Laboratorio di Programmazione di Rete

Lezione del 10 Maggio 2010

Docente: Novella Bartolini
Ricevimento: Mercoledì ore 12:30-14:00
Via Salaria 113, terzo piano, stanza 309
Email: bartolini@di.uniroma1.it

Redirezione di richieste

Configurazione di una servlet

- Il container utilizza un oggetto corrispondente all'interfaccia ServletConfig per passare informazioni alla servlet nel momento della sua creazione
- Si può ottenere tale oggetto per una specifica servlet con il metodo Servlet.getServletConfig()
- **Metodi:**
  - java.lang.String **getInitParameter**(java.lang.String name)
    Fornisce una stringa corrispondente al valore del parametro indicato.
  - java.util.Enumeration **getInitParameterNames**()
    Fornisce un elenco di nomi di parametri di inizializzazione
  - java.lang.String **getServletName**()
    Fornisce il nome dell'istanza corrente della servlet
  - ServletContext **getServletContext**()
    Fornisce il riferimento al contesto operativo in cui viene eseguita la servlet chiamante (segue).

Contesto di una servlet (1)

- Un oggetto rappresentativo del contesto di esecuzione della servlet è contenuto nell'oggetto ServletConfig:
  - implementa Interface ServletContext
- Definisce un insieme di metodi che una servlet usa per comunicare con il proprio container (il motore servlet)
- Esiste un solo contesto per ciascuna web application
- Nel servlet container esistono uno o più contesti servlet e ogni servlet deve essere contenuta in un contesto

Contesto di una servlet (2)

- Il contesto delle servlet è un contenitore di oggetti condivisi e può essere usato per comunicazioni tra servlet di una stessa web application

- Esempio: per trasferire una richiesta da una servlet ad un'altra dello stesso contesto si può usare il metodo di ServletContext:
  getRequestDispatcher(java.lang.String path)
  che fornisce un oggetto RequestDispatcher associato al percorso path

Interface RequestDispatcher (1)

- Si tratta di un oggetto che riceve richieste da un client e le inoltra a qualsiasi risorsa (servlet, pagine HTML o JSP) sul server.
- Un oggetto che implementa l'interfaccia RequestDispatcher viene richiamato attraverso il metodo
  - ServletContext.getRequestDispatcher("<URL>");

Interface RequestDispatcher: forward()

- public void **forward**(ServletRequest request, ServletResponse response)
  - Inoltra una richiesta da una servlet ad un'altra risorsa (servlet o pagina JSP o HTML) sullo stesso server.
  - Se l'oggetto RequestDispatcher è stato ottenuto attraverso una chiamata del metodo getRequestDispatcher(<URL>), il percorso di destinazione dell'oggetto ServletRequest è stato riconfigurato con l'<URL> specificato.
    - Il metodo forward deve essere chiamato prima che l'oggetto risposta venga inviato altrimenti si genera una IllegalStateException.
    - **Parametri:**
      - richiesta – un oggetto ServletRequest: la richiesta ricevuta dal client
      - risposta – un oggetto ServletResponse: la risposta che deve essere inviata al client
  - Questo metodo consente di effettuare un'elaborazione preliminare della richiesta da parte di una risorsa e demandare l'elaborazione definitiva della risposta ad un'altra risorsa.

Interface RequestDispatcher: include()

- public void **include**(ServletRequest request, ServletResponse response)
  - Include il contenuto di una risorsa (servlet o pagina JSP o HTML) nella risposta.
  - L'oggetto ServletResponse è lo stesso utilizzato dalla risorsa chiamante e i percorsi non vengono riconfigurati perché il client riceve la risposta dalla servlet che ha eseguito il metodo include.
    - **Parametri:**
      - richiesta – un oggetto ServletRequest: la richiesta ricevuta dal client
      - risposta – un oggetto ServletResponse: la risposta che deve pervenire al client

