Java

progettazione di applicazioni web -- jsp

G. Prencipe
prencipe@di.unipi.it

JavaServer Pages – JSP

- Le pagine JSP sono normali documenti testuali
- Contengono un mix di markup standard (es., HTML) e di istruzioni speciali
- Concettualmente simili alle pagine ASP (ma più potenti e portabili)

JSP e Servlet

- Le Servlet funzionano bene per pagine con tanto contenuto e poca "grafica"
- Però *tutta* la pagina deve essere generata in Java...
  - ...bisogna mettere mano al codice Java anche solo per cambiare il colore di sfondo!
- JSP (come ASP) immerge codice (Java) in pagine HTML
  - Separazione fra *presentazione* e *logica*

JSP

- Le JSP hanno lo stesso scopo delle servlets (generare pagine HTML con contenuto dinamico)
- Non sono programmi Java, ma documenti HTML con all'interno codice Java fornito tramite tags in stile HTML
- In questo modo la responsabilità della scrittura di pagine web dinamiche torna a chi gestisce le pagine (e non va ai programmatori)
- Inoltre viene eliminata tutta la parte noiosa di scrittura di **out.println** che è necessaria nelle servlet per generare codice HTML

Esempio: una pagina JSP

```
<HTML>
<%@ page language="java" imports=="com.wombat.JSP" %>
<H1>Benvenuto!</H1>
Oggi è
<jsp:useBean id=="calendario" class=="JSPCalendar" />
<UL>
<LI>giorno: <%== calendario.getDayOfMonth() %>
<LI>mese: <%== calendario.getMonth() %>
<LI>anno: <%== calendario.getYear() %>
</UL>
<% if (calendario.get(Calendar.AM_PM)==Calendar.AM) { %>
buon giorno!
<% } else { %>
buona seeeeeraaaaa...
<% } %>
</HTML>
```

Elementi di una pagina JSP

- Una pagina JSP può contenere cinque tipi di elementi:
  1. direttive JSP, fra **<%@** e **%>**
  2. dichiarazioni, fra <%! e %>
  3. espressioni, fra **<%==** e **%>**
  4. scriptlet, ovvero codice fra **<%** e **%>**
  5. commenti, fra <%-- e --%>
  6. azioni JSP, fra **<jsp:** e **/>**
  7. parti fisse, tutto il resto
- Vediamole a turno....

Direttive JSP

- Ci sono tre tipi di direttive:
  - di pagina (dimensioni buffer, trattamento errori, linguaggio da usare, ecc.)
  - *include* (per inserire altri file dentro la pagina)
  - *taglib* (estensioni del linguaggio JSP)

Direttive JSP

- <%@ DIRETTIVA {attributo=valore} %>

Esempi:

- <%@ page language=java %>
- <%@ page import=java.awt.*.java.util.* session=true %>
- <%@page errorPage="error.jsp" %>
- <%@ include file="myFile.html" %>
- <%@ taglib uri="myTagLib" %>

Dichiarazioni JSP

- <%! DICHIARAZIONE %>

Esempi:

- <%! String nome=pippo %>
- <%! public String getName() {return nome} %>

Espressioni JSP

- L'espressione fra <%= e %> viene *valutata*
- Il valore ottenuto viene converito in stringa
- La stringa viene sostituita al posto di

<%= .... %>

Espressioni JSP

- <%= ESPRESSIONE%>

Esempi:

- <%= getName() %>
- <%= new Date().toString(); %>

Esempio

```
<html><body>
<%! String nome=pippo %>
<%! public String getName() {return nome} %>
<H1>
Buongiorno
<%= getName() %>
</H1>
 </body></html>
```

Scriptlet JSP

- Il codice fra `<%` e `%>` viene eseguito
- L'immersione può anche essere parziale, come nel nostro esempio precedente:

```
<% if (calendario.get(Calendar.AM_PM)==Calendar.AM) { %>
buon giorno!
<% } else { %>
buona seeeeeraaaaa...
<% } %>
```

- Le dichiarazioni possono essere fatte nelle scriptlet

Scriptlet JSP

```
<% SCRIPTLET %>

Esempi:
<% z=z+1; %>
<%
      // Get the Employee's Name from the request
      out.println("<b>Employee: </b>" +
      request.getParameter("employee"));
      // Get the Employee's Title from the request
      out.println("<br><b>Title: </b>" +
      request.getParameter("title"));
 %>
```

