096000
18	6402373705728000
19	121645100408832000
20	2432902008176640000

Java Server Pages: Gestione delle Eccezioni

```
<%! public long fact (long x) throws IllegalArgumentException {
      if ((x < 0) || (x > 20))
        throw new IllegalArgumentException("Out of range.");
      else if (x == 0) return 1;
      else return x * fact(x-1);
    } %>

<% try { %>
  <p align=center> <%= x %>! = <%= fact(x) %></p>
<% } catch (IllegalArgumentException e) { %>
  <p>Sorry, factorial argument is out of range.</p>
<% } %>
```

Java Server Pages: Gestione delle Eccezioni




```
<% try {
      long result = fact(x); %>
  <p align=center> <%= x %>! = <%= result %></p>
<% } catch (IllegalArgumentException e) { %>
  <p>Sorry, factorial argument is out of range.</p>
<% } %>
```


Java Server Pages: Commenti ai contenuti

```
<!-- comment -->
```

- Commenti in HTML (o XML)
- Vengono inseriti nell'output esattamente senza essere processati
- Possono anche essere generati dinamicamente poiché sono parte della pagina che viene restituita al client
- Esempio:

```
<!-- Java longs are 64 bits, so 20! = <%= fact(20) %> is the limit. -->
```

Java Server Pages: Commenti JSP

<%-- comment --%>

- Sono indipendenti dal tipo di contenuto generato
- Sono indipendenti dal linguaggio di scripting utilizzato
- **Non vengono trasferiti sull'output**
- Il JSP container ignora il contenuto quando processa la pagina
- Esempio:

```
5! = <%= fact(5) %><br>
<%--
 6! = <%= fact(6) %><br>
 7! = <%= fact(7) %><br>
 8! = <%= fact(8) %><br>
--%>
 9! = <%= fact(9) %><br>
```


Java Server Pages: Commenti del linguaggio

<% /* comment */ %>

- Viene utilizzata la **sintassi nativa del linguaggio di scripting**
- **Non appaiono nell'output della pagina**
- **Appaiono** nel codice **sorgente della servlet** generata dalla JSP
- Esempi:

```
<% long valid = fact(20);
        long overflow = fact(21); /* Exceeds 64-bit long! */
   %>
<%= /* Comment before expression */ fact(5) %>
<%= fact(7) /* Comment after expression */ %>
<%= fact(9 /* Comment inside expression */) %>
<% long valid = fact(20);// This one fits in a 64-bit long.
        long overflow = fact(21);// This one doesn't.
 %>
Lora's brother is over <%= fact(3) // Strange ruler... %> feet tall!
```

Java Server Pages: Implicit Objects

- **Oggetti** automaticamente **disponibili nelle pagine JSP**
- **Gestiti e forniti dal JSP Container**
- Ci sono **nove implicit object** forniti dalle JSP. Gli implicit object possono essere **divisi** in **quattro** principali **classi**:
  - oggetti legati alla **servlet** relativa alla pagina JSP
  - oggetti legati all'**input** e all'**output** della pagina JSP
  - oggetti che forniscono **informazioni** sul **contesto** in cui la JSP viene **eseguita**
  - oggetti **risultanti da eventuali** e possibili **errori**

Java Server Pages: Implicit Objects

Object	Class or *Interface*	Description
page	*javax.servlet.jsp.HttpJspPage*	Page's servlet instance.
config	*javax.servlet.ServletConfig*	Servlet configuration data.
request	*javax.servlet.http.HttpServletRequest*	Request data, including parameters.
response	*javax.servlet.http.HttpServletResponse*	Response data.
out	javax.servlet.jsp.JspWriter	Output stream for page content.
session	*javax.servlet.http.HttpSession*	User-specific session data.
application	*javax.servlet.ServletContext*	Data shared by all application pages.
pageContext	javax.servlet.jsp.PageContext	Context data for page execution.
exception	java.lang.Throwable	Uncaught error or exception.

