Mauro Bogliaccino

Corso Java

Programma del corso Java SE - Java SE 11 Programmer II

Java SE 11 Programmer II

Elementi fondamentali JSP

JSP utilizza gli oggetti impliciti (built-in)

- Gli oggetti impliciti sono oggetti istanziati automaticamente dall'ambiente JSP, non dobbiamo preoccuparci di importarli e istanziarli.
- Per utilizzarli è sufficiente usare la sintassi nomeOggetto.nomeMetodo

questi oggetti sono disponibili per l’uso in pagine JSP e sono

- `out` : per scrivere codice HTML nella risposta (System.out di Java)
- `session` : dati specifici della sessione utente corrente
- `request` : richiesta HTTP ricevuta e i suoi attributi, header, cookie, parametri, etc.
- `page` : la pagina e le sue proprietà.
- `config` : dati di configurazione
- `response` : risposta HTTP e le sue proprietà.
- `application` : dati condivisi da tutte le pagine della web application
- `exception` : eventuali eccezioni lanciate dal server; utile per pagine di errore
- `pageContext` : dati di contesto per l’esecuzione della pagina

Gli oggetti impliciti possono essere

- 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 errori

Ambito

Definisce dove e per quanto tempo saranno accessibili gli oggetti: oggetti impliciti, JavaBeans, ...

- `page` di **pagina**: l'oggetto è accessibile dal servlet che rappresenta la pagina
- `request` di **richiesta**: l'oggetto viene creato e poi distrutto dopo l'uso
- `session` di **sessione**: l'oggetto è accessibile durante tutta la sessione
- `application` di **applicazione**: l'oggetto è accessibile dal servlet che rappresenta la pagina

Ciclo di vita di una pagina JSP

La pagina viene salvata in una cartella pubblica del server web alla prima richiesta ricevuta dal Web server la pagina JSP è automaticamente:

- tradotta in un sorgente Java chiamato **Servlet**
- compilata come programma Java
- caricata in memoria ed eseguita

successivamente la pagina JSP (la servlet) viene solo eseguita. In fase di debug il web server verifica se la pagina JSP è più recente del servlet corrispondente.

Rispetto ad altre tecnologie server side come PHP o ASP, questa è una differenza vantaggiosa in termini di velocità di risposta: dopo la prima esecuzione, il codice risulterà già compilato e disponibile immediatamente.

Con PHP e ASP il webserver interpreterà il codice ad ogni richiesta prima di servire la pagina di risposta.

- direttive al server
- `<%@ direttive %>`
- elementi di scripting
- `<%! dichiarazioni %>`
- `<%= espressioni %>`
- `<% scriptlet %>`
- `<%-- commenti --%>`
- azioni standard
  - `<jsp:include>`
  - `<jsp:forward>`
  - `<jsp:param>`
  - `<jsp:useBean>`

JSP: Azioni standard

jsp:include

Chiama una pagina JSP da un'altra

Al termine, la pagina di destinazione restituisce il controllo alla pagina chiamante.

```
<jsp:include page="path" flush="true"/>
<jsp:include page="path" flush="true">
<jsp:param name="paramName" value="paramValue" /> ..
</jsp:include>
```

jsp:forward

Chiama una pagina JSP da un'altra

L'esecuzione della pagina chiamante viene terminata dalla chiamata

```
<jsp:forward page="path" />
<jsp:forward page="path">
<jsp:param name="paramName" value="paramValue" /> ..
</jsp:forward>
```

jsp:forward

L'azione forward serve per trasferire l'utente da una pagina jsp ad un'altra. Come l'azione jsp:include è possibile utlizzare forward con o senza parametri. In quest'ultimo caso occorre terminare l'istruzione inserendo / prima della parentesi angolare di chiusura tag :

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

Nel caso si vogliano utilizare dei parametri, invece, utilizzeremo il tag di chiusura del blocco di codice </jsp:forward> :

```
<jsp:forward page="URL" >
<jsp:param name="ParamName1" value="ParamValue1" />
<jsp:param name="ParamName2" value="ParamValue2" />
</jsp:forward>
```

Esempio di codice (incluso in tomcat 7)

per trovarne altri, dopo l'indirizzo del server (es. localhost:8080) scrivi /examples/jsp/

```
<html>
<%
   double freeMem = Runtime.getRuntime().freeMemory();
   double totlMem = Runtime.getRuntime().totalMemory();
   double percent = freeMem/totlMem;
   if (percent < 0.5) {
%>

<jsp:forward page="one.jsp"/>

<% } else { %>

<jsp:forward page="two.html"/>

<% } %>

</html>
```