Inoltrare una richiesta da una servlet ad un'altra risorsa (servlet, jsp, html)

- L'oggetto RequestDispatcher fornisce un metodo alternativo al metodo ServletResponse.sendRedirect(<URL>)

- 1. Si invoca il metodo **getRequestDispatcher** di **ServletContext**
  - Si fornisce la URL relativa al server o alla applicazione web (NO PERCORSI ASSOLUTI!)

```
String url = "/welcome1";
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher(url);
```

Inoltrare una richiesta... metodo alternativo

- 2. Si invoca il metodo **`forward`** per trasferire il controllo completo alla pagina di destinazione
  - non è prevista nessuna comunicazione con il client, al contrario di quanto avviene con **`response.sendRedirect`**)

- 2 bis. Si invoca il metodo **`include`** per inserire l'output della pagina di destinazione e continuare l'elaborazione

Inoltrare una richiesta: esempio

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  String operation = request.getParameter("destinazione");
  if (operation = = null) {
    operation = "unknown";
  }
  if (operation.equals("destinazione1")) {
    gotoPage("/operations/paginadidestinazione.jsp",
             request, response);
  } else if (operation.equals("destinazione2")) {
    gotoPage("/operations/servletdidestinazione",
             request, response);
  } else {
    gotoPage("/operations/servletdierrore",
             request, response);
  }
}
```

Inoltrare una richiesta: esempio (cont.)

```
private void gotoPage(String address,
                      HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
  RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

Inoltrare una richiesta... (continua)

- Metodo **`forward(req, resp)`** della classe **`RequestDispatcher`**
  - La locazione deve essere all'interno della applicazione web, non può essere un URL esterno
  - La redirezione non coinvolge il client. Avviene in modo trasparente al client
  - Può essere utilizzata sia per richieste di GET che per richieste di POST
  - E' più efficiente del metodo **`sendRedirect()`** e va preferito a quest'ultimo ove possibile

Inoltrare una richiesta... (continua)

- Domanda:
  - Ritenete che l'uso del metodo encodeURL dell'interfaccia HttpServletResponse sia necessario nel caso di ridirezione tramite RequestDispatcher.forward( )?
  - E nel caso del metodo sendRedirect( ) di HttpServletResponse?

Attenzione alle richieste di POST...

- Al contrario delle richieste di GET, non possono essere inoltrate a normali pagine HTML.
- Se avete l'esigenza di inoltrare una richiesta di POST ad una pagina HTML statica rinominatela con estensione **`*.jsp`**
  - nomefile.html non può gestire richieste di POST
  - nomefile.jsp restituisce la stessa risposta sia alla richiesta di GET che alla richiesta di POST

Come fornire dati alla pagina/servlet di destinazione

- **Se** la richiesta deve essere inoltrata a più pagine di destinazione e richiede l'elaborazione di dati contenuti nell'oggetto **`HttpServletRequest`**
- conviene spostare l'elaborazione dei dati nella servlet da cui la richiesta ha origine e passare alla pagina/servlet di destinazione i dati già elaborati

Come fornire dati alla pagina di destinazione

- I dati preventivamente elaborati dalla servlet di origine possono essere inclusi come attributi dell'oggetto richiesta
  - `request.setAttribute("key1", value1);`
- La pagina di destinazione può prelevare questi dati
  - `Type1 value1 = (Type1)request.getAttribute("key1");`
- L'operazione di casting è necessaria (l'attributo è un oggetto (classe Object)