JSP implicit objects and their API's for HTTP applications

Java Server Pages: page Object

- **Rappresenta** la **pagina JSP** (una istanza della classe della servlet associata la JSP)
- **Poco usato** nella pratica poiché gli **scripting elments** sono **tradotti** in codice **Java** ed **hanno accesso** ai vari **metodi** della servlet
- Per linguaggi di **scripting differenti** da **Java** è **utile** per **dare accesso** alla interfaccia **javax.servlet.jsp.JspPage**
- Esempio:

```
<%@ page info="Page implicit object demonstration." %>
Page info:
<%= ((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>
```


Java Server Pages: config Object

- Oggetto **legato** alla **servlet relativa** alla **pagina JSP**
- L'oggetto contiene la **configurazione** della servlet (parametri di inizializzazione)
- **Poco usato** in pratica (**poco usati i parametri di inizializzazione**)
- è istanza di **javax.servlet.Servlet-Config**

Method	Description
`getInitParameterNames()`	Retrieves the names of all initialization parameters.
`getInitParameter(name)`	Retrieves the value of the named initialization parameter.

Java Server Pages: config Object

```
<%! static private DbConnectionPool pool = null;
public void jspInit () {
if (pool == null) {
String username = config.getInitParameter("username");
String password = config.getInitParameter("password");
pool = DbConnectionPool.getPool(this, username, password);
}
} %>
```

In questo caso **invece** di **memorizzare username** e **password nella JSP** si utilizzano i **parametri** di **configurazione** che sono **acceduti dall'oggetto implicito**. I valori di inizializzazione sono memorizzati nel **descrittore di deployment**

Java Server Pages: request Object

- Oggetto **legato all'I/O** della pagina **JSP**
- L'oggetto request **rappresenta** la **richiesta** per la pagina **JSP**
- implementa **javax.servlet.ServletRequest** o nel caso di protocollo HTTP **javax.servlet.http.HttpServletRequest**
- **Consente l'accesso** a tutte le **informazioni** della request:
  - l'indirizzo di provenienza
  - l'URL richiesto
  - tutti gli headers
  - i cookies
  - i parametri associati alla richiesta

Java Server Pages: request Object

```
<% String xStr = request.getParameter("num");
    try {
      long x = Long.parseLong(xStr); %>
      Factorial result: <%= x %>! = <%= fact(x) %>
<%    } catch (NumberFormatException e) { %>
      Sorry, the <b>num</b> parameter must specify an integer value.
<%    } %>
```

Andiamo a **raccogliere** il parametro **num** dalla **request**. Si noti che poiché tutti i **parametri** sono memorizzati come **stringhe** è necessaria una **conversione** per passare a **rappresentazioni numeriche qualora** siano **necessarie**.

Java Server Pages: request Object

Method	Description
`getParameterNames()`	Returns the names of all request parameters
`getParameter(name)`	Returns the first (or primary) value of a single request parameter
`getParameterValues(name)`	Retrieves all of the values for a single request parameter.

Method	Description
`getHeaderNames()`	Retrieves the names of all of headers associated with the request.
`getHeader(name)`	Returns the value of a single request header, as a string.
`getHeaders(name)`	Returns all of the values for a single request header.
`getIntHeader(name)`	Returns the value of a single request header, as an integer.
`getDateHeader(name)`	Returns the value of a single request header, as a date.
`getCookies()`	Retrieves all of the cookies associated with the request.

Java Server Pages: request Object

Method	Description
`getMethod()`	Returns the HTTP (e.g., `GET`, `POST`) method for the request.
`getRequestURI()`	Returns the request URL, up to but not including any query string.
`getQueryString()`	Returns the query string that follows the request URL, if any.
`getSession(flag)`	Retrieves the session data for the request (i.e., the `session` implicit object), optionally creating it if it doesn't already exist.
`getRequestDispatcher(path)`	Creates a request dispatcher for the indicated local URL.
`getRemoteHost()`	Returns the fully qualified name of the host that sent the request.
`getRemoteAddr()`	Returns the network address of the host that sent the request.
`getRemoteUser()`	Returns the name of user that sent the request, if known.
`getSession(flag)`	Retrieves the session data for the request (i.e., the `session` implicit object), optionally creating it if it doesn't already exist.

