



CORSO DI LAUREA IN INFORMATICA

TECNOLOGIE SOFTWARE PER IL WEB

JSP: JAVA SERVER PAGES

a.a 2020-2021

Java Server Pages

- Le JSP sono uno dei due componenti di base della tecnologia J2EE, relativamente alla parte Web:
 - template per la generazione di contenuto dinamico
 - estendono HTML con codice Java, Expression Language (EL), JSTL, , etc.
- Quando viene effettuata una richiesta a una JSP:
 - Viene chiamata la sua traduzione in servlet creata dal servlet container
 - **Il codice Java viene eseguito** per la generazione del contenuto della pagina (HTML, CSS, javascript, etc.)
 - la pagina così formata (parte statica + parte generata dinamicamente) viene inviata al client nella risposta
- Assimilabili ad un linguaggio di script (es. PHP, Perl, ...)

Esempio: Hello world

- Consideriamo una JSP, denominata *helloWorld.jsp*, che realizza il classico esempio “Hello World!” in modo parametrico:

```
<html>
  <body>
    <% String visitor=request.getParameter("name");
      if (visitor == null) visitor = "World"; %>
    Hello, <%= visitor %>!
  </body>
</html>
```

<http://myHost/myWebApp/helloWord.jsp>

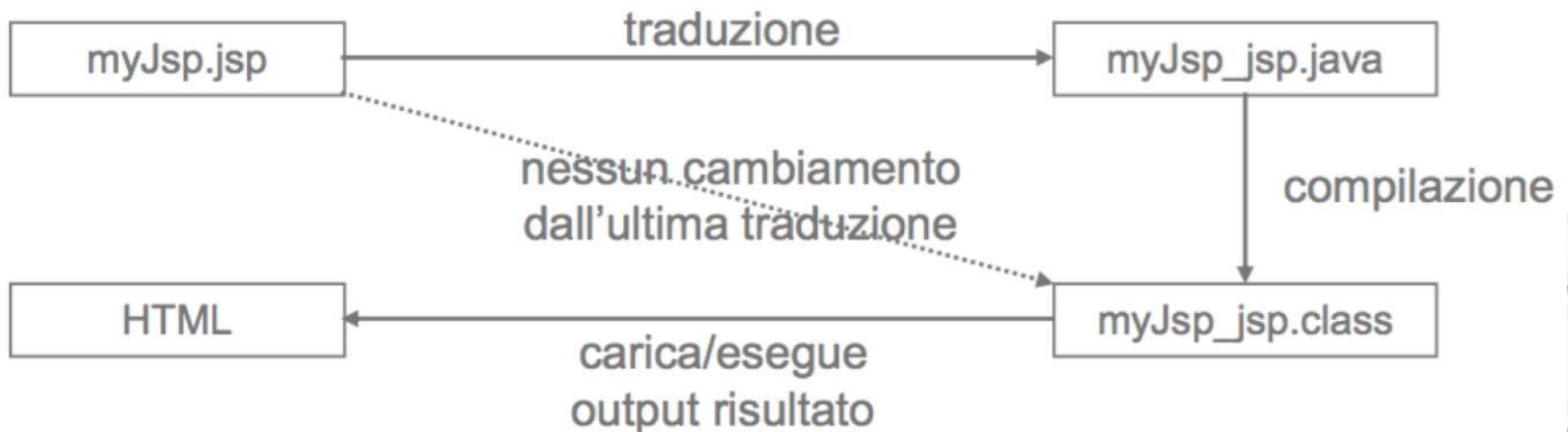
```
<html>
  <body>
    Hello, World!
  </body>
</html>
```

<http://myHost/myWebApp/helloWord.jsp?name=Mario>

```
<html>
  <body>
    Hello, Mario!
  </body>
</html>
```

