Java Enterprise Edition

Gabriele Tolomei

DAIS – Università Ca' Foscari Venezia

Programma del Corso

- 09/01 – Introduzione
- 10/01 – Java Servlets
- 16-17/01 – JavaServer Pages (JSP)
- 23-24/01 – Lab: Applicazione “AffableBean”
- 30-31/01 – Enterprise JavaBeans (EJB) + Lab

Deployment: Memorandum

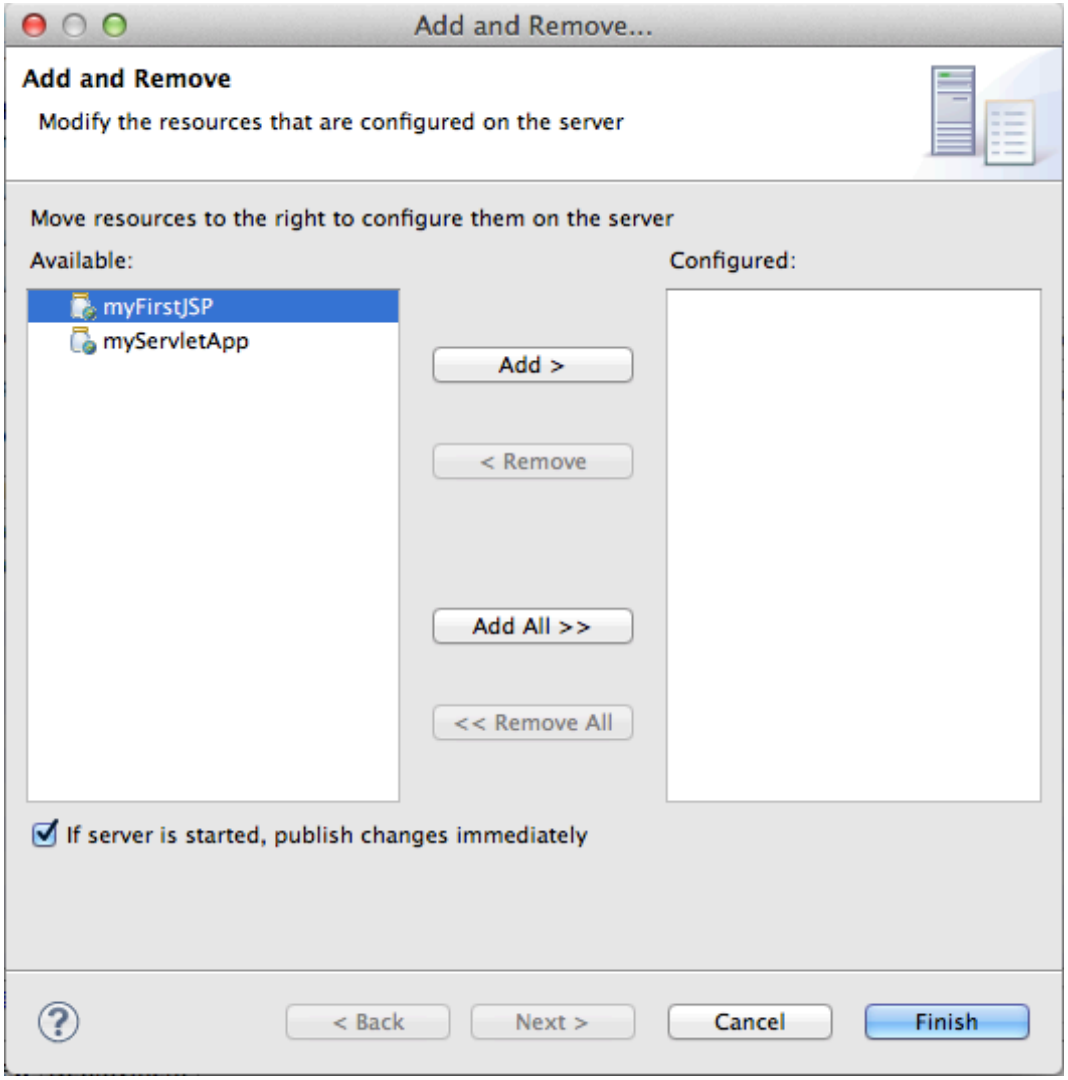
- Il server (ad es. JBoss) è stato avviato da **shell**
  - Eseguire il deployment copiando il file **.war** (eventualmente generato con Eclipse) nell'apposita directory del server (ad es. **$JBOSS_HOME/server/default/deploy/**)
  - La modifica di un componente software all'interno di Eclipse **non** causa il re-deployment automatico dell'applicazione (di default)
    - Il re-deployment deve essere eseguito rigenerando il file **.war** e copiandolo nell'apposita directory (come sopra)