Java Server Pages: request Object

```
<%  String nome=new String();
      String _HERE= request.getRequestURI();
      java.util.Enumeration eNames=request.getParameterNames();

      util.Debug.println("--"+_HERE+" (BEGIN)----------------------");
      while (eNames.hasMoreElements()) {
        nome=(String)eNames.nextElement();
        String[] values=request.getParameterValues(nome);
        for (int pc=0;pc<values.length;pc++)
          util.Debug.println(nome+": ["+pc+"]:"+values[pc]+" ");
        }
      util.Debug.println("--"+_HERE+" (END)----------------------"); %>
```

Java Server Pages: response Object

- Oggetto **legato** all'**I/O** della **pagina JSP**
- **Rappresenta** la **risposta** che viene **restituita** al **client**
- implementa l'interfaccia **javax.servlet.ServletResponse** o nel caso di HTTP **javax.servlet.http.HttpServletResponse**
- Consente di **inserire** nella **risposta** diverse **informazioni**:
  - il **content type** e l'enconding
  - **eventuali header** di risposta
  - **URL Rewriting**
  - i **cookies**

Java Server Pages: response Object

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

Il settaggio di Expires a 0 è un trucco per **evitare** il **caching della pagina**. Il valore no-cache di **Pragma** per indicare che i browser ed i proxy non devono mettere in cache la pagina. **CacheControl** ha analoga funzione a pragma per HTTP 1.1. Usiamo sia Pragma che CacheControl per **mantenere** la **compatibilità** anche nel caso di **connessioni HTTP 1.0**.

Java Server Pages: response Object

Method	Description
`setContentType()`	Set the MIME type and, optionally, the character encoding of the response's contents.
`getCharacterEncoding()`	Returns the character encoding style set for the response's contents.

Method	Description
`addCookie(cookie)`	Adds the specified cookie to the response.
`containsHeader(name)`	Checks whether the response includes the named header.
`setHeader(name, value)`	Assigns the specified string value to the named header.
`setIntHeader(name, value)`	Assigns the specified integer value to the named header.
`setDateHeader(name, date)`	Assigns the specified date value to the named header.
`addHeader(name, value)`	Adds the specified string value as a value for the named header.
`addIntHeader(name, value)`	Adds the specified integer value as a value for the named header.
`addDateHeader(name, date)`	Adds the specified date value as a value for the named header.

Java Server Pages: response Object

Method	Description
setStatus(code)	Sets the status code for the response (for non-error circumstances).
sendError(status, msg)	Sets the status code and error message for the response.
sendRedirect(url)	Sends a response to the browser indicating it should request an alternate (absolute) URL.

Method	Description
encodeRedirectURL(url)	Encodes a URL for use with the `sendRedirect()` method to include session information.
encodeURL(name)	Encodes a URL used in a link to include session information.

Java Server Pages: out Object

- Oggetto legato all'I/O della pagina JSP
- **Rappresenta** lo **stream di output** della **pagina**
- è istanza della classe **javax.servlet.jsp.JspWriter**
- Esempio:

```
<P>Counting eggs
<% int count = 0;
      while (carton.hasNext()) {
        count++;
        out.print(".");
      } %>
<BR>
There are <%= count %> eggs.</P>
```


Java Server Pages: out Object

Method	Description
`isAutoFlush()`	Returns `true` if the output buffer is automatically flushed when it becomes full, `false` if an exception is thrown.
`getBufferSize()`	Returns the size (in bytes) of the output buffer.
`getRemaining()`	Returns the size (in bytes) of the unused portion of the output buffer.
`clearBuffer()`	Clears the contents of the output buffer, discarding them.
`clear()`	Clears the contents of the output buffer, signaling an error if the buffer has previously been flushed.
`newLine()`	Writes a (platform-specific) line separator to the output buffer.
`flush()`	Flushes the output buffer, then flushes the output stream.
`close()`	Closes the output stream, flushing any contents.

Java Server Pages: session Object

- Oggetto che fornisce **informazioni sul contesto** di esecuzione della JSP
- **Rappresenta** la **sessione corrente** per un utente
- La **direttiva session** deve essere **true**
- Esempio:

```
<% 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);
  } %>
```


Java Server Pages: session Object

Method	Description
`getId()`	Returns the session ID.
`getCreationTime()`	Returns the time at which the session was created.
`getLastAccessedTime()`	Returns the last time a request associated with the session was received.
`getMaxInactiveInterval()`	Returns the maximum time (in seconds) between requests for which the session will be maintained.
`setMaxInactiveInterval(t)`	Sets the maximum time (in seconds) between requests for which the session will be maintained.
`isNew()`	Returns `true` if user's browser has not yet confirmed the session ID.
`invalidate()`	Discards the session, releasing any objects stored as attributes.

