Applicazioni web

Parte 7

Java Server Pages

Sommario

- JSP
  - Architettura e sintassi di base
- JavaBeans
  - Concetti fondamentali
  - Beans in pagine JSP
- JSTL
  - Linguaggio di espressione
  - Azioni principali
  - Azioni per SQL

Java Server Pages

- JSP è una tecnologia Java per lo sviluppo di applicazioni Web che forniscono contenuti dinamici in formato HTML o XML.
- Si basa su un insieme di speciali tag con cui possono essere invocate funzioni predefinite o codice Java (JSTL).
- Permette di creare librerie di nuovi tag che estendono l'insieme dei tag standard (JSP Custom Tag Library).
- Le librerie di tag JSP si possono considerare estensioni indipendenti dalla piattaforma delle funzionalità di un Web server.
- La tecnologia JSP è correlata con quella dei servlet. All'atto della prima invocazione, le pagine JSP vengono infatti tradotte automaticamente da un compilatore JSP in servlet.
- L'uso della tecnologia JSP richiede la presenza, sul Web server, di un servlet container, oltre che di un server specifico JSP detto motore JSP (che include il compilatore JSP); in genere, servlet container e motore JSP sono integrati in un unico prodotto (per esempio, Tomcat svolge entrambe le funzioni).
- JSP è una tecnologia alternativa rispetto a numerosi altri approcci alla generazione di pagine Web dinamiche, per esempio PHP, o ASP o la più tradizionale CGI.

Wikipedia

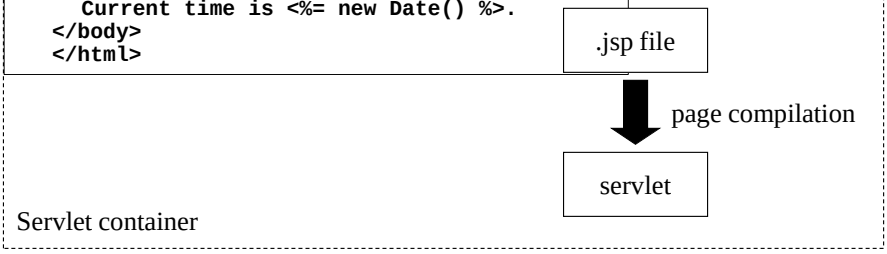
JSP - Architettura

- Le pagine JSP sono soggette a una fase di traduzione e una fase di gestione delle richieste
  - La fase di traduzione è compiuta una sola volta, a meno che la pagina JSP non cambi, nel qual caso viene ripetuta
  - Il risultato è un file di classe che implementa l'interfaccia Servlet

JSP - Compilazione

```
<%@ page import="java.util.*" %>
<html>
<head>
  <title>Current time</title>
</head>

<body>
  Current time is <%= new Date() %>.
</body>
</html>
```



JSP - Servlet generata

```
package jsp;
import [...]
import java.util.*;

public class _0005[...]_ extends HttpJspBase {
  [...]
  public void _jspService(HttpServletRequest request,
      HttpServletResponse response)
      throws IOException, ServletException {
    try {
      response.setContentType("text/html");
      JspWriter out = pageContext.getOut();
      [...]
      out.write("<html>\r\n<head>\r\n  <title>Current
        time</time>\r\n</head>\r\n\r\n<body>  Current time is ");
      out.print(new Date());
      out.write(".\r\n</body>\r\n</html>");
    } catch (Exception e) { [...] }
  }
}
```

Come utilizzare JSP

- Tomcat – run - start
- Scrivere le JSP in file .jsp
- Da memorizzare in ..programmi\Apache Software Foundation\Tomcat 5.5\webapps

JSP - Sintassi di base

- La sintassi delle pagine jsp è abbastanza semplice, e può essere classificata in:
  - *Scriptlets (<%)*
  - *Espressioni scriptlet (<%=)*
  - *Direttive (<%@)*
  - *Dichiarazioni (<%!)*
  - *Commenti (<%--)*
  - *Azioni (JSTL)*
  - *Espressioni EL*
- scriptlet e codice java
  - Limitare il codice java inserito in pagine jsp
  - Incapsulare computazione e logica applicativa in componenti (bean)
  - Preferire espressioni EL e azioni JSTL