Deployment: Memorandum

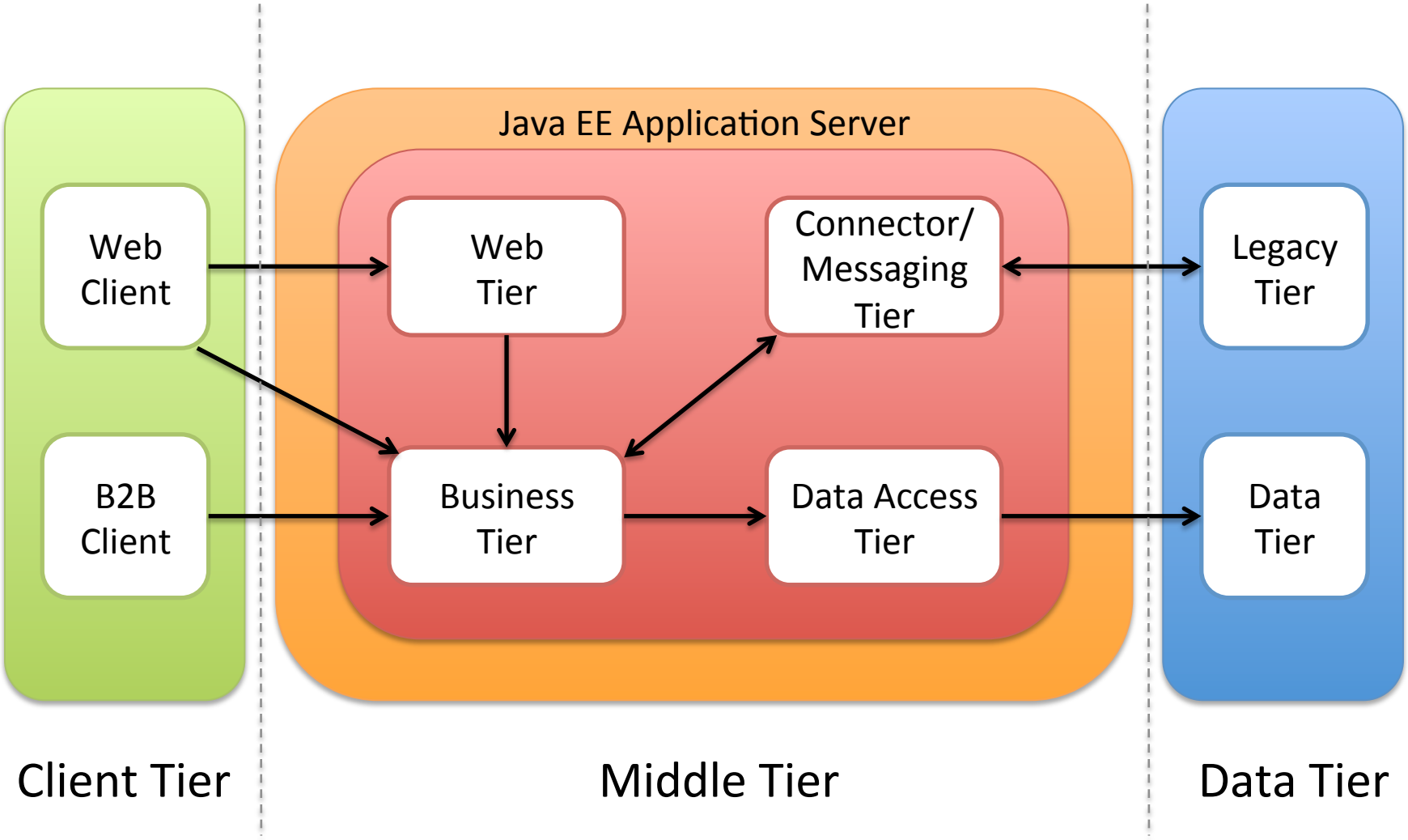
- Il server (ad es. JBoss) è stato avviato all'interno di **Eclipse**
  - Eseguire il deployment aggiungendo l'applicazione all'ambiente server di runtime nel tab "Servers"
  - Questo **non** causa la copia del file .war all'interno della directory di deploy del server (ad es. **$JBOSS_HOME/server/default/deploy/**)
  - Ogni modifica comporta il re-deploy automatico
  - Di default Eclipse salva i file **.war** in una propria directory interna al workspace

    ${workspace_dir}/.metadata/plugins/org.jboss.ide.eclipse.as.core/JBoss_5.1_Runtime_Server1387444525540/deploy

Deployment: Memorandum



Java EE: Architettura Multi-tier



Cos'è una JSP?