Java Server Pages: application Object

- Oggetto che **fornisce informazioni** sul **contesto di esecuzione** della **JSP**
- **Rappresenta** l'**applicazione** a cui la JSP appartiene
- **Consente** di **interagire** con **l'ambiente di esecuzione**:
  - **versione** del **JSP Container**
  - **accesso** a **risorse** server-side
  - **supporto al log**
  - accesso ai **parametri** di **inizializzazione** relativi all'**applicazione**
  - possibilità di **gestire gli attributi** di una **applicazione**

Java Server Pages: pageContext Object

- Oggetto che **fornisce informazioni** sul **contesto di esecuzione** della **JSP**
- **Rappresenta** l'insieme degli **oggetti impliciti** di una JSP
- **Consente** l'**accesso** a tutti gli **oggetti impliciti** e ai **loro attributi**
- Consente il **trasferimento** del **controllo** ad **altre pagine**
- **Poco usato per lo scripting**, utile per costruire **custom tags**

Java Server Pages: exception Object

- Oggetto connesso alla **gestione degli errori**
- **Rappresenta** l'**eccezione** che **non** viene **gestita** da **nessun blocco catch**
- Non è automaticamente **disponibile** in tutte le pagine (**solo** nelle **Error Page**)
- Esempio:

```
<%@ page isErrorPage="true" %>
<H1>Warning!</H1>
The following error has been detected:<BR>
<B><%= exception %></B><BR>
<% exception.printStackTrace(out); %>
```


Java Server Pages: exception Object

Method	Description
`getMessage()`	Returns the descriptive error message associated with the exception when it was thrown.
`printStackTrace(out)`	Prints the execution stack in effect when the exception was thrown to the designated output stream.
`toString()`	Returns a string combining the class name of the exception with its error message (if any).

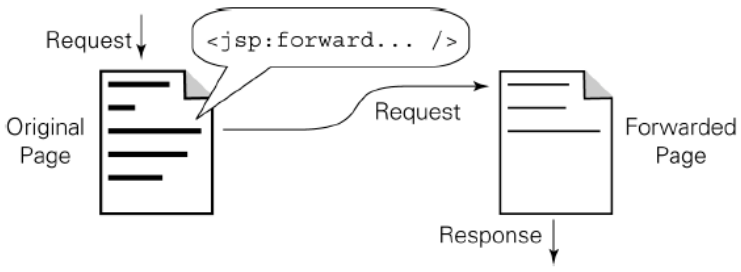
Java Server Pages: Actions

- Tag che **consentono di eseguire particolari azioni**
- Consentono di **trasferire il controllo** dalla pagina corrente **ad altre pagine**
- Consentono di **dialogare con i componenti** (JavaBeans)
- Viene **usata la sintassi XML**

Java Server Pages: Forward

```
<jsp:forward page="localURL" />
```

- Consente il **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**



Java Server Pages: Forward

<jsp:forward page="*localURL*" />

- E' possibile **generare dinamicamente l'attributo page**

  <jsp:forward page='<%= "message" + statusCode + ".html" %>' />
- Gli oggetti **session** e **request** della pagina d'arrivo **sono gli stessi** della **pagina chiamante**, **ma viene istanziato un nuovo contesto**
- E' **possibile aggiungere parametri** all'oggetto **request** della **pagina chiamata utilizzando** il tag **<jsp:param>**

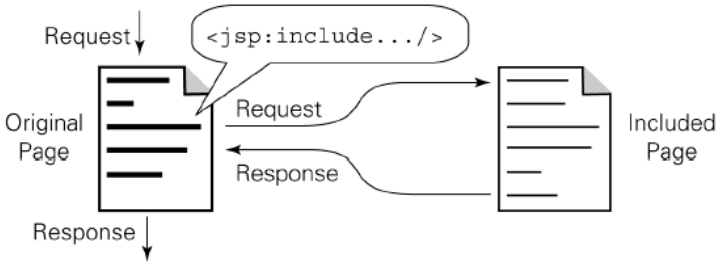
```
<jsp:forward page="localURL">
<jsp:param name="parameterName1“ value="parameterValue1"/>
  …
<jsp:param name="parameterNameN“ value="parameterValueN"/>
</jsp:forward>
```

- La **redirezione** è **possibile soltanto se non** è già stato **fatto** un **flush** del **buffer** di **output o se** la **chiamata avviene prima** della **creazione** di qualsiasi **output**

Java Server Pages: Include

<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**
- Questo tag **trasferisce temporaneamente** il **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** viene **eseguito** un **flush prima** di **effettuare l'inclusione** (**deve essere true**)
- Gli oggetti **session** e **request** per la **pagina** da **includere** sono gli **stessi** della pagina **chiamante**, ma viene **istanziato un nuovo contesto**



