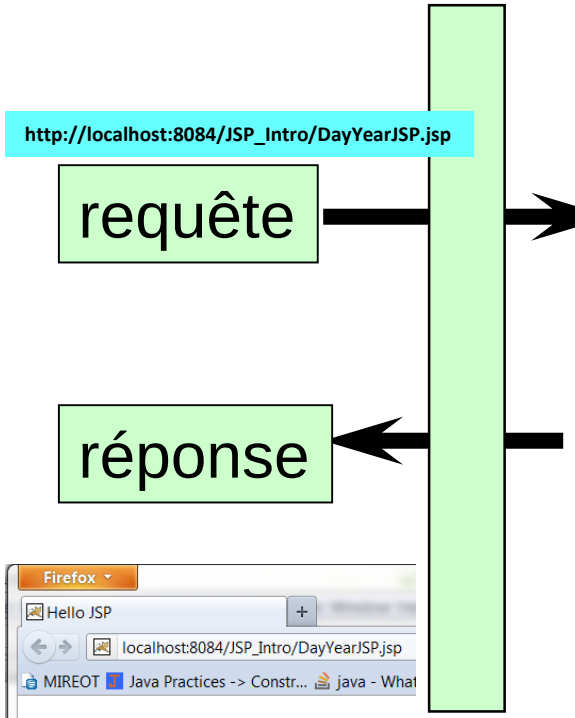
Introduction aux Java Server Pages

- JSP : Java Server Pages.
  - *Extension standard aux Servlets*
- Servlet : code HTML inclus dans le programme.
  - *Tout le code HTML n'est pas forcément dynamique*
  - *Parfois lourd à mettre en œuvre.*
- JSP : programme (java) inclus dans le code HTML
  - *cf Perl versus PHP.*
  - *Equivalent JAVA de ASP (Application Server Pages de Microsoft) et de PHP*

http://www.oracle.com/technetwork/java/javaee/jsp/index.html

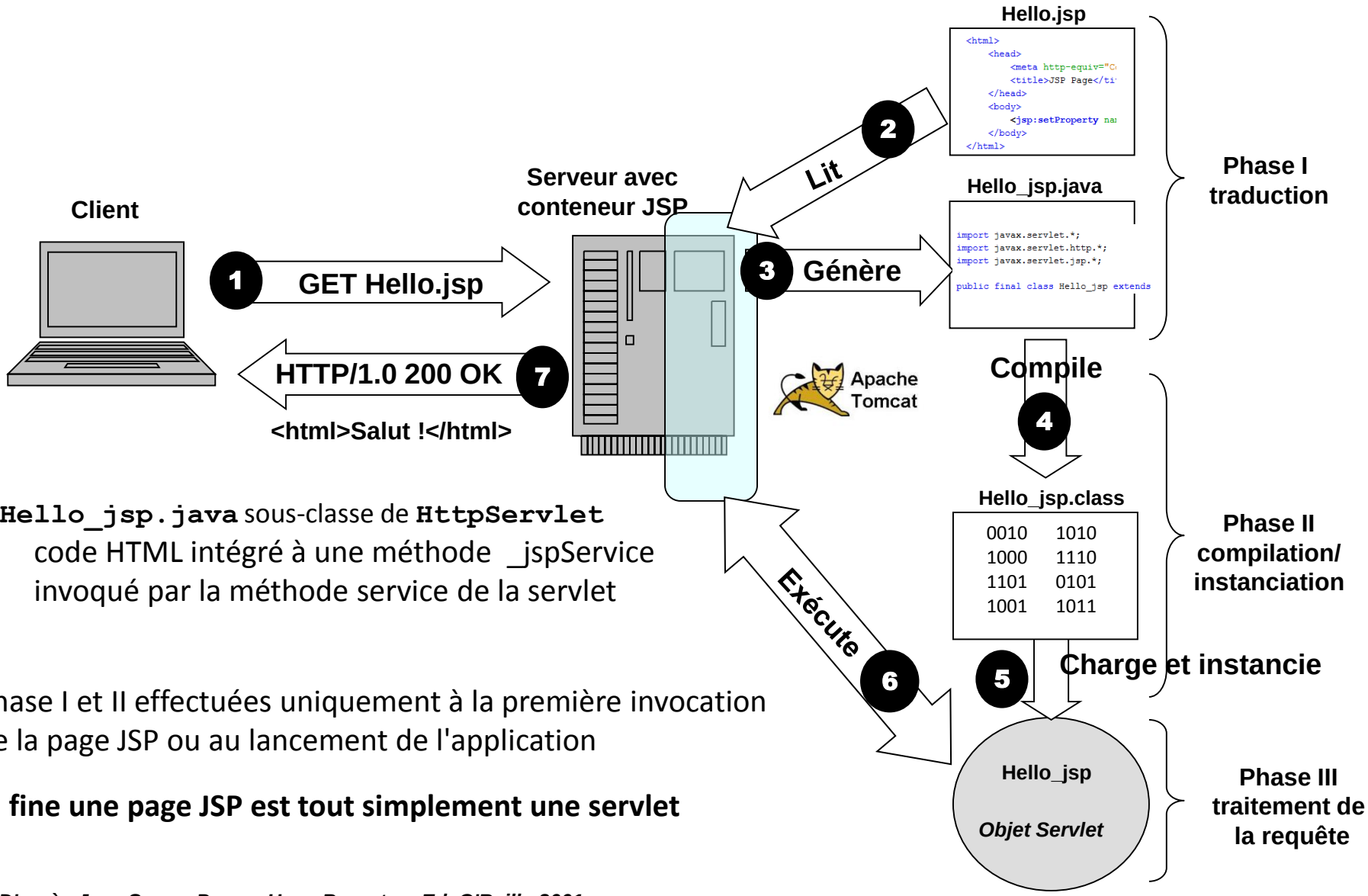
- **Java Server Pages**
  - ***Fichier texte qui décrit comment créer une réponse à partir d'une requête particulière.***
  - ***tags HTML + extensions + JAVA comme langage de script***



DayYearJSP.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.text.SimpleDateFormat"%>
<jsp:useBean id="date" class="java.util.Date"/>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Hello JSP</title>
    </head>
    <body>
        <h1>Bonjour</h1>
        <% SimpleDateFormat formatter = new SimpleDateFormat("dd:MM:yyyy");%>
        <p>Nous somme le <%=formatter.format(date)%></p>
    </body>
</html>
```

cycle de vie des JSP



`Hello_jsp.java` sous-classe de **`HttpServlet`**
code HTML intégré à une méthode _jspService
invoqué par la méthode service de la servlet

Phase I et II effectuées uniquement à la première invocation de la page JSP ou au lancement de l'application

in fine une page JSP est tout simplement une servlet

D'après Java Server Pages, Hans Bergsten, Ed. O'Reilly 2001

- Trois types de tags
  - `<%@ ... %>` Tags de **directives**
    contrôlent la structure de la servlet générée
  - `<% ... %>` Tags de **scripting**
    insertion de code java dans la servlet
  - `<jsp:... >` Tags d'**actions**
    facilitent l'utilisation de composants

Les directives

- Syntaxe :