- Di fatto una pagina JSP è una Servlet!
- Oltre ai vantaggi delle Servlet offre:
  - Look-and-feel HTML (plain-text)
  - Facilità per i web designer
  - Sviluppo senza programmare in Java tramite l'uso di custom tags e expression language (EL)
  - Compilazione e deployment automatico da parte del Servlet container
- Si dovrebbe concentrare solo sul livello di "presentazione" (GUI)
  - La logica applicativa implementata altrove

Java Servlet vs. JSP (1)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class HelloWorldServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("\t<head><title>Hello World</title></head>");
    out.println("\t<body>");
    out.println("\t\t<p>Hello " + req.getRemoteAddr() + "!</p>");
    out.println("\t\t<p>Server time is " + new Date() + "</p>");
    out.println("\t</body>");
    out.println("</html>");
    out.close();
  }
}
```

Java Servlet vs. JSP (2)

```
<%@page import="java.util.Date" %>
<html>
  <head><title>Hello World</title></head>
  <body>
    <p>Hello <%=request.getRemoteAddr()%>!</p>
    <p>Server time is <%=new Date()%></p>
  </body>
</html>
```

JSP: Ciclo di Vita

- Simile a quello di una Servlet “classica”
- Se l’istanza della Servlet corrispondente alla JSP richiesta non esiste, il Servlet container:
  - Compila la JSP nella Servlet corrispondente
  - Carica la classe e la istanzia