JSP - Scriptlet

- I frammenti di codice (*scriptlet)* sono racchiusi tra i tag *<% ... %>*
- Il codice è eseguito quando la jsp risponde alle richieste
- Si può inserire qualsiasi codice java valido
- Uno scriptlet non è limitato ad una sola riga di codice
- Per esempio il seguente codice visualizza la stringa “Hello” tra tag h1, h2, h3, h4:

```
<% for (int i = 1; i <= 4; i++) { %>
  <h<% out.print(i); %>>Hello</h<% out.print(i); %>>
<% } %>
```


JSP - Espressioni scriptlet

- Le espressioni di sciptlet sono racchiuse tra i tag *<%= ... %>*, **senza** il punto e virgola finale
  - `<%= new Date() %>`
  - `<%= fooVariable %>`
  - `<%= fooBean.getName() %>`
- Il risultato della valutazione di una espressione è convertito in una stringa e incluso nella pagina (stream) di output
- Le espressioni sono di solito usate per visualizzare semplicemente il valore di una variabile o il valore restituito dall’invocazione di un metodo

Esempio 01

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Esempio JSP 01</title>
</head>
<body>
<% out.println("Prima pagina JSP"); %>
<br />
Data di oggi <%= new java.util.Date() %>
</body>
</html>
```


Esempio02 – Gestione form

```
<html>
<head><title>Gestione Form</title>
</head>
<body>
<form id="modulo02" action="jsp02-gestione_form.jsp“
   method="post">
    Nome: <input type="text" name="nome" id="nome">
    <br />
    Cognome: <input type="text" name="cognome" id="cognome">
    <br />
    <input type="submit" value="Invia Modulo">
    </form>
</body>
</html>
```

Esempio02 – Pagina di risposta

```
<html>
   <head>
        <meta http-equiv="Content-Type" content="text/html;
   charset=iso-8859-1">
        <title>Gestione Form - Risposta</title>
   </head>
   <body>
        Valori ricevuti: <br />
        nome: <%= request.getParameter("nome") %> <br />
        cognome: <%= request.getParameter("cognome") %> <br />
        User-Agent: <%= request.getHeader("User-Agent") %>

   </body>
</html>
```


Esempio03 – Scrivi Cookie

```
…
   <body>
        <%
        // Scrittura di un cookie temporaneo
        String saluti="Buongiorno";
        Cookie messaggio = new Cookie("messaggio",saluti);
        response.addCookie(messaggio);
        // Scrittura di un cookie della durata di 1 giorno
        Cookie contatore = new Cookie("Numero_visite","1");
        int durata = 60*60*24;    // durata in secondi di un giorno
        contatore.setMaxAge(durata);
        response.addCookie(contatore);
        %>
        Sono stati scritti 2 cookie
   </body>
…
```

Esempio04 – Leggi Cookie

```
…
<body>
    <p>Valori Cookie</p>
    <table border=1>
    <tr>
    <th>Nome</th>
    <th>Valore</th>
    </tr>
          <%
          Cookie[] mieiCookie = request.getCookies();
          Cookie mioCookie;
          int i, contatoreVisite;
          for(i=0;i<mieiCookie.length; i++)       //scorro i cookie
          {
                    out.println("<tr><td>" + mioCookie.getName() + "</td>");
                    out.println("<td>" + mioCookie.getValue() + "</td></tr>");
          }
          %>
    </body>
…
```


JSP - Direttive

- Le direttive sono sempre racchiuse in tag <%@ ... %>
- Sono messaggi per il motore jsp
  - Non producono direttamente un output visibile, ma indicano al motore cosa fare col resto della pagina
  - Sono interpretate in fase di traduzione
- Le direttive principali sono
  - *Page*
    - *Esempio* `<%@ page import="java.util.*, com.foo.*" buffer="16k" %>`
  - *Include*
    - *Esempio* `<%@ include file="common-header.html" %>`
  - *taglib*

JSP - Dichiarazioni

- Le dichiarazioni si trovano tra i tag <%! ... %>
- Permettono di definire campi e metodi a livello di pagina
- Il contenuto deve essere codice java valido
- Le dichiarazioni di campi terminano sempre con un punto e virgola
  - `<%! int i = 0; %>`
- Si possono anche dichiarare metodi
  - Per esempio si può sovrascivere l'evento di inizializzazione
  - ```
    <%! public void jspInit() {
      //some initialization code
    }%>
    ```


