

Svantaggi delle servlet

- Le servlet sono la tecnologia principale di J2EE per comunicare con i browser dei client.
- Tuttavia ci sono degli inconvenienti per gli sviluppatori di applicazioni web:
 - le servlet “mescolano” la logica con la presentazione;
 - è scomodo dover “inglobare” il codice HTML nel codice Java per mezzo di **println** (soprattutto se il codice HTML è complesso);
 - il “rapid prototyping” non è possibile: ogni modifica richiede una ricompilazione esplicita e/o il cambiamento del file **web.xml** con il riavvio dell’applicazione o del server.

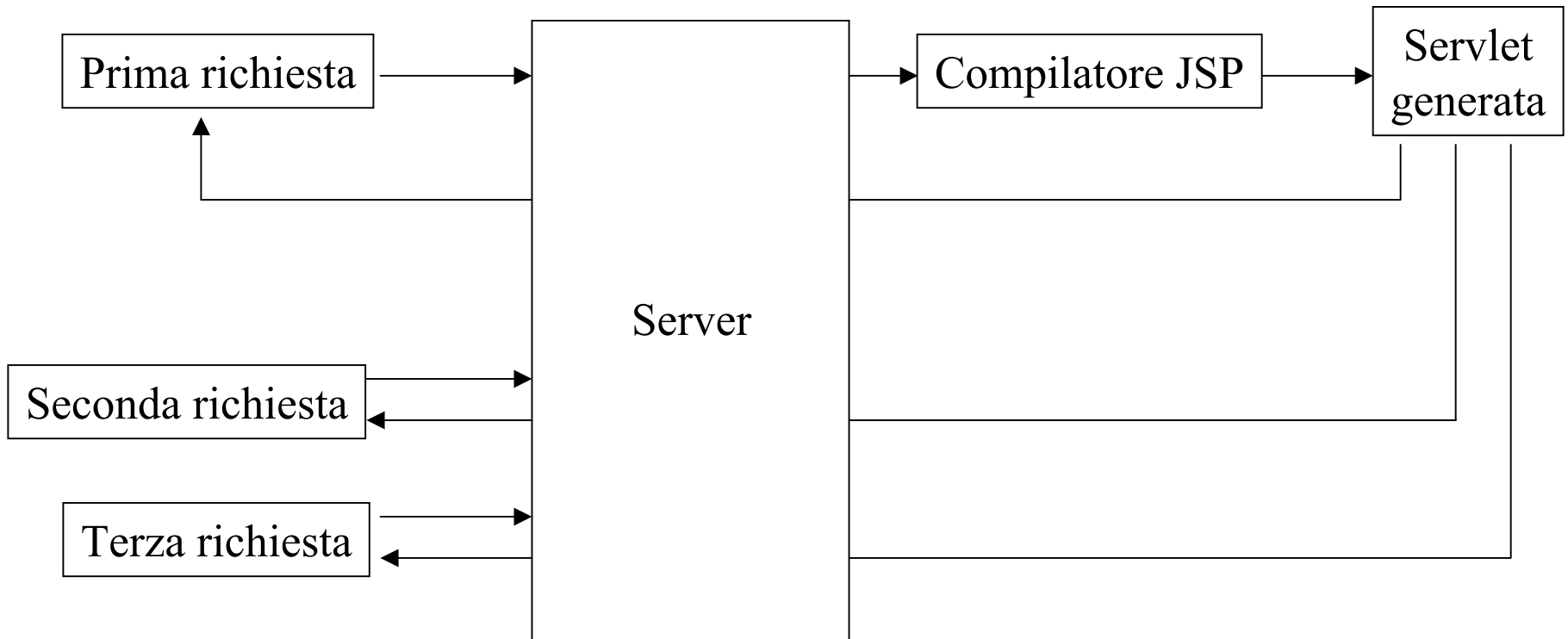
Java Server Pages (JSP)

- JSP è una tecnologia basata sull'API delle servlet che permette di fornire contenuti dinamici inserendo dei tag appositi in documenti HTML (template).
- Il funzionamento di una pagina JSP è quindi analogo a quello di una pagina PHP:
 - vi sono dei tag immersi nell'HTML;
 - un programma preprocessore elabora la risorsa (quando viene richiesta da un client) rimpiazzando i tag con codice HTML;
 - la pagina risultante viene spedita al browser del client.

JSP vs. PHP

- In realtà vi sono alcune differenze significative fra JSP e PHP:
 - i tag di una pagina JSP contengono codice Java (con le stesse potenzialità del codice di una servlet);
 - una pagina JSP **non** è interpretata:
 - alla prima richiesta, viene tradotta in una servlet, compilata e caricata in memoria centrale;
 - le richieste successive vengono gestite dalla servlet;
 - in caso di modifiche alla pagina JSP, il server se ne accorge e ritraduce/ricompila la pagina;
 - il server deve avere a disposizione il compilatore java (**javac**): non basta il JRE!

