Tools for model-view separation
1. Tags

Juan M. Gimeno, Josep M. Ribó

February, 2008

Contents

Tools for model-view separation: tags and beans

1-Tags. Contents

Introduction. The problem

With the elements we have discussed up to now we are able to design simple web applications with the following tools:

1. **Servlets**
   - Cumbersome to program
   - Difficult to read
2. **JSP pages**

   More comprehensible/easier to design than Servlets, but.....

   (a) Java code within an html page: Inelegant. Difficult to read
   (b) Presentation aspects are intertwined with logic and model aspects
   (c) Lack of abstraction (e.g., database name hardcoded in the jsp page)

How these problems may be solved?

Introduction. Solution guidelines

(a) **Java code within a html page**

This can be solved using **java beans** and **tag libraries** (presented in **this** module)

(b) **Presentation mixed up with business aspects**

This can be improved using:

- **Java beans** (presented in **this** module)
- **Tag libraries** (presented in **this** module)
- **Design patterns** like **Model-view-controller** (presented in the **Design patterns** module)

(c) **Lack of abstraction**

This can be improved by using desing patterns and JNDI (presented in the **Design patterns** module

What are personalized tags

Tags defined by the web application designer that can be used in a JSP page in order to get a specific behaviour

example.jsp:

```
<html>

<util:getValue  attribute="name" />

</html>
```

The tag `util:getValue` may get the value of an attribute called `name` that has been stored in the session scope

(This is just an example. The tag `util:getValue` has not been defined and does not exist)

Tags are gathered in **tag libraries**
A **tag library** is a set of tags that perform some related functionality

How to define and use a library of personalized tags

1. **Create a `.tld` file that describes the tag library** and provides for each tag:

   - A name
   - A list of attributes (if any)
   - The name of a Java class that contains the code associated to the tag

2. **Implement for each tag a class (subclass of `TagSupport`)** that redefines the methods `doStartTag` and/or `doEndTag` and/or `release`

   These methods describe the tag behaviour

3. **Insert to the deployment descriptor (web.xml) an entry for each tag library**

   This entry provides:

   - A name for the tag library in that application
   - The name and location of the tag library descriptor (.tld file)

4. **Use the tag library in any JSP page by:**

- Adding a `taglib` directive into the JSP page which specifies:
  - The name of the tag library and
  - The prefix that will be used in that page to refer to the tags of that library
- Calling any tag of the library within the JSP page

Example: Tag library to store the text of a form parameter

Example location: beans+tags/examples/ex1.1

Goal:

If the submission of an html form with some text fields generates an error (e.g., a text field has been filled up incorrectly) the form should be returned to the client *with the text written by him/her* (i.e., in order that he/she should not have to fill it again)

To achieve this goal we propose the following tag:

```
<html:parameterValue property="email"/>'
```

This tag returns the value of the text field `email` that has been filled by the client in an html form. If the client has not filled this text field, the tag returns "" (the empty string)

In the following slides we will show how to create a tag library that contains this tag

1. Creation of the tag library descriptor (.tld)

- It is a XML document (`tagLibDesc.tld`) located at

  `WEBAPPROOT/WEB-INF/tlds/tagLibDesc.tld`

  The location may be different. However, this is the most usual one

- This document describes each tag of the library being defined

- Each tag is described in terms of:
  - Its name
  - The class that implements it (it should be a subclass of `TagSupport`)
  - Its attributes

Document `html.tld`

```
<taglib>
  <tlibversion>0.0</tlibversion>
  <jspversion>1.2</jspversion>
  <shortname>HTML Library</shortname>
  <info>Tags to support html form creation</info>

  <tag>
    <name>parameterValue</name>
    <tagclass>taglibs.html.ParameterValueTag</tagclass>
    <bodycontent>empty</bodycontent>
    <description>
       This recalls the last value of the form parameter
       given as attribute value
     </description>
     <attribute>
        <name>property</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
        <description>
           The attribute property
           gets the parameter name
         </description>
      </attribute>
   </tag>
</taglib>
```