  - Inizializza l’istanza della classe Servlet tramite il metodo jspInit
- Per ogni richiesta utente, il container invoca il metodo _jspService della Servlet associata alla JSP, passando gli oggetti request e response
- Se rimossa, il container invoca il metodo jspDestroy

JSP: Compilazione

- La compilazione da JSP a Servlet avviene in 2 passi:
  1. Il file .jsp viene compilato in un file .java dal compilatore del container
     - Jasper su JBoss/Tomcat
  2. Il file .java generato dal compilatore viene a sua volta compilato in un file .class dal compilatore Java standard (javac)
     - Ecco perché il Servlet container necessita dell'intero JDK e non della sola piattaforma JRE

JSP: Note sulla compilazione

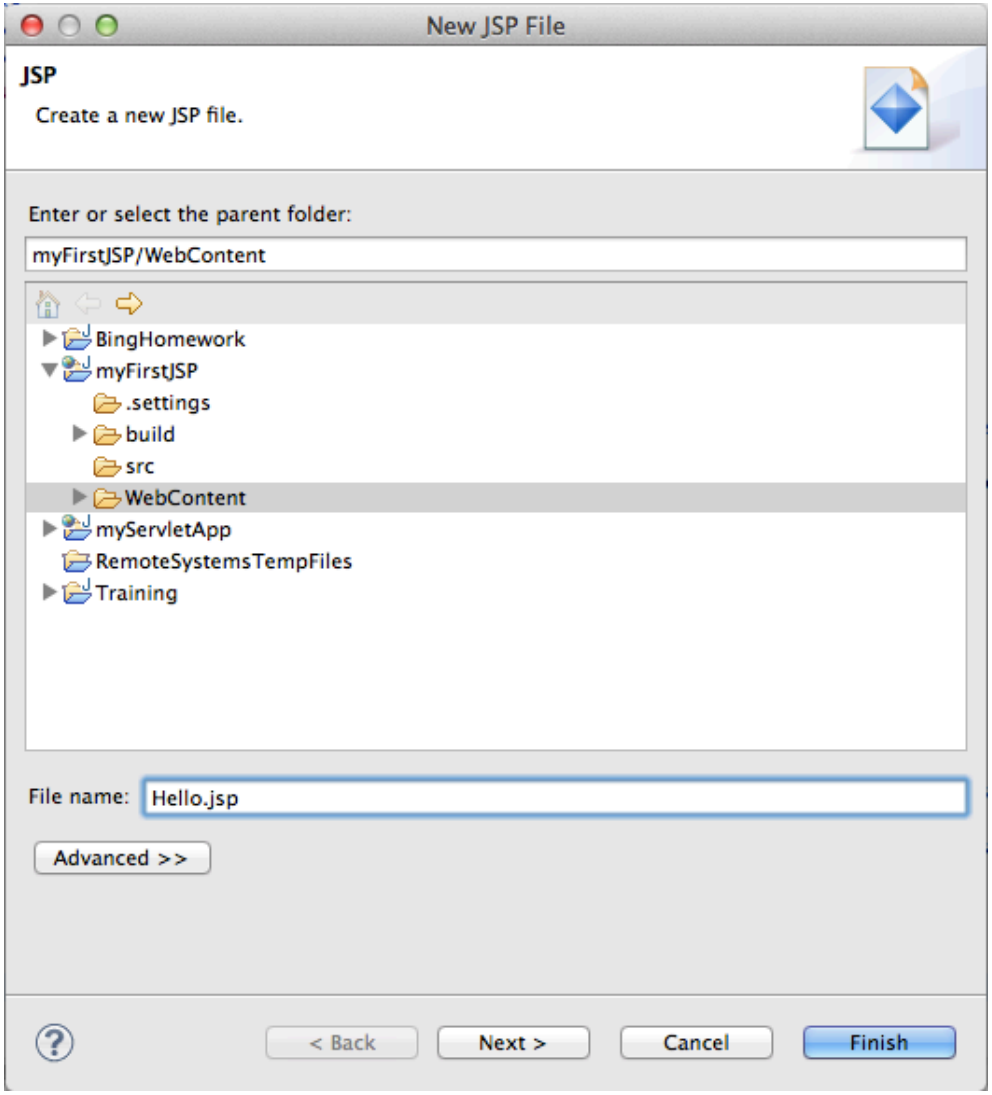
- La conversione .jsp → .java → .class avviene soltanto una volta o a fronte di una modifica del file .jsp
- I file generati (.java e .class) su JBoss si trovano in:

server/default/work/jboss.web/localhost/myapplication/org/apache/jsp/MyJSP_jsp.java

Lab: MyFirstJSP con Eclipse

- Creare un nuovo Dynamic Web Project
- Creare una JSP tramite Eclipse
- Packaging tramite WAR
- Deployment
- Testing

Lab: MyFirstJSP con Eclipse



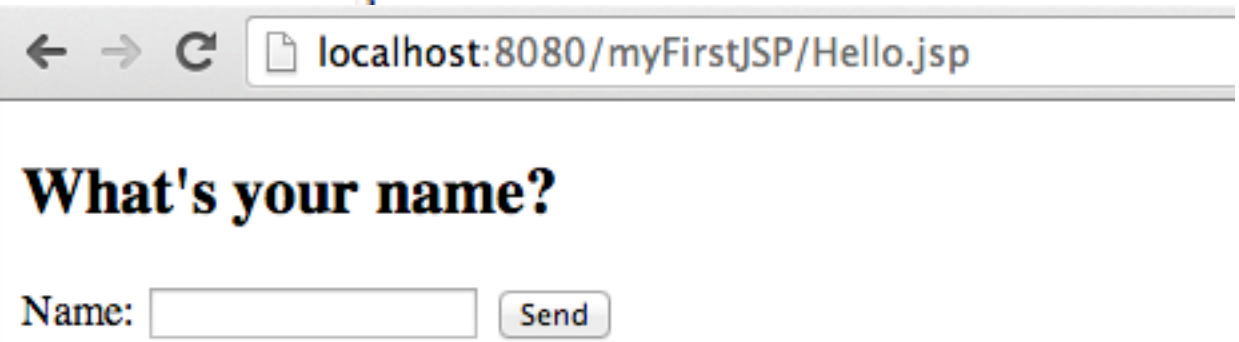
Lab: MyFirstJSP con Eclipse

Hello.jsp

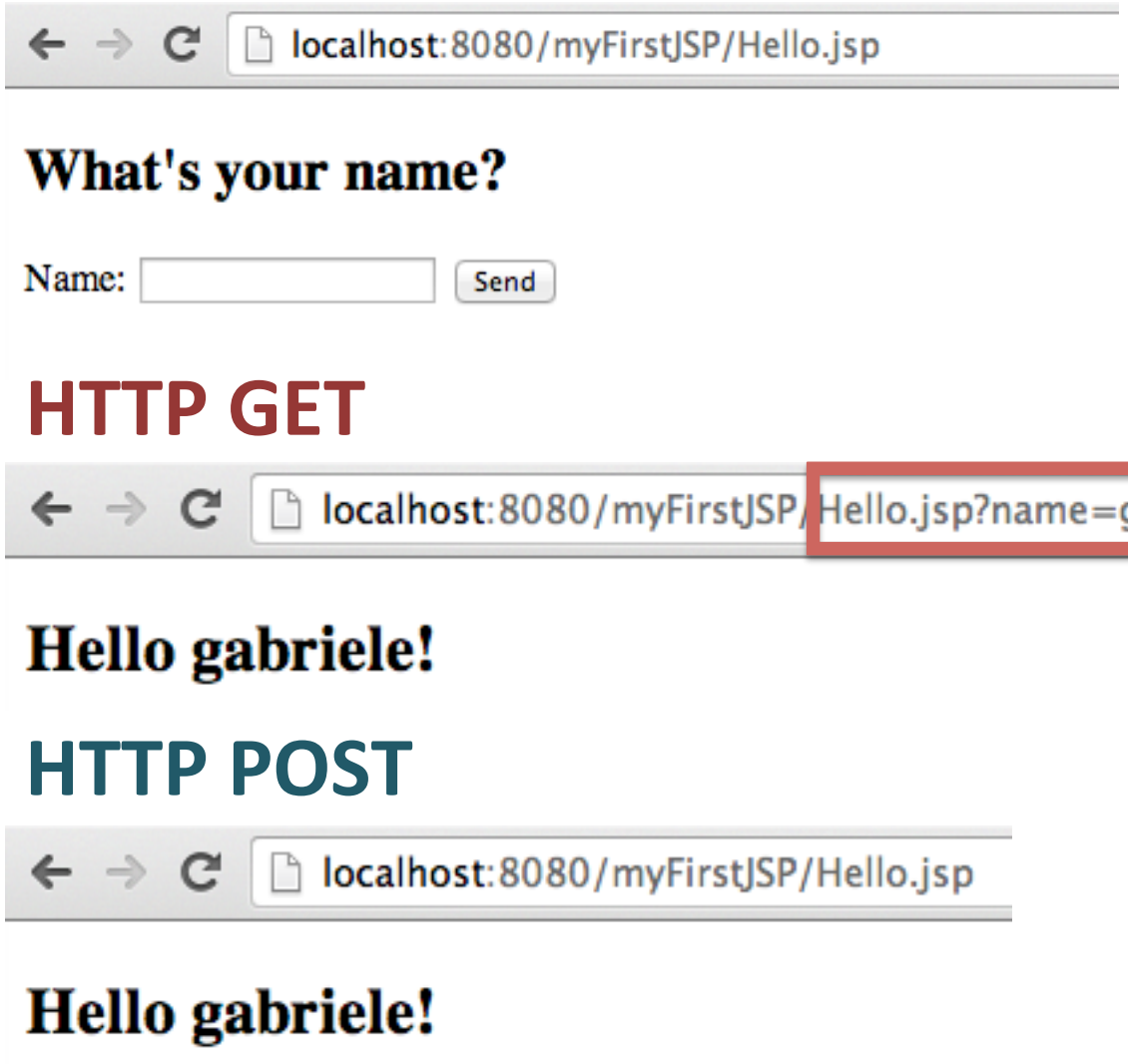
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello JSP</title>
</head>
<body>

</body>
</html>
```