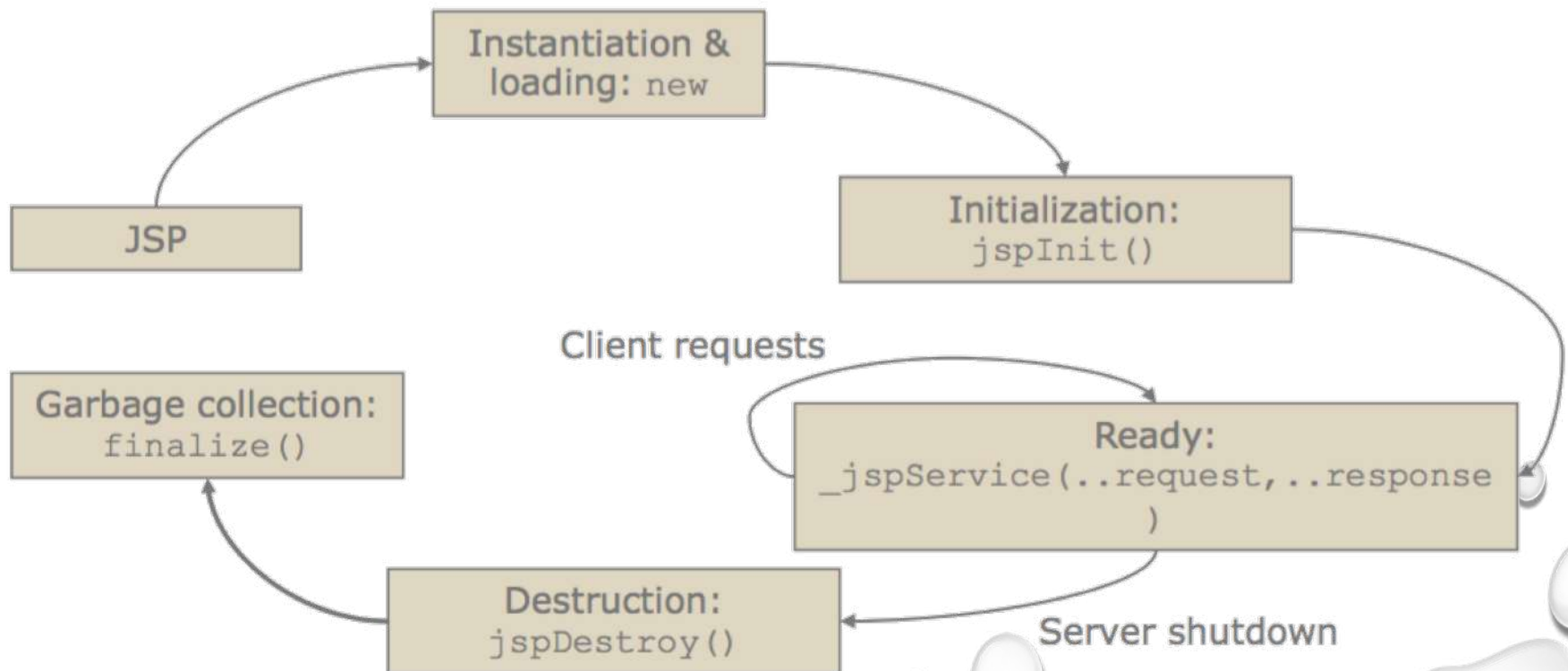
JspServlet

- Le richieste verso JSP sono gestite da una particolare Servlet (in Tomcat si chiama **JspServlet**) che effettua le seguenti operazioni:
 - traduzione della JSP in una Servlet
 - compilazione della Servlet risultante in una classe
 - esecuzione della JSP
- I primi due passi vengono eseguiti **solo quando cambia il codice** della JSP (che include la prima chiamata alla jsp)



Ciclo di vita delle JSP

- Dal momento che le JSP sono compilate in Servlet, il ciclo di vita delle JSP (**dopo la compilazione**) è controllato sempre dal servlet container (Tomcat, nel nostro caso)



Servlet e JSP: perché usare JSP?

- Nella Servlet la logica per la generazione del documento HTML è implementata completamente in Java
 - Il processo di generazione delle pagine è time-consuming, ripetitivo e soggetto a errori (sequenza di *println()*)
 - L'aggiornamento delle pagine è scomodo
- JSP nascono per facilitare la progettazione grafica e l'aggiornamento delle pagine
 - Si può separare agevolmente il lavoro fra grafici e programmatori
 - I Web designer possono produrre pagine senza dover conoscere i dettagli della logica server-side
 - La generazione di codice dinamico è implementata sfruttando il linguaggio Java ed altri linguaggi high-level quali EL e JSTL (che comunque poi vengono tradotti in java)

Servlet o JSP?