- `type="bean|applet"`

opzionali

- `code="objectCode"`
- `codebase="objectCodebase"`
- `align="alignment"`
- `archive="archiveList"`
- `height="height"`
- `hspace="hspace"`
- `jreversion="jreversion"`
- `name="componentName"`
- `vspace="vspace"`
- `width="width"`
- `nspluginurl="url"`

sintassi alternativa

- `<jsp:params><jsp:param name="paramName"value="paramValue" /></jsp:params>`
- `<jsp:fallback>testo in caso di fallimento del plugin</jsp:fallback> >`

Utilizzo di JavaBeans

jsp:useBean

Definisce un'istanza di Java bean.
<jsp:useBean id="name" scope="page|request|session|application" typeSpec />
<jsp:useBean id="name" scope="page|request|session|application"typeSpec >body</jsp:useBean>

dove `typespec` è una delle seguenti possibilità:

- `class="className"`
- `class="className" type="typeName"`
- `beanName="beanName" type=" typeName"`
- `type="typeName"`

jsp:setProperty

Imposta il valore di una o più proprietà in un bean.

<jsp:setPropertyname="beanName" prop_expr />

dove `prop_expr` è una delle seguenti possibilità:

- `property="*"`
- `property="propertyName"`
- `property="propertyName" param="parameterName"`
- `property="propertyName" value="propertyValue"`

Scrive il valore di una proprietà bean come stringa nell'oggetto out.
<jsp:getProperty name="name" property="propertyName" />

Le dichiarazioni

- JSP: usare le dichiarazioni `<%! ... %>`
- le espressioni `<%= ... %>`
- Racchiuse tra i tag `<%! ... %>`, contengono codice Java e sono utilizzate per definire metodi e proprietà della classe generata.
- Le dichiarazioni sono esterne al servlet, a differenza degli scriptlet.
- Queste variabili o metodi dichiarati diventeranno variabili di istanza della classe servlet generata.
- Questo significa che saranno globali a tutto il servlet generato per la pagina.
- Può servire per definire dei metodi da usarsi con il servlet, dichiarare attibuti

Sintassi

- `<%! Dichiarazione %>`

esempio

```
<%@page contentType="text/html" pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>JSP Page</title>
</head>
<body>
<h1>Le dichiarazioni: crea una funzione per stampare 5 numeri random</h1>
<h2>NB: non ci sono i controlli per ordinare i numeri e verificare l'assenza di duplicati, prova a completare il codice usando gli array</h2>
<%!
 private String lottoGame() {
 return ("<h2>" + (int)Math.floor(Math.random() * 90 +1 )+ "</h2>");
 }
 %>
<%= lottoGame() %>
<%= lottoGame() %>
<%= lottoGame() %>
<%= lottoGame() %>
<%= lottoGame() %>
</body>
</html>
```

JSP - Le direttive

Le direttive permettono di definire la struttura di tutto il documento JSP.
Indicano gli aspetti principali del servlet in cui sarà convertito il file JSP.
Sono processati al momento della conversione in servlet: a compile-time, cioè a tempo di compilazione.

Le direttive esistenti sono: page, include e taglib.

- `<%@ page ... %>`
- `<%@ include file="path - percorso relativo al file" %>`
- `<%@ taglib ...%>`

Introdotte dal simbolo `@` possono contenere diversi attributi, in alcuni casi concatenabili come per la direttiva import.

direttiva page

```
<%@ page attributi %>
```

dove attributi sono coppie: nome="valore"

Esempio:

```
<%@ page language=”Java” import="java.sql.*,java.util.*" session="true"
buffer="10kb" %>
```

Esiste una lista di attributi che possono essere usati, ne elenco alcuni:

```
import ="package.class"
```

Lista di package o classes, separati da virgola, che saranno importati per essere utilizzati nel codice java.

```
session ="true | false"
```

specifica se la pagina fa parte di una sessione HTTP. Se si inizializza a true, è disponibile l'oggetto implicito sessione.

```
buffer ="dimensionekb"
```

specifica la dimensione di un buffer di uscita di tipo stream, per il client.

`errorPage="url"`

specifica una pagina JSP che sarà processata nel caso si verifichi un errore.

`isErrorPage="true|false"`

Indica che la pagina è una pagina di errore JSP e può mostrare a video l'output dell'errore verificatosi. Per default è settata false.