Pagine JSP

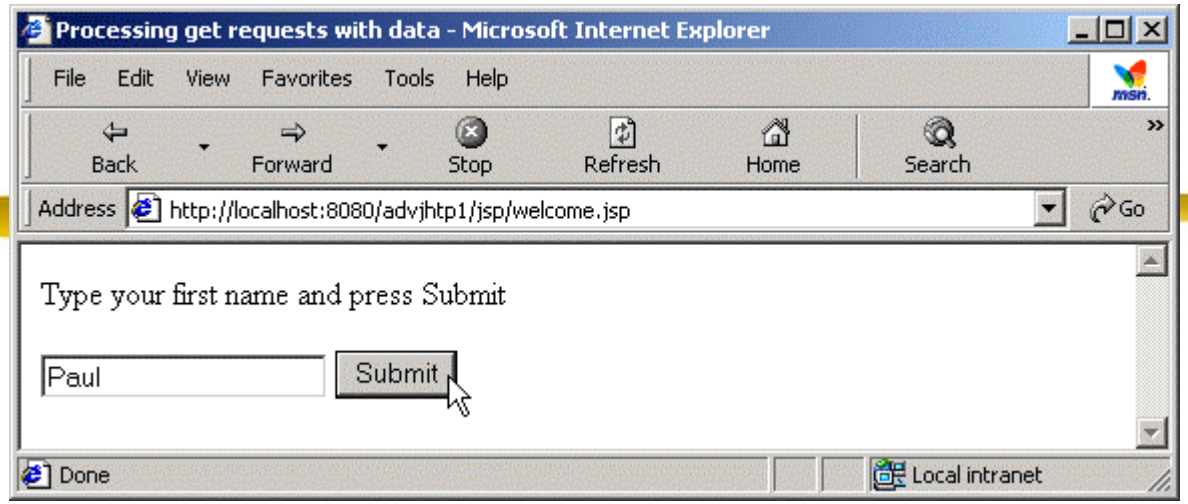
Un esempio, prima di cominciare la trattazione teorica

```
4
5  <!-- welcome.jsp -->
6  <!-- JSP that processes a "get" request containing data. -->
7
8  <html>
9
10    <!-- head section of document -->
11    <head>
12       <title>Processing "get" requests with data</title>
13    </head>
14
15    <!-- body section of document -->
16    <body>
17       <% // begin scriptlet
18
19          String name = request.getParameter( "firstName" );
20
21          if ( name != null ) {
22
23       %> <%-- end scriptlet to insert fixed template data --%>
24
25             <h1>
26                Hello <%= name %>, <br />
27                Welcome to JavaServer Pages!
28             </h1>
29
30       <% // continue scriptlet
31
32          }  // end if
```

Uso di scriptlet per inserire codice java

Uso dell'oggetto implicito **`request`** per ottenere il valore di un parametro

```
33          else {
34
35       %> <%-- end scriptlet to insert fixed template data --%>
36
37             <form action = "welcome.jsp" method = "get">
38                <p>Type your first name and press Submit</p>
39
40                <p><input type = "text" name = "firstName" />
41                   <input type = "submit" value = "Submit" />
42                </p>
43             </form>
44
45       <% // continue scriptlet
46
47          }  // end else
48
49       %> <%-- end scriptlet --%>
50    </body>
51
52 </html>  <!-- end XHTML document -->
```


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/advjhtp1/jsp/welcome.jsp
Go
Type your first name and press Submit
Paul
Submit
Done
Local intranet


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/advjhtp1/jsp/welcome.jsp?firstName=Paul
Go
Hello Paul,
Welcome to JavaServer Pages!
Done
Local intranet

Java Server Pages (JSP)

- Java Server Pages
  - Costituiscono un'estensione della tecnologia delle servlet
- Classi e interfacce specifiche per la definizione di pagine JSP:
  - Package **`javax.servlet.jsp`**
  - Package **`javax.servlet.jsp.tagext`**

Java Server Pages (JSP)

- Il JSP container (al momento della prima invocazione della pagina):
  - Legge la pagina JSP
  - Scrive una servlet (corrispondente alla pagina letta)
  - La compila (default usa javac, ma configurabile)
  - La invoca secondo il ciclo di vita della servlet stessa (inizializzazione, servizio)
- Alle invocazioni successive il container fa riferimento alla servlet già caricata in memoria e inizializzata, pertanto esegue solo il metodo di servizio.

