Laboratorio di Basi di Dati

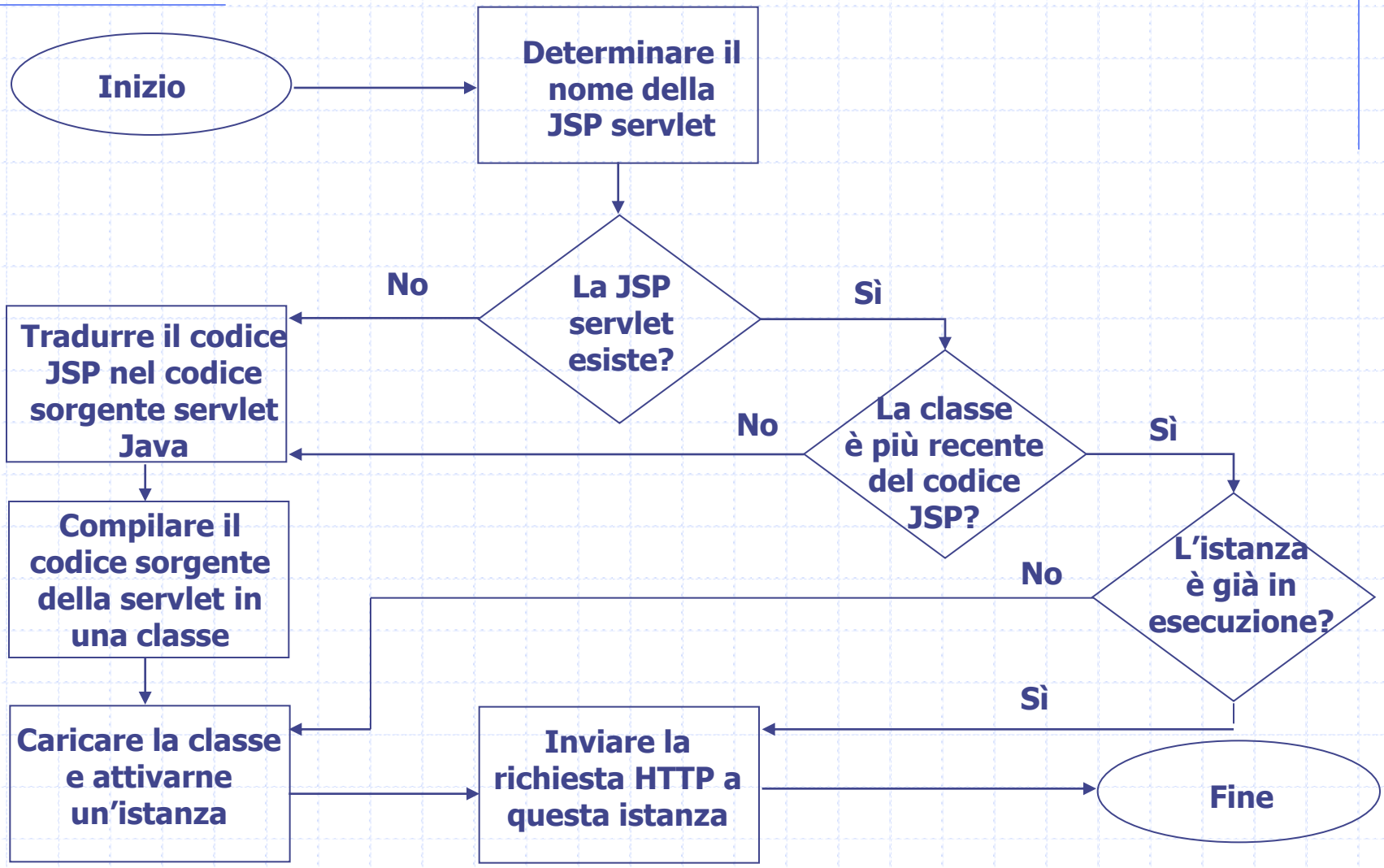
Docente: Alberto Belussi

Lezione 8

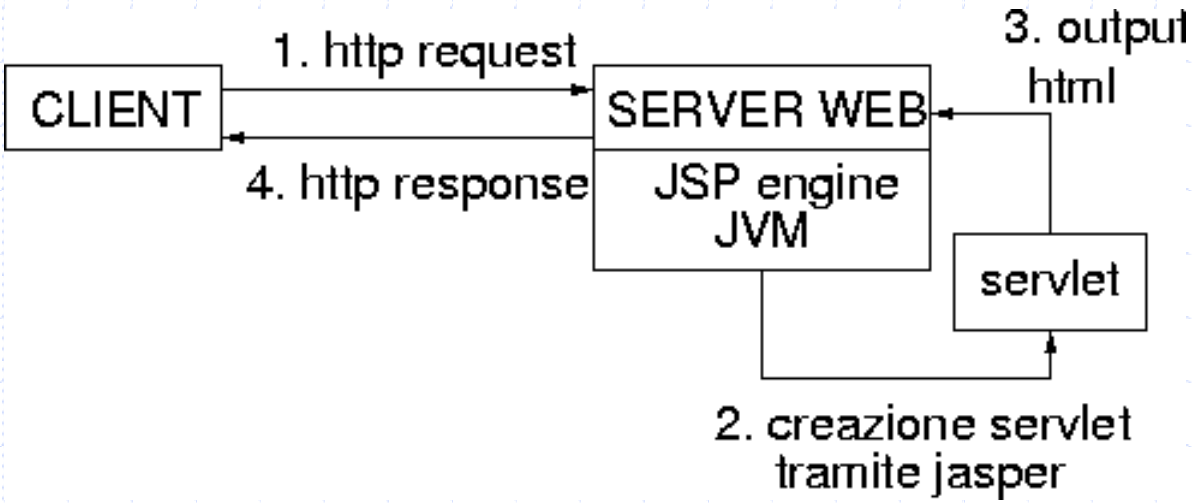
Java Server Pages

- Java Server Pages è essenzialmente un modo più "comodo" per il programmatore dell'applicazione web di scrivere una Servlet.
- Consente di separare il codice che genera le parti dinamiche della pagina dagli aspetti di presentazione.
- Una pagina JSP può quindi essere vista come uno schema di pagina Web che utilizza codice Java per generare le sue parti dinamiche e HTML per le parti statiche.
- Le pagine JSP vengono eseguite in un componente operante sul server chiamato *container JSP* che le traduce nelle corrispondenti servlet Java.
- Riferimento: http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Funzionamento JSP



Architettura JSP in Tomcat



- ogni pagina JSP (solitamente in webapps/*nomecontext*/) viene trasformata dal compilatore jasper in una servlet
- la servlet viene generata automaticamente ed il bytecode prodotto viene posto in un'opportuna sottodirectory di work/
- il compilatore jasper, per come è stato configurato in Tomcat (file server.xml), ricompila le pagine jsp quando necessario
- il log del processo viene salvato in logs/localhost_log.*data*.txt

Sintassi JSP

Il 'linguaggio' JSP fornisce 4 gruppi principali di marcatori speciali:

- Direttive
- Scripting:
  - Dichiarazioni
  - Espressioni
  - Scriptlet
- Azioni
- Commenti

Direttive

- Le direttive non influenzano la gestione di una singola richiesta HTTP ma influenzano le proprietà generali della JSP e di come questa deve essere tradotta in una servlet.
- Una direttiva è introdotta con il tag

  <%@ *tipo* ...%>

  o il suo equivalente XML <jsp:directive.*tipo* .../>.
- Direttive principali
  - <%@page attributi %>

    Attributi significativi
    - ***import***: consente di precisare classi o package da importare per la compilazione della jsp sevlet