Funzionamento di JSP



La “filosofia” alla base di JSP

- Il passaggio dalla programmazione di servlet a quella di pagine JSP dovrebbe essere naturale.
- Il programmatore JSP
 - pensa come il programmatore PHP (“frammenti” di codice immersi nell’HTML);
 - è consapevole che il tutto sarà trasformato in una servlet equivalente;
 - può utilizzare la servlet generata per il debugging.

La prima JSP (Data.jsp)

- Scriviamo una semplice JSP che visualizzi la data e ora correnti:

```
<%@ page import="java.util.Date"
import="java.text.*"%>
<%! DateFormat formatoData=new
SimpleDateFormat("dd/MM/yyyy");%>
<html>
  <body>
    La data corrente &egrave;
    <%= formatoData.format(new Date()) %>
  </body>
</html>
```

Dove salvare i file JSP

- Il codice precedente può essere memorizzato in un file `Data.jsp` nella directory radice di una web application (e.g., `/home/<nome-utente>/servlets`).
- URL:
`http://latoSERVER.dimi.uniud.it:8080/<nome-utente>/Data.jsp`.
- La servlet generata da Tomcat viene salvata in `$TOMCAT_HOME/work/Catalina/localhost/<nome-utente>/org/apache/jsp:`
 - `Data_jsp.java`
 - `Data_jsp.class`

La servlet generata da Tomcat (I)

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.util.Date;
import java.text.*;
public final class Data_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
    DateFormat formatoData=new SimpleDateFormat("dd/MM/yyyy");
    ...
    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;
    ...
```


La servlet generata da Tomcat (II)

```
...
try {
    _jspxFactory = JspFactory.getDefaultFactory();
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                                              null, true, 8192, true);
...
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
...
    out.write('\r');
    out.write('\n');
    out.write("\r\n");
    out.write("<html>\r\n");
    out.write("  <body>\r\n");
    out.write("    La data corrente &grave; \r\n");
    out.write("    ");
    out.print( formatoData.format(new Date()) );

```

La servlet generata da Tomcat (III)

```
        out.write("\r\n");
        out.write("  </body>\r\n");
        out.write("</html>\r\n");
    } catch (Throwable t) {
        if (!(t instanceof SkipPageException)){
            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0)
                out.clearBuffer();
            if (_jspx_page_context != null)
                _jspx_page_context.handlePageException(t);
        }
    } finally {
        if (_jspxFactory != null)
            _jspxFactory.releasePageContext(_jspx_page_context);
    }
}
}
```

Tipi di tag JSP

- Sintassi: `<%...%>` o `<jsp:...>`
- Si dividono in tre categorie:
 1. **Direttive** (*directives*) `<%@ ... %>`
 2. **Elementi di script** (*scripting elements*)
`<%!...%>` `<%= ... %>` `<% ... %>`
 3. **Azioni standard, predefinite** (*standard, built-in actions*) `<jsp: ...>`

Tipi di tag JSP: direttive

- Le direttive JSP influenzano la struttura della servlet generata dalla pagina JSP.

- Sintassi:

```
<%@ nomeDirettiva attributo="valore" %>.
```

- Tipi di direttive:

- **page** (controlla la servlet, importando classi, definendo il content type ecc.);
- **include** (permette di inserire dei file nella pagina JSP);
- **taglib** (permette al programmatore di definire dei tag custom).

Tipi di tag JSP: scripting element

- *Espressioni* della forma **<%= espressione %>**, che vengono valutate ed il valore risultante finisce nell'output generato dalla servlet.
- *Scriptlet* della forma **<% codice %>**, che vengono inseriti nel metodo **_jspService** (chiamato da service).
- *Dichiarazioni* della forma **<%! dichiarazione%>**, che vengono inserite nel corpo della classe della servlet generata (esternamente ad ogni metodo).

Tipi di tag JSP: azioni standard

- Solitamente nello sviluppo di web application vi sono due figure:
 - designer HTML,
 - programmatore.
- Lo scopo delle azioni standard è quello di permettere ai designer HTML di scrivere pagine JSP senza dover scrivere codice Java.

Tipi di tag JSP: azioni standard

- Un tipico esempio di azione standard sono i tag che riguardano gli Enterprise Java Bean:
 - `<jsp:useBean id="name" class="package.Class" />`
 - `<jsp:setProperty>`
 - `<jsp:getProperty>`
- Altre azioni standard:
 - `<jsp:include>`
 - `<jsp:forward>`
 - `<jsp:plugin>`
 - `<jsp:param>`

La direttiva `page`

- Sintassi: `<%@ page attributo="valore"%>`
- Attributi di `page`:
 - **import**: permette di importare i package specificati alla servlet generata;
 - **contentType**: indica il tipo MIME del contenuto inviato al browser;
 - **isThreadSafe**: determina se la servlet risultante implementi o meno `SingleThreadModel`;
 - **session**: determina se alla pagina JSP debba essere associata una sessione;
 - **buffer**: dimensione del buffer da usare per l'output;
 - **autoflush**: indica se il buffer di output debba essere automaticamente svuotato una volta pieno oppure debba essere generata un'eccezione;
 - **extends**: indica la superclasse della servlet che verrà generata;
 - **info**: definisce una stringa che può essere recuperata tramite `getServletInfo()`;
 - **errorPage**: specifica una pagina JSP (tramite un URL) che gestisca le eccezioni;
 - **isErrorPage**: indica se la pagina può fungere da pagina di errore per un'altra JSP;
 - **language**: al momento Java è l'unica possibilità.

La direttiva `include`

- Sintassi: `<%@ include file="nomeFile"%>`
- E' possibile includere sia file statici (HTML) che altre JSP.
- Anche se l'inclusione avviene a tempo di compilazione, nel caso si includa un file `.jsp`, viene incluso il codice non l'HTML generato.
- Questa direttiva è utile per gestire in file separati, le parti di un'applicazione web che restano fisse (menu, banner ecc.).

Esempio di `include`

- La seguente JSP include il file della nostra pagina JSP precedente:

```
<html>
  <body>
    <p>Questa JSP include un'altra JSP.
    Inizio inclusione:
    <%@ include file="Data.jsp"%>
    Fine inclusione.</p>
  </body>
</html>
```

Dichiarazioni

- Sintassi: `<%! dichiarazioni di variabili e metodi%>`
- Gli elementi dichiarati in uno scripting element `<%! ... %>` sono disponibili nel resto della JSP.
- Le dichiarazioni non generano alcun flusso di output.

Scriptlet

- Sintassi: `<% Blocco di codice da eseguire %>`
- Il blocco di codice specificato viene eseguito ad ogni richiesta della servlet.
- Infatti esso verrà inserito all'interno del metodo **`service()`** della servlet corrispondentemente generata (più precisamente all'interno del metodo **`_jspService()`**, richiamato da **`service()`**).

Oggetti impliciti

- Il codice all'interno di uno scriptlet (`<% ... %>`) finirà “dentro” il metodo **service** di una servlet.
- Quindi avremo a disposizione i seguenti oggetti:
 - **out** (**PrintWriter**)
 - **request** (**HttpServletRequest**)
 - **response** (**HttpServletResponse**)
 - **config** (**ServletConfig**)
 - **application** (**ServletContext**)
 - **session** (**HttpSession**)

Esempio: un contatore di accessi

```
<html>
  <body>
    <%!
      int x = 0;
      int inc(int x) {return x+1;}
    %>
    <% x = inc(x); %>
    <p>Numero di accessi: <%= x %> </p>
  </body>
</html>
```

Contatore con sessione

```
<html>
  <body>
    <% Integer x = new Integer(1);
      if(session.isNew())
        session.setAttribute("contatore", x);
      else {
        x=(Integer) session.getAttribute("contatore");

        if(x == null)
          session.setAttribute("contatore", new Integer(1));
        else
          session.setAttribute("contatore",
                                new Integer(x.intValue()+1));
      }
    %>
    <p>Numero di accessi: <%= x.intValue() %> </p>
  </body>
</html>
```

Esempio: form e parametri (I)

```
<% String nome = request.getParameter("nome"); %>
<HTML>
<HEAD>
<TITLE>
  <% if(request.getParameter("invio") != null) { %>
    Ciao, <%= nome.equals("") ? "Sconosciuto" : nome %>!
  <% } else { %>
    Form
  <% } %>
</TITLE>
</HEAD>
```


Esempio: form e parametri (II)

```
<BODY>
  <% if(request.getParameter("invio") != null) { %>
    Ciao, <%= nome.equals("") ? "Sconosciuto" : nome %>!
  <% } else { %>
  <FORM METHOD="post" ACTION="Ciao.jsp">
    Qual &egrave; il tuo nome?
    <INPUT TYPE="text" NAME="nome">
    <P>
    <INPUT TYPE="submit" NAME="invio" VALUE="Invia &gt;&gt;">
  </FORM>
  <% } %>
</BODY>

</HTML>
```

Esercizio

- Provare a riscrivere come pagina JSP l'esempio della servlet Bilancio della lezione 4.
- Per implementare la thread safety provare sia con il metodo `synchronized` che con la direttiva `<%@ page isThreadSafe="false" %>`.