`contentType = "MIME-Type"`

oppure

`contentType = "MIME-Type; charset = Character-Set"`

Il valore MIME di default è text/html

direttiva include

Indica al motore JSP di includere il contenuto del file corrispondente, inserendolo al posto della direttiva nella pagina JSP.
Il contenuto del file incluso è analizzato al momento della traduzione del file JSP e si include una copia del file stesso nel servlet generato.
Una volta incluso, se si modifica il file non sarà ricompilato nel servlet.

Il tipo di file da includere può essere un

- file html (statico)
- file jsp (dinamico)
- Sintassi: `<%@ include file="nome del file" %>`

esempio

```
<html>

<head>

<title> pagina di prova Direttive  </title>

</head>

<body>

<h1> pagina di prova Direttive inclusione  </h1>

      <%@ include file="/hello_world.html" %>

      <%@ include file=”/login.jsp” %>

</body>

</html>
```

direttiva taglib

Permette estendere i marcatori di JSP con etichette o marcatori generati dall'utente (etichette personalizzate).

Sintassi

```
<%@ taglib uri="taglibraryURI" prefix="tagPrefix" %>
```

Esempi librerie standard

- JSTL core
- JSTL sql
- JSTL function

Esempio JSTL Core

JSTL e EL

JSP Standard Tag Library e Expression Language

```
<%--
    Document   : core
    Created on : 26-giu-2013, 15.16.15
    Author     : mauro
--%>
<!-- collega la taglib e assegna il prefisso, in questo caso usa la lettera c -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Esempi core</title>
</head>
<body>
    <h1>Esempio con tag: out</h1>
<!-- la lettera del prefisso permette di accedere agli oggetti ed ai metodi definiti nella taglib vedi gli esempi -->
<c:out value="${'<body> , %'}"/>

    <h1>Esempio con tag: set</h1>
<c:set var="prodotto" scope="session" value="${2000*2}"/>
<c:out value="${prodotto}"/>
    <h1>Esempio con tag: remove</h1>
<c:remove var="prodotto"/>
<c:out value="${prodotto}"/>

    <h1>Esempio con tag: catch e if</h1>
<c:catch var ="catchException">
   <% int x = 10/0;%>
</c:catch>

<c:if test = "${catchException != null}">
   <p>The exception is : ${catchException} <br />
   There is an exception: ${catchException.message}</p>
</c:if>
    <h1>Esempio con tag: choose</h1>

    <c:set var="prodotto" scope="session" value="${5000+500+50+5}"/>

<c:choose>
    <c:when test="${prodotto <= 0}">
       Il valore del prodotto è minore di 0.
    </c:when>
    <c:when test="${prodotto > 1000}">
       Il valore del prodotto è maggiore di 1000.
    </c:when>
    <c:otherwise>
        <c:out value="${prodotto}"/>
    </c:otherwise>
</c:choose>

<h2>Esempio con import: importa il sorgente di una pagina web con JSTL</h2>
<%--<c:import var="data" url="http://www.bogliaccino.it"/>--%>
<h2>stampa il codice importato</h2>
<%--<c:out value="${data}"/>--%>

</body>
</html>
```

Esempio JSTL Functions

```
<%--
    Document   : newjspfucntions
    Created on : 26-giu-2013, 16.27.22
    Author     : mauro
--%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
<h2>la funzione replace fn:replace()</h2>
<c:set var="stringa1" value="Questa è la prima stringa."/>
<c:out value="${stringa1}"/>
<c:set var="stringa2" value="${fn:replace(stringa1,
                            'prima', 'seconda')}" />

<p>Stringa modificata: ${stringa2}</p>
<h2>la funzione split e la funzione join fn:split() fn:join()</h2>
<c:set var="stringa1" value="Questa è la prima stringa."/>
<c:set var="stringa2" value="${fn:split(stringa1, ' ')}" />
<c:set var="stringa3" value="${fn:join(stringa2, '|')}" />

<p>Stringa (3) : ${stringa3}</p>

<c:set var="stringa4" value="${fn:split(stringa3, '|')}" />
<c:set var="stringa5" value="${fn:join(stringa4, ' ')}" />

<p>Stringa (5) : ${stringa5}</p>
    </body>
</html>
```

Ciclo DO WHILE

- Ciclo DO WHILE con JSP

  (il ciclo viene eseguito almeno una volta)

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Corso JSP</title>
</head>
<body>
<h1>Uso del ciclo DO WHILE</h1>