```
<%@directive attribut="valeur" …>
```

- Permettent de spécifier des informations globales sur la page
- 3 types de directives
  - **`page`** *options de configuration de la page*
  - **`include`** *inclusions de fichiers statiques*
  - **`taglib`** *pour définir des tags d'actions personnalisées*

La directive page

Principaux attributs de la directive `page`

- **`<%@page import="java.util.*,java.sql.Connection" %>`**
- **`<%@page contentType="text/html;charset=ISO-8859-1" %>`**
- **`<%@page session="true|false" %>`**
  - *Indique si la page est incluse ou non dans une session. Par défaut **`true`**, ce qui permet d'utiliser un objet de type **`HttpSession`** pour gérer des données de session*
- **`<%@page errorPage="relativeURL" %>`**
  - *Précise la JSP appelée au cas où une exception est levée*
    - *URL relative par rapport au répertoire qui contient la page JSP ou relative par rapport au contexte de l'application Web si elle débute par /*
- **`<%@page isErrorPage=" true|false" %>`**
  - *Précise si la page JSP est une page de gestion d'erreur (dans ce cas l'objet `exception` peut être utilisée dans la page), `false` par défaut.*
- **`<%@page isThreadSafe=" true|false" %>`**
  - *Précise si la servlet générée est multithreadée ou non.*
- …

La directive include

- Syntaxe : **<%@include file="chemin relatif du fichier" %>**
  - *chemin relatif par rapport au répertoire qui contient la page JSP ou relatif par rapport au contexte de l'application Web si il débute par /*

- Inclus le fichier dans le source JSP avant que celui-ci ne soit interprété (traduit en servlet) par le moteur JSP
- Le fichier peut être un fragment de code JSP, HTML ou Java
- Tag utile pour insérer un élément commun à plusieurs pages (en-tête, pied de page)

Insertion à la traduction et non pas à l'exécution

➔ un changement du fichier inclus ne provoque pas une régénération de la servlet

Tags de scripting

- Permettent d'insérer du code Java qui sera inclus dans la servlet générée
- Trois types de tags
  - ***<%!*** *…* ***%>*** *tag de déclaration*
    - *Code inclus dans le corps de la servlet (déclaration de membres, variables ou méthodes)*
  - ***<%=expression%>*** *tag d'expression*
    - *L'évaluation de l'expression est insérée dans le flot de sortie dans la méthode* **`service()`** *de la servlet <==>* **`out.println(expression)`**
  - ***<%...%>*** *tag de scriptlet*
    - *Le code Java est inclus dans la méthode* **`service()`** *de la servlet*

Variables implicites

- les spécifications des JSP définissent plusieurs objets implicite utilisables directement dans le code Java

Variable	Classe	Rôle
`out`	`javax.servlet.jsp.JspWriter`	Flux en sortie de la page HTML générée
`request`	`javax.servlet.http.HttpServletRequest`	Contient les informations de la requête
`response`	`javax.servlet.http.HttpServletResponse`	Contient les informations de la réponse
`session`	`javax.servlet.http.HttpSession`	Gère la session
`application`	`javax.servlet.ServletContext`	Informations du contexte de l'application
`config`	`javax.servlet.Servletconfig`	Informations de configuration de la servlet
`exception`	`java.lang.Throwable`	L'objet exception pour une page d'erreur

Commentaires

- **<!-- ... -->**
  - *Commentaires HTML*
  - *Intégralement reconduits dans le fichier HTML généré*
- **<%-- ... --%>**
  - *Commentaires cachés*
  - *Contenu ignoré par le moteur JSP*

```
<%@page contentType="text/html"%>
<%@page import="java.util.Date"%>
<html>
    <head><title>Commentaires dans une page JSP</title></head>
    <%-- page pour montrer l'utilisation des commentaires --%>
    <!-- cette page a été générée le <%= new Date()%> -->
    <body>
    <H1>BONJOUR MONDE CRUEL</H1>
    </body>
</html>
```

Commentaire JSP

Commentaire HTML

page JSP

```
<html>
    <head><title>Commentaires dans une page JSP</title></head>
    <!-- cette page a été générée le Fri Feb 04 19:04:29 CET 2005 -->
    <body>
    <H1>BONJOUR MONDE CRUEL</H1>
    </body>
</html>
```

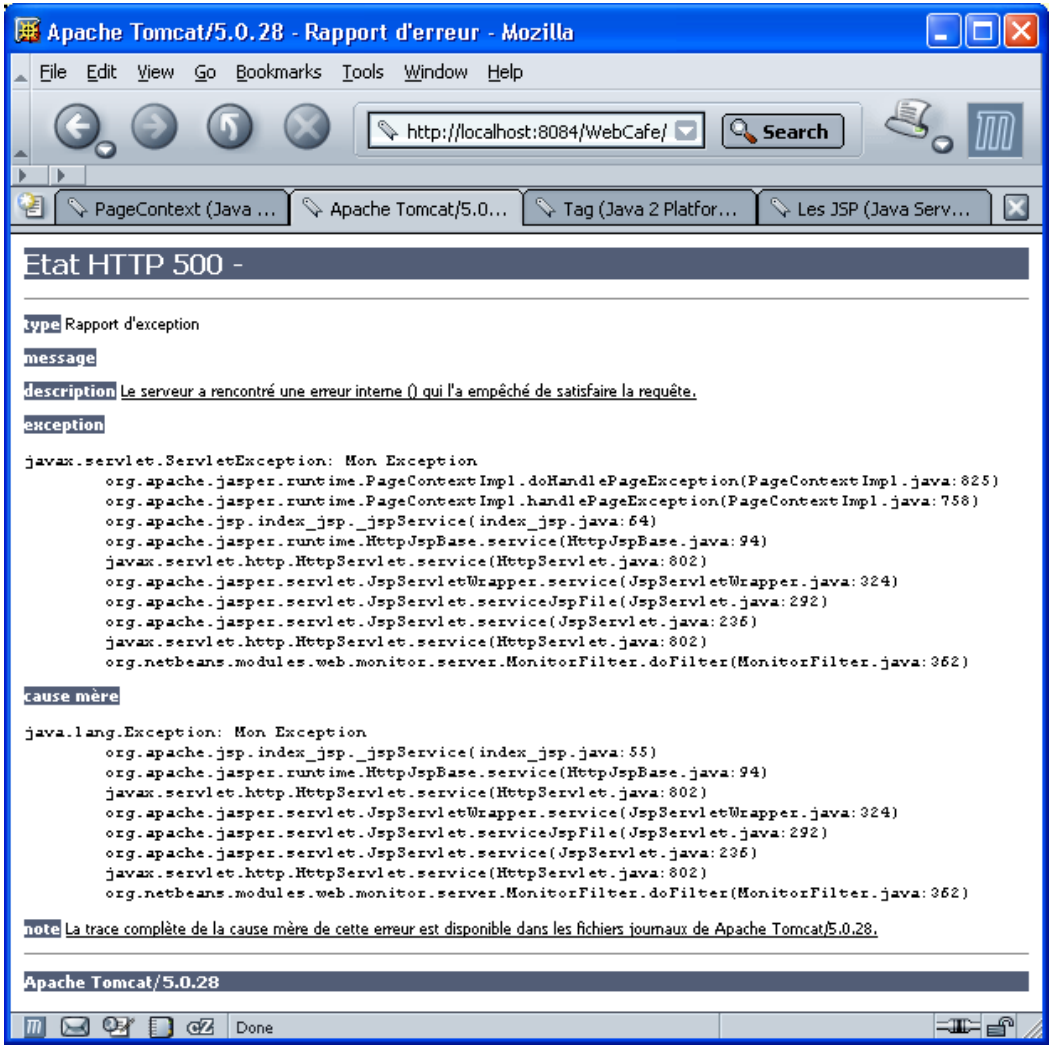
HTML généré

Gestion des erreurs

- Si une exception est levée dans une page JSP et n'est pas capturée
  - *Si pas de page d'erreur associée à la JSP :*

    affichage de la pile d'exécution