Commenti JSP

- Simili ai commenti HTML
- Sono rimossi dalla pagina prima di essere inviati al browser
- Racchiusi fra `<%--` e `--%>`

Azioni JSP

- **`jsp:useBean`** (istanzia un bean e gli assegna un nome)
- **`jsp:setProperty`** (assegna un valore a una proprietà di un bean)
- **`jsp:getProperty`** (legge il valore di una proprietà, lo converte in stringa, e lo manda in output)
- ... e alcune altre

Azioni JSP

- <jsp:AZIONE>

Esempi:

- <jsp:forward page=URL>
- <jsp:include page=URL flush=true>
- <jsp:useBean>

Oggetti JSP predefiniti

1.	out	Writer
2.	request	HttpServletRequest
3.	response	HttpServletResponse
4.	session	HttpSession
5.	page	this nel Servlet
6.	application	servlet.getServletContext
7.	config	ServletConfig
8.	exception	solo nella errorPage
9.	pageContext	sorgente degli oggetti, raramente usato

request

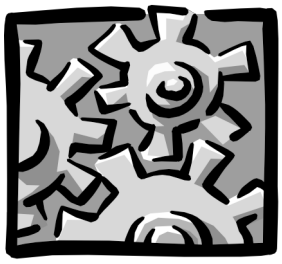
```
<%@ page errorPage="errorpage.jsp" %>
<html>
  <head>
    <title>UseRequest</title>
  </head>
  <body>
    <%
        // Get the User's Name from the request
        out.println("<b>Hello: " + request.getParameter("user") + "</b>");
    %>
  </body>
</html>
```

session

```
<%@ page errorPage="errorpage.jsp" %>
<html> <head> <title>UseSession</title> </head>
  <body>
    <%
      // Try and get the current count from the session
      Integer count = (Integer)session.getAttribute("COUNT");
      // If COUNT is not found, create it and add it to the session
      if ( count == null ) {
        count = new Integer(1);
        session.setAttribute("COUNT", count);
      }
 else {
        count = new Integer(count.intValue() + 1);
        session.setAttribute("COUNT", count);
      }
      // Get the User's Name from the request
      out.println("<b>Hello you have visited this site: "
        + count + " times.</b>");
    %>
  </body> </html>
```

Trattamento delle pagine JSP

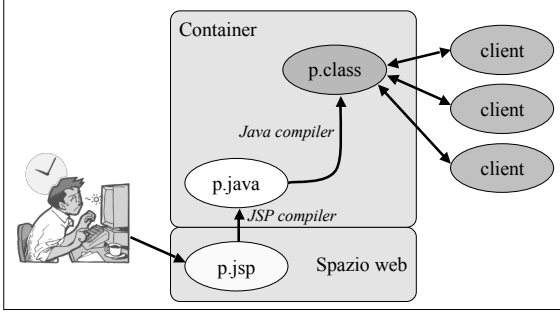
- Ma, esattamente, cosa accade quando un web server deve trattare una pagina JSP?
- Quali caratteristiche derivano alle JSP da questo trattamento?



Trattamento delle pagine JSP

- La prima volta che al server viene richiesta una certa pagina JSP, la pagina viene **compilata** e diventa una *servlet* (**.java**)
- La *servlet* viene *a sua volta* compilata e produce un file eseguibile (**.class**)
- Il codice del **.class** viene caricato nel server, viene eseguito, e tenuto pronto per eventuali altre richieste successive

Trattamento delle pagine JSP



Esempio di compilazione

JSP

```
<HTML>
<%@ page language="java" imports=="com.wombat.JSP" %>
<H1>Benvenuto!</H1>
Oggi è
<jsp:useBean id=="calendario" class=="JSPCalendar" />
<UL>
...
```

Servlet

```
import com.wombat.JSP.*;
public class XXX extends HttpServlet {
  public void doGet(HttpServletRequest r, HttpServletResponse q) {
    PrintWriter out = new PrintWriter(q.getWriter());
    out.println("<HTML>");
    out.println("<H1>Benvenuto!</H1>");
    out.println("Oggi è");
    JSPCalendar calendario = new JSPCalendar();
    out.println("<UL>");
    /* ... */
```