Hello.jsp: Implementazione

```
<body>
<%
String name = request.getParameter("name");
if (name == null || name.length() == 0) {
%>
<form action="">
<h2>What's your name?</h2>
<p>Name: <input type="text" name="name"/>
    <input type="submit" value="Send"/>
</p>
</form>
<%
}
```



Hello.jsp: Deployment+Testing



Da JSP a Java Servlet: _jspService

```
...
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response) throws IOException, ServletException {
  ...
  out.write("\r\n");
  out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.w3.org
  out.write("<html>\r\n");
  out.write("<head>\r\n");
  out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");
  out.write("<title>Hello</title>\r\n");
  out.write("</head>\r\n");
  out.write("<body>\r\n");
  String name = request.getParameter("name");
  if (name == null || name.length() == 0) {
    out.write("\r\n");
    out.write("<form>\r\n");
    out.write("  <h2>What is your name?</h2>\r\n");
    out.write("  <p>Name: <input type=\"text\" name=\"name\" /> \r\n");
    out.write("  <input type=\"submit\"\tvalue=\"Send\" />\r\n");
    out.write("  </p>\r\n");
    out.write("</form>\r\n");
  } else {
    out.write("\r\n");
    out.write("<h2>Hello ");
    out.print(name);
    out.write("!</h2>\r\n");
  }
  out.write("\r\n");
  out.write("</body>\r\n");
  out.write("</html>");
  ...
}
...
```