```
<%@page contentType="text/html"%>
<%@page import="java.util.Date"%>
<html>
    <body>
       <% if (true) throw new
        Exception("Mon Exception"); %>
    </body>
</html>
```



Gestion des erreurs

- Si une exception est levée dans une page JSP et n'est pas capturée
  - *Si une page d'erreur est associée à la JSP : redirection vers cette page*

```
<%@page contentType="text/html"%>
<%@page import="java.util.Date"%>
<%@page errorPage="/MaPageErreur.jsp" %>
<html>
    <body>
       <% if (true) throw new Exception("Mon Exception"); %>
    </body>
</html>
```

MaPageErreur.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page isErrorPage="true"%>
<html>
    <head><title>Page d'erreur</title></head>
    <body>
       <H1>ERREUR</H1>
       <p>Erreur lors du traitement</p>
       <p><b><%=exception.getMessage()%></b></p>
    </body>
</html>
```

JSP et Java Beans

- Un des objectif des JSP / Servlets
  - *Ne pas mélanger du code HTML au code Java des Servlets*
- D'un autre coté si c'est pour mettre du code Java dans le code JSP qu'est ce qu'on y gagne ?
- Un point important dans la conception de pages JSP est de minimiser le code Java embarqué dans les pages
- Déporter la logique métier dans des composants objets qui seront accédés depuis les pages JSP
  - *Simplification des traitements inclus dans la JSP*
  - *Possibilité de réutilisation des composants depuis d'autres JSP ou d'autres composants*
- Les spécifications JSP définissent une manière standard d'intérargir avec des composants Java Beans

Java Bean

- Qu'est-ce qu'un composant Java Bean ?
  - *Un composant java (objet) dont la classe respecte certaines règles d'écriture*
- *Le respect de ces règles d'écriture permet ensuite d'utiliser ces composants de manière standard*
  - Composants Java Beans depuis JSP (on va le voir tout de suite)
  - Composants graphiques pour la construction d'interface utilisateurs
  - …

- Un classe Java Bean
  - *Classe publique*
  - *Possède un constructeur public sans arguments*
  - *Regroupe un ensemble de propriétés*
    - *accessibles par des méthode de la forme* **`getXXX()`** *où* **`XXX`** *est le nom de la propriété*
    - *éventuellement modifiables par une méthode* **`setXXX()`** *où* **`XXX`** *est le nom de la propriété*
  - *Implémente en option l'interface* **`java.io.Serializable`**

- Un exemple de bean

Propriété privée

Accesseur (« getter »)

Modifieur (« setter »)

```
package test;
public class Personne {
  private String nom;
  private String prenom;
  private int age = 0;

  public Personne() {
    this.nom = "X";  // nom par défaut
    this.prenom = "x"; // prénom par défaut
  }

  public String getNom() {
    return (this.nom);
  }

  public void setNom(String nom) {
    this.nom = nom;
  }
  ...
  public int getAge () {
    return (this.ge);
  }

  public void setAge(int age) {
    this.age = age;
  }
}
```

Utiliser un bean depuis une JSP

- Le tag <jsp:useBean>
  - *Permet de localiser une instance ou bien d'instancier un bean pour l'utiliser dans la JSP*

- Syntaxe

```
<jsp:useBean
        id="beanInstanceName"
        class="package.class"
        type="package.class"
        scope="page|request|session|application"
/>
```

Nom utilisé pour la variable qui servira de référence sur le bean

La classe du bean

Optionnel, le type de la référence `beanInstanceName` si ce n'est pas le type défini par l'attribut class

Optionnel, détermine la portée durant laquelle le bean est défini et utilisable

- le bean demandé est déjà instancié pour la portée précisée :
  → renvoi de sa référence
- le bean demandé n'est pas déjà présent pour la portée précisée :
  → instanciation du bean
  → ajout de celui-ci à la portée spécifiée

Utiliser un bean depuis une JSP

- L'attribut scope

page	**valeur par défaut** bean utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus.
request	bean accessible durant la durée de vie de la requête. La méthode **`getAttribute()`** de l'objet **`request`** permet d'obtenir une référence sur le bean.
session	bean utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent. La JSP qui créé le bean doit avoir l'attribut session = "true" dans sa directive page
application	bean utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean. Le bean n'est instancié que lors du rechargement de l'application

Utiliser un bean depuis une JSP

- exemple

```
<jsp:useBean
      id="utilisateur"
      class="test.Personne"
      scope="session"
/>
```

Définition dans la session d'un java bean instance de la classe `Personne` du package `test` et désigné par la référence `utilisateur`