Caratteristiche di JSP

- Ereditano tutte le buone caratteristiche delle Servlet
- In più, rendono facile fare modifiche:
  - la compilazione è automatica – si può fare prototipazione rapida
  - se si vuole modificare l'aspetto delle pagine HTML, non c'è bisogno di toccare il codice – il livello di competenze richiesto è più basso

Uso tipico di JSP

- Il merito principale delle JSP è di separare la **presentazione** (parte HTML) dalla **logica applicativa** (parte codice)
- Per sfruttare al meglio questa separazione, è bene ridurre al minimo le scriptlet `<% ... %>`, e spostare tutta la logica all'interno di JavaBean appropriati

Uso tipico di JSP

- Con una chiara separazione fra presentazione e logica applicativa:
  - La pagina JSP può essere creata inizialmente dal programmatore, ma poi migliorata graficamente e mantenuta da un designer HTML
  - Il programmatore può cambiare il comportamento dell'applicazione modificando i bean, senza più toccare la pagina JSP

Sintassi JSP

- Sintassi "tradizionale"
  - Mark-up simile a quello di ASP
  - Il documento risultante non è più HTML, né rispetta altri standard
  - Non si possono usare strumenti standard (es. editor HTML) su pagine JSP
- Sintassi XML
  - Il mark-up è completamente basato su XML
  - Il documento risultante rispetta un DTD ben definito
  - Si possono usare strumenti standard per XML su pagine JSP

Sintassi JSP (tradizionale vs. XML)

elemento	tradizionale	XML
commenti	<%-- … %>	<!-- … -->
dichiarazioni	<%! … %>	<jsp:declaration> … </jsp:declaration>
direttive	<%@ include … %>	<jsp:directive.include … />
	<%@ page … %>	<jsp:directive.page … />
	<%@ taglib … %>	<html xmlns:*pfx*="*URL*" … >
espressioni	<%= … %>	<jsp:expression> … </jsp:expression>
scriptlet	<% … %>	<jsp:scriptlet> … </jsp:scriptlet>

Direttive

- Le **direttive** di una pagina JSP servono a comunicare informazioni sulla pagina al container (tipicamente, l'application server)
- Fra l'altro, indicano alcune opzioni relative al modo in cui la pagina deve essere compilata in una servlet

Direttive

- Ogni direttiva ha tipicamente uno o più **attributi**, a cui è possibile assegnare dei valori
- La sintassi è

  `<%@ direttiva attr`$_1$`=val`$_1$ … `attr`$_n$`=val`$_n$ `%>`
- Consideriamo tre direttive principali:
  - **include** – specifica l'inclusione di un'altra risorsa
  - **page** – dichiara proprietà della pagina
  - **taglib** – importa nuovi tag da una tag library

Direttiva **include**

- La direttiva **include** inserisce il contenuto di una risorsa data all'interno del testo della pagina JSP, **a tempo di compilazione**
- Si tratta dunque di un **`#include`** stile C
- Sintassi:
  - `<%@ include file="nome file" %>`
  - `<jsp:directive.include file="nome file" />`
- "nome file" può essere una URL (relativa)

Direttiva include

data.jspf

```
<%@ page import="java.util.*" %>
<%= (new Date()).toLocaleString() %>
```

oggi.jsp

```
<html><head>Esempio include</head>
<body>
L'ora corrente è
<b><%@ include file="data.jspf" %></b>
</body></html>
```

L'ora corrente è **10/20/06 11:29 AM**

Direttiva **page**

- La direttiva **page** specifica un certo numero di proprietà della pagina
- Alcune proprietà vengono usate al momento della compilazione (per costruire il corpo di alcuni metodi della servlet)
- Altre vengono usate a tempo di esecuzione

