Gestion des tags

Guillaume DEFRAIN

Tag JSP

Pourquoi définir de nouveaux tags :

- Eviter le mélange HTML et JAVA dans les JSP
- Permet une meilleure réutilisation du code
- Permet de créer des bibliothèque utilisable par plusieurs projet

Tag JSP

Pourquoi définir de nouveaux tags :

- Les balises personnalisées sont adaptées pour supprimer du codes Java inclus dans les JSP est le déporter dans une classe dédiée
- La classe dédiée est comparable à un Java Bean qui implémente une interface particulière

- Caractéristiques intéressantes des tags
  accès aux objets de la JSP (HttpResponse)
  peuvent recevoir des paramètres envoyés à partir de la JSP
  peuvent avoir un corps qu'ils manipulent ou pas

Tag JSP

Qu'est ce qu'un tag JSP

- Un tag JSP est une simple balise XML associée à une classe Java
- A la compilation d'une JSP, les balises sont remplacées par le résultat des classes Java

```
<prefixe:nomDuTag attribut1="valeur" attribut2="valeur" >
Corps du Tag
</prefix:nomDuTag>
```

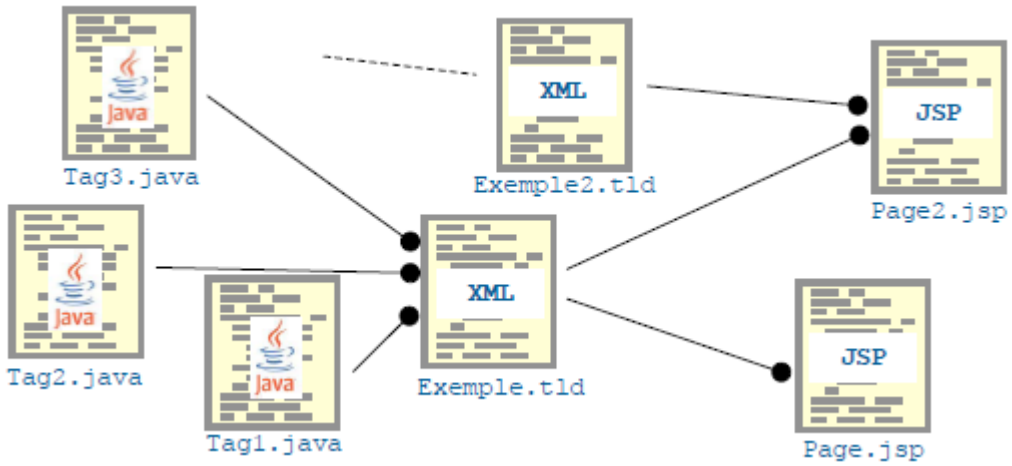
Nous retrouvons les éléments suivants

- préfixe : permet de distinguer les différents tags utilisés
- nomDuTag : identifie le nom du tag de la librairie « préfixe »
- Un certain nombre de couples d'attribut/valeur (peut-être au nombre de zéro)
- Un corps (peut ne pas exister)

Tag JSP

Qu'est ce qu'un tag JSP

- Un tag personnalisé est composé de plusieurs éléments
  - Tag Library Descriptor (TLD) ou description de la bibliothèque de balises
    - effectue le mapping entre les balises et les classes Java
    - Fichier de type XML
    - Le format porte obligatoirement l'extension « tld »
  - Une classe appelée « handler » pour chaque balise qui compose la bibliothèque



Tag JSP

Interface Tag

Propose 6 méthodes qui sont appelées (pour 5 d'entre elles séquentiellement à l'évaluation du tag :

- setParent(Tag)
- setPageContext(PageContext)
- doStartTag() : méthode appelée lors du passage sur la balise ouvrante. Son retour peux valoir :
  - Tag.SKIP_BODY : le corps de la balise est ignorée
  - Tag.EVAL_BODY_INCLUDE : le corps est évalué et écris dans le JSPWriter de la page
- doEndTag () : méthode appelée lors du passage sur la balise fermante. Son retour peut valoir :
  - Tag.EVAL_PAGE : le reste de la page est évalué
  - Tag.SKIP_PAGE : le reste de la page est ignorée
- release () :

Pour plus de facilité on utilisera une implémentation par défaut : javax.servlet.jsp.tagext.TagSupport

