

Programmazione WEB server-side JSP

# TECNICO SUPERIORE PER LO SVILUPPO SOFTWARE 2016 - JSP

# Introduzione al linguaggio JSP e configurazione dell'ambiente di sviluppo

# Sommario block1

- 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
- Ciclo di vita di una pagina Jsp, tipi di tag JSP: dichiarazioni, espressioni, scriptlet, direttive
- Scriptlet: usare i costrutti del linguaggio JAVA nelle JSP
- **Esercitazione di verifica delle competenze acquisite**



# Esplorazione dell'ambiente di sviluppo, creazione, modifica pagine JSP passaggio dei dati tra pagine

- Crea la tua directory CorsoJSP
- Esplora CorsoJSP
- Crea la pagina HelloWorld.jsp
- Stampa la data corrente (vd esercizi esempio2.jsp, esempio3.jsp)
- Vedi gli altri esempi (esempio1-6)
- Crea una pagina con form html per l'invio dei dati
- Ricevi e pubblica i dati ricevuti
- Gestione degli errori (introduzione)



# *Esplora CorsoJSP*

All'interno della directory *webapps* saranno presenti già alcune Web Application predefinite ed in più una cartella di nome *ROOT*. Questa è la cartella che contiene la radice del sito web gestito da Tomcat, la pagina *index.jsp* che troviamo al suo interno è la risorsa che abbiamo aperto durante il test del web server.

Ora creiamo la nostra prima Web Application. Secondo le specifiche J2EE, una WA deve avere una determinata struttura a directory. Creiamo quindi una nuova cartella di nome *CorsoJSP* (sarà il nome della nostra WA) all'interno di *webapps* e diamole la seguente struttura:

```
\CorsoJSP\  
\CorsoJSP\WEB-INF\  
\CorsoJSP\WEB-INF\web.xml  
\CorsoJSP\WEB-INF\classes\  
\CorsoJSP\WEB-INF\lib\
```



# CorsoJSP

*CorsoJSP* diventerà la **radice** della nostra **applicazione**, tutto ciò che sarà inserito al suo interno sarà reso **pubblico** e quindi disponibile a tutti gli utenti. La risorsa che Tomcat aprirà di default sarà un file di nome *index.htm* o ***index.jsp***.

E' opportuno organizzare l'applicazione in **più sottocartelle** (cercando di associare ogni sottolivello ad una sottostruttura **logica** della nostra applicazione), ogni sottocartella dovrà contenere **un file *index.jsp* o *.htm*** così da permettere una **navigazione** lineare per livelli.



# CorsoJSP

Differente invece sarà la funzione della cartella **WEB-INF** che non sarà esposta dall'Application Server e quindi diventa il luogo ideale per riporre le **classi** (all'interno di *classes*), gli **archivi jar** (all'interno di *lib*) o le **risorse** che dovranno essere protette dagli utenti esterni.

Importante è la funzione del ***Deployment Descriptor*** ovvero il file **web.xml** (un file xml editabile con qualsiasi editor testuale), che conterrà la descrizione delle informazioni e delle risorse aggiuntive necessarie al funzionamento della Web Application.



# esempio1

```
<html>
  <body>
<h1>Hello world in JSP</h1>
<h2><a href="index.jsp">home</a></h2>
<br/>  <!-- This is the JSP file-->
  <%
    out.println ("Hello World");
  %>
</body>
</html>
```





# esempio2

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
  <h1>Data in JSP</h1>
  <%!
Date theDate = new Date();
Date getDate()
{
  System.out.println( "In getDate() method" );
  return theDate;
}
%>
Hello! The time is now <%= getDate() %>
</BODY>
</HTML>
```

forma deprecata

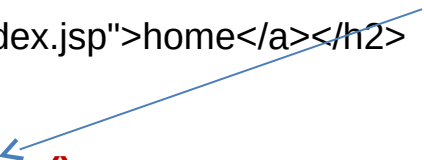


# esempio3

