

JSP [2]

Uso di Bean
Interazione fra pagine JSP

Slides parzialmente tratte da materiale di Giansalvatore Mecca (*Tecnologie di Sviluppo per il Web*) e Marty Hall (<http://www.coreservlets.com>)

Uso dei Costrutti JSP

Applicazioni
Semplici

Applicazioni
Complesse

- Elementi di scripting che svolgono le elaborazioni direttamente in JSP
- Elementi di scripting che svolgono elaborazioni indirettamente
 - impiego di classi di utilità
- Uso di oggetti **Bean**
- Custom tags
- Uso combinato di JSP, Servlet e Bean
 - pattern di sviluppo *Model-View-Control* (MVC)

Tag di azione

- Strategia: separare la presentazione dall'elaborazione
 - non conviene inserire scriptlet complessi nelle pagine destinate alla presentazione dei contenuti
 - meglio inserire il codice in apposite classi usate dalle pagine JSP
- Vantaggi
 - **Sviluppo**: si può sviluppare il codice Java in un ambiente specializzato, piuttosto che in un ambiente per HTML
 - **Debug**: si individuano gli errori di sintassi a tempo di compilazione
 - **Manutenzione**: aumenta la leggibilità e la modularità del codice
 - **Riuso**: la stessa classe può essere utilizzata da varie pagine
- Una soluzione: i tag di azione
 - tag speciali per l'esecuzione di operazioni frequenti

Tag di azione:

Uso di bean in JSP

JavaBean

- è una classe Java che rispetta alcune convenzioni
- ha un costruttore senza argomenti
- non ha variabili di istanza pubbliche
- per ogni variabile di istanza con nome xyz esistono i metodi:
 - getXyz()
 - setXyz(<tipo di xyz>)

Esempio

```
public class studente {
    private String nome;
    private String matricola;
    private int eta;
    public Studente() {}
    public String getNome() {
        return nome;
    }
    public String getMatricola () {
        return matricola;
    }
    public int getEta () {
        return eta;
    }
    public void setNome ( String nome ) {
        this.nome = nome;
    }
    public void setMatricola (String matr) {
        this.matricola = matr;
    }
    public void setEta (int eta) {
        this.eta = eta;
    }
}
```

Tag di azione:

Uso di bean in JSP

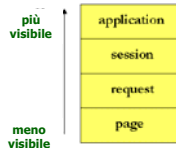
- Tre tag di azione per manipolare bean
 - **jsp:useBean**
 - per creare o rendere disponibile un oggetto bean in una pagina JSP
 - **jsp:setProperty**
 - per impostare proprietà di un bean
 - **jsp:getProperty**
 - per recuperare valori associati alle proprietà di un bean
- Utilità
 - è più facile impostare il valore di un oggetto dai parametri di una richiesta
 - è più facile condividere oggetti tra pagine e servlet

Uso di bean in JSP

- Uso di oggetti bean
 - Le istanze dei bean possono memorizzate in contesti (scope) associati agli oggetti impliciti
 - request, session, application, pageContext
- Azione **jsp:useBean**
 - può essere usato per istanziare un bean
 - può essere usato per salvare un'istanza di bean istanziato in un contesto
 - può essere usato per recuperare un bean precedentemente istanziato da un'altra componente

Visibilità (scope) dei Bean

- Esistono 4 campi di visibilità (scope) in cui si possono memorizzare gli oggetti bean
 - Pagina
 - Richiesta
 - Sessione
 - Applicazione
- Ogni contesto corrisponde ad un oggetto predefinito, in cui sono memorizzati i dati del contesto
- I contesti sono visibili a più componenti
 - pagine JSP, Servlet, thread di una stessa Servlet
 - Esse possono, così, condividere informazioni



Visibilità (scope) dei Bean

- Pagina (Page scope)**
 - Oggetti che esistono solo nelle pagine in cui sono definiti
 - Per ogni accesso alla pagina esiste un'istanza di tali oggetti
- Richiesta (Request scope)**
 - Esistono per la durata della richiesta del client
 - Muiono quando la risposta è stata inviata al client
- Sessione (Session scope)**
 - Esistono per la durata della sessione di browsing del client
 - Muiono quando l'utente lascia il sito o avviene un timeout della sessione
- Applicazione (Application scope)**
 - Oggetti associati all'intera applicazione web
 - Ogni Servlet/JSP nell'applicazione può accedervi

