Le *JavaServer Pages*

A cura di Giuseppe De Pietro (depietro_giuseppe@yahoo.it)

Introduzione

La presente guida è rivolta ad utenti con buone conoscenze di HTML, CSS e mediocri conoscenze di un qualsiasi linguaggio di programmazione. In ogni caso ho cercato di mantenere un livello base adatto anche a chi è alle prime armi con la programmazione Web.
È preferibile comunque avere delle basi di Java o della programmazione ad oggetti per poter rielaborare gli esempi proposti.
Le prime lezioni sono per gli utenti inesperti (chi non conosce Java può comunque creare semplici applicazioni Web). Le lezioni successive sono studiate per coloro che hanno già un minimo di esperienza e vogliono sfruttare le notevoli potenzialità delle Web Application in Java.
In tanti casi ho evitato spiegazioni dettagliate (anche se spesso inevitabili) per non rendere ostici degli argomenti che potevano essere illustrati anche con spiegazioni sommarie. Ho omesso altre informazioni che, a chi è già esperto della tecnologia, potrebbero rendere la guida inesatta.
In futuro e, su vostra gradita segnalazione, prometto di arricchirla con informazioni aggiuntive e più particolareggiate.

Perchè JSP?

Ho scelto la tecnologia JSP perché è basata su Java e quindi permette lo sviluppo di applicazioni Web senza dover essere legati ad un ambiente in particolare. E' risaputo ormai di come il linguaggio di Sun Microsystems sia multipiattaforma (Windows, Linux, Unix, MacOS ed altri) offrendo i notevoli vantaggi di scrivere applicazioni portabili in tutti i sensi.
Inoltre è una tecnologia Open Source: possiamo costruirci il Web Server con il nostro sito spendendo solo i soldi per l'hardware (inevitabile), il dominio e la linea veloce e utilizzando come software Linux, Java 2 SDK, MySQL come Server di Database e Tomcat come Web Server.
Inoltre le applicazioni web in Java sono tra le più complete e potenti. Nella guida io farò riferimento solo ad alcuni aspetti delle WA (*Web Application*) solo per non complicare il discorso, ma esse in realtà rappresentano una validissima soluzione al problema della separazione tra logica gestionale e l'interfaccia grafica. Quest'ultimo aspetto è fondamentale nelle applicazioni più complesse, dove alla realizzazione di una WA collaborano più figure, dai web designer ai programmatori che potranno così suddividersi i compiti evitando i problemi di condivisione.

Lezione 1

Introduzione a JSP

A cura di Giuseppe De Pietro (depietro_giuseppe@yahoo.it)

Differenze tra tecnologie statiche e dinamiche

Ogni volta che con il nostro browser digitiamo l'URL della pagina desiderata, avviene una interazione tra il nostro computer (client) ed il Web Server remoto che ospita la risorsa richiesta.
In particolare il client si connette al server remoto ed invia la richiesta del documento. Il server verifica la disponibilità del documento e risponde al client comunicando l'esito della richiesta e successivamente inviando il file HTML desiderato. La comunicazione tra gli host avviene sfruttando il protocollo HTTP (*Hypertext Transfer Protocol*) come illustrato di seguito:

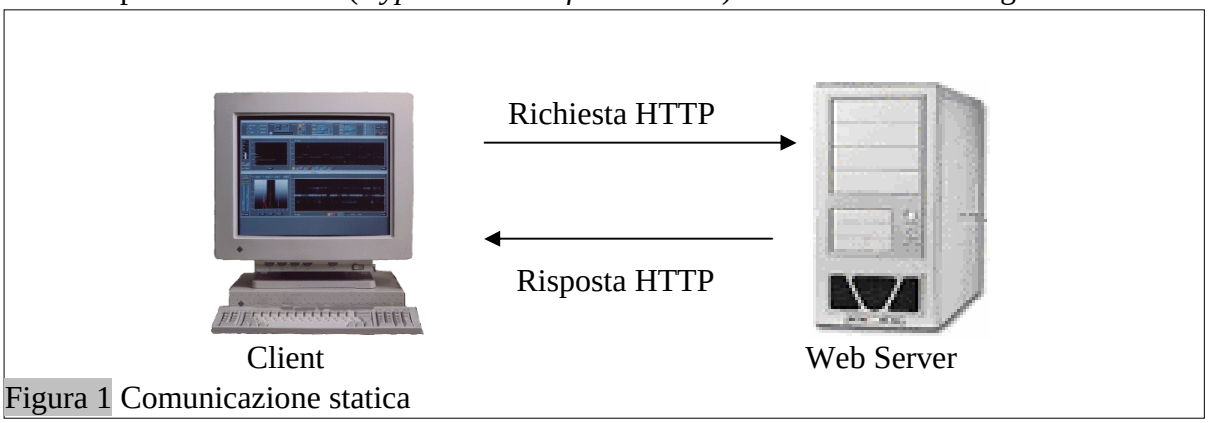

Figura 1 Comunicazione statica

Questo tipo di comunicazione viene definita **statica**, perché il compito del Web Server è solo quello di verificare l'esistenza delle pagine HTML e restituirle al client richiedente.
Ben presto ci si è resi conto che le sole pagine Web statiche non sono sufficienti a gestire servizi quali ricerche di dati, consultazioni on-line di archivi (prodotti, articoli e così via), acquisti effettuati sul Web. Si è passati quindi alle tecnologie Web dinamiche, in grado cioè di interpretare le diverse richieste dei client e di generare dinamicamente il risultato HTML.
Attualmente sono disponibili numerose tecnologie dinamiche, tutte molto valide:

- **CGI**: le *Common Gateway Interfaces* sono state le prime a comparire e di conseguenza presentano notevoli limitazioni prestazionali e di concorrenza (intesa come gestione degli accessi simultanei). Sono basati su applicativi compilati (redatti in C o in Perl) che girano sul server e rispondo alle richieste dei client.
- **ASP**: le *Active Server Pages* sono basate su script (VBScript o JavaScript) che vengono eseguiti sul Server e generano dinamicamente il contenuto HTML. Richiedono sistemi Windows e IIS (Internet Information Services) come Web Server per gestire le richieste.
- **ASP.NET**: evoluzione della tecnologia ASP. Si basa su codice VB.NET o C# che viene compilato sul Web Server IIS utilizzando le classi del Framework .NET. Grazie ad essa è possibile implementare i Web Services basati XML ed il protocollo SOAP.
- **PHP**: *Hypertext Preprocessor* è una tecnologia basata su codice scritto in PHP (un misto di C e Perl) e richiede Apache come web server. Può girare sia su sistemi Windows che Linux (anche se l'ambiente ideale è Linux).

- **JSP**: le Java Server Pages sono una componente fondamentale delle applicazioni Web in Java (tra le più complete e potenti tecnologie Web, vedi ***figura 2***). Fanno parte delle specifiche Java 2 Enterprise Edition (J2EE). Richiedono un *container* per poter essere eseguite che spesso funge anche da Web Server. Tra i più utilizzati c'è Tomcat di Jakarta (progetto Open Source, disponibile sia per Linux che per Windows) che può funzionare come Web Server standalone (autonomo), ma anche essere integrato in Apache.