- Le JSP non rendono inutili le Servlet
 - Le Servlet forniscono agli sviluppatori delle applicazioni Web, un completo controllo dell'applicazione
- *Se si vogliono fornire contenuti differenziati a seconda di diversi parametri quali l'identità dell'utente, condizioni dipendenti dalla business logic, etc. è conveniente continuare a lavorare con le Servlet*
- Le JSP rendono viceversa molto semplice presentare documenti HTML (o loro parti) all'utente; dominanti per la realizzazione di pagine dinamiche semplici e di uso frequente

Esempio: negative-balance.jsp

- Consideriamo la JSP, denominata *negative-balance.jsp*, dall'esercitazione modelloMVC

```
<!DOCTYPE html>
<html>
<head><title>You Owe Us Money!</title>
  <link href="./css/styles.css"
        rel="stylesheet" type="text/css"/>
</head>
<body>
<div align="center">
  <table class="title">
    <tr>
      <th>
        We Know Where You Live!
      </th>
    </tr>
  </table>
<p/>


  <!-- the View uses the "customer" javabean to
  <h2>Watch out, ${customerkey.firstName},
    we know where you live. </h2>
  <h2>Pay us the ${customerkey.balanceNoSign}
    you owe us before it is too late!</h2>

</div>
</body>
</html>
```


Servlet corrispondente a jsp 1/2

- negative-balance.jsp del progetto modelloMVC, dopo la traduzione in servlet, diventa il file java con il seguente path:

Apache-tomcat-9.0.43/work/Catalina/localhost/modelloMVC-0/org/apache/jsp/WEB_002dINF/results/
negative_002dbalance_jsp.java

...

```
public final class negative_002dbalance_jsp extends org.apache.jasper.runtime.HttpJspBase. ... {
```

...

```
public void _jspInit() {}  
public void _jspDestroy() {}
```

```
public void _jspService(final javax.servlet.http.HttpServletRequest request, final  
javax.servlet.http.HttpServletResponse response)
```

```
    throws java.io.IOException, javax.servlet.ServletException {
```

...

```
    final javax.servlet.jsp.PageContext pageContext;  
    javax.servlet.http.HttpSession session = null;  
    final javax.servlet.ServletContext application;  
    final javax.servlet.ServletConfig config;  
    javax.servlet.jsp.JspWriter out = null;  
    final java.lang.Object page = this;
```

...

```
    try {  
        ...  
        out.write("<!DOCTYPE html>\r\n");  
        out.write("<html>\r\n");  
        out.write("<head><title>You Owe Us Money!</title>\r\n");
```

Servlet corrispondente a jsp 2/2

Suggerimento: dopo aver fatto il deploy manuale di un .war in Tomcat/webapps andate nella cartella Tomcat/work e cercate le traduzioni in java delle vostre jsp. (Naturalmente dovete averle chiamate almeno una volta)

```
out.write("    <link href=\"./css/styles.css\">\r\n");
out.write("        rel=\"stylesheet\" type=\"text/css\">\r\n");
out.write("</head>\r\n");
out.write("<body>\r\n");
out.write("<div align=\"center\">\r\n");
out.write("    <table class=\"title\">\r\n");
out.write("        <tr>\r\n");
out.write("            <th>\r\n");
out.write("                We Know Where You Live!\r\n");
out.write("            </th>\r\n");
out.write("        </tr>\r\n");
out.write("    </table>\r\n");
out.write("    <p>\r\n");
out.write("        <img src=\"./images/club.gif\" align=\"left\">\r\n");
out.write("\r\n");
out.write("    <!-- the View uses the \"customer\" javabean to extract the\n        information retrieved by the Model -->\r\n");
out.write("    <h2>Watch out, ");
out.write((java.lang.String) org.apache.jasper.runtime.PageContextImpl
    .proprietaryEvaluate("${customerkey.firstName}", java.lang.String.class,
    (javax.servlet.jsp.PageContext) jspcx_page_context, null));
out.write(",\r\n");
out.write("        we know where you live. </h2>\r\n");
out.write("    <h2>Pay us the $");
out.write((java.lang.String) org.apache.jasper.runtime.PageContextImpl
    .proprietaryEvaluate("${customerkey.balanceNoSign}",
    java.lang.String.class, (javax.servlet.jsp.PageContext) jspcx_page_context,
    null));
out.write("\r\n");
out.write("        you owe us before it is too late!</h2>\r\n");
out.write("</div>\r\n");
out.write("</body>\r\n");
out.write("</html>");
} catch (java.lang.Throwable t) {
```