Java Server Pages: Include

<jsp:include page="*localURL*" flush="true" />

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

```
<jsp:include page="localURL“ flush=“true”>
  <jsp:param name="parameterName1“
             value="parameterValue1"/>
  …
  <jsp:param name="parameterNameN“
             value="parameterValueN"/>
</jsp: include >
```


Java Server Pages: Modello a componenti

- **Scriptlet** e **espressioni** consentono uno **sviluppo centrato** sulla **pagina**
- Lo sviluppo centrato sulla pagina **non consente** una **forte separazione** tra **sviluppo dell'applicazione** e **presentazione** dei **contenuti**
- Un'**applicazione** molto **complessa necessita** di un **Architettura** a più **livelli**
- **Le JSP consentono uno sviluppo basato sul modello a componenti**
- Il Modello a Componenti consente di avere una **maggiore separazione** fra **logica dell'applicazione** e **contenuti**
- Il Modello a Componenti consente **architetture molto più articolate**

Java Server Pages: JavaBeans

- I **Componenti** solo **elementi software autonomi**, **riusabili** che **incapsulano** un certo **insieme di funzionalità**
- I Componenti **dialogano** fra loro e con l'esterno **attraverso** una **interfaccia ben definita**
- I **JavaBeans** sono **componenti software** scritti in **Java**
- I JavaBeans hanno delle **proprietà** che ne **definiscono lo stato**:
  - possono essere **read-only**, **write-only** o **readable** and **writable**
  - possono essere **trigger** o **linked**
  - possono essere **indexed**
  - **possono contenere qualsiasi tipo di dato Java**

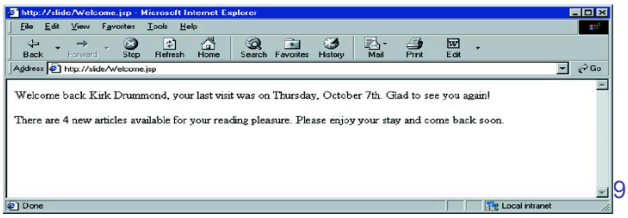
Java Server Pages: Bean Tags

- Esistono dei **tag** per **agganciare** un **bean** e le sue **proprietà**
- Ci sono **tre tipi di tag**:
  - Tag per creare un **riferimento al bean**
  - Tag per **settare il valore** delle **proprietà** del **bean**
  - Tag per **leggere** il **valore** delle **proprietà** del **bean**

Java Server Pages: Bean Tags

```
<jsp:useBean id="user" class="RegisteredUser" scope="session"/>
<jsp:useBean id="news" class="NewsReports" scope="request">
<jsp:setProperty name="news" property="category" value="financial"/>
<jsp:setProprety name="news" property="maxItems" value="5"/>
</jsp:useBean>

<html>
  <body>
    Welcome back
    <jsp:getProperty name="user" property="fullName"/>,
    your last visit was on
    <jsp:getProperty name="user" property="lastVisitDate"/>.
    Glad to see you again!
    <P>
      There are <jsp:getProperty name="news" property="newItems"/>
      new articles available for your reading pleasure. Please enjoy your
      stay and come back soon.
  </body>
</html>
```



Java Server Pages: <jsp:useBean>

```
<jsp:useBean id="beanName" class="class name"/>
```

- **Inizializza e crea la reference al bean**
- E possibile specificare il **tipo di bean** ed **assegnarli un nome**
- Gli attributi principali sono id e class ma è possibile specificarne altri

Attribute	Value	Default	Example Value
id	Java identifier	none	myBean
scope	page, request, session, application	page	session
class	Java class name	none	java.util.Date
type	Java class name	same as class	com.manning.jsp.AbstractPerson
beanName	Java class or serialized Bean	none	com.manning.jsp.USCurrency.ser

Java Server Pages: <jsp:useBean>

- Attributo **id**
  - **identificatore unico per il bean** (Java identifier)
- Attributo **class**
  - **indica il nome della classe del bean**
  - **la classe deve essere nel classpath**

```
<%@ page import="com.taglib.wdjsp.*" %>
<jsp:useBean name="user" class="RegisteredUserBean" />
                                    oppure
<jsp:useBean name="user" class="com.taglib.wdjsp.RegisteredUserBean" />
```