Java Server Pages (JSP)

- Come per le servlet, le pagine JSP utilizzano un oggetto:
  - request (rappresentativo della richiesta HTTP pervenuta)
  - response (rappresentativo della risposta HTTP da inviare)
  - hanno inoltre accesso a tutti i dati della richiesta, del contesto e dell'applicazione web.

Java Server Pages - componenti

- Elementi che costituiscono una pagina JSP:
  - Direttive, cioè istruzioni dirette al servlet/JSP container che specificano come gestire la JSP
  - Azioni
    - Scriptlet (e varianti, come espressioni, dichiarazioni ecc.)
    - Azioni standard
    - Tag personalizzati

Java Server Pages - direttive

- Direttive
  - Istruzioni dirette al JSP container
    - i.e. programma che gestisce le pagine JSP fino alla loro esecuzione
  - Consentono al programmatore di specificare
    - Impostazioni e opzioni della pagina
    - Contenuti esterni da includere o package da importare
    - Librerie di tag personalizzati utilizzabili nella pagina

Java Server Pages - direttive

- Sintassi:
  - <%@ direttiva {attr="valore"}%>
    - Es: <%@ page language="java"%>
      specifica il linguaggio di scripting
      (la spec. JSP 1.1 riconosce solo Java)
    - Es: <%@ include file="relativeURLspec"%>
      specifica il percorso relativo (URL) di un file che deve essere incluso

Java Server Pages - direttiva page

Direttiva	Descrizione
Direttive **page**	
import="importlist"	Fornisce un elenco di package da importare. Utile per non dover scrivere tutto il percorso dei package (nomi di classi pienamente qualificati). Per impostazione predefinita vengono importati: *java.lang.*, javax.servlet.*, javax.servlet.jsp.** e *javax.servlet.http.**
session="true|false"	Se *true* la pagina ha accesso alla variabile implicita *session*, che fa riferimento alla sessione attuale. Per impostazione predefinita è *true*.
isThreadSafe="true|false"	Se *true*, il container può inviare alla pagina nuove richieste prima che vengano completate le richieste in corso. Se questo attributo è su *false* le richieste verranno inviate progressivamente, con possibili conseguenze a livello di prestazioni. L'impostazione predefinita è *true*.
errorPage="error_url"	Se su questa pagina si verifica un'eccezione non catturata, il container passerà alla pagina qui indicata.

Java Server Pages - azioni

- Azioni
  - Sono codificate in un linguaggio di programmazione (JSP 1.1 consente soltanto Java)
  - Specificate sotto forma di
    - scriptlet
      - » codice puro <% sorgente scriptlet %>
      - » Varianti: espressioni <%= espressione %>, dichiarazioni <%! Dichiarazione %>, commento <%-- commento --%>
    - azione standard

      <jsp:actionName attributo="valore">body</jsp:faiqualcosa>
    - tag personalizzati

      <tagPrefix:tagName attributo="valore">body</tag>

JavaServer Pages - scriptlet

- Scriptlet
  - Sono blocchi di codice eseguiti nel contesto della pagina
  - Consentono l'inserimento di codice Java all'interno della pagina JSP
  - Realizzano l'elaborazione della richiesta
    - Interagiscono con gli elementi della pagina e altre componenti per creare pagine dinamiche

JSP: azioni standard e tag personalizzati

- Azioni standard: tag JSP dal comportamento predefinito, comportano l'esecuzione di codice, parametrizzato in base agli attributi del tag.