Tag

- Le parti variabili della pagina sono contenute all'interno di **tag** speciali
- Sono possibili due tipi di sintassi per questi tag:
 - **Scripting-oriented tag**
 - **XML-Oriented tag**
- Le **scripting-oriented tag** sono definite da delimitatori entro cui è presente lo scripting (self-contained)
- Sono di quattro tipi:
 - **<%! %> Dichiarazione**
 - **<%= %> Espressione**
 - **<% %> Scriptlet**
 - **<%@ %> Direttiva**

Servlet corrispondente a jsp

- `<%@ %>`, `<%! %>`, `<% %>` e `<%= %>` inseriscono codice java direttamente nel file java tradotto dalla jsp come segue:

... Per inserire, in questo punto, import ed altre specifiche a livello di file usa `<%@ %>`

```
public final class negative_002dbalance_jsp extends org.apache.jasper.runtime.HttpJspBase. ... {
```

... Per inserire, in questo punto, dichiarazioni a livello di classe ed altri metodi in questa classe usa `<%! %>`

```
public void _jspInit() {}  
public void _jspDestroy() {}
```

```
public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse  
response)
```

```
    throws java.io.IOException, javax.servlet.ServletException {
```

```
        ...
```

```
        final javax.servlet.jsp.PageContext pageContext;  
        javax.servlet.http.HttpSession session = null;  
        final javax.servlet.ServletContext application;  
        final javax.servlet.ServletConfig config;  
        javax.servlet.jsp.JspWriter out = null;  
        final java.lang.Object page = this;
```

```
        ...
```

```
        try {  
            ...  
            out.write("<!DOCTYPE html>\r\n");  
            out.write("<html>\r\n");  
            out.write("<head><title>You Owe Us Money!</title>\r\n");  
            ...  
        }
```

Per inserire qualsiasi codice java in questo metodo usa `<% %>`

Per inserire espressioni Java che restituiscono un valore (in questo metodo) usa `<%= %>`

```
}
```

Dichiarazioni

- Si usano i delimitatori `<%!` e `%>` per dichiarare variabili e metodi
- Variabili e metodi dichiarati possono poi essere referenziati in qualsiasi punto del codice JSP
- I metodi diventano metodi della Servlet quando la pagina viene tradotta

```
<%! String name = "Paolo Rossi";  
      double[] prices = {1.5, 76.8, 21.5};  
  
      double getTotal() {  
          double total = 0.0;  
          for (int i=0; i<prices.length; i++)  
              total += prices[i];  
          return total;  
      }  
%>
```

Espressioni

- Si usano i delimitatori **<%=** e **%>** per valutare espressioni Java
- Il risultato dell'espressione viene convertito in stringa inserito nella pagina al posto del tag

(continuando l'esempio della pagina precedente)

JSP

```
<p>Sig. <%=name%>,</p>  
<p>l'ammontare del suo acquisto è: <%=getTotal()%> euro.</p>  
<p>La data di oggi è: <%=new Date()%></p>
```



Pagina HTML risultante

```
<p>Sig. Paolo Rossi,</p>  
<p>l'ammontare del suo acquisto è: 99.8 euro.</p>  
<p>La data di oggi è: Tue Feb 20 11:23:02 2010</p>
```

Esempio 1 (declarations.jsp)

```
<%@ page language="java" import="java.util.Date, java.text.SimpleDateFormat"
    contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Declarations</title>
</head>
<body>

<%! String name = "Paolo Rossi";
    double[] prices = {1.5, 76.8, 21.5};

    double getTotal() {
        double total = 0.0;
        for(int i=0; i <prices.length; i++)
            total += prices[i];
        return total;
    }

    String formattedDate(Date today) {
        SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy HH.mm.ss");
        return formatter.format(today);
    }
%>

<h3>JSP Declarations</h3>
<p>Sig. <%=name %>, </p>
<p>l'ammontare del suo acquisto &grave;: <%=getTotal() %> euro.</p>
<p>La data di oggi &grave;: <%=formattedDate(new Date()) %></p>
</body>
</html>
```