JSP - Commenti

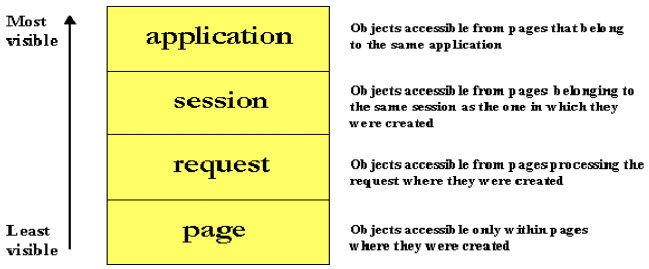
- Si possono sempre includere commenti html in pagine jsp
  - Gli utenti possono leggere questi commenti se visualizzano il codice html della pagina
- Se si vuole evitare questo, bisogna racchiudere i commenti in tag <%-- ... --%>:
  - `<!-- commento inviato al client -->`
  - `<%-- commento solo per la parte server --%>`
- I commenti jsp possono essere usati per escludere alcuni scriptlet o tag dalla compilazione
  - Ruolo importante in fase di debugging e testing
    ```

JSP - Azioni

- Eseguite durante la fase di gestione delle richieste
  - Per esempio, una azione può creare oggetti o componenti
  - Una azione può essere *standard* (definita nelle specifiche jsp) oppure *custom* (fornita tramite meccanismi di estensione dei tag)
- Un elemento di azione segue la sintassi degli elementi xml
  - Tag di apertura che include il nome dell'elemento
  - Attributi opzionali
  - Corpo opzionale
  - Corrispondente tag di chiusura
  - Oppure un tag semplice, con eventuali attributi
  - `<mytag attr1="value" ... >body</mytag>`
  - `<mytag attr1="value" ... />`
  - `<mytag attr1="value" ... ></mytag>`

JSP - Visibilità degli oggetti

- Le pagine jsp possono gestire vari tipi di oggetti
  - Oggetti impliciti, creati automaticamente dal container jsp
  - Componenti, creati dagli autori della pagina

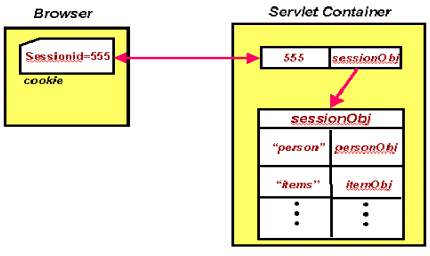


JSP - Gestione application

- Si può risalire al contesto dell'applicazione tramite l'oggetto implicito *application*, di classe *ServletContext*
- C'è un contesto per ciascuna applicazione web
  - Una *applicazione web* è una collezione di servlet e contenuto installato in una specifica sottocartella di webapps
  - Può essere installata per mezzo di un file *.war*
- Si può memorizzare e recuperare qualsiasi oggetto java, identificandolo con una chiave univoca

JSP - Gestione session

- Per default, tutte le pagine jsp partecipano ad una sessione http
- Dagli scriptlet si può accedere alla sessione tramite l'oggetto implicito *session* di classe *HttpSession*
- Utilizzare le sessioni per memorizzare oggetti che devono essere condivisi tra varie pagine jsp e servlet cui l'utente ha accesso
- L'oggetto *session* è identificato da un *session ID* e memorizzato come cookie dal browser