    - ***errorPage***: consente di precisare l'URI di una pagina jsp da invocare in caso di errore
    - ***isErrorPage***: indica se la pagina è una pagina di gestione dell'errore o no. Se true consente l'accesso all'oggetto implicito exception
  - <%@include ***file***="nomeFile" %>

Scripting

- È il gruppo dei marcatori che permettono di inserire del codice all'interno di un documento.
- Un elemento di scripting è introdotto con uno dei seguenti tag: <%!...%> o <%=...%> o <%...%> che individuano i seguenti tipi di scripting:
  - Dichiarazione <%!...%>
  - Espressione <%=...%>
  - Scriptlet <%...%>

Scripting: Dichiarazione

<%! dichiarazione**;** [dichiarazione**;**]* ... %>

- Si inseriscono dichiarazioni di **variabili di classe** (comuni a più istanze) o **metodi statici** (possono essere chiamati senza richiedere l'accesso ad una istanza, ad esempio: nomeClasse.nomeMetodo())

- Esempio:

<%! private int x = 4; %>

Scripting: Espressione

<%= espressione %>

oppure

<jsp:expression> espressione </jsp:expression>

- Viene valutata l'espressione JAVA e il risultato viene sostituito al tag <%= espressione %> nella pagina HTML generata.
- Esempio:

<%= bean.getMatricola() %>

- Questo tag viene sostituito nella pagina HTML con il valore della proprietà Matricola contenuta nel bean.

Scripting: Scriptlet

<% codice su una o più linee %>

oppure

<jsp:scriptlet> espressione </jsp:scriptlet>

- Frammento di codice Java che può modificare anche il flusso del codice HTML generato.
- Solitamente gli operatori condizionali (if, ?, ..) ed i cicli (for, while, ..) possono essere utilizzati per produrre dinamicamente porzioni diverse di codice HTML in funzione dei parametri della richiesta HTTP o dei dati estratti dalla base di dati.
- Tale codice diventerà parte dei metodi doGet (doPost) della servlet che viene associata la JSP.