Avec TOMCAT, les beans doivent être **nécessairement** définis dans des packages

```
<html>
   <body>
      <ul>
         <li>NOM : <%=utilisateur.getNom()%></li>
         <li>PRENOM : <%=utilisateur.getPrenom()%></li>
         <li>AGE : <%=utilisateur.getAge()%></li>
      </ul>
   </body>
</html>
```

Le langage EL (Expression Language)

- Objectif
  - *fournir un moyen simple d'accéder aux données nécessaires à une JSP en s'affranchissant de la syntaxe Java*
  - *Destiné aux non programmeurs Java*

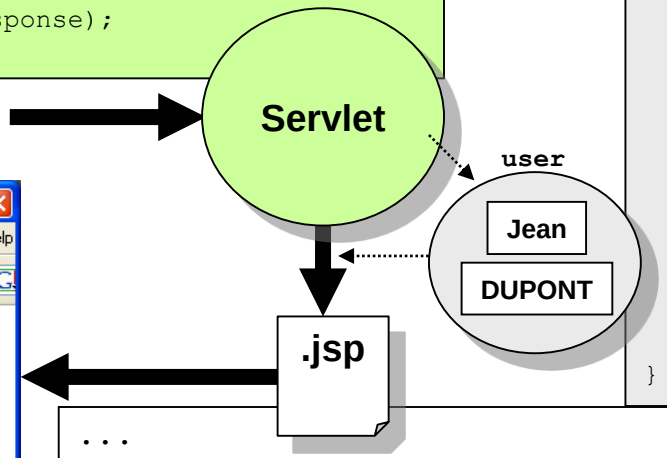
- EL (Expression Language)
  - *langage particulier constitué d'expressions qui permet en particulier d'utiliser et de faire référence à des objets java accessibles dans les différents contextes (page, requête, session ou application) d'une JSP*
  - *Initialement introduit avec la JSTL (JSP Standard Tag Library)*
  - *Supporté de manière standard à partir de la version 1.4 J2EE (Servlets 2.4, JSP 2.0)*
    - *une expression EL peut être utilisée directement dans n'importe quelle page JSP (à la place d'une expression* `<%=expressionJava%>` *)*

Le langage EL (Expression Language)

- exemple : Accéder à un attribut dans la requête

```
...
model.User utilisateur = new model.User();
utilisateur.setNom("DUPONT");
utilisateur.setPrenom("Jean");
request.setAttribute("user",utilisateur);
getServletContext().getRequestDispatcher("/testEL.jsp")
        .forward(request, response);
...
```

```
public class User {

    private String nom;
    private String prenom;

    public User() {
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String name) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}
```




```
...
<% model.User user = (model.User) request.getAttribute("user"); %>
<H1 align="center">Au revoir<BR>
<%=user.getPrenom()%> <%=user.getNom()%></H1>
...
```

```
...
<jsp:useBean id="user" type="model.User" scope="request"/>
<H1 align="center">Au revoir<BR>
<%=user.getPrenom()%> <%=user.getNom()%> </H1>
...
```

L'interpréteur EL recherche l'attribut dans la portée (scope) suivant l'ordre :
- portée de la page
- portée de la requête
- portée de la session
- portée de l'application

Expression EL →

```
...
<H1 align="center">Au revoir<BR>
${user.prenom} ${user.nom} </H1>
...
```

Le langage EL (Expression Language)

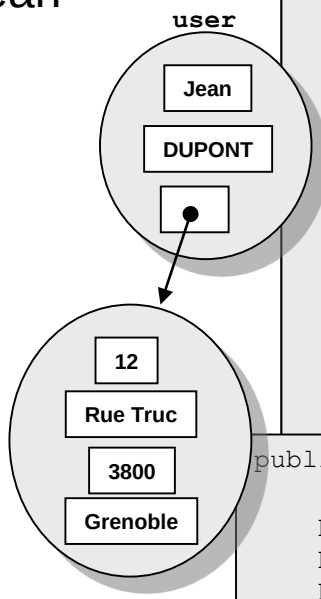
- Accéder à une propriété d'un JavaBean

```
${user.prenom}
```
→ **Jean**

- Accéder à une propriété imbriquée

```
${user.adresse.ville}
```
→ **Grenoble**

```
${user.adresse}
```
→ **12 Rue Truc 38000 Grenoble**




```
public class User {

    private String nom;
    private String prenom;
    private Adresse adresse;

    public User() {
    }
    public String getNom() {
        return nom;
    }
    public Adresse getAdresse() {
        return adresse;
    }
    public void setNom(String name) {
```

```
public class Adresse {

    private int no;
    private String rue;
    private int codePostal;
    private String ville;
    ...
    private String toString() {
       return "" + no + " " + rue + " " +
            codePostal + " " + ville;
}
```

- Accéder à une map

```
map.put("a","toto");
map.put("b","titi");
map.put("u",new User("Jean","Dupont"));
```

```
${map.a}
```
→ **toto**

```
${map.b}
```
→ **titi**

```
${map.u.prenom}
```
→ **Jean**

```
${map.u.adresse}
```
→ **12 Rue Truc 38000 Grenoble**

- Accéder à un tableau ou une liste

```
${array[0]}<BR>
${array[1]}<BR>
```

```
${liste[0]}<BR>
```

[ ] utilisable à la place de .

```
${user.prenom}   <==>   ${user["prenom"]}
${map.a}                ${map["a"]}
```

Le langage EL (Expression Language)

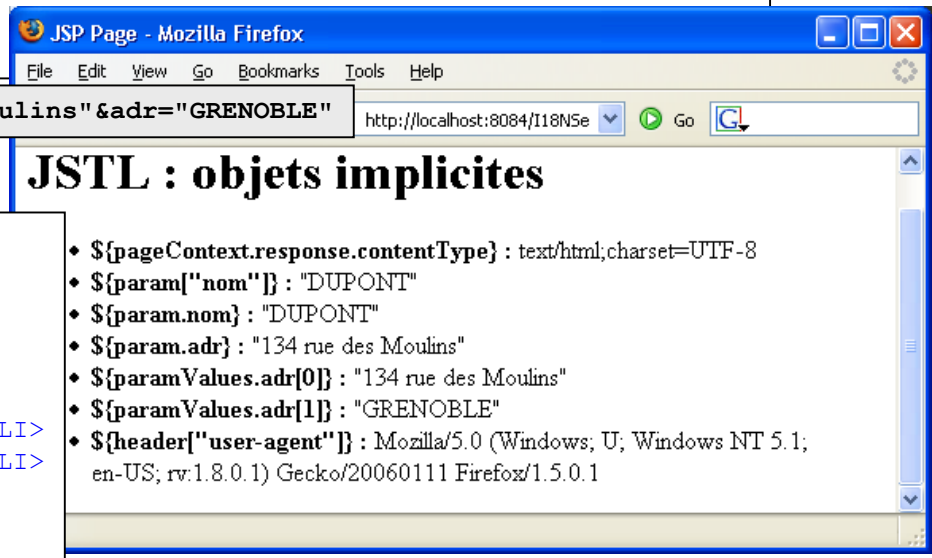
- Variables (objets) implicites définies par EL