JSP - Gestione session

- Si può memorizzare in una sessione qualsiasi oggetto java
- Deve essere identificato da una chiave univoca
- Non c'è limite al numero di oggetti memorizzati in una sessione
  - Tuttavia, porre grossi oggetti nella sessione può degradare le prestazioni, consumando prezioso spazio in memoria heap
- È possibile evitare che una pagina jsp appartenga alla sessione
  - Attributo in direttiva page
  - `<%@ page session="false" %>`
- I server impostano il tempo di vita di un oggetto sessione
  - Spesso nei server il default è 30 minuti
  - Si può modificare invocando il metodo *setMaxInactiveInterval(int secs)* sull'oggetto *session*

JSP - Gestione delle eccezioni

- L'attributo *errorPage* della direttiva *page* permette di inoltrare una eccezione non catturata ad una pagina di gestione degli errori
  - Per esempio, la seguente direttiva informa il motore jsp di inoltrare qualsiasi eccezione non catturata alla pagina *err.jsp*
  - `<%@ page isErrorPage="false" errorPage="err.jsp" %>`
- È necessario che la pagina *err.jsp* si segnali come pagina di gestione errori, tramite la seguente direttiva
  - `<%@ page isErrorPage="true" %>`
  - Questo permette di accedere all'oggetto implicito *exception* di classe *Throwable*, che descrive l'eccezione

JSP
Oggetti impliciti per scriptlet

- Il container jsp rende disponibili degli oggetti impliciti
  - Possono essere usati negli scriptlet e nelle espressioni di scriptlet, senza che l'autore della pagina debba prima crearli
- classi e interfacce definite dalle API delle servlet
  - ***request***: rappresenta la richiesta (*HttpServletRequest)* che attiva l'invocazione di servizio (visibilità a livello di richiesta)
  - ***response***: reppresenta la risposta (*HttpServletResponse)* alla richiesta; non usata spesso (visibilità pagina)
  - ***out***: un oggetto stream (*JspWriter)* per scrivere sull'output (visibilità pagina)
  - ***session***: una sessione (*HttpSession*) (visibilità sessione)
  - ***application***: rappresenta il contesto (*ServletContext*) di configurazione dell'applicazione (visibilità applicazione)

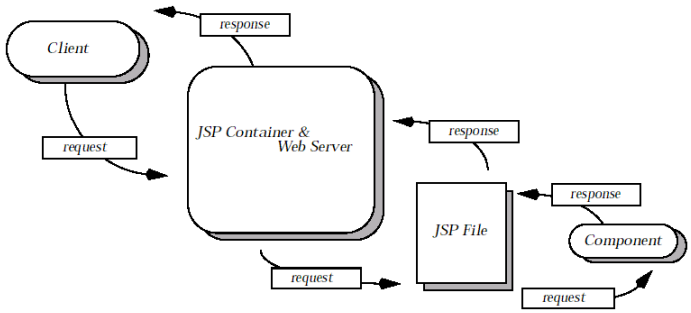
Sintassi XML vs scriplets

- JSP può usare, in alternativa alla sintassi standard, la sintassi XML
- <jsp:root xmlns="…" version="…">
- <jsp:scriplet > ↔ <% >
- <jsp:expression > ↔ <%= >
- <jsp:declaration > ↔ <%! >
- <jsp:directive > ↔ <%@ >

JavaBeans

JavaBeans
Elaborazione e presentazione

- La sola funzione delle pagine jsp dovrebbe essere di *presentare* contenuti dinamici ai loro utenti
- La reale *elaborazione* dei dati dovrebbe essere tenuta fuori dal codice jsp
  - I disegnatori di pagine non dovrebbero essere necessariamente abili sviluppatori



JavaBeans - Componenti Java

- Sono classi Java scritte seguendo particolari convenzioni.
- Le API dei *JavaBeans* rendono possibile scrivere *componenti* software in linguaggio java
- I componenti sono unità software **auto-contenute e riutilizzabili**
- … che possono essere **composte** in applicazioni, servlet e altri componenti
- .. usando strumenti di sviluppo **visuali**