- Librerie di tag personalizzati
  - Meccanismo di estensione dell'insieme dei tag JSP predefiniti
  - Consente la definizione di nuovi tag da parte del programmatore
    - Nuovi tag possono incapsulare complesse funzionalità

Traduzione della pagina JSP in una servlet

- Il codice contenuto nella pagina JSP
  - Costituirà un blocco di codice all'interno della definizione di un servlet
    - Metodo di inizializzazione _jspInit()
    - Il metodo **`_jspService()`** conterrà il codice degli scriptlet e una sequenza di istruzioni di write("…") per tutti i tag HTML presenti nella pagina .jsp
    - Metodo di distruzione _jspDestroy()

- Ciclo di vita della JSP simile a quello di una servlet.
  - La prima volta che viene invocata la pagina, il container la traduce in una servlet, che viene compilata e mandata in esecuzione (esecuzione del metodo _jspInit).
  - Il container invoca il metodo _jspService ad ogni richiesta successiva della stessa pagina JSP.

Errori JSP

- Errori al momento della traduzione
  - Si verificano nel momento in cui viene generata la servlet corrispondente alla JSP
- Errori al momento della richiesta
  - Si verificano durante l'elaborazione della richiesta

JSP o Servlet?

- JSP
  - Hanno l'aspetto e la struttura di pagine XHTML
    - Contengono markup HTML o XHTML
  - Vengono utilizzate quando la maggior parte del contenuto che deve essere visualizzato segue una struttura fissata.
    - In generale una piccola parte del contenuto deve essere generata dinamicamente
- Servlet
  - Utilizzate invece quando solo una piccola porzione del contenuto deve seguire una struttura fissata
    - La maggior parte del contenuto deve essere generata dinamicamente

Esempio di pagina JSP

Esempio in cui

- Viene creata la struttura della pagina attraverso markup XHTML
- Viene creato un oggetto java (**java.util.Date**)
- Viene effettuata la conversione automatica di un'espressione JSP in un'oggetto **String**
- Viene usato un **meta**-elemento per ricaricare la pagina a specifici intervalli di tempo