Esempio 2 (expressions.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Expressions</title>
</head>
<body>
<h3>JSP Expressions</h3>
<ul>
<li>Current time: <%= new java.util.Date() %>
<li>Server: <%=application.getServerInfo() %>
<li>Application name: <%=application.getContextPath() %>
<li>Session Id: <%=session.getId() %>
<li>The <code>testParam</code> form parameter:
    <%=request.getParameter("testParam") %>
</ul>
</body>
</html>
```

JSP Expressions

- Current time: Sun May 01 18:43:21 CEST 2016
- Server: Apache Tomcat/8.0.32
- Application name: /jsp
- Session Id: 84F1ED2B7DD60439D6FD5BE749CF0043
- The testParam form parameter: null

- <http://myHost/myWebApp/expressions.jsp>
- <http://myHost/myWebApp/expressions.jsp?testParam=PW>

Scriptlet

- Si usano `<%` e `%>` per aggiungere un frammento di codice Java eseguibile alla JSP (**scriptlet**)
- Lo **scriptlet** consente tipicamente di inserire logiche di controllo di flusso nella produzione della pagina
- La combinazione di tutti gli scriptlet in una determinata JSP deve definire un blocco logico completo di codice Java

```
<% if (userIsLogged) { %>  
    <h1>Benvenuto Sig. <%=name%></h1>  
<% } else { %>  
    <h1>Per accedere al sito devi fare il login</h1>  
<% } %>
```

Come scrivere una scriptlet mista ad HTML

1. Scrivete java ed HTML come se fossero un unico linguaggio
2. Taggare con `<% %>` ciascun frammento di codice java e con `<%= %>` ciascuna espressione java che restituisce un valore

• Es.

`if (userIsLogged) {`

`<h1> Benvenuto Sig. name </h1>`

`} else {`

`<h1> Per accedere al sito devi fare il login </h1>`

`}`

```
<% if (userIsLogged) { %>
    <h1>Benvenuto Sig. <%=name%></h1>
<% } else { %>
    <h1>Per accedere al sito devi fare il login</h1>
<% } %>
```

Esempio 3 (scriptlet.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    info="simple jsp examples" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Scriptlet</title>
</head>
<body>
<h3>JSP Scriptlet</h3>
<ul>
    <li>Current time: <%= new java.util.Date() %>
    <li>Server: <%=application.getServerInfo() %>
    <li>Application name: <%=application.getContextPath() %>
    <li>Session Id: <%=session.getId() %>
    <% String param = request.getParameter("testParam");
    if(param != null) { %>
    <li>The <code>testParam</code> form parameter:
        <%=request.getParameter("testParam") %>
    <% } else { %>
    <li>No form parameter
    <% } %>
    <li>Info: <%=this.getServletInfo() %>
</ul>
</body>
</html>
```

Direttive

- Sono comandi JSP valutati a tempo di compilazione
- Le più importanti sono:
 - **page**: permette di importare package, dichiarare pagine d'errore, definire modello di esecuzione JSP relativamente alla concorrenza (ne discuteremo a breve), ecc.
 - **include**: include un altro documento
 - **taglib**: carica una libreria di custom tag implementate dallo sviluppatore
- *Non producono nessun output visibile*

```
<%@ page info="Esempio di direttive" %>  
<%@ page language="java" import="java.net.*" %>  
<%@ page import="java.util.List, java.util.ArrayList" %>  
<%@ include file="myHeaderFile.html" %>
```

La direttiva page

- La direttiva **page** definisce una serie di attributi che si applicano all'intera pagina
- Sintassi:

```
<%@ page
  [ language="java" ]
  [ extends="package.class" ]
  [ import="{package.class | package.*}, ..." ]
  [ session="true | false" ]
  [ buffer="none | 8kb | sizekb" ]
  [ autoFlush="true | false" ]
  [ isThreadSafe="true | false" ]
  [ info="text" ]
  [ errorPage="relativeURL" ]
  [ contentType="mimeType [ ;charset=characterSet ]" |
    "text/html ; charset=ISO-8859-1" ]
  [ isErrorPage="true | false" ]
%>
```