JavaBeans - Introspezione

- L'introspezione è una tecnica che permette allo sviluppatore di osservare il contenuto di un Bean tramite una lista delle sue proprietà e dei suoi metodi.
- I componenti espongono le loro caratteristiche agli strumenti di sviluppo per la manipolazione visuale
- Gli strumenti di sviluppo scoprono le caratteristiche di un bean (proprietà, metodi ed eventi) con un processo noto come *introspezione*
- Un bean non deve estendere nessuna particolare classe o implementare alcuna interfaccia
- I bean supportano l'introspezione in due modi
  1. Aderendo a specifiche regole, note come **design pattern**, nella denominazione di proprietà, metodi ed eventi; la classe *Introspector* sfrutta la *reflection* di java per scoprire le caratteristiche del bean
  2. Fornendo esplicitamente informazioni su proprietà, metodi ed eventi con una classe associata, che implementi l'interfaccia *BeanInfo*

JavaBeans - Caratteristiche

- Le *proprietà* esprimono l'aspetto e il comportamento di un bean
  - Possono essere modificate in fase di sviluppo o di esecuzione
- I bean usano gli *eventi* per comunicare con altri bean
- I metodi di un bean non sono diversi da altri metodi, e possono essere invocati da altri bean o da script
  - Normalmente, tutti i metodi pubblici sono esportati
- *La persistenza* permette ai bean di salvare e recuperare lo stato dei bean
  - Si usa la serializzazione di oggetti java

JavaBeans - Proprietà

- Le proprietà sono valori privati cui si accede tramite metodi getter e setter
- I nomi dei metodi getter e setter seguono specifiche regole, chiamate design pattern
- In questo modo, gli strumenti di sviluppo possono:
  - Scoprire le proprietà di un bean
  - Determinare i tipi di queste proprietà
  - Individuare un editor appropriato per ognuno dei tipi di proprietà
  - Visualizzare i valori delle proprietà (di solito in una tabella)
  - Alterare le proprietà (in fase di sviluppo)

JavaBeans
Metodi getter e setter

```
public class Persona implements java.io.Serializable {
  public Persona() { // costruttore
  }

  private String nome = null;
  public String getNome() { // metodo getter
    return nome;
  }
  public void setNome(String nome) { // metodo setter
    this.nome = nome;
  }

  private int eta = 0;
  public int getEta() { // metodo getter
    return eta;
  }
  public void setEta(int eta) { // metodo setter
    this.eta = eta;
  }
}
```


JavaBeans - Azione useBean

- I componenti JavaBean non sono altro che oggetti java che rispettano un preciso pattern di progettazione e denominazione
  - Il bean incapsula le sue proprietà dichiarandole private
  - Fornisce metodi pubblici di accesso (getter/setter) per leggere e modificare i loro valori
- Prima di poter accedere a un bean da una pagina JSP, è necessario identificarlo e ottenere un riferimento
- L'azione *useBean* tenta di recuperare un riferimento ad una istanza esistente
  - Il bean può essere stato creato precedentemente e posto nello scope sessione o application nell'esecuzione di una diversa richiesta
  - Il bean è istanziato nuovamente (in accordo all'attributo *class*) solo se un riferimento non viene trovato

JavaBeans - Azione useBean

- Si consideri il tag
  - `<jsp:useBean id="utente" class="miopackage.Persona" scope="session" />`
  - In questo esempio, l'istanza di *Persona* viene creata una sola volta e memorizzata con visibilità sessione
  - Se questo tag *useBean* viene incontrato nuovamente in una richiesta diversa, viene recuperato dalla sessione un riferimento alla istanza originale già creata