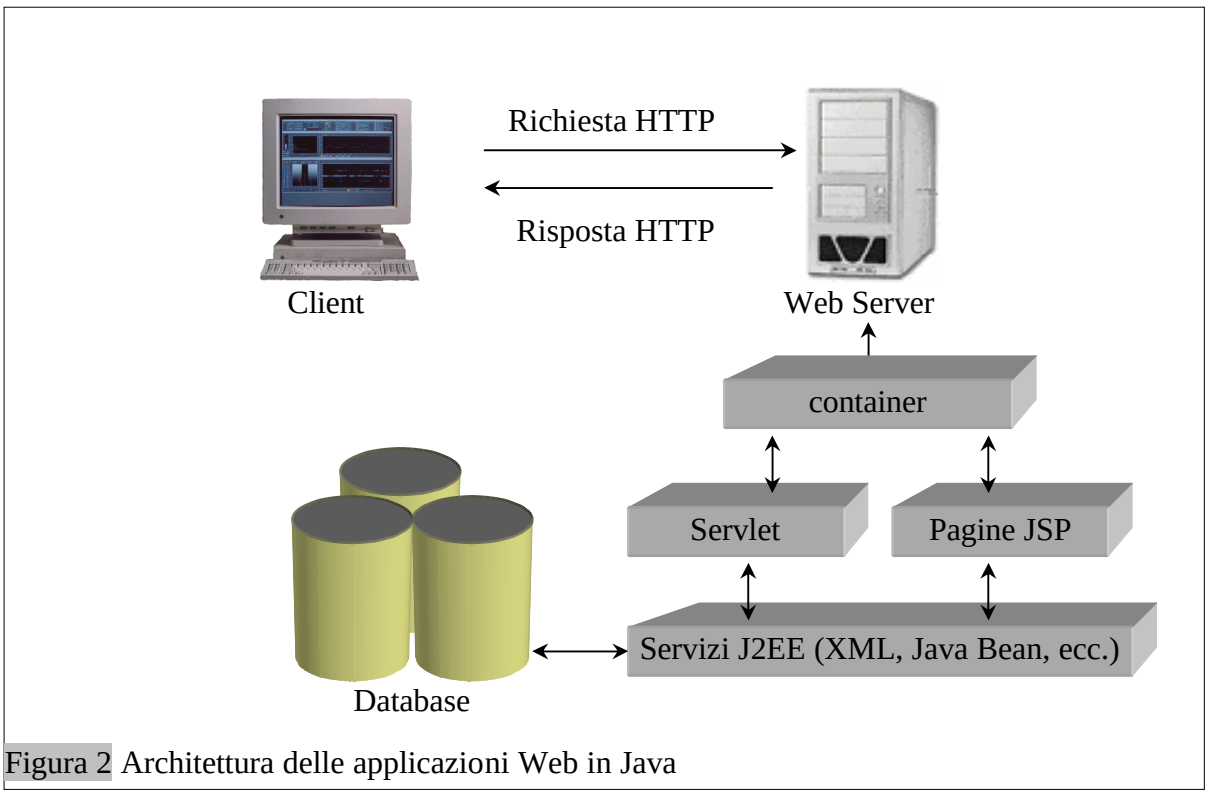

Figura 2 Architettura delle applicazioni Web in Java

Java Web Application, Servlet o pagine JSP?

Spesso si tende ad associare lo stesso significato ai tre termini. Chiariamo meglio i concetti.
Una **Java Web Application** è un insieme di Servlet, pagine JSP, componenti Bean, pagine HTML e fogli di stile CSS.
Una **Servlet** è una classe Java che utilizza il protocollo http per ricevere ed inviare delle richieste tra client e Server.
Una **pagina JSP** è una pagina che racchiude del codice statico (HTML) e del codice dinamico (Java) che verrà eseguito sul Server (ha la stessa struttura di una pagina PHP o ASP).
Le Java Web Application inizialmente si basavano solo sulle Servlet, offrivano comunque notevoli potenzialità ma erano una tecnologia alla portata dei programmatori più esperti (in una classe Java bisognava inserire sia il codice gestionale che il codice che curava il layout della pagina Web).
Successivamente furono introdotte le JSP per semplificare il lavoro dei web designer che potevano così dedicarsi al layout della pagina senza entrare in merito della logica di gestione dell'applicazione. Una pagina JSP infatti, dovrebbe contenere poco codice Java e definire solo il template HTML dell'applicazione.
Vediamo un primo semplice esempio. Realizziamo una pagina JSP di nome *hello.jsp* che visualizza la data corrente:

```
<%@ 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 JSP</h1>
   <br/>
   <%
   Calendar data = Calendar.getInstance();

   out.print("Oggi è il giorno "
data.get(data.DATE)+ "/"+(data.get(data.MONTH)+1)+"/"+data.get(data.YEAR));
    %>
  </body>
</html>
```