N.B. valori sottolineati sono quelli di default

Attributi di page


- **language="java"** linguaggio di scripting utilizzato nelle parti dinamiche, allo stato attuale l'unico valore ammesso è "java"
- **import="{package.class | package.*},..."** lista di package da importare.
 - Gli import più comuni sono impliciti e non serve inserirli (java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*)
- **session="true | false"** indica se la pagina fa uso della sessione (altrimenti non si può usare session)
- **buffer="none | 8kb | sizekb"** dimensione in KB del buffer di uscita
- **autoFlush="true | false"** dice se il buffer viene svuotato automaticamente quando è pieno
 - Se il valore è **false** viene generata un'eccezione quando il buffer è pieno

Attributi di page (2)

- **isThreadSafe**="true | false" indica se il codice contenuto nella pagina è thread-safe
 - Se vale **false** le chiamate alla JSP vengono serializzate
- **info**="**text**" testo di commento
 - Può essere letto con il metodo **Servlet.getServletInfo()**
 - Es: `<%= this.getServletInfo()%>` oppure `<%= page.getServletInfo()%>`
- **errorPage**="**relativeURL**" indirizzo della pagina a cui vengono inviate le eccezioni
- **isErrorPage**="**true** | false" indica se JSP corrente è una pagina di errore
 - Si può utilizzare l'oggetto **exception** solo se l'attributo è **true** (la traduzione in java della jsp contiene una dichiarazione per una variabile con nome exception)
- **contentType**="**contentType [;charset=charSet]" | "text/html; charset=ISO-8859-1"** indica il tipo MIME e il codice di caratteri usato nella risposta


Esempio 4 (error.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isErrorPage="true"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Error</title>
</head>
<body>
<h3> Error</h3>
<% if(exception != null) { %>
<p>An exception was raised: <%= exception.toString() %></p>
<p>Exception message is: <%= exception.getMessage() %></p>
<br>
<%
    StackTraceElement[] st = exception.getStackTrace();
    for(StackTraceElement e: st){
        out.println(e.toString());
    }
%>
</body>
</html>
```



Esempio 4b (call error.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    errorPage="error.jsp"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Call Error Page</title>
</head>
<body>
<h3>JSP Call Error Page</h3>
    <% String param = request.getParameter("testParam");
        if(param.equals("PW")) { %>
            The <code>testParam</code> form parameter: <%=param %>
        <% } else { %>
            No form parameter
        <% } %>
</body>
</html>
```



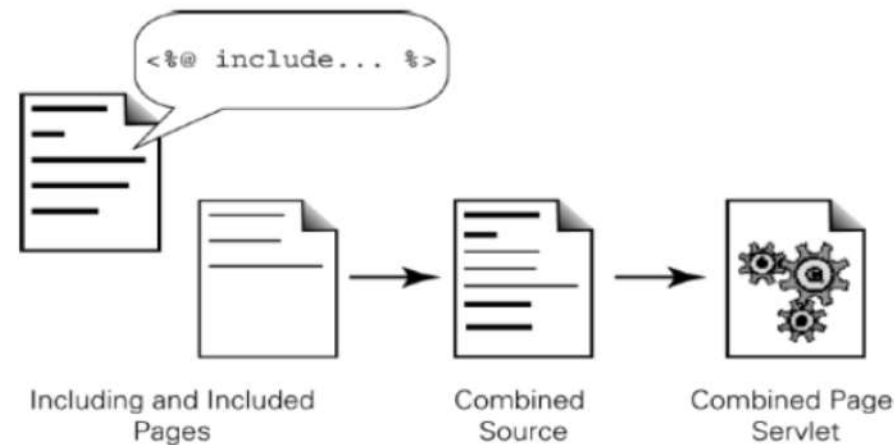
Errore: null pointer
assegnato a testParam

- <http://myHost/myWebApp/callerror.jsp>

La direttiva include

- Sintassi: `<%@ include file = "localURL"%>`
- Serve ad includere il contenuto del file specificato
 - È possibile nidificare un numero qualsiasi di inclusioni
 - L'inclusione viene fatta a tempo di compilazione: eventuali modifiche al file incluso non determinano una ricompilazione della pagina che lo include **PERICOLOSO!!**

La ricompilazione del file che include va forzata perché non automatica come al solito.