Accesso ai bean da JSP azione *useBean*

- Sintassi

```
<jsp:useBean id="nome" [class="classe"]
                    [type="type"] [scope="contesto"] />
```

 - l'attributo *scope* può assumere come valori i vari contesti:
 - session, application, request, page (valore di default)
 - I valori degli attributi *class* e *type* sono nomi di classi Java
 - includono l'indicazione del package
- Esempio:

```
<jsp:useBean id="studente" class="StudenteBean" scope="request"/>
```

Accesso ai bean da JSP azione *useBean*

- Semantica - se viene specificato "class"**
 - viene cercato un oggetto con nome uguale all'id specificato nel contesto indicato dall'attributo *scope*
 - se c'è, viene restituito il valore, dopo aver fatto il cast sulla classe specificata (può esserci una *ClassCastException*)
 - altrimenti viene creata una nuova istanza del bean e salvata nel contesto
- Semantica - se viene specificato "type"**
 - viene cercato l'attributo nel contesto indicato da *scope*
 - se c'è, viene restituito il valore, dopo aver fatto il cast sulla classe specificata con *type*
 - altrimenti **NON** viene creata una nuova istanza del bean e si genera un'eccezione

Accesso ai bean da JSP azione *useBean*

- Esempio:

```
<jsp:useBean id="auto" class="acibase.Automobile"
                    scope="session" />
```

 - semantica**
 - verifica se nella sessione esiste un bean chiamato *auto*
 - se c'è, prova il cast su *acibase.Automobile* e lo associa al riferimento *auto*
 - se non c'è lo crea, lo associa al riferimento *auto* e lo salva nel contesto di sessione

Accesso ai bean da JSP azione *getProperty*

- Funzione:**
 - serve a prelevare il valore di una proprietà, a convertirla in stringa e a stamparla nella risposta
- Sintassi**

```
<jsp:getProperty name="nomeBean"
                    property="nomeProprieta" />
```
- Semantica**
 - esegue il metodo *get* per accedere la proprietà, e poi esegue una *out.print*
 - è necessario utilizzare il nome di una proprietà esistente
 - se la proprietà è null viene restituita la stringa vuota
- Esempio:**
 - il codice:

```
<jsp:getProperty name="auto" property="cilindrata" />
```
 - è equivalente a:

```
<%= auto.getCilindrata() %>
```

Esempio: StringBean

```
package coreservlets;
public class StringBean {
    private String message = "No message specified";
    public String getMessage() {
        return(message);
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

- **Installazione**

Il bytecode deve essere nella stessa directory dei servlet (eventuali sottodirectory per il package)
`install_dir\webapps\ROOT\WEB-INF\classes\coreservlets`

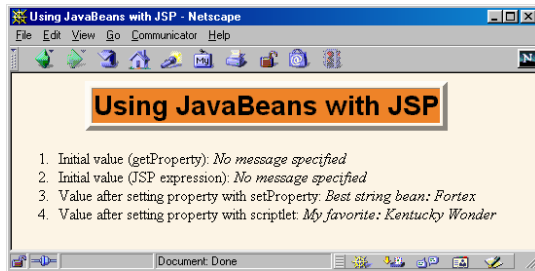
- **Consiglio**

Le classi bean (e di utilità) devono essere *sempre* definite in package!

Pagina JSP che usa StringBean

```
<jsp:useBean id="stringBean" class="coreservlets.StringBean" />
<OL>
<LI>Initial value (getProperty):
    <I><jsp:getProperty name="stringBean" property="message" /></I>
<LI>Initial value (JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
    property="message"
    value="Best string bean: Fortex" />
    Value after setting property with setProperty:
    <I><jsp:getProperty name="stringBean" property="message" /></I>
<LI><%=stringBean.setMessage("My favorite: Kentucky Wonder");%>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>
```

Pagina JSP che usa StringBean



azione setProperty – forma 1

- **Funzione:**

- serve a modificare il valore di una proprietà del bean
- varie forme per specificare il valore
 - valore costante
 - espressione Java
 - valore di uno o di tutti i parametri della richiesta

- **Prima forma**

```
<jsp:setProperty name="nomeBean"
property="nomeProprieta" value="valoreCostante" />
```

- **Semantica**

Si richiama il metodo set corrispondente al nome indicato da *property* (+set), passando come parametro il valore di *value*

- **Esempio**

```
<jsp:setProperty name="auto" property="cilindrata" value="1800" />
equivalente a: <%= auto.setProperty(1800); %>
```

azione setProperty – forma 2

- **Sintassi**