Direttiva **page**

attributo	descrizione
language="java"	Specifica il linguaggio usato negli scriptlet
extends="*classe*"	Dichiarazione extends della servlet
import="*package*" o "*classe*"	Dichiarazione di import della servlet
session="true" o "false"	Supporto alle sessioni (default true)
buffer="none" o "*n*kb"	Dimensione del buffer di output (default 8kb)
autoFlush="true" o "false"	Flush automatico del buffer (default true)
isThreadSafe="true" o "false"	La pagina è thread-safe (possibili più thread) (default true)
info="*testo*"	Stringa di informazioni generiche
errorPage="URL relativa"	Pagina da visualizzare in caso di errori
contentType="*tipo MIME*"	Tipo MIME della pagina (default text/html)
isErrorPage="true" o "false"	True se questa è una pagina di errore
pageEncoding="*charset*"	Encoding della pagina (default ISO-8859-1)

Direttiva **taglib**

- La direttiva taglib consente di importare una **tag library**
- Ciascuna tag library definisce un certo numero di tag addizionali che possono essere usati nella pagina
- Esistono tag library standard, di terze parti, e custom (potete definire le vostre)

Direttiva **taglib**

- La sintassi è
  - <%@ taglib uri="*libreria*" prefix="*pfx*" %>
- Dopo la direttiva taglib, è possibile usare i tag definiti in libreria, prefissandoli con pfx
- Es: se *libreria* definisce un tag <orario/>e ho usato come *pfx* "hr", potrò accedere al tag con <hr:orario/>

Direttiva **taglib**

- In generale, chi scrive una pagina JSP può scegliere i prefissi a suo piacimento, in modo da evitare conflitti di nomi quando più librerie definiscono gli stessi tag
- Alcuni prefissi sono riservati:
  - jsp, jspx, java, javax, servlet, sun, sunw

Tag "storici" <jsp:…/>

- Versioni "vecchie" (pre-2.0) della specifica JSP definiscono un certo numero di tag di base, introdotti dal prefisso **`jsp:`**
- A questi tag "storici" se ne sono poi aggiunti molti altri
- Vediamo dapprima i tag più antichi, rimandando a dopo le librerie più moderne

jsp:forward <jsp:…/>

- jsp:forward implementa il metodo forward() delle servlet
- Permette di inoltrare una richiesta a un'altra pagina JSP (o a una servlet, o a una risorsa statica)

```
<jsp:forward page="url">
  <jsp:param name="n"
             value="v"/>
  <jsp:param name="n"
             value="v"/>
    …
</jsp:forward>
```

I **`jsp:param`** *sono opzionali.*

I valori di **`page`** *e* **`value`** *possono essere dati con espressioni JSP delimitate da* **`<%= %>`**

jsp:include

<jsp:…/>

- jsp:include implementa il metodo include() delle servlet
- Permette di incorporare il risultato dell'invocazione di un'altra JSP (o servlet, o altra risorsa)
- Nota bene:
  - `<%@ include %>` a tempo di compilazione
  - `<jsp:include />` a tempo di esecuzione

```
<jsp:include page="url"
             flush="t/f">
  <jsp:param name="n"
             value="v"/>
  <jsp:param name="n"
             value="v"/>
    ...
</jsp:include>
```

I **`jsp:param`** *sono opzionali, come anche l'attributo* **`flush`**

I valori di **`page`** *e* **`value`** *possono essere dati con espressioni JSP delimitate da* **`<%= %>`**

jsp:plugin

<jsp:…/>

- jsp:plugin genera codice HTML per includere un componente client-side (es., una applet o un bean eseguito dal plugin Java)
- Il codice generato è quello "giusto" per il browser e il tipo di plugin
  - <applet>, <embed>, ecc.

```
<jsp:plugin
      ...attributi vari...  >
  <jsp:param name="n"
             value="v"/>
  <jsp:param name="n"
             value="v"/>
    ...
  <jsp:fallback>
      testo da mostrare se il plugin
      non è disponibile
  </jsp:fallback>
</jsp:plugin>
```