pageContext	objet `PageContext` de la page JSP
pageScope	`Map` pour accéder aux attributs définis dans la portée de la page (PageContext)
requestScope	`Map` pour accéder aux attributs définis dans la portée de la requête (HttpServletRequest)
sessionScope	`Map` pour accéder aux attributs définis dans la portée de la session (HttpSession)
applicationScope	`Map` pour accéder aux attributs définis dans la portée de l'application (ServletContext)
param	`Map` pour accéder aux paramètres de la requête http sous forme de String
paramValues	`Map` pour accéder aux paramètres de la requête http sous la forme de tableau de String
header	`Map` pour accéder aux valeurs de l'en-tête de la requête
headerValues	`Map` pour accéder aux valeurs de l'en-tête de la requête sous la forme de tableau de String
initParam	`Map` pour accéder aux paramètres d'initialisation (`init-params` du `web.xml`)
cookie	`Map` pour accéder aux cookies

Exemple :

```
/index.jsp?nom="DUPONT"&adr="134 rue des Moulins"&adr="GRENOBLE"
```

```
<UL>
  <LI><B>\${pageContext.response.contentType}:</B>
         ${pageContext.response.contentType}</LI>
  <LI><B>\${param["nom"]}: </B>${param["nom"]}</LI>
  <LI><B>\${param.nom}: </B>${param.nom}</LI>
  <LI><B>\${param.adr}: </B>${param.adr}</LI>
  <LI><B>\${paramValues.adr[0]}:</B>${paramValues.adr[0]}</LI>
  <LI><B>\${paramValues.adr[1]}:</B>${paramValues.adr[1]}</LI>
  <LI><B>\${header["user-agent"]}:</B>
            ${header["user-agent"]}</LI>
</UL>
```



Le langage EL (Expression Language)

- Opérateurs définis par EL

Opérateur	Rôle	Exemple

`.`
`[]`
	Obtenir une propriété d'un objet
	Obtenir une propriété par son nom ou son indice
		`${param.nom}`
		`${param["nom"]}`
		`${tab[0]}`

`== eq != ne`
`< lt > gt`
`<= le >= ge`
	Opérateurs relationnels	`<c: if test="${user.age ge 18}">`

`+ - * / div` `% mod`	Opérateurs arithmétiques	`${article.prixHT * 0.055}`

`&& and`
`||`
`or ! not`
	Opérateurs logiques	`${(user.age ge 7)&&(user.age le 77)}`

`empty`	Teste si un objet est `null` ou vide si c'est une chaîne de caractère.	`${empty param.nom}`

Le langage EL (Expression Language)

- actions personnalisées permettent aux développeurs d'étendre la syntaxe JSP,
- la notion de fonctions permet aux développeurs d'étendre les possibilités de l'EL

définies comme des méthodes publiques statiques

ELFunctions.java

```
package el;
public class ElFunctions {
  public static final double TAUX_TVA=0.1976;

  public static double prixTTC(double p) {
        return p + tva(p);
  }
  public static double tva(double p) {
        return p * TAUX_TVA;
  }
```

Doivent être déclarées dans un fichier TLD

elFunctions.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-jspta
  <tlib-version>1.0</tlib-version>
  <short-name>el</short-name>
  <uri>/WEB-INF/tlds/elFunctions</uri>
  <function>
       <name>prixTTC</name>
       <function-class>el.ElFunctions</function-class>
       <function-signature>double prixTTC(double)</function-signature>
   </function>
```

Utilisées dans les JSP de manière analogue aux actions personnalisées

prixArticle.jsp

```
<%@taglib uri="/WEB-INF/tlds/elFunctions.tld" prefix="el"%>
...
<html>
...
    <h1>
    Article : <I> ${article.prixHT} </I>
    </h1>
    <h2>
    Prix HT : ${article.prixHT} €<BR>
    Prix TTC : ${el:tva(article.prixHT)} €<BR>
    Prix TTC : ${el:prixTTC(article.prixHT)} €<BR>
    </h2>
...
</html>
```

Pour en savoir plus sur EL

- Supports de cours et articles de Mickael BARON
  - http://mbaron.developpez.com/
  - *Introduction aux JSP 2 :* http://mbaron.developpez.com/javaee/jsp2
- Présentation des Expressions Languages. Par F. Martini
  - http://adiguba.developpez.com/tutoriels/j2ee/jsp/el/
- Tutorial JEE6 - Chapter 6 Expression Language
  - http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html

Fixer les propriétés d'un bean depuis une JSP

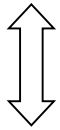
- Le tag <jsp:setProperty>
  - *Permet de mettre à jour la valeur de un ou plusieurs attributs d'un bean*

- Syntaxe

```
<jsp:setProperty name="beanInstanceName"
      property="propertyName"
      value="string|<%=expression%>"
/>
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<jsp:setProperty name="utilisateur"  property="nom"  value="Toto"/>
<jsp:setProperty name="utilisateur"  property="age"  value="34"/>
```

Quand le type de la propriété du Bean n'est pas `String` une conversion automatique est effectuée en utilisant la méthode `valueOf` de la classe enveloppe



```
<%utilisateur.setAge(Integer.valueOf("34"));%>
```

Fixer les propriétés d'un bean depuis une JSP

- Possibilité de fixer les propriétés du bean à l'aide des paramètres de la requête

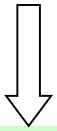
- Syntaxe

```
<jsp:setProperty name="beanInstanceName" property="propertyName"/>
```

Le nom de la propriété est le même que le nom du paramètre de la requête

```
<jsp:setProperty name="beanInstanceName" property="propertyName"
      param="paramétrage"/>
```

Le nom de la propriété est différent du nom du paramètre de la requête



```
MaPageJSP?name="Toto"&age="24"
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<jsp:setProperty name="utilisateur"  property="nom"  param="name"/>
<jsp:setProperty name="utilisateur"  property="age" />
```

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

Fixe toutes les propriétés correspondant à des paramètres de la requête

Accéder aux propriétés d'un bean depuis une JSP

- Le tag <jsp:getProperty>
  - *Permet d'obtenir la valeur d'un attribut d'un bean*

- Syntaxe