Document elements:

- `<tag>...</tag>`

  Description of a tag of the library

  One block `<tag>...</tag>` for each tag in the library

  - `<tagclass>...</tagclass>`
    Name of the class that implements the tag behaviour
    This class should extend `TagSupport` (see below)

  - `<bodycontent>...</bodycontent>`
    Useful for tags with body (see ***). If the tags cannot have body, it has the value `empty`:
    `<bodycontent>empty</bodycontent>`

- `<attribute>...</attribute>`
  Description of a tag attribute
  One block `<attribute>...</attribute>` for each tag attribute
  * `<name>...</name>`
    Attribute name
  * `<required>...</required>`
    True if it is an attribute required for the tag
  * `<rtexprvalue>...</rtexprvalue>`
    True if the attribute accepts a run-time value (e.g., a JSP scriptlet)
    Example:
    `<html:parameterValue property="<%=propertyValue%>"/>'`

2. Implement a subclass of TagSupport

- This should be located in a `.class` file within the `WEB-INF/classes` or `WEB-INF/lib` directories

- The following methods may be redefined:

  - `doStartTag`
    This method is executed when the beginning of the tag is encountered
    `<html:parameterValue....`
    If it returns `SKIP_Body`, it means that the tag body should be ignored, in the case that it exists

  - `doEndTag`
    This method is executed when the end of the tag is encountered
    `.../>` or `</html:parameterValue>`
    It is redefined if some behaviour is needed when the end of the tag is reached
    The return value `EVAL_PAGE` means that the servlet container should continue with the process of the JSP page

  - `release`
    It is called before `doEndTag`. Useful to release the resources used by the tag (e.g., connection to a database)

```
package taglibs.html;

import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

public class ParameterValueTag extends TagSupport{
   private String property;

   public void setProperty(String property) {

    this.property = property;
   }

   public int doStartTag() throws JspException{

    ServletRequest req = pageContext.getRequest();
    String value = req.getParameter(property);

    try {
      if (value == null) pageContext.getOut().print("");
      else pageContext.getOut().print(value);
    }
    catch(java.io.IOException ex) {
       throw new JspException(ex.getMessage());
    }
    return SKIP_BODY;
  }
}
```

Setting the attributes

- There must be a `setNameOfAttribute(..)` method for each tag attribute (e.g., `property`):

```
public void setProperty(String property) {

 this.property = property;
}
```

- It is guaranteed that before the invokation of `doStart()` the servlet container will set the tag attributes by means of a call to the corresponding `setNameOfAttribute(..)` method using the values given to those attributes at the JSP page

Access to the page context

The methods of `ParameterValueTag` may access the page context by means of the attribute `pageContext`.

This attribute is inherited from the `TagSupport` class:

```
protected  PageContext pageContext;
```

By means of the `PageContext` operations, the `doStartTag()` method may access the date needed in order to perform its behaviour:

- `Object findAttribute(String attribName)`

  Searches for an attribute with id. `attribName` which has been defined in some scope (page, request, session, application)

- `Object getAttribute (String attributeName, int scope)`

  Searches for an attribute with id. `attribName` which has been defined in the scope `scope`

  Four scopes:

  - `PageContext.APPLICATION_SCOPE`
  - `PageContext.SESSION_SCOPE`
  - `PageContext.REQUEST_SCOPE`
  - `PageContext.PAGE_SCOPE`

- `void setAttribute (String attribName, Object val)`

- `void setAttribute (String attrNam, Object val , int scope)`

  The corresponding operations to set the attributes

- `JspWriter getOut()`

  Returns the output flow to send data to the client

- `ServletRequest getRequest()`

  Returns the request associated to this page

- `ServletResponse getResponse()`

  Returns the response object that is to be sent to the client

- `ServletContext getServletContext()`

  Returns the application object

- `HttpSession getSession()`

  Returns the session to which this request is associated

- `void forward(String path)`

  Redirects the current request/response to another service (servlet, jsp page...)