JSP API

- Il contenuto della pagina JSP viene eseguito all'interno del metodo generato _jspService
- Tutti i contenuti statici sono convertiti in chiamate a out.write()
- Tutti contenuti dinamici (inclusi nei tag <% %>) vengono eseguiti come codice Java "normale"

JSP: Oggetti “impliciti”

- Il metodo generato automaticamente con la compilazione .jsp → .java definisce e inizializza alcuni oggetti
- Questi oggetti possono essere riferiti all’interno della stessa pagina JSP

```
HttpServletRequest request
HttpServletResponse response
PageContext pageContext
HttpSession session
ServletContext application
ServletConfig config
JspWriter out
```

JSP: Oggetti “impliciti”

```
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;

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

Sintassi JSP: Direttive

- Le direttive JSP controllano la compilazione da JSP a Servlet
- Non hanno alcun effetto sul comportamento a run-time
- Controllano solo la fase di generazione di codice

`<%@page attributes %>` or in XML as `<jsp:directive.page .../>`
`<%@include file="path" %>` or in XML as `<jsp:directive.include .../>`
`<%@taglib prefix="prefix" uri="uri" %>` or in XML as `<html xmlns:prefix="uri" ...>`

Sintassi JSP: Direttiva **Page**

`contentType` - specifies the content type (and encoding) used for the page response. Defaults to "text/html".
`session` - indicates that the page participates in a session. Defaults to true.
`buffer` - the size of the page buffer in KB, or none for no buffer. Buffering allows developers to change HTTP headers late in the request processing. Defaults to 8 KB.
`autoFlush` - tells the JSP to flush the page buffer when it fills. Defaults to true.
`errorPage` - designates the page to display if an exception occurs in the JSP page
`isErrorPage` - if set to true, gives the page access to the implicit exception variable. Defaults to false.
`import` - adds to the Java import list for the generated Java file. Can be comma-delimited and/or specified multiple times (each in its own page directive).
`extends` - changes the generated servlet's super class
`info` - gives a brief description for the page (servlet info)
`isThreadSafe` - tells the servlet container that multiple requests can concurrently access the page. Defaults to true.
`language` - sets the JSP script language. Defaults to java.

Sintassi JSP: Direttiva **Include**

- Include il contenuto di un altro file JSP a tempo di traduzione/compilazione
- Analogo ad editare in contenuto del file JSP incluso direttamente nel file JSP che lo include

Sintassi JSP: Direttiva **Taglib**

- Fornisce accesso a librerie di tag standard e custom
- Più avanti esamineremo JSTL

Sintassi JSP: Scriptlets

- Rappresentano porzioni di codice Java all'interno di una pagina JSP
- La sintassi per inserire scriptlets è la seguente:

  <% java_code %> oppure <jsp:scriptlet>…</jsp:scriptlet>
- Una volta tradotta la pagina JSP diventano parte del metodo _jspService
- Come qualsiasi porzione di codice Java:
  - tutti gli statement devono terminare con il ";"
  - tutte le parentesi devono essere bilanciate
- L'uso massiccio di scriptlet è sconsigliato poiché "snatura" il vero ruolo di una pagina JSP
  - interfaccia vs. logica applicativa

Sintassi JSP: Scriptlets (esempio)

```
...
<body>
  <% for (int i = 0; i < 10; i++) { %>
    <p>Hello world!</p>
  <% } %>
  <p>Bye!</p>
  <% System.out.println("Done processing"); %>
</body>
...
```

Sintassi JSP: Expressions

- Simili agli scriptlets ma più specifiche
- Il risultato di un'espressione è inviato allo stream che gestisce la risposta
- La sintassi è la seguente:

```
<%= java expr %>
```

oppure

```
<jsp:expression>java expr</jsp:expression>
```

- Le espressioni **non** devono essere terminate con ";" perché vengono convertite in

```
out.print(java expr);
```

Sintassi JSP: Expressions (esempi)

```
Timestamp is <%= System.currentTimeMillis() %>
Hello <%= request.getParameter("name") %>!
This is <%= true %>!
2 + 2 is <%= 2 + 2 %>
Max of a and b is <%= a > b? a : b %>.
```

Sintassi JSP: Declarations

- Usate per dichiarare **metodi** e **variabili** al di fuori dello *scope* del metodo _jspService
  - sia di istanza che statici
- La sintassi è la seguente:

<%! java decl %>

oppure

<jsp:declaration>java decl </jsp:declaration>

- Consentono il riuso del codice sebbene vi siano alternative migliori
  - custom tags e beans

Sintassi JSP: Declarations (esempio)