```
<jsp:getProperty name="beanInstanceName" property="propertyName" />
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<body>
<ul>
   <li>Nom : <jsp:getProperty name="utilisateur"  property="nom"/></li>
   <li>Age : <jsp:setProperty name="utilisateur"  property="age"/></li>
</ul>
...
```

Tag de redirection

- Le tag <jsp:forward>
  - *Permet de rediriger la requête vers un fichier HTML, une autre page JSP ou une Servlet*
- Syntaxe

```
<jsp:forward page="relativeURL|<%=expression%>"/>
```

```
...
<jsp:forward page="uneAutrePage.jsp" />
```

Si URL commence par un / elle est absolue (contexte de l'application) sinon elle est relative à la JSP

Ce qui suit l'action forward est ignoré, et tout ce qui a été généré dans cette page JSP est perdu

- Possibilité de passer un ou plusieurs paramètres vers la ressource appelée

```
<jsp:forward page="relativeURL|<%=expression%>">
    <jsp:param name="parametre" value="string|<%=expression%>">
    ...
</jsp:forward>
```

Tag d'inclusion

- Le tag <jsp:include>
  - *Permet d'intégrer **dynamiquement** un contenu généré par une autre page JSP ou une autre servlet.*
- Syntaxe

```
<jsp:include page="relativeURL" flush="true|false"/>
```

spécifie si le tampon doit être envoyé au client et vidé

- Comme pour <jsp:forward> possibilité de passer un ou plusieurs paramètres vers la ressource incluse en utilisant le tag <jsp:param>

Tags personnalisés

- Possibilité de définir ses propres tags basés sur XML :
  - *tags personnalisés (custom tags)*
  - *regroupés dans des bibliothèques de tags (taglibs)*

- Objectifs
  - *Déporter dans des classes dédiées le code java contenu dans les scriptlets de la JSP et appeler ce code en utilisant des tags particuliers*
  - *Améliorer la séparation des rôles :*
    - *page JSP : présentation – concepteur de pages Web*
    - *scriplets / code Java – développeur Java*

- Tags personnalisés / Java Beans
  - *"philosophie" similaire*
  - *Java Beans : objets métier pour stocker et échanger des données*
  - *Tag personnalisé : interagit directement avec environnement JSP dans lequel il s'exécute*

Tags personnalisés

- Que peuvent faire des tags personnalisés :
  - *Produire du contenu pour une page JSP*
  - *Recevoir des paramètres envoyés à partir de la JSP qui les appelle*
  - *Avoir un corps qu'ils peuvent manipuler.*
    - *Possibilité d'imbriquer un tag personnalisé dans un autre avec un nombre d'imbrications illimité*
  - *Accéder aux Java Beans définis dans la page JSP*
  - *Introduire de nouveaux Java Beans*
  - *Introduire de nouvelles variables de scripting*

Tags personnalisés

- Les Tags personnalisés son regroupés en bibliothèques de Tag (Tag Lib)

- Un tag personnalisé est défini par :
  - *Une classe Java (Gestionnaire de balise : Tag Handler)*
    - *code exécuté par le conteneur de JSP lorsque ce Tag est invoqué dans une page JSP*
    - *implémente interface* **`javax.servlet.jsp.tagext.JSPTag`**
    - *Accède à un objet* **`javax.servlet.jsp.JSPWriter`** pour générer une réponse
  - *Une entrée dans le fichier de description de la bibliothèque à laquelle il est associé (document XML TLD Tag Library Descriptor)*
    - *la syntaxe du tags*
      - *Nom, attributs ….*
    - *La classe du Tag Handler associé*

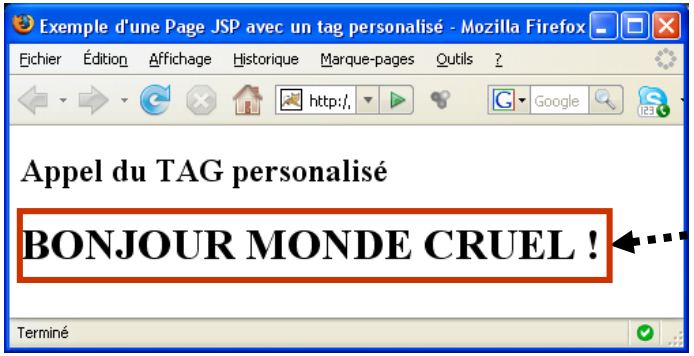
Tags personnalisés

Page JSP

```
<%@ taglib uri="/WEB-INF/tlds/mesTags.tld"
      prefix="mesTags"%>
<html>
   ...
   <mesTags:HelloTag/>
   ...
</html>
```

Fichier TLD

```
<taglib>
    ...
    <tag>
        <name>HelloTag</name>
        <tag-class>mestags.HelloTagHandler1</tag-class>
        ...
    </tag>
    ...
</taglib>
```



HelloTagHandler1.class

Implémentation du TAG (Tag Handler)

Balises personnalisés

- Différentes formes de balises (tag) personnalisées (syntaxe XML)
  - *Balise sans corps ni attribut*

