Lezione 12 - Extension Tag

Esempi di Tag

Tag senza corpo
```
<jsp:useBean id="bean" class="MyBean" scope="session" />
```

Tag con corpo

```
<jsp:forward page="urlSpec">
<jsp:param name="nome" value="valore"/>
</jsp:forward>
```

```
<mylib:traudci lang="it">
Hello world!
</mylib:traduci>
```

Librerie di Tag

Una libreria di tag è una ulteriore potenzialità che permette agli sviluppatori delle servlet di fornire nuovi tag potenti e facilmente utilizzabili per la realizzazione di pagine JSP. In questo modo è possibile scrivere una pagina JSP anche a chi non conosce il linguaggio di programmazione Java.
Per aggiungere un nuovo tag dobbiamo fornire le seguenti cose:

1. la pagina JSP che utilizza il tag;
2. un file di configurazione che illustra gli aspetti sintattici (ma non solo) del nuovo tag;
3. il codice Java che indica le operazioni da compiere.

Vediamo prima di tutto come si utilizzano i nuovi tag dentro una pagina jsp. Nella prima riga, utilizziamo la direttiva **`taglib`** per indicare dove si trova il file di configurazione che descrive i tag presenti nella pagina. La direttiva ha due attributi: uri indica il file di configurazione (`/esempio`), `prefix` indica il prefisso con cui vengono indicati i tag di questa libreria (`ex`). Una volta dichiarata nella pagina la libreria, possiamo utilizzare tutti i tag che sono definiti nel file di configurazione.

```
<%@ taglib uri="/esempio" prefix="ex" %>
<HTML><HEAD>
<TITLE>Esempio Tag</TITLE></HEAD>
<BODY>
Testo libero.
<h1><ex:mytag /></h1>
Altro Testo
</BODY>
</HTML>
```

Ora vediamo il file di configurazione disponibile all'url "/esempio":
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library 1.1//EN" "http://java.sun.com/j2ee/dtds/web-
jsptaglibrary_1_1.dtd">
<taglib>
<tlibversion>1.0</tlibversion>
<jspversion>1.1</jspversion>
```

```
<shortname>esempio</shortname>
<info>Libreria di tag.</info>
<tag>
<name>mytag</name>
<tagclass>provatag.MyTag</tagclass>
<bodycontent>empty</bodycontent>
<info>Esempio</info>
</tag>
</taglib>
```

Per fare in modo che nell'url "/esempio" si trovi il contenuto di sopra possiamo configurare il file `web.xml` nel seguente modo:

```
...
<taglib>
<taglib-uri>/esempio</taglib-uri>
<taglib-location>/WEB-INF/tlds/esempio.tld</taglib-location>
</taglib>
```

La configurazione nel file web.xml non è necessaria, ma facilita l'installazione. Infatti anche se le librerie risiedono in cartelle diverse da quella data, non è necessario cambiare il codice delle pagine JSP, ma solo il file di configurazione.

Vediamo ora come programmare il gestore del tag. Il metodo `doStartTag` del gestore del tag verrà richiamato dal motore delle servlet in corrispondenza di ogni tag di apertura, `<ex:mytag>` in questo caso, mentre il metodo `doEndTag` verrà richiamato ad ogni tag di chiusura, `</ex:mytag>` nel nostro caso. Nell'esempio che segue, verrà semplicemente inviata alla JSP del testo per indicare quando viene invocato il metodo `doStartTag` e `doEndTag`. In pratica è come se i `<ex:mytag>` venisse sostituito con `"Start tag"`, mentre `</ex:mytag>` con `"End tag"`.

```
package provatag;
import java.io.IOException;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class MyTag extends TagSupport
{
public int doStartTag() throws JspTagException
{
try
{
pageContext.getOut().write("Start tag");
}
catch (IOException e)
{
throw new JspTagException("Error: write to JSP out");
}
return EVAL_BODY_INCLUDE; // return SKIP_BODY;
}

public int doEndTag() throws JspTagException
{
try
{
pageContext.getOut().write("End tag");
}
catch (IOException e)
{
throw new JspTagException("Error: could not write to JSP out");
}
```