La maggior parte del codice è costituito da tag HTML, mentre la parte in Java è racchiusa tra <%..%>.
Anche per chi non sa nulla di Java può intuire il funzionamento della pagina, si crea una nuova istanza della classe *Calendar* (equivalente della classe *Data* che è stata deprecata nelle nuove specifiche J2SE), si prelevano il giorno, il mese (aggiungendo 1 perché Gennaio equivale a 0) e l'anno e si utilizza il canale *out* per inviare i dati al client.
Il client riceverà solo l'equivalente HTML della pagina:

```
<!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 JSP</h1>
   <br/>
     Oggi è il giorno 7/7/2004
  </body>
</html>
```

In realtà il container (nel nostro caso Tomcat 5.0.25) ha eseguito svariate operazioni in seguito alla richiesta del browser:

- ha convertito la pagina JSP in una Servlet (classe Java)
- ha compilato la classe in bytecode
- ha istanziato la classe nel servlet container
- ha esaudito la richiesta restituendo la risposta al browser.

Tutto questo accade solo alla prima richiesta della pagina, perché alle successive richieste Tomcat avrà già la pagina compilata ed in “memoria”, ottenendo quindi un notevole miglioramento delle prestazioni.
Le pagine JSP difatti sono molto performanti rispetto alle altre tecnologie (ASP e PHP) che ad ogni richiesta interpretano il codice lato server senza compilarlo.
Diamo ora un’occhiata al lavoro di traduzione che Tomcat ha svolto, generando la seguente classe:

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.lang.*;
import java.util.*;
public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
      implements org.apache.jasper.runtime.JspSourceDependent {
  private static java.util.Vector _jspx_dependants;
  public java.util.List getDependants() {return _jspx_dependants; }
  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;
      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();
        _jspx_out = out;
        out.write("\r\n\r\n\r\n\r\n\r\n\r\n<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML
1.0 Transitional//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd\">\r\n<html xmlns=\"http://www.w3.org/1999/xhtml\"
xml:lang=\"IT\">\r\n  <head>\r\n   \r\n    \r\n    <title>Prima pagina
JSP</title>\r\n    \r\n   \r\n    <meta http-equiv=\"description\"
content=\"Questa è la mia prima pagina JSP\"/>\r\n    \r\n    <!--\r\n    <link
rel=\"stylesheet\" type=\"text/css\" href=\"styles.css\">\r\n    -->\r\n
</head>\r\n  \r\n  <body>\r\n   <h1>Prima pagina JSP</h1>\r\n    <br/>\r\n
");
      Calendar data = Calendar.getInstance();
      out.print("Oggi è il giorno " + data.get(data.DATE)+
"/"+(data.get(data.MONTH)+1)+"/"+data.get(data.YEAR));
        out.write("\r\n  </body>\r\n</html>\r\n");
      } catch (Throwable t) {
        if (!(t instanceof SkipPageException)){
            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0) out.clearBuffer();
            if (pageContext != null) pageContext.handlePageException(t);
        }
      } finally {
        if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
      }
  }
}
```

Risulta evidente di come sia molto più semplice scrivere una JSP che una Servlet. Esaminando meglio il codice generato, notiamo come le istruzioni Java che noi abbiamo inserito e le istruzioni di scrittura dei tag HTML sono stati copiati nel metodo `_jspService()`. Nel corso delle successive richieste il *container* avrà già in memoria la servlet creata dalla pagina JSP e richiamerà sempre il metodo `_jspService()` sull'istanza conservata.
In conclusione, alla base di tutto ci sono comunque le classi Java, solo che anche un web designer senza troppa esperienza in programmazione Java, potrà realizzare le proprie applicazioni con uno sforzo minimo perché il *container* farà tutto il lavoro di conversione.
A questo punto si potrebbe pensare di realizzare la nostra applicazione solo con pagine JSP, certo per chi si cimenta per la prima volta in questa tecnologia potrebbe essere una soluzione valida, ma è nella realizzazione delle Servlet e dei componenti Bean che risiedono le maggiori differenze tra le Java Web Application e le altre tecnologie tradizionali come ASP o PHP.