```
<prefixe:nomDuTag></prefix:nomDuTag>
```

```
<prefixe:nomDuTag/>
```

  - *Balise sans corps avec 2 attributs*

```
<prefixe:nomDuTag attribut1="valeur1" attribut2="valeur2" />
```

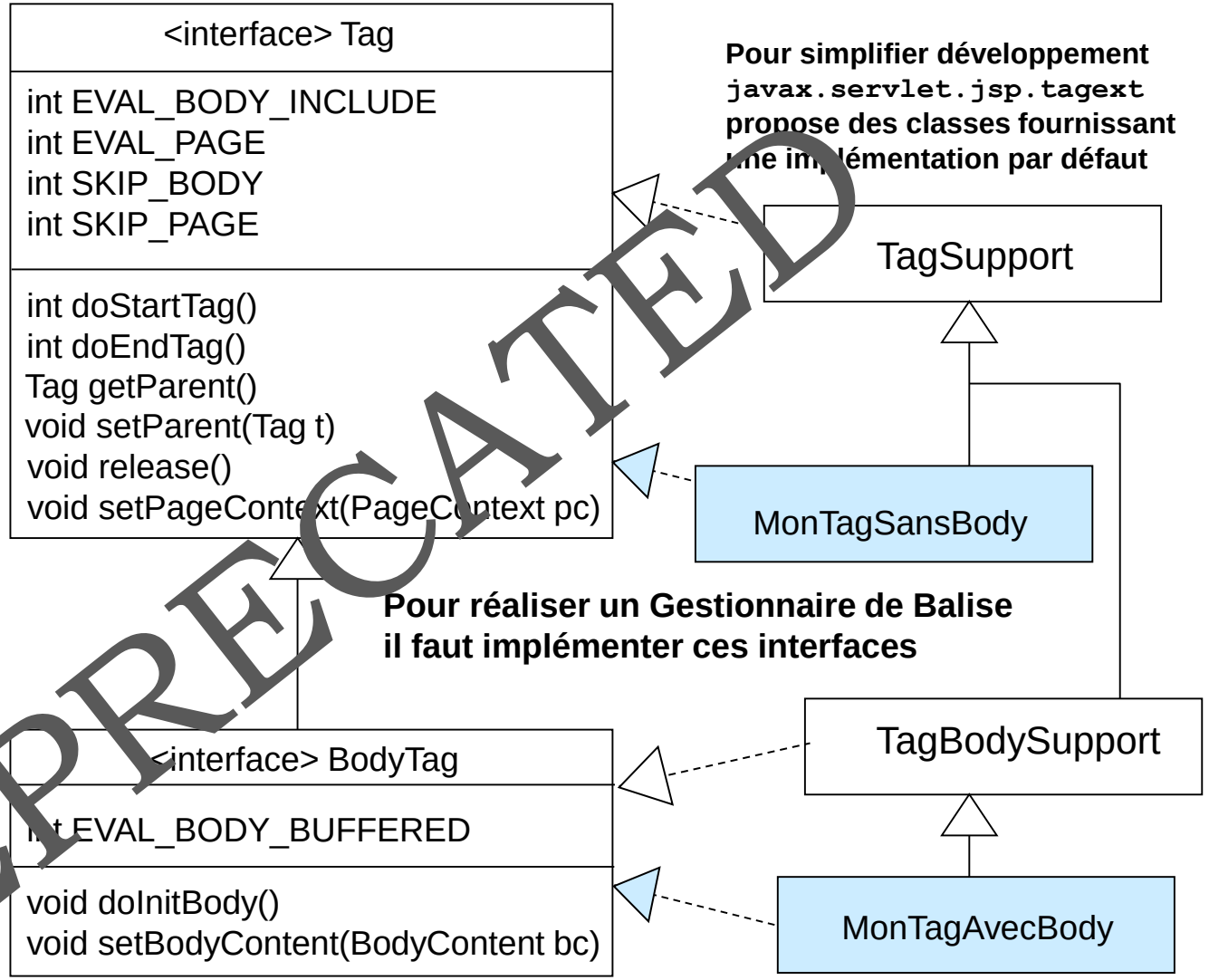
  - *Balise avec corps avec 2 attributs*