```
<%!
private static final String USERNAME_PARAM = "username";
private static final String DOMAIN_PARAM = "domain";
private String userAtDomain(String user, String domain) {
  return user + "@" + domain;
}
%>
...
<html>
 <head>...</head>
 <body>
  <p>
   Your email address is
   <%= userAtDomain(request.getParameter(USERNAME_PARAM), request.getParameter(DOMAIN_PARAM)) %>.
  </p>
 </body>
</html>
```

Sintassi JSP: Comments

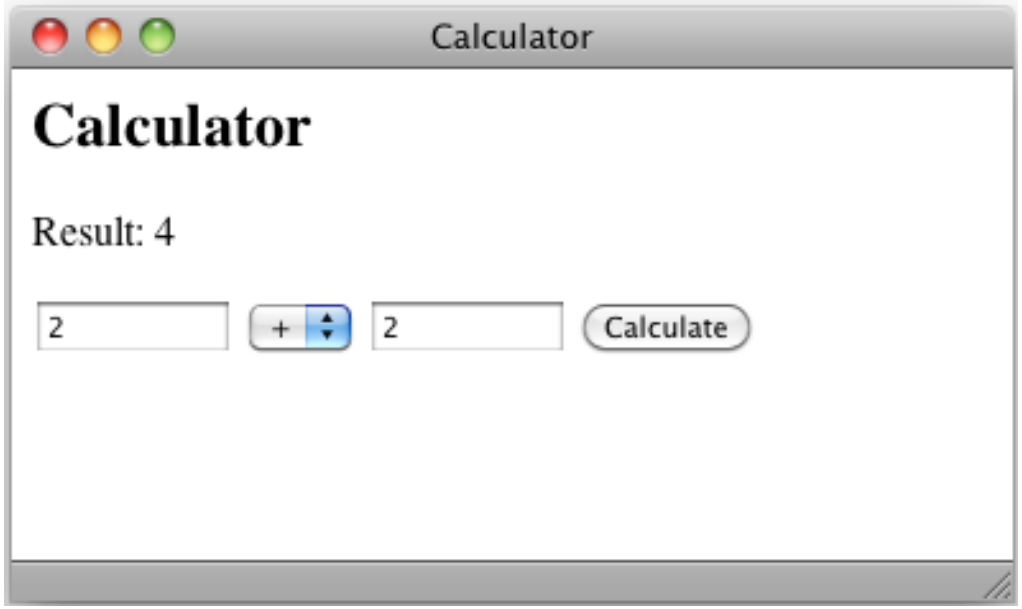
- Usati per commentare porzioni di pagina JSP
- Sia contenuti statici

  <% -- comment --%>
- Sia contenuti dinamici

  <% -- Hello <%= getUser() %> --%>
- Non ci possono essere commenti annidati
- **NOTA**: I commenti Java tradizionali si possono usare all'interno degli scriptlets, espressioni (tranne "//") e le dichiarazioni

Lab: Calcolatrice JSP

- Creare una Calcolatrice attraverso una pagina JSP che prende **2 argomenti numerici** ed **uno tra 4 possibili operatori** (**+,-,*,/**) e produce il risultato corrispondente



Sintassi JSP: Actions

`<jsp:include page="path"/>` - includes the contents of a local file during runtime
`<jsp:forward page="path"/>` - forwards the request to another page (internal redirect)
`<jsp:useBean id="name" ...>` - creates a new bean and variable for the page
`<jsp:getProperty name="name" .../>` - prints a bean property
`<jsp:setProperty ... value="val" | param="param"/>` - sets a bean property

```
<jsp:useBean id="beanName" class="ClassName"
scope="page|request|session|application"/>
```

- Same effect as: `<% scope.setAttribute("beanName", new ClassName()); %>` with the exception that beanName is first checked for existence

```
<jsp:getProperty name="beanName" property="propertyName"/>
```

- Same effect as: `<%= beanName.getPropertyName() %>`

```
<jsp:setProperty name="beanName" property="propertyName" value="propertyValue"/>
```

- Same effect as: `<% beanName.setPropertyName(propertyValue); %>`

```
<jsp:setProperty name="beanName" property="propertyName" param="paramName"/>
```

- Same effect as: `<% beanName.setPropertyName(request.getParameter(paramName)); %>`