JavaBeans - Azione setProperty

- Modificare una proprietà di un componente JavaBean richiede l'uso del tag *setProperty*
- Per questo tag, serve identificare il bean, la proprietà da modificare e il nuovo valore da memorizzare:
  - `<jsp:setProperty id="utente" property="nome" value="Al" />`
- Il tag *useBean* opzionalmente può anche includere un corpo, che viene eseguito solo quando il bean viene istanziato
  - ```
    <jsp:useBean id="utente" class="miopackage.Persona"
    scope="session">
      <jsp:setProperty id="utente" property="eta" value="25" />
    </jsp:useBean>
    ```
    ```

JavaBeans - Azione setProperty

- Per elaborare i dati di un form, si possono far corrispondere i nomi delle proprietà di un bean con i nomi di elementi di input del form
  - Con un'unica istruzione, si analizzano tutti i valori che arrivano dal form html e si assegnano alle corrispondenti proprietà del bean
  - Questo processo, chiamato *introspezione*, è gestito dal motore jsp e implementato attraverso il meccanismo della java reflection
  - `<jsp:setProperty id="utente" property="*"/>`
- Se i nomi delle proprietà del bean non corrispondono agli elementi di input del form, allora le proprietà devono essere mappate esplicitamente, indicando il parametro:
  - `<jsp:setProperty id="utente" property="eta" param="campoEta" />`

JavaBeans - Azione getProperty

- Si può accedere al valore di una proprietà di un bean usando il tag *getProperty*
- Si specifica il nome del bean da usare (dal campo *id* del tag *useBean*) assieme al nome della *proprietà* che si vuole leggere
- Il valore viene visualizzato direttamente in output
  - `<jsp:getProperty id="utente" property="nome" />`

Reflection

- La reflection è una caratteristica molto potente del linguaggio Java che permette di ottenere informazioni su classi e interfacce direttamente in fase di run-time.
- È possibile determinare la classe a cui un oggetto appartiene, i suoi attributi, i costruttori e i metodi.
- I tool visuali utilizzati per manipolare i Java Beans utilizzano la reflection per analizzare a fondo le proprietà specifiche degli stessi beans.

EL

Expression Language

EL - Linguaggio per espressioni

- Nella versione 2.0 delle specifiche per le pagine JSP, sono state aggiunte delle istruzioni note come expression language statements (EL). Attraverso tali istruzioni è possibile arrivare, sovente, allo stesso risultato che si ottiene utilizzando gli elementi di scripting di base. La differenza è che la sintassi utilizzata dalle espressioni EL è più semplice.
- EL invocato attraverso il costrutto *${expr}*
- Un *identificatore* in EL fa riferimento a variabili JSP
- I parametri di richiesta, le intestazioni e i cookie sono accessibili tramite degli oggetti impliciti: *param*, *header* e *cookie*
  - *param["foo"]* (o *param.foo*) restituisce come stringa il valore associato al parametro di richiesta *foo*

EL - Componenti e collezioni

- I dati dell'applicazione di solito consistono di oggetti che aderiscono alle specifiche dei *JavaBeans*, o che rappresentano *collezioni* come liste, mappe o array
  - EL fornisce due *operatori*, *"."* e *"[]"*, per rendere facile l'accesso ai dati incapsulati in questi oggetti
  - EL segue ECMAScript nell'unificare il trattamento di *"."* e *"[]"*
  - L'operatore *"."* può essere usato come conveniente abbreviazione per accedere alle proprietà, quando il loro nome segue le convenzioni degli identificatori java
  - L'operatore *"[]"* permette un accesso più generale

EL - Operatori

- Sono disponibili gli operatori standard relazionali, aritmetici e logici
  - `==` o *eq*, `!=` o *ne*, `<` o *lt*, `>` o *gt*, `<=` o *le*, `>=` o *ge*
  - +, -, *, / o *div*, *%* o *mod*
  - `&&` o *and*, `||` o *or*, `!` o *not*
  - Operatore condizionale *? – A ? B : C*
  - Fornito inoltre l'operatore *empty*
- Esempio

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %> [...]
<c:if test="${book.price <= user.spendingLimit}">
  The book ${book.title} fits your
  budget!
</c:if>
```


EL - Oggetti impliciti

- Il linguaggio per espressioni JSP definisce un insieme di oggetti impliciti
- *pageContext*: il contesto della pagina jsp, che fornisce accesso a vari oggetti, tra cui:
  - *servletContext*: il contesto della pagina e di tutti i componenti della stessa applicazione
  - *session*: l'oggetto sessione per il client
  - *request*: la richiesta che attiva l'esecuzione della pagina jsp
  - *response*: la risposta restituita dalla pagina jsp