Java Server Pages: <jsp:useBean>

```
<jsp:useBean id="myclock" class="com.manning.jsp.ClockBean"/>

<html>
  <body>
    There is a Bean hiding in this page!
  </body>
</html>
```

- Questa **JSP userà** il **bean** definito dalla **classe com.manning.jsp.ClockBean**
- **Si farà riferimento al bean** mediante l'**identificatore myclock**
- Un bean va **definito prima di essere usato**
- **<jsp:useBean>** crea un'**istanza** del **bean** e gli **assegna** l'**id**
- Il bean, **quando istanziato**, **esegue determinate operazioni definite** dallo **sviluppatore**
- Ciascun tag crea **<jsp:useBean>** una diversa istanza di un bean

```
<jsp:useBean id="clock1" class="com.manning.jsp.ClockBean" />
<jsp:useBean id="clock2" class="com.manning.jsp.ClockBean" />
```

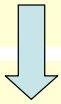
Java Server Pages: <jsp:getProperty>

```
<jsp:getProperty name="bean name"
                 property="property name"/>
```

- **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**, definisce il **nome del bean** a cui faccio riferimento
  - **property**, definisce il **nome della proprietà** di cui **voglio visualizzare il valore**

Java Server Pages: <jsp:getProperty>

```
<jsp:useBean id="style" class="beans.CorporateStyleBean"/>
<html>
 <body bgcolor="<jsp:getProperty name="style" property="color"/>">
  <center>
   <img src="<jsp:getProperty name="style" property="logo"/>">
   Welcome to Big Corp!
  </center>
 </body>
</html>
```



```
<html>
 <body bgcolor="pink">
  <center>
   <img src="http://imageserver/logo.gif">
   Welcome to Big Corp!
  </center>
 </body>
</html>
```

Java Server Pages: <jsp:setProperty>

```
<jsp:setProperty name="bean name“
                 property="property name“
                 value=“property value”/>
```

- Consente di **modificare** il **valore** delle **proprietà del bean**
- Il **bean eseguirà determinate operazioni** in **funzione del valore assegnato alle proprietà**
- Esempi:

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


Java Server Pages: Indexed Properties

- I **bean tag non** supportano le **proprietà indicizzate**
- Un bean è un **normale oggetto Java**: è possibile accedere a variabili e metodi.
- Esempio:

```
<b>Tomorrow’s Forecast</b>: <%= weather.getForecasts(0) %>
 <br/>
 <b>The Rest of the Week</b>
 <ul>
   <% for (int index=1; index < 5; index++) { %>
     <li><%= weather.getForecasts(index) %></li>
   <% } %>
 </ul>
```

Java Server Pages: Inizializzare un bean

- E' **possibile inizializzare** un **bean settando** il valore delle **proprietà** al momento **dell'inizializzazione**:

```
<jsp:useBean id="clock" class="com.manning.jsp.ClockBean">
  <jsp:setProperty name="clock" property="timezone" value="CST"/>
</jsp:useBean>
```

- E' possibile inizializzare un bean con i **parametri** della **request HTTP**:

```
<jsp:setProperty name="calculator" property="interestRate“
                 value="<%= request.getParameter(“interestRate") %>"/>
<jsp:setProperty name="calculator" property="years“
                 value="<%= request.getParameter(“years") %>"/>

<jsp:setProperty name="calculator" property="interestRate“ param="interestRate"/>
<jsp:setProperty name="calculator" property="years" param="years"/>

<jsp:setProperty name="calculator" property="interestRate"/>
<jsp:setProperty name="calculator" property="years"/>

<jsp:setProperty name="calculator" property="*">
```


Java Server Pages: Bean Life Cycle

- **Ogni volta** che una **pagina JSP** viene **richiesta** e **processata** il **bean viene istanziato**
- E' **possibile estendere** la **vita** del **bean oltre** la **singola richiesta**
- **scope** è **l'attributo** che **consente** di **estendere** la **vita del bean**
- Il bean viene identificato nel suo ciclo vitale dall'attributo id del tag <jsp:useBean>

Scope	Accessibility	Life span
page	current page only	until page is displayed or control is forwarded to a new page
request	current page and any included or forwarded pages	until the request has been completely processed and the response has been sent back to the user
session	the current request and any subsequent request from the same browser window	life of the user's session
application	the current and any future request that is part of the same web application	life of the application