<h2>Uso dell'oggetto implicito OUT per stampare il valore a video</h2>
<table >
<tr>
<td></td>
<td>numero</td>
<td>aggiungo +1 al numero base </td>
<td>moltiplico *2 il numero base </td>
</tr>
<%
  int i=0;
  do{
  out.print("<tr><td>numero</td><td>"+i+"</td>"+"<td>"+(i+1)+"</td>"+"<td>"+(i*2)+"</td></tr>");
  i++;
  }
  while(i<=10);
  %>
```

Funzione contatore accessi + esempio uso dichiarazioni e espressioni

- Funzione contatore accessi + esempio uso dichiarazioni e espressioni

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>

<%! int conta = 0; %>
<%! public void contatore()
 {
 conta ++;
 }
 %>
Questa pagina è stata caricata <%= conta %> volte.
<% contatore(); %>
</body>
</html>
```

- Assegna un nome al progetto e salvalo sul disco
- Scegli un server web
- Scegli il framework Java Server Faces
- Clicca su 'Fine'

In Source Packages aggiungi un nuovo package (p.es login), all'interno del package aggiungi una classe java (p. es accesso.java)

Incolla il codice seguente

```
package login;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

/**
*
* @author mauro
*/

@ManagedBean
@RequestScoped
public class accesso {
public String login;

public String getLogin() {
return login;
```

Il file index.xhtml contiene il form per l'invio del nome utente, incolla il codice all'interno dei tag h:body

```
<h:form>

<h2>login</h2>
Login: <h:inputText id="login" size="30" required="true" requiredMessage="inserisci nome utente" value="#{accesso.login}" />

<h:/form>
Infine crea il file login.xhtml che visualizza il testo inviato. Attenzione! Anche questo codice dovrà compreso all'interno dei tag h:form, così come il campo input per l'invio del testo.

<h:form>

Benvenuto: <h:outputText value="#{accesso.login}" />
           <h:commandLink value="torna alla home" action="index"/>


<h:/form>---
![bg](./background.jpg)
```

JSP Java Server Pages

Primo approccio a JSP

- Cos’è e a cosa serve JSP
- Hello World con JSP
- Primo esempio stampare la data corrente a video
- Uso degli oggetti impliciti ed esempi

- JSP è una specifica di Sun Microsystems, poi acquistata da Oracle
- La tecnologia viene presentata verso la fine degli anni '90 formendo il supporto al server-side in ambito Java
- Serve per creare e gestire pagine web dinamiche
- Permette di mescolare in una pagina codice HTML per generare la parte statica, con contenuto dinamico generato a partire da marcatori speciali <% .... %>
- Il contenuto dinamico si ottiene grazie alla possibilità di incorporare nella pagina codice Java di differenti forme
- L'obiettivo finale è separare l'interfaccia (presentazione visuale) dalla implementazione (logica di esecuzione)

pagina JSP -> Servlet

- La pagina JSP si converte in un servlet
- La conversione la realizza il motore o contenitore JSP (tomcat, glassfish, ...), la prima volta che si sollecita la pagina JSP
- Questo servlet generato processa qualsiasi richiesta per questa pagina JSP
- Se si modifica il codice della pagina JSP, allora si rigenera e ricompila automaticamente il servlet e si ricarica la volta successiva

Installazione ed esecuzione della prima pagina JSP

- Installazione dell' SDK di Java e di Tomcat configurazione dell’IDE e prima pagina Hello world con JSP
- Esplorazione dell’ambiente di sviluppo, creazione, modifica pagine JSP, passaggio dei dati tra pagine
- Oggetti impliciti, oggetti principali, uso di Request, Out, etc.
- Inclusione di file, primo utilizzo delle direttive.
- Tipi di tag JSP: dichiarazioni, espressioni, scriptlet, direttive.
- Compile-time e Request-time

Elementi fondamentali di JSP

Sintassi e caratteristiche degli oggetti principali: scriptlet, azioni standard, servlet

- Direttive `<%@ (page | include | taglib)`
- Codice Java `<% (dichiarazioni | java | espressioni)`
- Azioni standard `jsp:action (include | forward | usebean | getproperty | setproperty)`

Utilizzo degli elementi fondamentali

- Ciclo di vita di una pagina Jsp, Scriptlet: usare i costrutti del linguaggio JAVA nelle JSP
- Cookie e gestione delle sessioni – Impostare una procedura di login da riutilizzare
- Teoria ed esercizi sulle standard actions - jsp:forward
- Standard action JavaBeans: estendere le funzionalità della web application, separare la logica dell'app;
- Standard action: come usare le altre azioni standard
- Lavorare con le date in JSP.