I **`jsp:param`** *sono opzionali, come anche il* **`jsp:fallback`**

jsp:plugin

<jsp:…/>

attributo	descrizione
type="bean" o "applet"	Tipo di componente
code="nome classe"	Classe da eseguire
codebase="directory base"	Codebase
name="nome istanza"	Il nome con cui quest'istanza del componente può essere riferita
archive="archivio jar"	Archivio JAR contenente le classi
align, height, width, hspace	Dimensioni e formattazione
jreversion="versione"	Versione minima del JRE richiesta
nspluginurl="URL jre"	Dove scaricare il JRE per Netscape
iepluginurl="URL jre"	Dove scaricare il JRE per Internet Explorer

jsp:useBean

<jsp:…/>

- jsp:useBean consente di includere un Javabean all'interno della pagina JSP
- Il bean non è visuale: tipicamente, si usa per mantenere il modello
- Se un bean con il nome dato non esiste ancora nell'ambito, viene istanziato, altrimenti si accede al bean già esistente

```
<jsp:useBean
  id="nome istanza"
  scope=ambito
  class="classe" … >
</jsp:useBean>
```

L' **`ambito`** *può essere* **`page`** *(che è il default),* **`request`**, **`session`** *o* **`application`**, *e stabilisce quanto deve essere persistente il bean.*

Il nome dell'istanza viene usato in altri tag per riferire il bean.

jsp:getProperty e jsp:setProperty

<jsp:…/>

- jsp:getProperty legge una proprietà del bean
  - Il risultato, trasformato in stringa, viene aggiunto all'output
- jsp:setProperty assegna un valore alla proprietà

```
<jsp:getProperty
  name="nome istanza"
  property="nome proprietà"
>

<jsp:setProperty
  name="nome istanza"
  property="nome proprietà"
  value="valore"
>
```

Il valore di **value** *può essere dato con espressioni JSP delimitate da* **<%= %>**

jsp:setProperty *ha anche altre forme*

Oggetti impliciti

- In aggiunta ai Javabean creati con **<jsp:useBean>**, ciascuna pagina ha accesso a un certo numero di **oggetti impliciti**, creati e mantenuti dal container
- Questi oggetti corrispondono a quelli accessibili dalle servlet

Oggetti impliciti

oggetto	funzione
pageContext	Contesto di esecuzione della pagina; fornisce accesso anche a servletContext, session, request, response
param	Parametri della richiesta (nome → valore singolo)
paramValues	Parametri della richiesta (nome → valori multipli)
header	Header della richiesta (nome → valore singolo)
headerValues	Header della richiesta (nome → valori multipli)
cookie	Cookie del client (nome → valore singolo)
initParam	Parametri di inizializzazione del container
pageScope	Oggetti di scoping (vd. servlet)
requestScope	
sessionScope	
applicationScope	

Oggetti impliciti

oggetto	descrizione
request	Richiesta della servlet
response	Risposta della servlet (di solito non è usata in JSP)
session	Sessione corrente
application	Oggetto ServletContext della servlet corrispondente
out	Writer connesso alla risposta (di solito non usato)
config	Configurazione iniziale
exception	Eccezione che ha causato il caricamento di una error page (disponibile solo se la pagina ha isErrorPage="true")

Comunicazione con le servlets

- Le servlets sono utilizzate spesso in combinazione con le JSP
- Quindi, è utile conoscere come esse possono collaborare fra loro
- I due principali motivi per cui le servlets e le JSP possono volersi scambiare informazioni sono
  - Per condividere informazioni legate alla sessione individuale di un utente
  - Per condividere informazioni legate all'ambiente dell'applicazione

Comunicazione con le servlets

- Per le JSP entrambe queste categorie di informazioni sono fornite dagli oggetti impliciti **session** e **application**
- Vediamo quali oggetti forniscono queste informazioni alle servlet, e come avviene lo scambio/condivisione

Comunicazione con le servlets

- Se un oggetto **Session** è stato creato da una servlet (nella stessa sessione) quando una JSP è riferita, allora l'oggetto JSP implicito **session** contiene tutte le coppie attributo-valore esistenti nell'oggetto **Session** originale
- Quindi l'oggetto implicito **session** può semplicemente utilizzare il suo metodo **getValue** per prelevare le informazioni memorizzate nella servlet

Comunicazione con le servlets

- La classe **HttpServlet** implementa l'interfaccia **ServletConfig** attraverso la sua superclasse **GenericServlet**
- Questa interfaccia ha un metodo chiamato **getServletContext** che restituisce un riferimento a **ServletContext**
  - Il contesto della servlet