See the `ContextPage` specification (in J2EE API) for a complet description of its operations

3. Add to the web.xml tag support

```
<taglib>

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

</taglib>
```

4. JSP page

```
<%@ taglib uri='WEB-INF/tlds/html.tld' prefix='html'%>

<html>
<form action="regpage.jsp" method="get">

<table border="0" cellpadding="3" cellspacing="0">

<tr valign="top">
   <td> <b>Nom:</b> </td>
   <td><input name="nom" type="text" size=50
              value='<html:parameterValue property="nom"/>'>
    </td>
</tr>
<tr valign="top">
   <td> <b>e-mail:</b> </td>
   <td><input name="email" type="text" size=30
              value='<html:parameterValue property="email"/>'>
</td>
</tr>
<tr valign="top">
   <td>   </td>
   <td>
       <input type="submit" value="Enviar"></textarea>
   </td>
</tr>
</table></form></html>
```

Example: Tags to manage java beans

Example location: beans+tags/examples/ex1.2

This example proposes a simplified implementation of the standard tags:

- `useBean`
- `setProperty`
- `getProperty`

These tags are defined in section **2- Java beans** (of this module)

Tags with body

****TO BE COMPLETED

JSP actions

JSP defines within its specification some tags called **JSP actions**

It is not necessary to reference/declare any library in order to use them

- `<jsp:useBean>`
- `<jsp:setProperty>`
- `<jsp:getProperty>`
- `<jsp:include>`
- `<jsp:forward>`
- `<jsp:param>`
- `<jsp:plugin>`

Standard JSP tags. Examples of use

- `<jsp:useBean>`, `<jsp:setProperty>` and `<jsp:getProperty>`

  They are related to java beans and are presented in section **2-Java beans** of this module

- `<jsp:forward>`

```
<jsp:forward page="aux.jsp">
   <jsp:param name="customer" value="Pepet"/>
</jsp:forward>">
```

  This performs a request to the page `aux.jsp` with one parameter named `customer` with value `pepet`

  The remainder of the calling page is not processed after the forward to `aux.jsp`

- <jsp:include>

```
<jsp:include page="aux.jsp" flush="true">
   <jsp:param name="customer" value="Pepet"/>
</jsp:forward>">
```

If this code is included in the page callingP.jsp the following steps are followed:

1. A request to aux.jsp with parameter customer=pepet is performed by the JSP container
2. The response from the request to aux.jsp is included verbatim into the calling page callingP.jsp
3. The processing of callingP.jsp is continued by the JSP container

jsp:include also works if the included page is a static one (in that case, the contents of that page is included into the calling one verbatim)

Notice that this behaviour is different from the page directive include which cannot deal with dynamic page inclusions

JSTL: The JSP Standard Tag Library

- The **JavaServer Pages Standard Tag Library (JSTL)** is a tag libary that extends the JSP specification
- It provides a collection of tags that encapsulate functionality which is common to many JSP applications
- It has been specified following the Java Community Process
- The current version is JSTL-1.2 (since May, 2006)

The JSP Standard Tag Library (JSTL)

The JSTL is constituted by:

- **Four libraries:**

Library	URI	Prefix
Core (General purpose tasks: control-flow, URL manipulation, imports...)	http://java.sun.com/jstl/core	c
XML processing	http://java.sun.com/jstl/xml	x
I18N formatting (Internationalization and formatting actions)	http://java.sun.com/jstl/fmt	fmt
Database access	http://java.sun.com/jstl/sql	sql

- **An expression language:**

  It allows the construction of expressions constituted by: **literals, operators, function calls** and **implicit objects**

JSTL: The expression language (EL)

JSTL defines a small language to evaluate expressions.