EL - Oggetti impliciti

- Diversi altri oggetti impliciti sono disponibili:
  - *param*: mappa un nome di parametro di richiesta al suo valore
  - *header*: mappa un nome di header della richiesta al suo valore
  - *cookie*: mappa un nome di cookie al suo valore
  - *initParam*: mappa un nome di parametro di inizializzazione al suo valore
  - *paramValues, headerValues*: per valori multipli, restituiscono array (es. *select* con attributo *multiple*)

Un esempio

```
<html>
   ...
   <body>
        <h1>
        Saluti Signor ${param.nome}!
        </h1>
   </body
</html>
```

JSTL - Usare una libreria di tag

- L'insieme di tag significativi che un container jsp può interpretare può essere esteso attraverso una *tag library*
  - Collezione di azioni che incapsulano funzionalità, da usare in una pagina jsp
- La direttiva *taglib* dichiara che la pagina usa una tag library
  - Identifica univocamente la tag library tramite una uri
  - Associa un prefisso di tag per distinguere l'uso delle azioni della libreria
  - `<%@ taglib uri="..." prefix="..." %>`
- **Suggerimento per gli utenti di Tomcat**
  - Copiare *jstl.jar* e *standard.jar* da *examples/WEB-INF/lib*
  - Anch'essi sono sviluppati sotto il progetto Jakarta

Alberto Ferrari 48

URI

- Uno Uniform Resource Identifier (URI, acronimo più generico rispetto ad "URL") è una stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, ecc.
- L'URL è un URI, o più comunemente chiamato indirizzo web.

Wikipedia

JSTL - JSP Standard Tag Library

- Organizzato in aree
  - Per identificare chiaramente l'area funzionale che coprono
  - Per dare a ciascuna area il suo spazioni di nomi

Area	Subfunction	Prefix
Core	Variable Support	c
	Flow Control	
	URL Management	
	Miscellaneous	
XML	Core	x
	Flow Control	
	Transformation	
I18n	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

JSTL - Azioni general-purpose

- L'azione *out* valuta una espressione e visualizza il risultato sull'oggetto *JspWriter* corrente
  - `<c:out value="${cart.numberOfItems}" />`
- Il linguaggio per espressioni (EL) era una caratteristica di JSTL
  - Ora è integrata appieno nella tecnologia jsp
  - Esteso per supportare funzioni
- EL può essere ora usato al posto di scriptlet
  - Scriptlet
    - `<p><%= book.getTitle() %></p>`
  - JSP 1.x + JSTL
    - `<p><c:out value="${book.title}" /></p>`
  - JSP 2.0
    - `<p>${book.title}</p>`

JSTL - Azioni general-purpose

- Il tag *set* imposta il valore di una variabile
  - Se la variabile non esiste, viene creata
  - La variabile con scope può essere impostata dall'attributo *value*:
  - `<c:set var="foo" scope="session" value="..."/>`
  - O dal corpo dell'elemento:
  - `<c:set var="foo">...</c:set>`
- Il tag *set* può anche impostare una proprietà di un oggetto
  - `<c:set value="value" target="objectName" property="propertyName"/>`
- *jsp:useBean* deve essere usato per rendere disponibile la variabile agli scriptlet…
  - ```
    <c:set var="bookId" value="${param.remove}"/>
    <jsp:useBean id="bookId" type="java.lang.String" />
    <% cart.remove(bookId); %>
    ```
- Per eliminare una variabile, c'è il tag *remove*
  - `<c:remove var="foo" scope="session"/>`
    ```

JSTL - Azioni condizionali

- Il tag *if* permette l'esecuzione condizionale del suo corpo, a seconda del valore dell'attributo *test*
  - `<c:if test="${cart.numberOfItems > 0}">...</c:if>`
- Il tag *choose* permette l'esecuzione condizionale di vari blocchi incapsulati nei suoi sotto-tag *when*
  - Esegue il corpo del primo tag *when* la cui condizione di *test* sia verificata
  - Se nessuna delle sue condizioni di *test* viene valutata a true, allora se presente viene eseguito il corpo del tag *otherwise*
  - ```
    <c:choose>
      <c:when test="${user.category=='trial'}">...</c:when>
      <c:when test="${user.category=='member'}" >...</c:when>
      <c:otherwise>...</c:otherwise>
    </c:choose>
    ```