Elementi fondamentali JSP## Costrutti del linguaggio

Supporta i costrutti e la sintassi standard di Java.

Commenti

Un commento JSP **non** viene inviato al client come parte dell'output della pagina JSP

```
<%-- Comment string..--%>
```

I commenti HTML vengono inviati al client

```
<!-- comments -->
```

direttiva `page`

Definisce gli attributi a livello di pagina.

```
<%@ page attribute="value" ..%>
```

valori di default degli attributi:

- attribute = language="java"
- session="true"
- contentType=text/html;charset="ISO-8859-1"
- import="package(s)"
- buffer="8kb"
- autoflush="true"
- isThreadSafe="true"
- info="text_string"
- errorPage="relativeURL"
- isErrorpage="true"
- extends="class_name"value = a string literal in single or double quotes

include

Inserisce testo in una pagina JSP

```
<%@ include file = "path" ..%>
```

taglib

Definisce una custom tag library per una pagina JSP.

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

Dopo la direttiva taglib, fai riferimento ai tag personalizzati usando la sintassi:

```
<tagPrefix:tagName>...</tagPrefix:tagName>
```

Le Tag Libraries

- JSTL
- core
- xml
- frm
- sql
- functions
- Librerie di Tag personalizzate
- EL - expressions language - espressioni dinamiche

esempio:

<%! private String foo = null;public String getFoo() {return this.foo;} %>

scriptlet

Contiene un blocco di codice di scripting

Una pagina JSP può contenere più blocchi di codice di scripting.

```
<% script code %>
```

esempio:

```
<% String greeting =request.getParameter("Greeting");out.println(greeting);
%>
```

expression

Definisce le dichiarazioni valutate sul server prima di inviare l'output della pagina al client.

response

La risposta al client.

Java type: `javax.servlet.HttpServletResponse`

session

L'oggetto sessione creato per il client richiedente.

Java type: `javax.servlet.http.HttpSession`

RequestDispatcher

- L'approccio servlet-centric prevede di utilizzare le pagine JSP solo per la presentazione e delegare il controllo ad una o ad un gruppo di servlet
- Le servlet quindi
- gestiscono le richieste (vengono cioè invocate tramite URL) elaborano i dati necessari a soddisfare le richieste (interagendo con la classe DBMS.java e utilizzando i Java Data Bean come componenti per rappresentare le informazioni di interesse)
- trasferiscono il controllo alla JSP designata a presentare i risultati.
- Se il gruppo di servlet che realizzano l'applicazione web viene ristretto a contenere una sola servlet, allora l'applicazione ha un solo punto di accesso e questa servlet ha i controllo totale sul flusso dell'applicazione.

Passaggio dati fra servlet-JSP:

- I Java Data Bean istanziati dalla servlet devono essere passati alla JSP prima di trasferire ad essa il controllo.
- A tal fine esiste una coppia di metodi della classe HttpServletRequest che permettono di inserire/recuperare in/da request (oggetto implicito della JSP) un numero arbitrario di oggetti.

Questi metodi sono:

- setAttribute(String, Object)
- getAttribute(String)

Trasferimento del controllo dalla servlet alla JSP

- Quando all'interno di una servlet, dopo aver preparato i dati in JDB e averli inseriti nell'oggetto request (parametro del metodo doGet o doPost), si vuole richiamare una JSP per visualizzare i dati, si dice che si trasferisce il controllo (forward) alla JSP.
- Per trasferire il controllo è necessario creare un oggetto di tipo RequestDispatcher associato alla JSP che si vuole 'invocare'.
- Ci sono due modi equivalenti per definire un oggetto RequestDispatcher associato ad una JSP all'interno di una servlet:

```
RequestDispatcher rd =
request.getRequestDispatcher(“PathRelativoJSP")
RequestDispatcher rd =
getServletContext.getRequestDispatcher(“PathAssolutoJSP")
```

- Una volta ottenuto l'oggetto RequestDispatcher rd, è sufficiente invocare il suo metodo forward:

`forward(HttpServletRequest, HttpServletResponse)`

per trasferire MOMENTANEAMENTE il controllo alla JSP.

`rd.forward(request, response)`

Non è un browser redirect e nemmeno una terminazione del metodo doGet o doPost della servlet: è una semplice chiamata di metodo.
Perciò tutto il codice eventualmente presente dopo rd.forward(request, response) verrà eseguito dopo che la JSP ha finito la sua esecuzione