```
<jsp:setProperty name="beanName" property="*"/>
```

- has the effect of invoking the following code for each property of the bean for which there exists a corresponding request parameter:
  `beanName.setPropertyX(request.getParameter("propertyX"));`

Sintassi JSP: Tag Libraries

- HTML può essere inserito all'interno di Java Servlets
- Analogamente il codice Java può essere inserito all'interno di pagine JSP
- **Desiderata**: Più separazione tra contenuti statici e dinamici!
- **Soluzione**: Incapsulamento di codice Java all'interno di specifici tag XML

Sintassi JSP: Tag Libraries (2)

- La sintassi per i tag XML è la seguente:

<prefix:tag attr1="val1"… attrN="valN">…</prefix:tag>

- Si possono usare tag "standard" definiti da JSTL oppure definirsene di propri
- I tag sono distribuiti in cosiddette tag libraries che contengono gli oggetti necessari ad implementare i tag in esse definite

Sintassi JSP: Tag Libraries (3)

- Prima che i tag possano essere usati devono essere dichiarati nel/i file JSP
- La sintassi per la dichiarazione è la seguente:

<%@ taglib prefix="pre" [tagdir="/WEB-INF/tags/dir" | uri="URI"] %>

- Il prefisso (prefix) deve essere univoco per ogni tag definito nella tag library di una pagina
- Gli attributi tagdir o uri devono riferirsi ai tag library descriptors (TLD)

Sintassi JSP: Tag Library Descriptors (TLD)

Note

For example: `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>` `<%@ taglib prefix="my" uri="/my.tld" %>` `<%@ taglib prefix="custom" tagdir="/WEB-INF/tags/custom" %>` As mentioned before, tag libraries can also be imported using the XML name-space syntax: `<html xmlns:c="http://java.sun.com/jsp/jstl/core"> ... </html>`

JSP Standard Tag Library (JSTL)

- Include funzionalità comuni a molte JSP
- Fornisce tags:
  - per il controllo del flusso delle pagine
    - if, choose, forEach, redirect, etc.
  - per la formattazione che supporta I18N
  - per la manipolazione di documenti XML
  - per l'accesso a RDBMS

JSTL: Esempio

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<html xmlns="http://www.w3.org/1999/xhtml">
...
<body>
<c:choose>
  <c:when test="<%=request.getParameter("name")==null%>">
  <form>
    <h2>What is your name?</h2>
    <p>
      Name: <input type="text" name="name"/>
            <input type="submit" value="Send"/>
    </p>
  </form>
  </c:when>
  <c:otherwise>
    <h2>Hello <%=request.getParameter("name")%>!</h2>
  </c:otherwise>
</c:choose>
</body>
</html>
```

Supporto JSTL

- NOTA: JBoss fornisce “nativamente” il supporto JSTL
- Altri ambienti (ad es. Tomcat) non contengono le librerie JSTL, in questo caso è necessario:
  - Scaricare l’implementazione JSTL, ad esempio: http://download.java.net/maven/1/jstl/jars/jstl-1.2.jar
  - Copiare il file jstl.jar dentro la directory WEB-INF/lib/ del progetto
  - Dichiarare le tag libraries usate nelle corrispondenti pagine JSP
  - Re-packaging + Re-deployment

JSP Expression Language (EL)

- Linguaggio per l'accesso ai dati memorizzati in JavaBeans o Map usato al posto delle espressioni viste in precedenza

  ${user.name} vs. <%= user.getName() %>

- Supporta aritmetica e logica

  ${2 + 2 > 3}

- Supporta il recupero di informazioni annidate

${user.name.first} vs. <%= user.getName().getFirst() %>

JSP Expression Language: Esempio

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
...
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
...
<title>Hello ${param.name}</title>
</head>
<body>
<c:choose>
  <c:when test="${empty param.name}">
  <form>
    <h2>What is your name?</h2>
    <p>
      Name: <input type="text" name="name"/>
            <input type="submit" value="Send"/>
    </p>
  </form>
  </c:when>
  <c:otherwise>
    <h2>Hello ${param.name}!</h2>
  </c:otherwise>
</c:choose>
</body>
</html>
```

JSP Expression Language: Operatori

- Aritmetici:

  +, -, *, /, %, mod

- Logici:

  and, &&, or, ||, not, !, ==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge, empty

- Condizionali:

  a ? b : c

JSP Expression Language: Oggetti “impliciti”

- **pageContext, servletContext, session, request, response**
- **param, paramValues, header, headerValues, cookie, initParam, pageScope, requestScope, sessionScope, applicationScope**