Azioni

- Questi marcatori permettono di supportare diversi comportamenti della pagina JSP.
- Vengono processati ad ogni invocazione della pagina JSP.
- Permettono di trasferire il controllo da una JSP all'altra, di interagire con Java Data Beans, ecc.
- Un'azione è introdotta con un tag del tipo:

  <jsp:*tipoAzione*.../>

- Ad esempio, se si vuole all'interno di una JSP, includere dinamicamente un'altra JSP, è sufficiente inserire nel punto dove si vuole includere l'altra JSP l'azione:
  <jsp:include page="*localURL*" flush="true"/>

Commenti

- Questi marcatori permettono di inserire diversi tipi di commenti all'interno delle JSP.
- Ci sono tre tipi di commenti:
  - <!-- commenti di contenuto -->: sono i tipici commenti di HTML.
  - <%-- commenti JSP --%>: sono i commenti visibili solo nel sorgente della JSP. (La servlet corrispondente non conterrà nulla di questi commenti).
  - <% /* commenti di scripting */ %>: sono i commenti all'interno della parte di codice della JSP e sono visibili anche nella servlet corrispondente.

Oggetti impliciti

- Ogni pagina JSP rende disponibile un insieme di oggetti che possono essere utilizzati all'interno della pagina.
- Questi *oggetti impliciti* sono accessibili sia attraverso azioni specifiche sia attraverso elementi di scripting.
- Oggetti più usati: request, page, exception.

Esempio di JSP: hello.jsp

```
<%-- hello.jsp stampa il classico saluto --%>
<%! static private String str = "world!";%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/REC-html40/loose.dtd">

<html>
    <head>
        <title>Hello!</title>
    </head>
    <body>
        <h1>Hello world!</h1>
        <b>Hello, <%= str.toUpperCase() %> </b>
    </body>
</html>
```

Esempio di JSP: hello.jsp (2)

- Supponiamo di memorizzare questa JSP nel context ROOT/
- La prima volta che viene invocata questa JSP, tramite l'URL http://localhost:8080/hello.jsp, Tomcat invoca il compilatore Jasper che produce la servlet equivalente ~/tomcat/work/Catalina/localhost/_/org/apache/jsp/hello_jsp.java

Servlet hello_jsp.java (1)

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class hello_jsp extends HttpJspBase {
    static private String str = "world!";
    public hello_jsp( ) { }
    private static boolean _jspx_inited = false;
    public final void _jspx_init()
            throws org.apache.jasper.runtime.JspException { }
    public void _jspService(HttpServletRequest request,
                            HttpServletResponse response)
            throws java.io.IOException, ServletException {
```

Servlet hello_jsp.java (2)

```
JspFactory _jspxFactory = null;
PageContext pageContext = null;
HttpSession session = null;
ServletContext application = null;
ServletConfig config = null;
JspWriter out = null; Object page = this;
String _value = null;
try {
        if (_jspx_inited == false) {
          synchronized (this) {
                if (_jspx_inited == false) {
                  _jspx_init();
                  _jspx_inited = true;
                }
          }
        }
```

Servlet hello_jsp.java (3)

```
_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();
out.write("\r\n\r\n<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01
          Transitional//EN\"\r\n\t \"http://www.w3.org/TR/REC-
          html40/loose.dtd\">\r\n\r\n<html>\r\n<head>\r\n <title>\r\n
          Esempio\r\n </title>\r\n</head>\r\n\r\n<body>\r\n <h1>Hello
          world!</h1>\r\n <b>Hello, ");
out.print( str.toUpperCase() );
out.write("</b>\r\n</body>\r\n</html>\r\n");
} catch (Throwable t) {
```

Servlet hello_jsp.java (4)

```
                    if (out != null && out.getBufferSize() != 0)
                            out.clearBuffer();
                    if (pageContext != null)
                            pageContext.handlePageException(t);
            } finally {
                    if (_jspxFactory != null)
            _jspxFactory.releasePageContext(pageContext);
            }
        }
    }
```

Esecuzione della servlet hello_jsp.java: risultato HTML

```
<!DOCTYPE HTML
    PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/REC-html40/loose.dtd">

<html>
    <head>
        <title>Hello!</title>
    </head>

    <body>
        <h1>Hello world!</h1>
        <b>Hello, WORLD!</b>
    </body>
</html>
```

Gestione degli errori

- Quando si verifica un errore nell'esecuzione di una jsp (ovvero nell'esecuzione della servlet corrispondente), il container inoltra il controllo ad una specifica pagina JSP alla quale viene fornito l'accesso all'oggetto implicito exception.
- Ogni pagina jsp definisce qual è la pagina di errore a cui inoltrare il controllo in caso di errore tramite la direttiva page:

```
<%@ page errorPage="/jsp/error.jsp"
         isErrorPage="false" %>
```

- Mentre un file JSP che DEVE gestire errori (error.jsp), conterrà la seguente direttiva:

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

Gestione degli errori: esempio

- Supponiamo di voler scrivere una pagina JSP che produce il risultato della divisione di due interi dati come parametri d'input.
- Se non si vuole gestire direttamente il caso di divisione per 0, ma si vuole sfruttare il meccanismo dell'eccezione, allora è necessario creare anche una pagina JSP per la gestione delle eccezioni.
- Avremo quindi un file div.jsp ed un file error.jsp

File div.jsp

```
<%--div.jsp Esegue una divisione tra due numeri dati in input --%>
<%@ page errorPage="error.jsp" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
    <head>
        <title>Esempio di jsp con gestione errori </title>
    </head>
    <body>
        <% String s;
        int x = ((s=request.getParameter("x"))==null) ? 10 : Integer.parseInt(s);
        int y = ((s=request.getParameter("y"))==null) ? 10 : Integer.parseInt(s);
        %>
        <h1>Divisione intera</h1>
        <pre> <%= x %>/<%= y %> = <%= x/y %> </pre>
        </body>
</html>
```

File error.jsp

```
<%@ page isErrorPage="true" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
    "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
    <head>
        <title>Errore!</title>
    </head>
    <body>
        <h2>Pagina per il trattamento dell'errore:</h2>
        <h3>È stata generata la seguente eccezione:</h3>
        <pre> <%= exception %> </pre>
        <hr> L'oggetto <code>exception</code> ha i seguenti metodi:
        <h4>getMessage()</h4> <%= exception.getMessage() %>
        <h4>printStackTrace(out)</h4>
        <pre> <% exception.printStackTrace(new java.io.PrintWriter(out)); %> </pre>
    </body>
</html>
```

Esempi da scaricare (1)

1. Scaricare nella directory ~/tomcat/webapps/ROOT il pacchetto EsempiJSP.zip dalla pagina web del modulo di Laboratorio di Basi di Dati e Web.
2. Scompattare il pacchetto: unzip EsempiJSP.zip.
3. Si otterranno le JSP hello.jsp, div.jsp e error.jsp nel contesto ROOT.
4. Per vedere la pagina web prodotta dalla JSP hello.jsp accedere via browser alla pagina: http://localhost:8080/hello.jsp
5. Provare anche la jsp div.jsp con parametri x e y corretti e con y=0. http://localhost:8080/div.jsp

Esempi da scaricare (2)

1. Scaricare nella directory ~/tomcat/webapps/CorsoStudi la JSP Corsi.jsp dalla pagina web del corso.
2. La JSP Corsi.jsp consente la visualizzazione dei corsi di studio dell'ateneo e dei dati che descrivono un singolo corso di studi.
3. Modificare il file .bashrc inserendo la riga seguente:

   CLASSPATH=$CLASSPATH:~/tomcat/webapps/CorsoStudi/WEB-INF/classes
4. Riaprire una nuova shell e riavviare Tomcat.

Esempi da scaricare (3)

- Occorre inoltre completare il codice della JSP utilizzando con opportune modifiche ed estensioni il frammento di codice JAVA presente nella servlet ServletCorsoStudi.java vista nella precedente esercitazione 7.
- Per vedere la pagina web prodotta dalla JSP: http://localhost:8080/CorsoStudi/Corsi.jsp

Esempio da scaricare (4)

L'applicazione che si sta costruendo costituisce un esempio di **MVC page-centric**.

Infatti la JSP Corsi.jsp viene invocata direttamente dal browser e gestisce sia il controllo di flusso che la presentazione, mentre la logica è delegata alla classe Java dbms.

Consegna Esercitazione 8

Inviare via email al docente i file di nome:

"ES8-<matricola>.jsp"

e

"ES8-<matricola>.java"

Contenenti rispettivamente la JSP Corsi.jsp e la classe JAVA DBMS.java completate secondo quanto richiesto dall'esercizio 8.

Il messaggio dovrà soddisfare il seguente formato:

- Oggetto: <Matricola> - Esercitazione 8
- Contenuto: <Matricola> - <Cognome> - <Nome>
- Allegati: file di nome: ES8-<Matricola>.jsp e ES8-<Matricola>.java

Il messaggio email va spedito entro le 24.00 del giorno 21 maggio 2013.

Riferimenti

- Marty Hall. “CORE. Servlets and JavaServer Pages”. Sun Microsystems Press.
- Phil Hanna. “JSP. La guida Completa.” McGraw-Hill.