Comunicazione con le servlets

- Per ottenere accesso in lettura/scrittura alle informazioni legate all'ambiente dell'applicazione, una servlet
  - Prima invoca **getServletContext** e memorizzail riferimento a **ServletContext**
  - Poi invoca i metodi **setAttribute** e **getAttribute** sul riferimento
  - Nello stesso modo questi metodi sono invocati sull'oggetto JSP implicito application

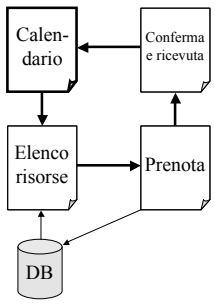
```
ServletContext context = getServletContext();
String userName = (String)context.getAttribute("name");
```

- Analogamente a **session**, **application** contiene le informazioni registrate dalla servlet

Un esempio

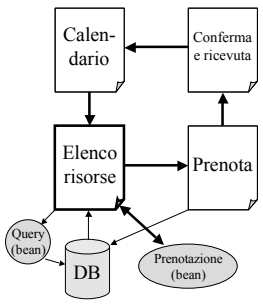
- Consideriamo per esempio una applicazione per la prenotazione di risorse (stanze, proiettori, traduttori, …) per un centro conferenze
- Come potremmo strutturarla?
  - Quali pagine JSP ci servono?
  - Come possono condividere i dati?

Un esempio



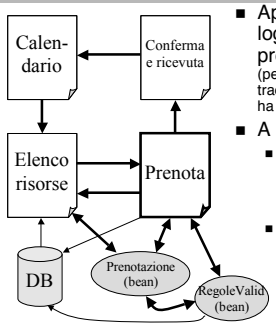
- Calendario.html usa una normale FORM HTML per far scegliere una data all'utente
- Al submit, i dati della FORM vengono inviati a **Elenco_risorse.jsp**
- (in alternativa, usiamo una JSP con un <jsp:plugin> per mostrare un calendario grafico)

Un esempio



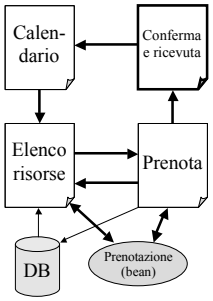
- **Elenco_risorse.jsp** accede ai dati della FORM tramite param, e li usa per fare una query al DB via JDBC o tramite un bean con scope di pagina
- Quindi manda in out l'elenco delle risorse disponibili, divise per categorie, con un checkbox accanto a ciascuna
- Anziché usare una lunga FORM, raccogliamo la prenotazione in un bean con scope di sessione

Un esempio



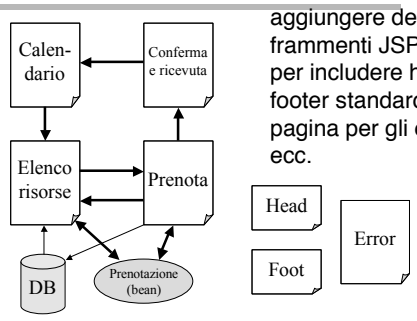
- **Prenota.jsp** recupera la Prenotazione dal contesto di sessione
- Applica un po' di business logic per decidere se la prenotazione è "sensata"
  (per esempio: non posso chiedere un traduttore se la stanza che ho scelto non ha un impianto audio adeguato)
- A seconda dei casi:
  - rimanda a **Elenco_risorse.jsp** (passando con <jsp:param> o nella Prenotazione una spiegazione del problema)
  - accetta la prenotazione e la registra nel DB, passando a **Conferma_e_ricevuta.jsp**

Un esempio



- **Conferma_e_ricevuta.jsp** informa l'utente che la prenotazione è stata accettata, fornisce un numero di ricevuta (che può essere usato da ulteriori pagine JSP per amministrazione e verifica)
- Infine, si ritorna alla pagina iniziale per una ulteriore prenotazione

Un esempio



- Potremmo anche aggiungere dei frammenti JSP (.jspf) per includere header e footer standard, una pagina per gli errori, ecc.

Esercizio 5

- Riscrivete l'esercizio della somma fra interi (Esercizio 2) usando le JSP

Errori

- Cosa succede se una pagina JSP produce degli errori?
- Provate ad esempio a inserire letter invece di numeri nell'esempio della somma….