```
return EVAL_PAGE; // return SKIP_PAGE;
}
}
```

file configurazione della libreria

- `<taglib>`: inizio della definizione della libreria, tutti i tag seguenti devono essere nidificati dentro questo tag;
- `<tlibversion>`: versione delle librerire;
- `<jspversion>`: versione delle JSP;
- `<shortname>`: nome della libreria;
- `<info>`: descrizione della libreria

tag

Per ogni tag definito nella libreria ci dovrà essere:

- `<tag>`: inizio della definizione del tag, tutti i tag seguenti devono essere nidificati dentro questo tag.
- `<name>`: il nome del tag, quello che useremo nella pagina JSP dopo i ':';
- `<tagclass>`: la classe Java che definisce il comportamento del tag;
- `<body>`: il tipo di trattamento del testo nel corpo del tag, cioè dell'eventuale testo che si trova tra `<ex:mytag>` e `</ex:mytag>`:
  - `empty`: nessun corpo, eventualmente il corpo viene ignorato;
  - `JSP`: il corpo è valutato prima dal motore delle servlet e poi eventualmente dal gestore del tag;
  - `tagdependent`: il corpo è valutato solo dal gestore del tag, l'eventuale codice JSP (scriplets, esepressioni) **non** viene valutato.
- `<info>`: una descrizione libera sul tag;
- `<teiclass>`: eventuale classe che definisce le eventuali variabili di scripting definite dal tag stesso (vedi esempio sotto);

attributi

Ogni tag definito può avere 0, 1 o più attributi, un attributo ha la stessa sintassi degli attributi html. Es `<ex:mytag attributo="valore">`. Gli attributi possono essere definiti solo sul tag di apertura. Per ogni attributo ci dovrà essere:

- `<attribute>`: inizio della definizione del singolo attributo, tutti i tag seguenti devono essere nidificati dentro questo tag:
- `<name>`: il nome dell'attributo ('attributo' nell'esempio di prima);
- `<required>`: true se l'attributo è obbligatorio, false atrimenti.
- `<rtexpvalue>`: se il valore dell'attributo è statico (deciso al momento della scrittura della pagina JSP o dinamico cioè deciso al momento dell'esecuzione.

Dal punto di vista della classe che gestisce gli attributi il problema si riduce semplicemente all'implementazione di un metodo **`set`** per ogni attributo definito. Di solito in aggiunta al metodo `set` viene implementato anche il metodo `get`. Ad esempio per definire un attributo nome, basterà che la classe implementi il metodo `void setNome(String nome)`. Bisogna prestare attenzione ai tipi: tutti i metodi `set` hanno come parametro una stringa, quindi se l'attributo Java (variabile d'istanza) ha un tipo diverso, la conversione da Stringa al tipo corretto dev'essere fatta nel metodo `set`. Ovviamente nel file JSP gli attributi sono tutti di tipo stringa.

handler dei tag

Il significato del tag viene definito tramite una classe che lo gestisce. A seconda del tipo di funzionalità richieste bisognerà implementare i seguenti metodi della classe `TagSupport`:

funzionalità tag	metodi addizionali da implementare	valore ritornato `doStartTag`
senza corpo	`doStartTag, doEndTag`	`SKIP_BODY`
con attributi	`get/set` attributi	
con corpo	`doStartTag`, `doEndTag`	`EVAL_BODY_INCLUDE`
modifica body	`doInitBody`, `doAfterBody`	`EVAL_BODY_TAG`

body

Vediamo ora un esempio di tag con corpo. Questo tag trasforma il testo del corpo in maiuscolo.

```
<%@ taglib uri="/esempio" prefix="ex" %>
<HTML><HEAD><TITLE>Esempio</TITLE></HEAD>
<BODY>
Testo statico
<ex:captag>corpo del tag</ex:captag>
Altro testo statico
</BODY>
</HTML>
```

Ecco la classe che implementa questo:

```
package provatag;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.util.*;

public class CapTag extends BodyTagSupport {
public int doStartTag() throws JspTagException {
return EVAL_BODY_TAG;
}

public int doAfterBody() throws JspTagException {
BodyContent bodyContent = getBodyContent();
if (bodyContent != null)
{// Do nothing if there was no body content
String output = bodyContent.getString().toUpperCase;
try
{
bodyContent.getEnclosingWriter().write(output);
}
catch (java.io.IOException ex)
{
throw new JspTagException("Fatal IO error");
}
}
return EVAL_PAGE;
```

```
}

public int doEndTag() throws JspTagException {
return EVAL_PAGE;
}
}
```

Tag e motore delle servlet, ovvero ciclo di vita del tag

Per capire a fondo come funzionano i tag bisogna conoscere come interagiscono con il motore delle servlet.

Il motore delle servlet richiama nell'ordine `setPageContext`, `setParent` e i metodi per impostare gli eventuali attributi prima di chiamare il metodo `doStartTag`. Il motore delle servlet ci garantisce anche che il metodo release verrà richiamato prima della chiusura della pagina.
Una tipica sequenza di invocazione è la sequente:

```
SomeTag tag = new someTag();
tag.setPageContext(...);
tag.setParent(...);//vedi tag nidificati
tag.setAttribute1(value1);
tag.setAttribute2(value2);
tag.doStartTag();
tag.doEndTag();
tag.release();
```

Il metodo release del gestore del tag dovrebbe resettare il proprio stato e liberare ogni eventuale risorsa usata. Di solito non è necessario ridefinire questo metodo (vedi esempi precedenti).

Se un tag è derivato da `BodyTagSupport`, ci sono dei metodi aggiuntivi: `setBodyContent`, `doInitBody` e `doAfterBody`. Questi metodi ci permettono di accedere al corpo del tag che è composto da tutto quello che si trova tra lo start tag e l'end tag.

L'oggetto `BodyContent` è una sottoclasse di `JspWriter`, che è l'oggetto usato internamente dalla variabile `out` delle JSP. L'oggetto `BodyContent` è disponibile attraverso la variabile `bodyContent` nei metodi `doInitBody`, `doAfterBody` e `doEndTag`. Questo è importante perché questo oggetto contiene i metodi che possono essere usati per leggere, scrivere cancellare e recuperare il contenuto e incorporarlo nel `JspWriter` originele durante il metodo `doEndTag`.

Il metodo `setBodyContent` crea un contenuto per il corpo e aggiunge al gestore del tag.Il metodo `doInitBody` è chiamato una sola volta prima della valutazione del corpo del tag è viene usato di solito per l'inizializzazione che dipende dal contenuto del tag stesso. Il metodo `doAfterBody` è chiamato dopo che il corpo del tag è stato valutato: se `doAfterBody` ritorna `EVAL_BODY_TAG`, il corpo e valutato di nuovo, altrimenti se viene ritornato `SKIP_BODY` la valutazione del corpo del tag è terminata.

Una tipica sequenza di chiamate fatte dal motore delle Servlet ai metodi di `BodyTagSupport` potrebbe essere la seguente:

```
tag.doStartTag();
out = pageContext.pushBody();
tag.setBodyContent(out);
// perform any initialization needed after body content is set
tag.doInitBody();
tag.doAfterBody();
// while doAfterBody returns EVAL_BODY_TAG we
// iterate the body evaluation
...
```

```
tag.doAfterBody();
tag.doEndTag();
tag.pageContext.popBody();
tag.release();
```

tag con corpo

Avere un tag che valuta il corpo è una possibilità molto interessante. Tra le varie possibilità offerte c'è anche quella di iterare sul corpo stesso: questo ci permette in maniera semplice di manipolare tutte quelle strutture dati che sono enumerazioni: enumeration, array, ResultSet, Collection etc.
Per iniziare a usare il corpo in un tag, dobbiamo ricordarci di fare due cose:

1) impostare il `<bodyContent>` del tag a JSP o tagdependent. Per esempio

```
<tag>
   ...
   <bodycontent>JSP|tagdependent</bodycontent>
</tag>
```

2) derivare il gestore del tag dalla classe `BodyTagSupport`, che è la classe di supporto per l'interfaccia `BodyTag`. In teoria non servirebbe implementare nessun altro metodo dell'interfaccia in quanto già presenti nella classe base. In questo modo si ottiene un tag che lascia inalterato il corpo del tag stesso.
Il modo di implementare il gestore del tag dipende da come il gestore ha bisogno di interagire col corpo. Interagire significa che il gestore legge o modifica il contenuto del corpo oppure produce più valutazioni del corpo stesso.

Se il gestore del tag interagisce con il corpo, il valore di ritorno del metodo `doStartTag` vine interpretato nel seguente modo:

- `SKIP_BODY`: il motore delle servlet non valuta il testo del corpo;
- `EVAL_BODY_TAG`: questo valore può essere ritornato solo dalla classe `BodyTagSupport`. (Se il gestore non ha bisogno di interagire con il corpo dovrebbe implementare solo l'intergaccia Tag o derivare da `TagSupport`).;
- `EVAL_BODY_INCLUDE`; questo valore può essere ritornato solo dalla classe `TagSupport`.

Esempio:

```
<tag>
  <name>loop</name>
  <tagclass>provatag.LoopTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>enum</name>
    <required>true</required>
    <rtexpvalue>true</rtexpvalue>
  </attribute>
</tag>
```

Nel seguente esempio vediamo il codice del gestore del tag che visualizza tutti i parametri di una richiesta HTTP: `import java.util.Enumeration;`

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.IOException;

public class LoopTag extend BodyTagSupport
```

```
{
  java.util.StringTokenizer enum =null

  public void setEnum(String enum)
  {
    this.enum = new java.util.StringTokenizer(enum,",;");
  }
  public int doStartTag() throws JspException
  {
     if (enum != null && enum.hasMoreElements())
     {
        pageContext.setAttribute("name", enum.nextToken());
        return EVAL_BODY_TAG;
     }
     return SKIP_BODY;
   }
   public void doAfterBody() throws JspException
   {
     if (enum!=null && enum.hasMoreElements())
     {
        pageContext.setAttribute("name", enum.nextToken());
        return EVAL_BODY_TAG;
     }
     return SKIP_BODY;
   }
   public int doEndTag() throws JspException
  {
     try
     {
      pageContext.getOut().print(bodyContent.getString());
      return EVAL_PAGE;
     }
     catch (IOException ioe)
     {
        throw new JspException(ioe.getMessage());
     }
   }

}
```

Una pagina JSP che usa il tag appena creato potrebbe essere:

```
<%@ taglib uri="/esempio" prefix="ex" %>
<html>
<body>
<h1>Esempio loop</h1>
<table border="2">
<tr>
 <th>Nome</th>
 <th>Valore</th>
 <ex:loop enum="name1,name2,name3" >
<tr>
  <td><%= pageContext.getAttribute("name") %></td>
  <td>
<%= request.getParamter((String)pageContext.getAttribute("name"))
%></td>
</tr>
```

```
</ex:loop>
</table>
</body>
<html>
```