Tag JSP

Fichier Tld

- Le fichier de description de la bibliothèque de tags décrit une bibliothèque de balises
- Les informations qu'il contient concerne la bibliothèque de tags et concerne aussi chacun des balises qui la compose
- Doit toujours avoir l'extension « .tld »
- Le format des descripteurs de balises personnalisées est défini par un fichier DTD
- La balise racine du document XML est la balise <taglib>
- La première partie du document TLD concerne les informations
- de la bibliothèque
- <tlib-version> : version de la bibliothèque (obligatoire)
- <jsp-version> : version des spécifications JSP (obligatoire)
- <short-name> : nom de la bibliothèque (obligatoire)
- <description> : description de la bibliothèque (optionnelle)
- <tag> : il en faut autant que de balises qui composent la bibliothèque

Tag JSP

- Chaque balise personnalisée est définie dans la balise <tag>

- La balise <tag> peut contenir les balises suivantes
- <name> : nom du tag, unique dans la bibliothèque (obligatoire)
- <tag-class> : nom de la classe du handler du tag (obligatoire)
- <body-content> : type du corps du tag (optionnelle)
  - JSP : le corps du tag contient des tags JSP
  - tagdependent : interprétation du corps est faite par le tag
  - empty : le corps doit obligatoirement être vide
- <description> : description du tag (optionnelle)
- <attribute> : décrit les attributs. Autant qu'il y a d'attributs

Tag JSP

Utilisation d'une bibliothèque

```
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="tag-prefix" %>
// ou si le fichier tld est dans le Jar :
<%@ taglib uri="/WEB-INF/lib/taglib.jar" prefix="tag-prefix" %>
```

Toutefois, il est préférable de définir la taglib dans le fichier **web.xml** de l'application web, avec le code suivant :

```
<taglib>
<taglib-uri>taglib-URI</taglib-uri>
<taglib-location>/WEB-INF/lib/taglib.jar</taglib-location>
</taglib>
```

Ainsi, dans les pages JSP, la directive **taglib** devient :

```
<%@ taglib uri="taglib-URI" prefix="tag-prefix" %>
```

Tag JSP

Exemple simple

La classe HelloTag

```
public class HelloTag extends TagSupport {

        public int doStartTag() throws JspException {
                try {
                        pageContext.getOut().println ("Hello World !");
                } catch (IOException e) {
                        throw new JspException ("I/O Error", e);
                }
                return SKIP_BODY;
        }
}
```

- On étend TagSupport afin de bénéficier des implémentations par défaut des méthodes de Tag.
- On surcharge doStartTag(), dans lequel on se contente d'écrire la chaîne "Hello World" dans la sortie de la page courante (pageContext est initialisé par l'implémentation par défaut desetPageContext()).
- On retourne Tag.SKIP_BODY car on ne veut pas traiter le corps de la balise.

Tag JSP

- Le mapping de notre tag dans le fichier de descripteur correspond à ceci :

```
<tag>
   <name>hello</name>
   <tagclass>exemple.taglib.HelloTag</tagclass>
   <bodycontent>empty</bodycontent>
</tag>
```

- Ainsi, dans une page JSP, le code suivant :

```
<tag11:hello/>
```

affichera dans le navigateur :

Hello World !

Tag JSP

Variables implicites

- Les balises personnalisées accèdent aux variables implicites

de la JSP dans laquelle elles s'exécutent via un objet de type PageContext

- Utilisation de l'attribut implicite pageContext
- La classe PageContext définit plusieurs méthodes
  - JspWriter getOut() : accès à la variable out de la JSP
  - ServletRequest getRequest() : accès à la variable request
  - ServletContext getServletContext() : instance du ServletContex
  - Object getAttribute(String) : retourne objet associé au paramètre (scope à page)
  - Object getAttribute(String, int) : retourne objet avec un scope précis
  - setAttribute(String, Object) : associe un nom à un objet (scope à page)
  - setAttribute(String, Object, int) : associe un nom à un objet avec un scope
  - Object findAttribute(String) : cherche l'attribut dans les différents scopes
  - removeAttribute(String) : supprime un attribut

Tag JSP

Utilisation d'attributs

```
public class HelloTag extends TagSupport {
        private String name = null;

        public void setName (String string) {
                this.name = string;
        }

        public int doStartTag() throws JspException {
                if (this.name==null)
                        this.name = "World";
                try {
                        pageContext.getOut().println ("Hello " + this.name + " !");
                } catch (IOException e) {
                        throw new JspException ("I/O Error", e);
                }
                return SKIP_BODY;
        }
}
```

- Les attributs du tags sont mappés avec des propriétés de la classe (possédant obligatoirement un setter)
- Ils ne sont pas forcement des chaines de caractères, n'importe quel objet peut être utilisé (un cast est fait lors de l'évaluation)