Errori

- Per gestite i casi di errore, si crea una pagina JSP d'errore, e si configura il sistema in modo tale che venga invocata in caso d'errore
- Ecco come fare:
  - Utilizzare la direttiva **page** con attributo **errorPage** per specificare quale pagina JSP invocare in caso d'errore
    <%@ page errorPage="ErroreSomma.jsp" %>
  - Scrivere la pagina **ErroreSomma.jsp**. In questa pagina va utlizzata la direttiva **page** impostando l'attributo booleano **isErrorPage** a **true** (specifica che questa pagina viene utilizzata come pagina d'errore)
    <%@ page isErrorPage="true" %>

Esercizio 6

- Aggiungete la gestione dell'errore all'esempio della somma di due interi
- La pagina d'errore deve visualizzare il messaggio legato all'eccezione generata e un pulsante per riprovare
  - L'eccezione si accede tramite l'oggetto implicito **exception**
  - Il pulsante **Riprova** riporta alla pagina html iniziale

Esercizio 7

- Riscrivete l'esercizio del carrello (Esercizio 3) usando le JSP

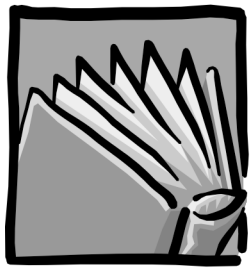
Java

progettazione di applicazioni web -- jsp

fine

JavaServer Pages – JSP

- Primo Teorema delle JSP
  - JSP sta a Java come ASP sta a Javascript
- Però hanno un certo numero di caratteristiche in più:
  - Funzionano su più server (non solo Microsoft IIS)
  - Sono portabili su OS diversi
  - Interagiscono con i Bean



Uso tipico di JSP

- Il merito principale delle JSP è di separare la **presentazione** (parte HTML) dalla **logica applicativa** (parte codice)
- Per sfruttare al meglio questa separazione, è bene ridurre al minimo le scriptlet `<% … %>`, e spostare tutta la logica all'interno di JavaBean appropriati
- Implementazione in stile **MVC** (Model-View-Controller)

Uso tipico di JSP

- L'architettura MVC prevede un **modello** astratto della realtà che si vuole modellare (**M**)
- Separatamente, uno strato di software legge lo stato del modello e ne produce una **vista** (**V**)
- Analogamente, uno strato di software detto **controller** interpreta le azioni dell'utente e le effettua apportando modifiche al modello (**C**)
- Nel nostro caso, **M** è implementato come un insieme di Javabean; pagine JSP fungono da **V** e **C**

Confronto fra CGI, Perl e Servlet

Caratteristica	CGI	mod_perl	Servlet
Portabile fra web server	**si**	**no** (solo Apache)	**si** (limitata)
Portabile fra S.O.	**si** (con poche varianti)	**no**	**si**
Un processo per ogni richiesta	**si**	**no**	**no**
Perdite di memoria (leak)	**no**	**si**	**no** (in teoria)
Linguaggio	C, Perl, sh, ecc.	Perl	Java
Tipi stretti	**no**	**no**	**si**
Persistenza	**no** (va fatta a mano)	**no** (va fatta a mano)	**si**
Persistenza verso DB	**no**	**si**	**si**
Portabilità fra DB	**no** (ad hoc)	**no** (ad hoc)	**si** (via JDBC)
Controllo dei diritti	**no** (solo via S.O.)	**no** (solo via S.O.)	**si** (diritti Java)
Supporto da IDE	**no**	**no**	**si** (es.: VisualAge)

Confronto fra ASP e JSP

Caratteristica	ASP	JSP
Portabile fra web server	**no** (solo Microsoft IIS)	**si** (limitata)
Portabile fra S.O.	**no** (solo Microsoft Windows)	**si**
Un processo per ogni richiesta	**no**	**no**
Perdite di memoria (leak)	**no** (in teoria)	**no** (in teoria)
Linguaggio	Javascript	Java
Tipi stretti	**no**	**si**
Persistenza	**si**	**si**
Persistenza verso DB	**si**	**si**
Portabilità fra DB	**si** (via ODBC)	**si** (via JDBC)
Controllo dei diritti	**no** (solo via S.O.)	**si** (diritti Java)
Supporto da IDE	**si**	**si** (ancora debole)