```
<jsp:setProperty name="nomeBean"
property="nomeProprieta" value="<%= espressione %>" />
```

- **esempio:**

```
<jsp:setProperty name="auto" property="cilindrata"
value="<%= request.getParameter("cilindrata") %>" />
```

- **Semantica equivalente alla precedente**

- **Attenzione:** conversione automatica del valore

- il contenitore riceve come parametro una stringa
- se il tipo della proprietà è diverso da *String*, applica in automatico la routine standard di conversione da stringa al valore del tipo
 - es: Integer.parseInt()
- se il valore fornito è scorretto si può generare un'eccezione

setProperty–forma 3: assegnazione di parametri di form

- **Sintassi 1**

```
<jsp:setProperty name="nomeBean"
property="nomeProprieta"
param="nomeParametro" />
```

- **Ipotesi:** Il valore viene da un parametro associato alla richiesta

- l'attributo *param* indica il nome del parametro

- **Semantica**

- si cerca il parametro nel contesto indicato
 - se il parametro c'è, si tenta di assegnare il valore alla proprietà
 - se il parametro non c'è, non si effettua nessuna operazione
- conversione automatica del valore del parametro
 - dal tipo String al tipo della variabile

setProperty-forma 3: assegnazione di parametri di form

- Sintassi 2

```
<jsp:setProperty name="nomeBean"  
property="nomeProprieta" />
```

- hp: controlli delle form con uguale nome delle proprietà dei bean
- Semantica:

se la richiesta contiene il parametro, si tenta di assegnare il valore alla proprietà, altrimenti non si effettua nessuna operazione

- Sintassi 3

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

- Es: <jsp:setProperty name="auto" property="*" />

- Semantica:

- assegna i valori di TUTTI i parametri della richiesta alle proprietà del bean con lo stesso nome, secondo la semantica descritta prima
- attenzione anche in questo caso ai valori scorretti forniti dall'utente
- Molto comodo per la creazione di "form beans"
- oggetti i cui campi sono riempiti mediante una o più form

Operazioni condizionali su Bean

- Creazione condizionale di Bean

- jsp:useBean crea una nuova istanza del bean solo se non esiste una con lo stesso id e lo stesso scope
- Altrimenti l'istanza pre-esistente è semplicemente riferita da id

- Impostazione condizionale di proprietà di un Bean

- La sintassi tradizionale <jsp:useBean ... /> può essere sostituita con

```
<jsp:useBean ...>  
operazioni  
</jsp:useBean>
```

- Le operazioni (elementi jsp:setProperty) sono eseguite solo se viene creato un nuovo bean, e non se il bean esiste già

Bean - esempio: un Bean per il conteggio degli accessi

```
package contatori;  
  
public class CounterBean {  
    private String firstPage;  
    private int accessCount = 1;  
  
    public String getFirstPage() {  
        return(firstPage);  
    }  
  
    public void setFirstPage(String firstPage) {  
        this.firstPage = firstPage;  
    }  
  
    public int getAccessCount() {  
        return(accessCount);  
    }  
  
    public void setAccessCount(int increment) {  
        accessCount = accessCount + increment;  
    }  
}
```

Bean - esempio 1: uso di Bean con scope differenti

```
<jsp:useBean id="session_counter" class="contatori.CounterBean"  
scope="session" />  
<jsp:useBean id="app_counter" class="contatori.CounterBean"  
scope="application" />  
  
<% session_counter.setAccessCount(1);  
synchronized(page) {  
    app_counter.setAccessCount(1);  
}  
%>  
<h3>  
Numero di accessi nella sessione corrente:  
<jsp:getProperty name="session_counter" property="accessCount" />  
</h3> <p>  
<h3>  
Numero totale di accessi nell'applicazione:  
<% synchronized(page) { %>  
    <jsp:getProperty name="app_counter" property="accessCount" />  
<% } %>  
</h3>
```

Bean - esempio 2: accesso al contatore da più pagine

- L'applicazione Web è costituita da 3 pagine JSP che accedono al contatore (sharedCounts1.jsp, sharedCounts2.jsp e sharedCounts3.jsp)
- Prima pagina (sharedCounts1.jsp):

```
<jsp:useBean id="counter"  
class="contatori.CounterBean"  
scope="application">
```

```
<jsp:setProperty name="counter"  
property="firstPage"  
value="sharedCounts1.jsp" />
```

Esecuzione
condizionale
(solo se si
crea il bean)