```
<%@ page language="java" import="java.lang.*,java.util.*" %>
<!DOCTYPE html>
<head>
<title>Prima pagina JSP</title>
<meta http-equiv="description" content="Questa è la mia prima pagina JSP"/>
</head>
<body>
<h1>Prima pagina JSP</h1>
<h2>con classe calendar <a href="index.jsp">home</a></h2>
<br/>
<%
Calendar data = Calendar.getInstance();
out.print("Oggi è il giorno " + data.get(data.DATE)+ "/" + (data.get(data.MONTH)
+1)+ "/" + data.get(data.YEAR));
%>
<hr/>
Ho utilizzato la classe Calendar (equivalente della classe Data che è stata deprecata nelle nuove
specifiche J2SE)

</body>
</html>
```

**forma corretta**



# esempio4

```
<%@ page language="java" import="java.lang.*,java.util.*" %>
```

```
<%@page errorPage = "PaginaErrore.jsp" %>
```

```
<!DOCTYPE html>
```

```
<head>
```

```
<title>Prima pagina JSP</title>
```

```
<meta http-equiv="description" content="Questa è la mia prima pagina JSP"/>
```

```
</head>
```

```
<body>
```

```
<h1>Prima con errore che richiama una pagina per gestirlo</h1>
```

```
<h2><a href="index.jsp">home</a></h2>
```

```
<br/>
```

```
<%= 7/0 %>
```

```
</body>
```

```
</html>
```




# esempio5

```
<%@ page language="java" import="java.lang.*,java.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="IT">
<head>
<title>Prima pagina JSP</title>
<meta http-equiv="description" content="Questa è la mia prima pagina JSP"/>
</head>
<body>
<h1>Prima pagina EL - Expression Language</h1>

<br/>
${ 3 + 2 }
<hr/>

</body>
</html>
```



# esempio6

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head><title>JSP Page</title></head>
<body>
<%
int a=1,b=2,c=3,d=a+b+c;
out.println("<h1>Posso fare i calcoli!</h1>");
out.println("<h2>1+2+3="+d+"</h2>");
%>
</body>
</html>
```



# nelle pagine JSP

Come si può notare le sezioni racchiuse tra **<%**  
**e %>** sono quelle che contengono le istruzioni  
in linguaggio Java  
(*gli stessi limitatori utilizzati anche nelle pagine ASP*).

Una JSP è una pagina in HTML con incapsulato  
al suo interno delle direttive JSP rappresentanti  
codice Java

***Nota bene:*** i file per essere riconosciuti dal  
server come pagine JSP **devono essere salvati**  
con



estensione **.jsp**

# Un semplice form html

Ora realizziamo una semplice pagina html che **chiederà l'inserimento di due valori** numerici ed il server dovrà risponderci con una pagina che visualizzi la somma dei due numeri inviati.

I file utilizzati saranno *index.htm* e *somma.jsp* che saranno collocati nella cartella *CorsoJSP* all'interno di *webapps*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="IT">
<head>
<title>Corso JSP - Lezione 2</title>
<meta http-equiv="description" content="Invio di due valori numerici"/>
</head>
<body>
<form method="post" action="somma.jsp">
<fieldset>
<legend>Inserisci due valori numerici</legend>
<label for="valore1">Valore 1:</label>
<input type="text" id="valore1" name="valore1"><br>
<label for="valore2">Valore 2:</label>
<input type="text" id="valore2" name="valore2"><br>
<input type="submit" value="Invia valori">
</fieldset>
</form>
</body>
</html>
```



# La pagina *somma.jsp*

Il valore dell'attributo action del tag form definisce la pagina che dovrà intercettare i valori. La pagina *somma.jsp* sarà così composta:

```
<%@ page language="java" import="java.lang.*,java.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="IT">
<head>
<title>Somma di due numeri</title>
</head>
<body>
<h1>Somma di due numeri</h1>
<br/>
<%
String risultato;
// si prelevano i due valori
String valore1=request.getParameter("valore1");
String valore2=request.getParameter("valore2");
// si convertono i valori stringa in valori numerici decimali e si sommano
try
{
int somma=Integer.parseInt(valore1)+Integer.parseInt(valore2);
risultato=valore1 +" + "+ valore2 +" = "+ Integer.toString(somma);
} catch (Exception e) {
//si verifica quando i valori sono errati
risultato="Valori numerici errati";
}
//si stampa il messaggio
out.write(risultato);
%>
</body>
</html>
```





# La pagina *somma.jsp* commento

La pagina *somma.jsp* preleva i valori inviati dal client grazie al metodo ***getParameter*** dell'oggetto implicito *request*

I valori passati saranno ovviamente in formato stringa, quindi dovremo convertirli in interi utilizzando il metodo ***parseInt*** della classe ***Integer***.

Dopodichè costruiremo la stringa di risposta (variabile risultato).

Il costrutto ***try...catch*** serve per gestire gli errori che potrebbero riscontrarsi nelle istruzioni contenute nel blocco try. In Java la **gestione degli errori** avviene in una modalità diversa da altri linguaggi come *C* o *Visual Basic*.



# La pagina *somma.jsp* commento

Con le eccezioni del linguaggio Java il programmatore è “**obbligato**” a **gestire le eccezioni** o comunque deve dichiarare esplicitamente di non volerle gestire, in questo modo viene sempre reso consapevole che una parte di codice potrebbe generare delle eccezioni e qui sta la robustezza del linguaggio Java.

Nel nostro caso i valori inviati potrebbero essere dei caratteri alfanumerici, o valori decimali (l'algoritmo presentato gestisce solo la somma di interi) che causerebbero errori nell'esecuzione della pagina, introducendo il ***try..catch*** ci assicuriamo sempre il controllo della nostra applicazione anche in caso di errore.



# La somma!

Dopo aver creato i file proviamo che il tutto funzioni. Apriamo il browser al seguente **indirizzo**

<http://localhost:8080/CorsoJSP/>

Inserisci due valori numerici

Valore 1:

Valore 2:

Inseriamo dei valori interi e premiamo il pulsante “Invia valori” dovremmo ottenere:

## Somma di due numeri

10 + 20 = 30



# Gestione Errori

## Errori al momento della compilazione

Questo tipo di errore si verifica al momento della prima richiesta, quando il codice JSP viene tradotto in servlet. Generalmente sono causati da **errori di compilazione** ed il motore JSP, che effettua la traduzione, si arresta nel momento in cui trova l'errore ed invia al client richiedente una pagina di **"Server Error" (o errore 500)** con il dettaglio degli errori di compilazione.

## Errori al momento della richiesta

Questi altri errori sono quelli su cui ci soffermeremo perché sono causati da errori **durante l'esecuzione della pagina** e non in fase di compilazione. (contenuto della pagina o qualche altro oggetto contenuto in essa).

I programmatori java sono abituati ad intercettare le eccezioni innescate da alcuni tipi di errori, nelle pagine JSP questo non è più necessario perché la **gestione dell'errore** in caso di eccezioni viene **eseguita automaticamente dal servlet generato** dalla pagina JSP. Il compito del programmatore si riduce al **creare un file .jsp** che si occupi di gestire l'errore e che permetta in qualche modo all'utente di tornare senza troppi problemi all'esecuzione dell'applicazione JSP.

# ErrorPage

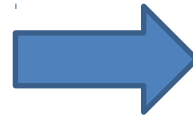
Creazione di una pagina di errore

Una pagina di errore può essere vista come una normale pagina JSP in cui si specifica, tramite l'opportuno **parametro** della **direttiva page**, che si tratta del codice per gestire l'errore.

# ErrorPage

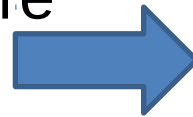
Ecco un semplice esempio: PaginaErrore.jsp

`<%@ page isErrorPage = "true" %>`



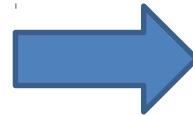
Direttiva JSP

Siamo spiacenti, si è verificato un errore  
durante l'esecuzione:



Messaggio html

`<%= exception.getMessage()%>`



Espressione JSP

# ErrorPage

A parte la direttiva page il codice Java è composto da un'unica riga che utilizza l'oggetto **exception** (implicitamente contenuto in tutte le pagine di errore) richiamando l'oggetto **getMessage()** che restituisce il messaggio di errore.

## Uso delle pagine di errore

Perché nelle nostre pagine JSP venga utilizzata una determinata pagina di errore l'unica cosa da fare è inserire la seguente direttiva:

```
<% page errorPage = "PaginaErrore.jsp" %>
```

che specifica quale pagina di errore deve essere richiamata in caso di errore in fase di esecuzione.



# Torna al sommario



## Ciclo di vita di una pagina Jsp, tipi di tag JSP: dichiarazioni, espressioni, scriptlet, direttive

- ciclo di vita
- dichiarazioni
- espressioni
- scriptlet
- direttive



# 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
- nelle chiamate successive la pagina JSP (i.e. la servlet corrispondente) viene solo eseguita



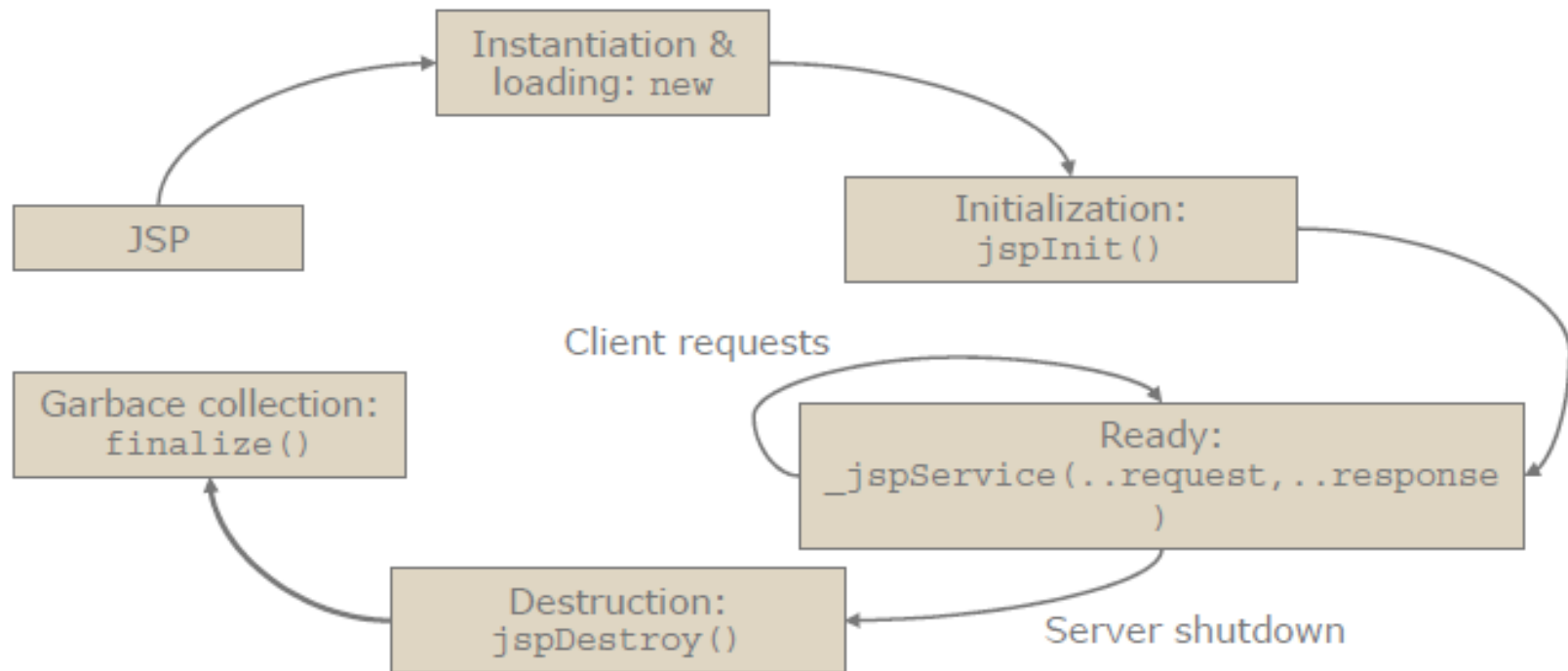
# Ciclo di vita di una pagina JSP

- ad ogni invocazione il server web verifica se la pagina JSP è più recente della corrispondente Servlet
- se lo è, perché ad esempio la pagina JSP è stata modificata, allora la pagina viene di nuovo tradotta, compilata, caricata ed eseguita
- caratteristica molto comoda in fase di sviluppo ma da disattivare al rilascio dell'applicazione perché è costosa in termini di risorse



# Ciclo di vita di una pagina JSP

**Dal momento che le JSP sono compilate in servlet, il ciclo di vita delle JSP è controllato dal web container**



# Ciclo di vita di una pagina JSP

- Il JSP container traduce la JSP in una classe servlet, che gestisce la richiesta corrente e quelle future
- Esecuzione in quattro passi:
  - 1.il *JSP engine* (o *JSP container*) parserizza la pagina e crea un sorgente Java (una classe servlet)
  - 2.il file viene compilato in un file Java *.class*
  - 3.il contenitore carica e inizializza il servlet
  - 4.il servlet elabora le richieste e restituisce i risultati

## Note:

- I passi 1 e 2 vengono eseguiti al primo utilizzo della pagina e ad ogni aggiornamento
- il passo 3 è eseguito solo alla prima richiesta del servlet
- il passo 4 può essere eseguito più volte



# Ciclo di vita del servlet generato

Quando si chiama per prima volta il file JSP, si genera un **servlet** con le seguenti operazioni

- **jspInit()**
  - inizializza il servlet generato
  - solo nella prima richiesta
- **jspService(richiesta,risposta)**
  - Maneggia le richieste. si invoca in ogni richiesta, incluso nella prima
- **jspDestroy()**
  - Invocata dal motore per eliminare il servlet



# Servlet generato da una JSP

Nel metodo `_jspService` si introduce automaticamente il contenuto dinamico della pagina JSP.

- il codice html si trasforma in una chiamata all'oggetto `out` dove si trova il contenuto Esempio:

```
out.write("\r\n\r\n<HTML>\r\n<head> <title>  
  ciao, mondo </title> </head>\r\n\r\n<body> <h1>  
  ;ciao, mondo! </h1> \r\nLa data di oggi es: ");
```

- il codice dinamico si traduce in funzione del contenuto

Es: il codice jsp `<%= new Date().toString() %>` Si traduce con `out.print( new Date().toString() );`





# Servlet generato da una JSP

nella prima invocazione di questa pagina si genera automaticamente il seguente servlet:

```
public class ej1_0005fciao$jsp extends HttpJspBase {
    ....
    public final void _jspx_init() throws JasperException {
        ...
    public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
    ....

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        String _value = null;
```



# Servlet generato da una JSP

```
try {  
    if (_jspx_inited == false) {  
        _jspx_init();  
        _jspx_inited = true;  
    }  
    _jspxFactory = JspFactory.getDefaultFactory();  
  
    response.setContentType("text/HTML; charset=ISO-8859-1");  
    pageContext =  
_jspxFactory.getPageContext(this, request, response, "",  
true, 8192, true);  
  
    application = pageContext.getServletContext();  
    config = pageContext.getServletConfig();  
    session = pageContext.getSession();  
    out = pageContext.getOut();  
}
```



# Servlet generato da una JSP

```
// HTML
// begin [file="C:\\web\\jakarta-
tomcat3.2.2\\webapps\\marian\\ciao.jsp";from=(0,40);to=(
6,20)]
    out.write("\r\n\r\n<HTML>\r\n<head> <title> ciao, mondo
</title> </head>\r\n\r\n<body> <h1> ;ciao, mondo!
</h1> \r\nLa data di oggi es: ");
// end
// begin [file="C:\\web\\jakarta-
tomcat3.2.2\\webapps\\marian\\ciao.jsp";from=(6,23);to=(
6,46)]
    out.print( new Date().toString() );
// end
// HTML // begin [file="C:\\web\\jakarta-
tomcat3.2.2\\webapps\\marian\\ciao.jsp";from=(6,48);to=(
11,0)]
    out.write("\r\n\r\n</body>\r\n</HTML>\r\n\r\n");
// end
```