Java Server Pages: Bean Life Cycle

```
<jsp:useBean id="beanName" class="class“
                scope="page|request|session|application"/>
```

- **Page Beans** (default scope)
  - **Ogni** volta che viene **richiesta** la pagina un **nuovo bean** viene **istanziato**
  - **bean transienti e non persistenti**
- **Request Beans**
  - Utilizzato per **estendere** la **vita** del **bean** alle **JSP richiamate** mediante i tag **<jsp:include>** e **<jsp:forward>**

Java Server Pages: Bean Life Cycle

```
<jsp:useBean id="beanName" class="class“
                scope="page|request|session|application"/>
```

- **Session Beans**
  - Il bean è **persistente**, viene **inserito nell'oggetto di sessione**
  - **Nella stessa sessione**, **qualsiasi altra JSP avrà accesso al bean**
  - Il bean **scade allo scadere della sessione**
- **Application Beans**
  - **Persistenza totale**: il bean **esiste finché esiste** il **JSP Container**
  - E' **disponibile per tutte** le **JSP** e **per ogni utente/sessione**
  - **Esiste solo un'istanza del bean per il server**

Java Server Pages: Cos'è un bean?

- Un **JavaBean** è una **normale classe Java**
- Una classe Java, per essere un bean, deve **seguire** un determinato **insieme di regole** e **convenzioni** di **naming** e di **struttura**
- E' **sempre possibile accedere** agli **elementi** di un **bean** secondo il **normale standard Java**
- Un **bean container** è in **grado** di **interfacciarsi** con i **bean** utilizzando i **meccanismi Java standard** della **introspection** e della **reflection**

Java Server Pages: Bean Constructor

- E' necessario implementare **un costruttore senza argomenti**
- Tale costruttore è quello **utilizzato** dal **JSP container** per **istanziare** il **bean** richiesto con il tag **<jsp:useBean>**
- Ciascuna classe Java ha un costruttore di default senza argomenti
- Esempi:

```
public class DoNothingBean { }

package com.taglib.wdjsp.components;
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);
  }
}
```

Java Server Pages: Bean Properties

- Per creare una **proprietà** è necessario creare una **variabile private** e i **metodi di accesso**
- I metodi di accesso (**access methods**) consentono di accedere (**getter method**) e di modificare (**setter method**) il valore di una proprietà
- I **metodi di accesso sono** metodi **pubblici** che hanno il **nome della proprietà preceduta** dal prefisso **get** o **set**
- Deve essere rispettata la tipizzazione
- Esempi:

```
private String rank;
public void setRank(String rank);
public String getRank();

private int age;
public void setAge(int age);
public int getAge();
```


Java Server Pages: Bean Properties

```
package com.taglib.wdjsp.components;
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;
  }
}
```

```
<jsp:useBean id="time" class="CurrentTimeBean"/>
<html><body>
It is now <jsp:getProperty name="time" property="minutes"/>
minutes past the hour.
</body></html>
```

Java Server Pages: Bean Properties

```
package com.taglib.wdjsp.components;
import java.util.*;
public class DiceBean {
  private Random rand;
  public DiceBean() {
    rand = new Random();
  }

  public int getDieRoll() {
    // return a number between 1 and 6
    return rand.nextInt(6) + 1;
  }

  public int getDiceRoll() {
    // return a number between 2 and 12
    return getDieRoll() + getDieRoll();
  }

}
```


Java Server Pages: Indexed Properties

```
package com.taglib.wdjsp.components;
import java.util.*;
public class StatBean {
  private double[] numbers;
  public StatBean() {
    numbers = new double[2];
    numbers[0] = 1;
    numbers[1] = 2;
  }
  public double getAverage() {
    double sum = 0;
    for (int i=0; i < numbers.length; i++)
      sum += numbers[i];
    return sum/numbers.length;
  }
  public double[] getNumbers() {
    return numbers;
  }
  public double getNumbers(int index) {
    return numbers[index];
  }
  public void setNumbers(double[] numbers) {
    this.numbers = numbers;
  }
  public void setNumbers(int index, double
value) {
    numbers[index] = value;
  }
  public void setNumbersList(String values) {
    Vector n = new Vector();
    StringTokenizer tok =
       new StringTokenizer(values, ",");
    while (tok.hasMoreTokens())
      n.addElement(tok.nextToken());
    numbers = new double[n.size()];
    for (int i=0; i < numbers.length; i++)
      numbers[i] =
        Double.parseDouble((String)
n.elementAt(i));
  }
  public String getNumbersList() {
    String list = new String();
    for (int i=0; i < numbers.length; i++) {
      if (i != numbers.length)
        list += numbers[i] + ",";
      else
        list += "" + numbers[i];
    }
    return list;
  }
  public int getNumbersSize() {
    return numbers.length;
  }
}
```