```
</jsp:useBean>  
Fra sharedCounts1.jsp (questa pagina),  
<A HREF="sharedCounts2.jsp"> sharedCounts2.jsp </A> e  
<A HREF="sharedCounts3.jsp"> sharedCounts3.jsp </A>,  
<jsp:getProperty name="counter" property="firstPage" />  
&grave; stata acceduta per prima.
```

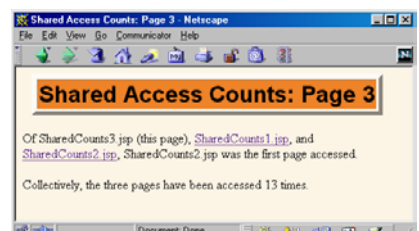
```
<p>Completivamente, le tre pagine sono state accedute  
<jsp:getProperty name="counter" property="accessCount" />  
volte.  
<jsp:setProperty name="counter"  
property="accessCount" value="1" />
```

Bean - esempio 2: accesso al contatore da più pagine

- Supponiamo che

- sharedCounts2.jsp è stata acceduta per prima
- Le tre pagine sono state accedute complessivamente 12 volte da un numero arbitrario di client

- Risultato:



Uso di Bean in JSP (sommario)

- Installazione della classe bean
Il bytecode del bean (file `.class`) deve essere nella directory `WEB-INF\classes` dell'applicazione
- Importazione della classe nella pagina
`<%@ page import="Studente"%>`
- Dichiarazione/creazione di un oggetto bean
`<jsp:useBean id="st1" class="Studente" />`
- Modifica del contenuto del bean
`<jsp:setProperty name="st1" property="nome" value="X" />`
- Accesso al contenuto di una proprietà del bean
`<jsp:getProperty name="st1" property="nome" />`

Tag di azione

- Tag di azione
 - un modo per eseguire operazioni ricorrenti senza dover programmare in Java
 - due tipologie
 - tag predefiniti
 - tag definiti dall'utente
- Funzioni principali dei tag di azione predefiniti
 - Utilizzo dei `JavaBeans`
 - Interazione con altre pagine o `Servlet`
 - inoltro delle richieste e inclusione delle risposte

Tag di azione: Inoltro delle richieste

- tag `forward`
 - sintassi: `<jsp:forward page="uri" />`
 - semantica:
il controllo passa alla pagina con l'`uri` specificato, che riceve la stessa richiesta
- `forward` ha sub-elementi `jsp:param` per i parametri
 - sintassi: `<jsp:param name="nomepar" value="valore" />`
 - semantica:
nella query string associata alla richiesta esempio compare la coppia `"nomepar=valore"`
 - esempio:
`<jsp:forward page="fine.jsp">`
`<jsp:param name="utente" value="Pippo" />`
`</jsp:forward>`
la pagina `fine.jsp` riceve nella query string `"utente=Pippo"`

Tag di azione: Inclusione delle risposte

- tag `jsp:include`
 - Sintassi:
`<jsp:include page="uri-relativo" flush=true />`
 - semantica:
 - nella risposta della pagina corrente si inserisce la risposta prodotta dalla pagina specificata in `uri`
 - anche in questo caso è possibile specificare parametri attraverso `<jsp:param>`
 - preferibile alla direttiva `include`
 - sensibile alle modifiche delle pagine incluse

Tag di azione: Inclusione delle risposte

<u>File annunci.jsp</u>	<u>File annuncio1.html</u>
<pre><html> <body> <h1>Annunci</h1> <jsp:include page="annuncio1.html" /> <jsp:include page="annuncio2.html" /> </body> </html></pre>	<pre>Il 15 marzo non ci sarà ricevimento studenti
 Gli studenti possono prendere un appuntamento tramite messaggio di posta elettronica al docente</pre>

	<u>File annuncio2.html</u>
	<pre>E' stato attivato il laboratorio per le esercitazioni del corso</pre>

Differenze tra `<jsp:include>` e la direttiva `@include`

- `jsp:include` include l'output della pagina indicata
 - `@include` include il codice della pagina
- `jsp:include` ha effetto al momento della richiesta
 - `@include` ha effetto durante la traduzione
- con `jsp:include`, la pagina principale e quella inclusa originano due servlet separate
 - con `@include`, esse diventano parte di un solo servlet
- `jsp:include` gestisce automaticamente modifiche al file incluso
 - con `@include` non è garantito (!!)