- Esempio:
`<%@ include file="/shared/copyright.html"%>`

```
<html>
<body>
<%@ include file="header.jsp" %>
<br>
Contact Us at: we@studytonight.com
<br/>
<%@ include file="footer.jsp" %>
</body>
</html>
```

This says insert the complete content of **header.jsp** into this JSP page

This says insert the complete content of **footer.jsp** into this JSP page

Esempio 5(compact.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>JSP Include</title>
```

```
</head>
```

```
<body>
```

```
<%@ include file="header.jsp" %>
```

```
<br>
```

```
Contact Us at: we@studytonight.com<br>
```

```
<br>
```

```
<%@ include file="footer.jsp" %>
```

```
</body>
```

```
</html>
```

header.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<div>
```

```
HEADER
```

```
</div>
```

footer.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<div>
```

```
FOOTER
```

```
</div>
```

HEADER

Contact Us at: we@studytonight.com

FOOTER

Built-in objects (con scope differenziati)

- Le specifiche JSP definiscono **8 oggetti built-in** (o impliciti) utilizzabili senza doverle definire ed istanziarle
- Rappresentano utili riferimenti ai corrispondenti oggetti Java veri e propri presenti nella tecnologia Servlet

| Oggetto | Classe/Interfaccia |
|-------------|---|
| page | <code>javax.servlet.jsp.HttpJspPage</code> |
| config | <code>javax.servlet.ServletConfig</code> |
| request | <code>javax.servlet.http.HttpServletRequest</code> |
| response | <code>javax.servlet.http.HttpServletResponse</code> |
| out | <code>javax.servlet.jsp.JspWriter</code> |
| session | <code>javax.servlet.http.HttpSession</code> |
| application | <code>javax.servlet.ServletContext</code> |
| pageContext | <code>javax.servlet.jsp.PageContext</code> |
| exception | <code>java.lang.Throwable</code> |

Oggetto page

- L'oggetto page rappresenta l'istanza corrente della Servlet
 - Ha come tipo l'interfaccia HTTPJspPage che discende da JSP page, la quale a sua volta estende Servlet
 - Può quindi essere utilizzato per accedere a tutti i metodi definiti nelle Servlet

JSP

```
<%@ page info="Esempio di uso page." %>  
<p>Page info:  
  <%=page.getServletInfo() %>  
</p>
```

Pagina HTML

```
<p>Page info: Esempio di uso di page</p>
```

Oggetto config

- Contiene la configurazione della Servlet (parametri di inizializzazione)
- *Poco usato in pratica in quanto in generale nelle JSP sono poco usati i parametri di inizializzazione*
- Metodi di config:
 - **getInitParameterNames()** restituisce tutti i nomi dei parametri di inizializzazione
 - **getInitParameter(name)** restituisce il valore del parametro passato per nome

Oggetto request

- Rappresenta la richiesta alla pagina JSP
- È il parametro request passato al metodo service() della servlet
- Consente l'accesso a tutte le informazioni relative alla richiesta HTTP:
 - indirizzo di provenienza, URL, headers, cookie, parametri, ecc.

```
<% String xStr = request.getParameter("num");  
try  
{  
    long x = Long.parseLong(xStr); %>  
    Fattoriale: <%= x %>! = <%= fact(x) %>  
<%}  
catch (NumberFormatException e) { %>  
    Il parametro <b>num</b>non contiene un valore intero.  
<%} %>
```


Oggetto response

- Oggetto legato all'I/O della pagina JSP
- Rappresenta la risposta che viene restituita al client
- Consente di inserire nella risposta diverse informazioni:
 - content type ed encoding
 - eventuali header di risposta
 - URL Rewriting
 - i cookie

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


Oggetto out

- Oggetto legato all'I/O della pagina JSP
- È uno stream di caratteri e rappresenta lo stream di output della pagina
- Esempio:

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

Metodi dell'oggetto out

- **isAutoFlush()** dice se output buffer è stato impostato in modalità autoFlush o meno
- **getBufferSize()** restituisce dimensioni del buffer
- **getRemaining()** indica quanti byte liberi ci sono nel buffer
- **clearBuffer()** ripulisce il buffer
- **flush()** forza l'emissione del contenuto del buffer
- **close()** fa flush e chiude lo stream

Oggetto session

- Oggetto che fornisce informazioni sul contesto di esecuzione della JSP
- Rappresenta la sessione corrente per un utente
- L'attributo **session** della direttiva **page** deve essere **true** affinché JSP partecipi alla sessione