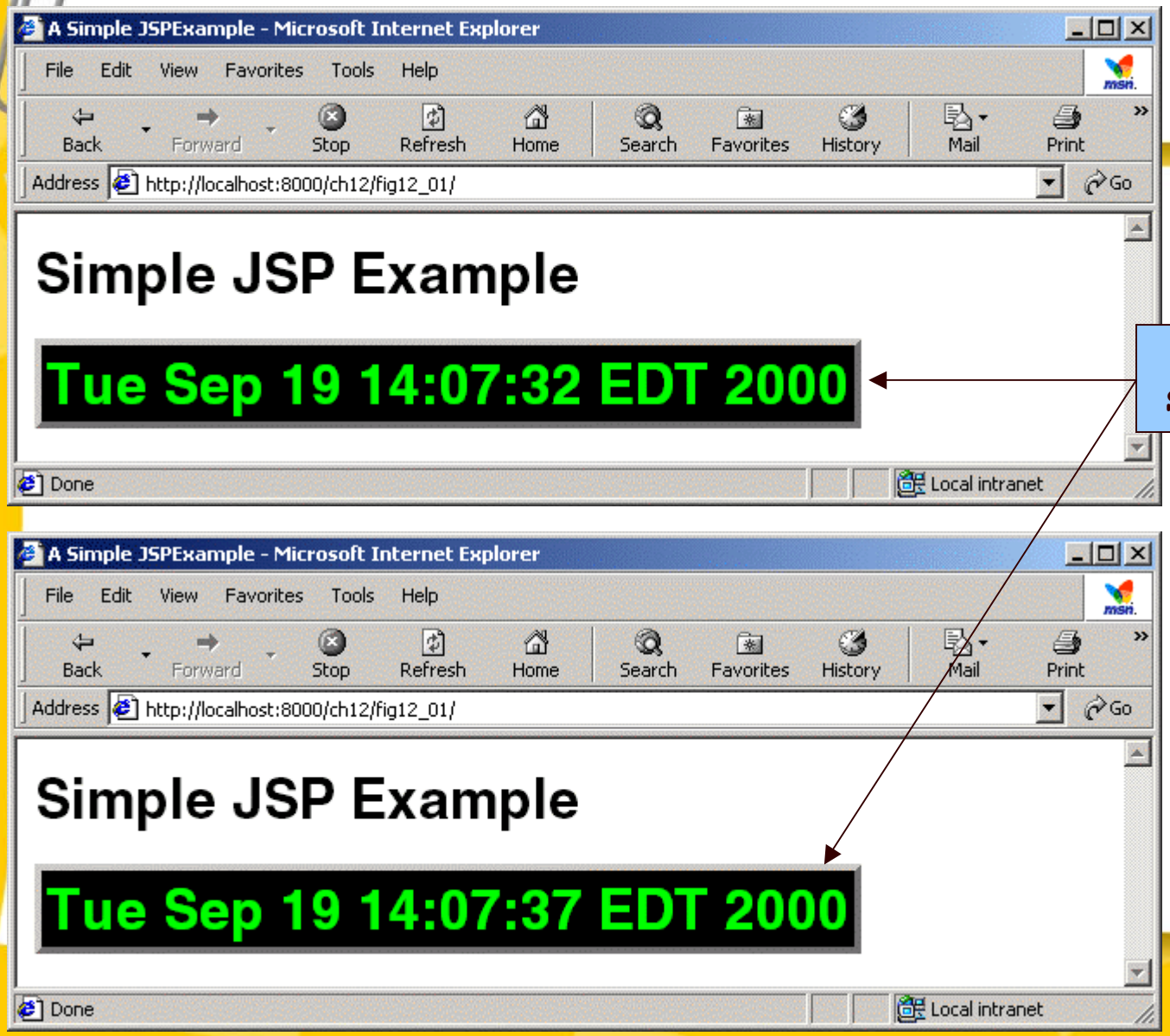
- Si noti alla prima invocazione di **clock.jsp** il ritardo con cui
  - Il JSP container traduce la pagina JSP in una servlet
  - Il JSP container compila la servlet
  - Il JSP container esegue la servlet
- Le successive richieste della pagina JSP non sperimentano questo ritardo.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5
6  <html xmlns = "http://www.w3.org/1999/xhtml">
7
8
9    <head>
10     <meta http-equiv = "refresh" content = "60" />
11
12     <title>A Simple JSP Example</title>
13
14     <style type = "text/css">
15       .big { font-family: helvetica, arial, sans-serif;
16            font-weight: bold;
17            font-size: 2em; }
18     </style>
19   </head>
20
21   <body>
22     <p class = "big">Simple JSP Example</p>
23
24     <table style = "border: 6px outset;">
25       <tr>
26         <td style = "background-color: black;">
27           <p class = "big" style = "color: cyan;">
28
29             <!-- JSP expression to insert date/time -->
30             <%= new java.util.Date() %>
31
32           </p>
33         </td>
34       </tr>
35     </table>
```

`meta` element refresh: ricarica la pagina ogni **`60`** seconds

Crea un oggetto **`Date`** che viene implicitamente convertito in un oggetto **`String:`** si tratta di un'espressione