JSTL - Azioni cicliche

- Il tag *forEach* permette di iterare su una collezione di oggetti
  - Si specifica la collezione tramite l'attributo *items*, l'elemento corrente è disponibile tramite una variabile denominata tramite l'attributo *var*
  - È supportato un gran numero di tipi di collezione
    - Tutte le implementazioni di *java.util.Collection* e *java.util.Map*
    - *Gli array di oggetti come pure gli array di tipi primitivi*
    - Le implementazioni di *java.util.Iterator* e *java.util.Enumeration*
    - Gli oggetti *java.lang.String,* se contengono liste di valori separati da virgola, per esempio: *"Monday,Tuesday,Wednesday,Thursday,Friday"*
  - ```
    <table>
      <c:forEach var="item" items="${cart.items}"><tr>
        <td>${item.name}</td>
        <td>${item.quantity}</td>
      </tr></c:forEach>
    </table>
    ```

JSP - Hello, user!

```
<html>
<head>
  <title>Hello user</title>
</head>
<body>

<i>Hello, ${param.user}!</i><br/>

</body>
</html>
```


JSP - Parametri della richiesta

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <title>Parameters example</title>
</head>
<body>

<i>Hello, ${param.user}!</i><br/>

<c:forEach var="paramName"
   items="${pageContext.request.parameterNames}">
  ${paramName} = ${param[paramName]}<br/>
</c:forEach>

</body>
</html>
```

JSP - Header della richiesta

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <title>Headers example</title>
</head>
<body>

<c:forEach var="headerName"
   items="${pageContext.request.headerNames}">
  ${headerName} = ${header[headerName]}<br/>
</c:forEach>

</body>
</html>
```


JSTL - Azioni SQL

- Il tag *query* è usato per eseguire query SQL, che restituiscono insiemi di risultati (JDBC result sets)
  - L'interfaccia *Result* è usata per recuperare le informazioni sul risultato della query
  - Per query sql parametrizzate, si usa un tag *param* innestato
  - ```
    <sql:query var="users">
      SELECT * FROM users WHERE country = ' ${country}'
    </sql:query>
    <table>
      <c:forEach var="row" items="${users.rows}"><tr>
        <td><c:out value="${row.lastName}"/></td>
        <td><c:out value="${row.firstName}"/></td>
        <td><c:out value="${row.address}"/></td>
      </tr></c:forEach>
    </table>
    ```
    ```

JSTL - Azioni SQL

- Il tag *setDataSource* permette di impostare la sorgente di dati, ossia di collegarsi al database
  - Si possono fornire i parametri per il *DriverManager*
  - `url[,[driver][,[user][,password]]]`
  - `jdbc:mysql://localhost/mydb,org.gjt.mm.mysql.Driver`
- Il tag *update* è usato per aggiornare righe di un database
- Il tag *transaction* è usato per eseguire una serie di istruzioni sql in maniera atomica (transazione)

JSTL Funzioni

Area	Function	Tags	Prefix
Functions	Collection length	`length`	

	String manipulation	`toUpperCase, toLowerCase`
		`substring, substringAfter,`
		`substringBefore`
		`trim`
		`replace`
		`indexOf, startsWith, endsWith,`
		`contains, containsIgnoreCase`
		`split, join`
		`escapeXml`
			`fn`

JSP vs. Servlet

- JSP → Aggiungere un po' di Java a HTML
- Servlet → Aggiungere un po' do HTML a Java
- In pratica è consigliabile utilizzare i servlet quando l'applicazione è complessa
- Tomcat converte le JSP in Servlet …