```
<% UserLogin userData = new UserLogin(name, password);  
    session.setAttribute("login", userData);  
%>  
<%UserLogin userData=(UserLogin)session.getAttribute("login");  
    if (userData.isGroupMember("admin")) {  
        session.setMaxInactiveInterval(60*60*8);  
    } else {  
        session.setMaxInactiveInterval(60*15);  
    }  
%>
```

Oggetto application

- Oggetto che fornisce informazioni su contesto di esecuzione della JSP (è **ServletContext**)
- Rappresenta la Web application a cui JSP appartiene
- Consente di interagire con l'ambiente di esecuzione:
 - fornisce la versione di JSP Container
 - garantisce l'accesso a risorse server-side
 - permette accesso ai parametri di inizializzazione relativi all'applicazione
 - consente di gestire gli attributi di un'applicazione

Oggetto pageContext

- Oggetto che fornisce informazioni sul contesto di esecuzione della pagina JSP
- Rappresenta l'insieme degli oggetti impliciti di una JSP
 - Consente accesso a tutti gli oggetti impliciti e ai loro attributi
 - Consente trasferimento del controllo ad altre pagine
- Utilizzando questo oggetto è possibile lavorare con gli attributi (e.g., find, get, set, remove) in qualsiasi livello
 1. JSP Page – Scope: **PAGE_CONTEXT** (*default*)
 2. HTTP Request – Scope: **REQUEST_CONTEXT**
 3. HTTP Session – Scope: **SESSION_CONTEXT**
 4. Application Level – Scope: **APPLICATION_CONTEXT**
- Es:

```
<% pageContext.setAttribute("role", "manager", PageContext.SESSION_SCOPE); %>
<% pageContext.getAttribute("mail", PageContext.APPLICATION_SCOPE); %>
```

Oggetto exception

- 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 ma solo nelle Error Page (quelle dichiarate con l'attributo `errorPage` impostato a `true`)
- Esempio:

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

JavaBeans

- JavaBeans è il modello di “base” per componenti Java, il più semplice...
- Un **JavaBean**, o semplicemente **bean**, non è altro che una classe Java dotata di alcune caratteristiche particolari:
 - Classe **public**
 - Ha un **costruttore** public di default (senza argomenti)
 - Espone proprietà, sotto forma di coppie di metodi di accesso (**accessors**) costruiti secondo una convenzione standard per i nomi dei metodi (get... set...)
 - La proprietà **prop** è definita da due metodi **getProp()** e **setProp()**
 - Il tipo del parametro di setProp(...) e del valore di ritorno di ... getProp() devono essere uguali e rappresentano il tipo della proprietà (può essere un tipo primitivo o una qualunque classe Java)
 - Es.: **void setLength(int value)** e **int getLength()** identificano proprietà **length** di tipo **int**

JavaBeans (2)

- Se definiamo solo il metodo get avremo una proprietà in sola lettura (*read-only*)
- Le proprietà di tipo boolean seguono una regola leggermente diversa: metodo di lettura ha la forma **isProp()**
- Es: la proprietà **empty** sarà rappresentata dalla **coppia**:
void setEmpty(boolean value) e **boolean isEmpty()**
- Ci sono anche proprietà indicizzate per rappresentare collezioni di valori
- Es: **String getItem(int index)** e **setItem(int Index, String value)** definiscono la proprietà indicizzata **String item[]**

Esempio

- Creiamo un bean che espone due proprietà in sola lettura (ore e minuti) e ci dà l'ora corrente

```
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; }
}
```