```
36    </body>
37
38  </html>
```



Rappresentazione tramite **`String`** dell'oggetto **`Date`**

Aree di visibilità (scope) in una pagina JSP

- Le pagine JSP possono accedere ad oggetti definiti in diverse aree di visibilità (scope):
  - Applicazione
    - Oggetti associati al **contesto servlet** della JSP;
    - Per recuperare tali oggetti si ricorre al metodo javax.servlet.ServletContext.**getAttribute**
  - Pagina
    - Oggetti che sono visibili solo al codice presente sulla stessa pagina
    - Per accedervi si usa javax.servlet.jsp.PageContext.**getAttribute**
    - Una volta completata la richiesta della pagina il container elimina il riferimento a tali oggetti

Aree di visibilità (continua)

- Richiesta
  - Visibilità uguale alla richiesta
  - Vi si accede con il metodo javax.servlet.ServletRequest.**getAttribute**
  - Viene eliminato il riferimento a tali oggetti quando viene inviata la risposta
- Sessione
  - Oggetti associati ad una sessione utente
  - Vi si accede con il metodo javax.servlet.http.**HTTPSession.getAttribute**
  - I riferimenti a tali oggetti vengono eliminati quando la sessione termina (per volere del client o per timeout)

Oggetti impliciti

Implicit Object	Description
Application Scope	
application	This **javax.servlet.ServletContext** object represents the container in which the JSP executes.
Page Scope	
config	This **javax.servlet.ServletConfig** object represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor.
exception	This **java.lang.Throwable** object represents the exception that is passed to the JSP error page. This object is available only in a JSP error page.
out	This **javax.servlet.jsp.JspWriter** object writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
page	This **java.lang.Object** object represents the **this** reference for the current JSP instance.
pageContext	This **javax.servlet.jsp.PageContext** object hides the implementation details of the underlying servlet and JSP container and provides JSP programmers with access to the implicit objects discussed in this table.

Oggetti impliciti (cont.)

Implicit Object	Description
`response`	This object represents the response to the client. The object normally is an instance of a class that implements **`HttpServletResponse`** (package **`javax.servlet.http`**). If a protocol other than HTTP is used, this object is an instance of a class that implements **`javax.servlet.ServletResponse`** .
Request Scope	
`request`	This object represents the client request. The object normally is an instance of a class that implements **`HttpServletRequest`** (package **`javax.servlet.http`**). If a protocol other than HTTP is used, this object is an instance of a subclass of **`javax.servlet.ServletRequest`** .
Session Scope	
`session`	This **`javax.servlet.http.HttpSession`** object represents the client session information if such a session has been created. This object is available only in pages that participate in a session.

Componenti di scripting JSP

- Come specificare componenti JSP
  - Scriptlets (delimitate da **<%** and **%>**)
  - Commenti (delimitati da **<%--** and **--%>**)
  - Espressioni (delimitati da **<%=** and **%>**)
  - Dichiarazioni (delimitati da **<%!** and **%>**)

Scriptlet

- Delimitate da <% e %>
- Blocchi di codice Java
- Inserite nel metodo **_jspService** al momento della traduzione

- Scriptlet, espressioni e codice XHTML possono essere intercalati per creare diverse risposte sulla base di informazioni incluse nella richiesta

Componenti di scripting JSP (seq. di escape)

Literal	Escape sequence	Description
`<%`	`<\%`	The character sequence `<%` normally indicates the beginning of a scriptlet. The `<\%` escape sequence places the literal characters `<%` in the response to the client.
`%>`	`%\>`	The character sequence `%>` normally indicates the end of a scriptlet. The `%\>` escape sequence places the literal characters `%>` in the response to the client.

`'`
`"`
`\`
	`\'`
	`\"`
	`\\`
		As with string literals in a Java program, the escape sequences for characters `'`, `"` and `\` allow these characters to appear in attribute values. Remember that the literal text in a JSP becomes string literals in the servlet that represents the translated JSP .

Commenti

- Sono supportati tre tipi di commento:
  - Commento JSP
  - Commento XHTML
  - Commento del linguaggio di scripting

Commento JSP

- Commento JSP
  - <%-- --%>
    - Non si usa all'interno di scriptlet
    - Non è visibile al client

Commento XHTML

- Commento XHTML
  - <!-- -->
    - Non si usa all'interno di scriptlet
    - E' visibile al client

Commento del linguaggio di scripting

- Commento del linguaggio di scripting
  - Single-line //, oppure Multi-line /* */
    - Si usa esclusivamente all'interno di scriptlet
    - E' visibile al client?

Espressioni

- Sono delimitate da <%= e %>
- Contengono espressioni Java che vengono valutate quando il client richiede la pagina che le contiene
- Il container converte il risultato di un'espressione in un oggetto **String** e lo invia in output come parte della risposta