An expression is shown in the following way:

```
${expression}
```

Examples:

```
${1+2+4}
${a==222}
${a<=222}
${param.nom}
```

JSTL: The expression language (EL)

Elements that may come up in an expression:

- **Literals**

  Booleans (true and false); integers; floating points; strings; null

- **Operators**

  - Arithmetic: +, -, *, div (/), mod (%)
  - Logical: and (&&), or (——), not (!)
  - Relational: lt (<), le (<=), gt (>), ge (>=), eq (==), ne (! =)
  - empty (Prefix operator to know if a value is null or empty)
  - Access to collections: ., []

- **Variables**

  (their value is searched for at different scopes: page, request, session, application).

- **Implicit objects (or implicit variables)**

  They provide information about the request, page context, accessible variables and initialization parameters

JSTL: The expression language (EL). Implicit objects

Certain objects of the application may be referenced by means of implicit variables (built-in).

They provide information about the request,page context, accessible variables and initialization parameters.

Implicit objects have a java type.

- **Accessible variables**

  The following implicit objects return the collection of variables accessible from the page at different scopes.

  - *pageScope* (type=`java.util.Map`). A collection of all page scope variables.
  - *requestScope* (type=`java.util.Map`). A collection of all request scope variables.
  - *sessionScope* (type=`java.util.Map`). A collection of all session scope variables.
  - *applicationScope* (type=`java.util.Map`). A collection of all application scope variables.

- **Request**

  The following implicit objects return some issues related to the request that has activated that jsp page.

  - *param* (type=`java.util.Map`). A collection of all request parameter values, considered as strings (just one value for each parameter).
  - *paramValues* (type=`java.util.Map`). A collection of all request parameter values, considered as a string array value for each parameter.
  - *header* (type=`java.util.Map`).
  - *headerValues* (type=`java.util.Map`).
  - *cookie* (type=`java.util.Map`).

- **initParam** (type=`java.util.Map`).

- **pageContext** (type=`javax.servlet.jsp.PageContext`).

  The java class `PageContext` provides access to several elements related to the jsp page (e.g., pageContext.request; pageContext.response; pageContext.session; pageContext.servletContext).

The JSP Standard Tag Library: The Core library

The core library is a part of the JSP standard tag library

This library presents some tags to express control-flow actions (conditionals, loops, *calls* to other pages, parameters of these calls....)

Some tags included in the core library:

- `c:out`
- `c:if`
- `c:forEach`
- `c:choose`
- `c:when`
- `c:otherwise`
- `c:set`
- `c:redirect`
- `c:param`

Further tag actions have been included in this library: `c:catch`, `c:forTokens`, `c:import`, `c:remove`, `c:url`

The JSP Standard Tag Library: Exemple 1

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

<html>
<body>

Static value:
<p><c:out value="value= 22222"/>  
<p>Dynamic value (computed by the JSP container)
<p><c:out value="value= ${1+2+3}"/>  

<p>Value of a variable
   (again, computed by the JSP container):

<c:set var="variable" value="${444}" />
<p><c:out value="variable=${variable}" />  
</body>
</html>
```

The JSP Standard Tag Library. Example 2

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

<html>
<body>
Values of the form parameters:

Nom: <c:out value="${param.nom}"/><br>
e-mail: <c:out value="${param.email}"/><br>
Receptor: <c:out value="${param.receptor}"/><br>
Contingut vomitera: <c:out value="${param.vomitera}"/><br>
Extra bilis: <c:out value="${param.bilis}"/><br>
Afegits:<br>
  <c:forEach items="${paramValues.addenda}" var="current">
     <c:out value="${current}"/>  
  </c:forEach>
</body>
</html>
```

Getting a JSTL implementation

An implementation of the JSTL can be found at:

http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

In order to use this distribution of the JSTL in a web applications:

1. Copy the JAR files (jstl.jar and standard.jar) in WEB-INF/lib

2. Import JSTL into your pages with the following directives:

   - **Core library:**

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

   - **XML library:**

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

   - **FMT library:**

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

   - **SQL library:**

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

   - **Function library:**

     ```
     <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"
     ```