Tag JSP

Utilisation d'attributs

```
<tag>
    <name>hello</name>
    <tagclass>exemple.taglib.HelloTag</tagclass>
    <bodycontent>empty</bodycontent>
    <attribute>
          <name>name</name>
          <required>false</required>
          <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

On ajoute une balise <attribute> par attribut

- <name> : nom de l'attribut
- <required> : obligatoire : si true est que l'attribut n'est pas présent dans l'appels de tag une exception est remontée
- <rtexprvalue> : indique si la valeur de l'attribut peut être le résultat d'une expression (<%=expression%>).

Tag JSP

Tag collaboratifs

- L'interface Tag définit les méthodes setParent() et getParent() afin de renseigner le tag sur son tag parent (si il existe).

- On prendra comme exemple un switch

```
<exemple:switch test="2">
    <exemple:case value="0">Zéro</exemple:case>
    <exemple:case value="1">Un</exemple:case>
    <exemple:case value="2">Deux</exemple:case>
    <exemple:case value="3">Trois</exemple:case>
</exemple:switch>
```

- On définira 2 classe Switch et Case (le case connaitra son père grâce à la méthode getParent())

Tag JSP

Tag collaboratifs

SwitchTag

```
public class SwitchTag extends TagSupport {

        private Object test = null;

        public void setTest (Object obj) {
                this.test = obj;
        }

        public int doStartTag() throws JspException {
                return EVAL_BODY_INCLUDE;
        }

        public boolean isValid (Object caseValue) {
                if (this.test==caseValue) return true;
                if (this.test!=null && this.test.equals(caseValue)) return true;
                return false;
        }
}
```

- On défini une méthode qui test l'égalité entre l'attribut de la classe et un objet (cette méthode sera appelée par le case)

Tag JSP

Tag collaboratifs

CaseTag

```
public class CaseTag extends TagSupport {

        public Object value = null;

        public void setValue (Object object) {
                this.value = object;
        }

        public int doStartTag() throws JspException {

                if ( getParent() instanceof SwitchTag ) {
                        SwitchTag parent = (SwitchTag) getParent();
                        if (parent.isValid(this.value))
                                return EVAL_BODY_INCLUDE;
                        return SKIP_BODY;
                }
                throw new JspException ("Le tag case doit être à l'intérieur du tag swith.");
        }
}
```

- Le tag case se contente de tester si sa value est égale à celle du switch. Si c'est le cas elle affiche son contenu.

Tag JSP

Tag itératif

```
<exemple:iterate count="3"> Cette ligne sera affichée trois fois<br/>
</exemple:iterate>
```

- Pour faire ceci on utilisera l'interface IterationTag qui est un extension de l'interface Tag. Elle possède une méthode supplémentaire doAfterBody () qui est appelée après la méthode doStartTag(). Elle renvoi :
  - IterationTag.EVAL_BODY_TAG : la méthode doAfterBody() sera ré-appeler
  - Tag.SKIP_BODY : le corps est traité, passage à la suite

Tag JSP

Tag itératif

IterateTag

```
public class IterateTag extends BodyTagSupport {

    private int count = 0;
    private int current;

    public void setCount(int i) {
        count = i;
    }

    public int doStartTag() throws JspException {
        current = 0;
        if (current < count)
            return Tag.EVAL_BODY_TAG;
        return Tag.SKIP_BODY;
    }

    public int doAfterBody() throws JspException {
        current++;
        if (current < count)
            return IterationTag.EVAL_BODY_TAG;
        return Tag.SKIP_BODY;
    }
}
```

```
Cette ligne sera affichée trois fois
Cette ligne sera affichée trois fois
Cette ligne sera affichée trois fois
```

Tag JSP

Autre possibilités :

- bufferisation du contenu des tags
- TagExtraInfo : permet gérer des variables de scripts, vérification des attributs
- Gestion des exceptions des tags (interfaces TryCatchFinally)

Taglibs 2.0

- Modification des interfaces et des classes à étendre (mais garde la rétrocompatibilité)
- Gestion dynamique des attributs
- Ecriture de fonction EL (méthode publique statique)
- Externalisation du code HTML dans des fichier .tag qui sont ensuite compilé en une classe (sur le même principe que jsp compilés en Servlet)

Questions

?

Documentation

- La page officielle chez Sun :

  http://java.sun.com/products/jsp/jstl/
- La page officielle de la spécification de la JSTL :

  https://jstl-spec-public.dev.java.net/
- L'implémentation du projet Jakarta de la JSTL 1.1 :

  http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html
- L'API des interfaces et classes de bases de la JSTL :

  http://java.sun.com/products/jsp/jstl/1.1/docs/api/index.html
- La documentation des différents tags :

  http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/