Java Server Pages: Indexed Properties

```
<jsp:useBean id="stat" class="com.taglib.wdjsp.StatBean">
  <% double[] mynums = {100, 250, 150, 50, 450};
       stat.setNumbers(mynums); %>
</jsp:useBean>

<html>
  <body>
    The average of
<% double[] numbers = stat.getNumbers();
      for (int i=0; i < numbers.length; i++) {
        if (i != numbers.length)
          out.print(numbers[i] + ",");
        else
          out.println("" + numbers[i]);
      } %>
    is equal to <jsp:getProperty name="stat" property="average"/>
  </body>
</html>
```


Java Server Pages: Boolean Properties

private boolean enabled;
public boolean isEnabled();
public void setEnabled(boolean b);

private boolean authorized;
public boolean isAuthorized();
public void setAuthorized(boolean b);

Java Server Pages: Type Conversion

- Ogni volta si accede ad una proprietà del bean col tag <jsp:getProperty> questa viene convertita in una stringa
- Ogni volta che viene settata una proprietà col tag <jsp:setProperty> avviene la conversione inversa

Property Type	Conversion to String
boolean	java.lang.Boolean.toString(boolean)
byte	java.lang.Byte.toString(byte)
char	java.lang.Character.toString(char)
double	java.lang.Double.toString(double)
int	java.lang.Integer.toString(int)
float	java.lang.Float.toString(float)
long	java.lang.Long.toString(long)

Property Type	Conversion from String
boolean or Boolean	java.lang.Boolean.valueOf(String)
byte or Byte	java.lang.Byte.valueOf(String)
char or Character	java.lang.Character.valueOf(String)
double or Double	java.lang.Double.valueOf(String)
int or Integer	java.lang.Integer.valueOf(String)
float or Float	java.lang.Float.valueOf(String)
long or Long	java.lang.Long.valueOf(String)

Confronto fra Servlet e JSP

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GreetingServlet extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException,
                                                                                    IOException
{       response.setContentType("text/html");
        response.setBufferSize(8192);
        PrintWriter out = response.getWriter();
        out.println("<html>" +"<head><title>Hello</title></head>");
       out.println("<body  bgcolor=\"#ffffff\">" +
                    "<img src=\"duke.waving.gif\">" +
                     "<h2>Hello, my name is Duke. What's yours?</h2>" +
                     "<form method=\"get\">" +
                     "<input type=\"text\" name=\"username\" size=\"25\">" +
                     "<p></p>" +
                     "<input type=\"submit\" value=\"Submit\">" +
                     "<input type=\"reset\" value=\"Reset\">" +
                     "</form>");
```

Confronto fra Servlet e JSP

```
String username = request.getParameter("username");
if ( username != null && username.length() > 0 ) {
      RequestDispatcher dispatcher =getServletContext().getRequestDispatcher("/response");
      if (dispatcher != null)
            dispatcher.include(request, response);
 }
out.println("</body></html>");
out.close();
 }

 public String getServletInfo() {
      return "The Hello servlet says hello.";
}

}
```


Confronto fra Servlet e JSP

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ResponseServlet extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException,
                                                                                    IOException
{
PrintWriter out = response.getWriter();
String username = request.getParameter("username");
if ( username != null && username.length() > 0 )
     out.println("<h2>Hello, " + username + "!</h2>");
}

public String getServletInfo() {
      return "The Response servlet says hello.";
}
}
```

Confronto fra Servlet e JSP

```
<html>
<head>
<title>Hello</title>
</head>
<body bgcolor="white">
<img src="duke.waving.gif">
<h2>My name is Duke. What is yours?</h2>

<form method="get"> <input type="text" name="username" size="25">
      <p></p>
      <input type="submit" value="Submit">
      <input type="reset" value="Reset">
</form>
<%     String username = request.getParameter("username");
       if ( username != null && username.length() > 0 )
{ %>
     <%@include file="response.jsp" %>
<%    } %>
</body>
</html>
```


Response.jsp

```
<h2><font color="black">Hello, <%=username%>!</font></h2>
```