```
<prefixe:nomDuTag attribut1="valeur1" attribut2="valeur2" >
        Corps du Tag
</prefixe:nomDuTag>
```

Gestionnaire de Tag (Tag Handler)

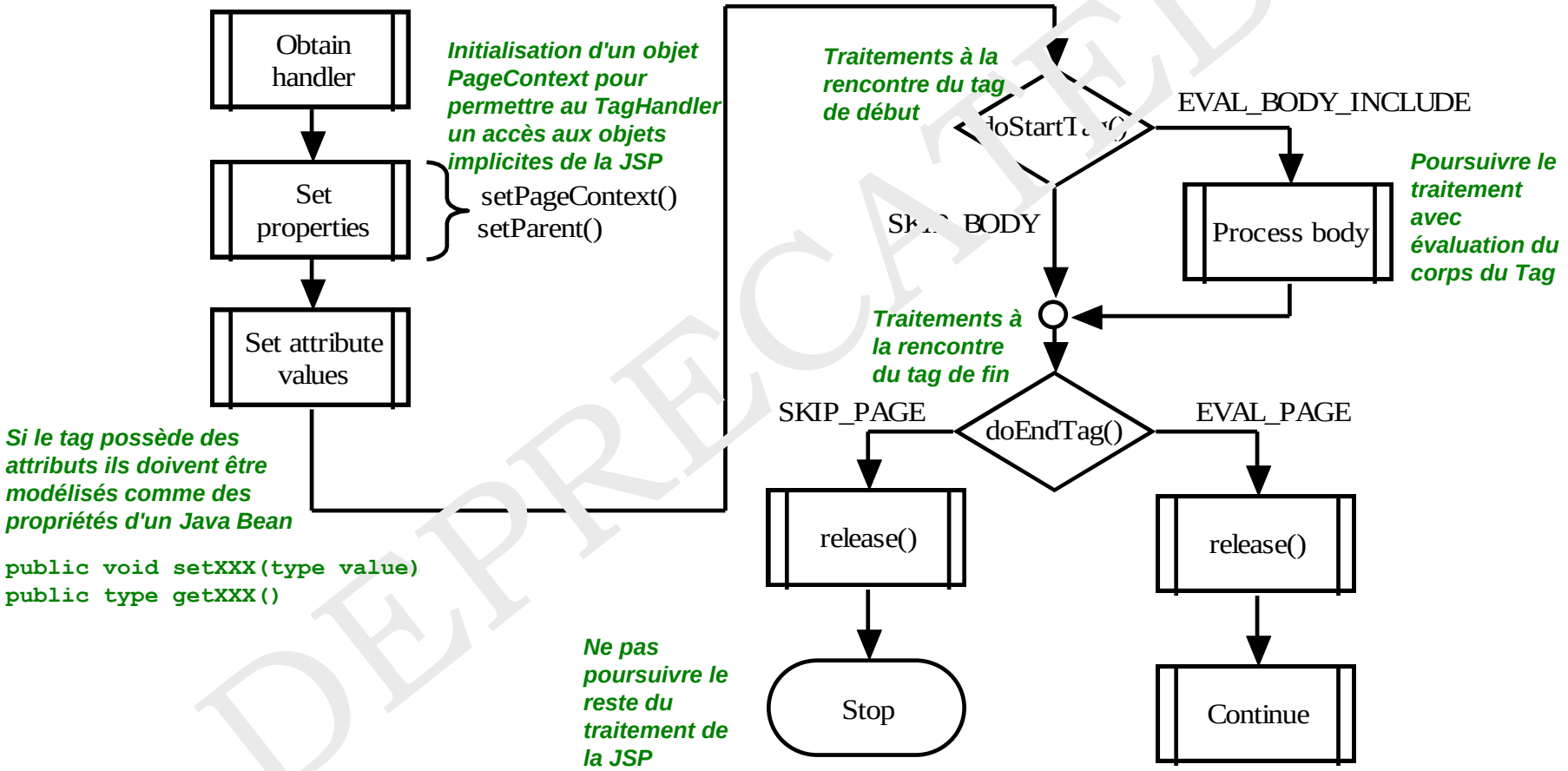
Package `javax.servlet.jsp.tagext`

Introduit avec JSP 1.2



Cycle de vie d'un Tag

- Opérations réalisées par le code généré lors de la compilation de la JSP



Accès aux variables implicites de la JSP

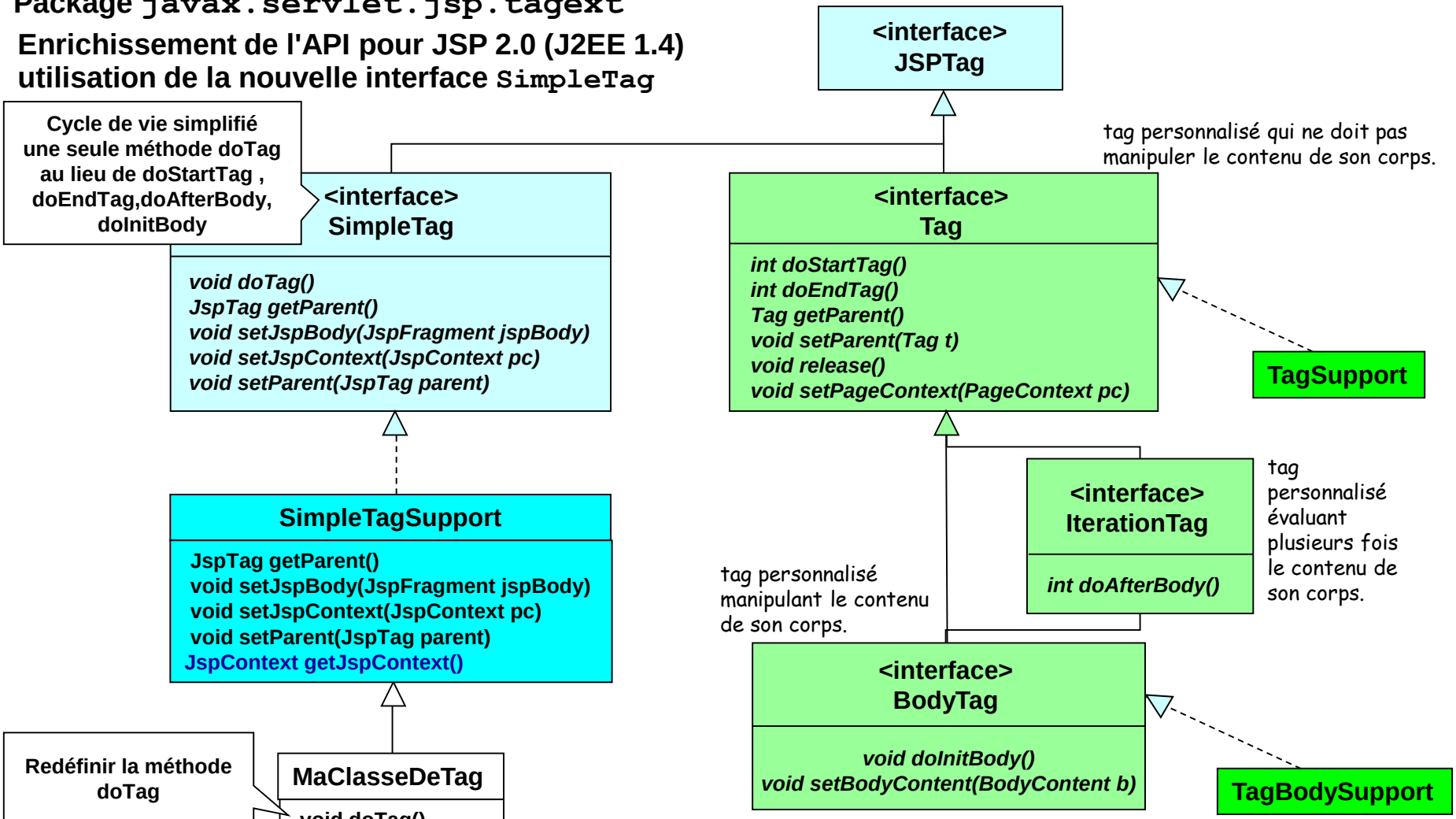
- Le Tag Handler accède à l'environnement JSP via un objet **`PageContext`**
  - *Variable d'instance **`pageContext`** dans les classes de support*
- Principales méthodes d'accès :

`JspWriter getOut()`	**Permet un accès à la variable `out` de la JSP**
`Exception getException()`	**Permet un accès à la variable `exception` de la JSP**
`Object getPage()`	**Permet un accès à la variable `page` de la JSP**
`ServletRequest getRequest()`	**Permet un accès à la variable `request` de la JSP**
`ServletResponse getResponse()`	**Permet un accès à la variable `response` de la JSP**
`ServletConfig getServletConfig()`	**Permet un accès à l'instance de la variable de type `ServletConfig`**
`ServletContext getServletContext()`	**Permet un accès à l'instance de la variable de type `ServletContext`**
`HttpSession getSession()`	**Permet un accès à la session**
`Object getAttribute(String)`	**Renvoie l'objet associé au nom fourni en paramètre dans la portée de la page**
`setAttribute(String, Object)`	**Permet de placer dans la portée de la page un objet dont le nom est fourni en paramètre**

API pour les Tags personnalisés

Package `javax.servlet.jsp.tagext`

Enrichissement de l'API pour JSP 2.0 (J2EE 1.4)
utilisation de la nouvelle interface `SimpleTag`



Pour des raisons de compatibilité avec les versions précédentes de JSP, les API antérieures ont été conservées

Accès aux contexte de la JSP

- Le Tag Handler accède à l'environnement JSP via l'objet **`JspContext`** qui lui a été transmis par le conteneur de servlet.
  - *la méthode **`getJspContext`** dans la classe **`SimpleTagSupport`** permet de récupérer sa référence*
- Principales méthodes de ***`JspContext`*** :

`JspWriter getOut()`	**Permet un accès à la variable `out` de la JSP**
`Object getAttribute(String name)`	**Renvoie l'objet défini dans la portée de la page associé au nom fourni en paramètre**
`Object getAttribute(String name, int scope)`	**Renvoie l'objet associé au nom fourni en paramètre et défini dans la portée définie par scope (`page, request, session, application`)**
`setAttribute(String name, Object obj)`	**Place dans la portée de la page un objet dont le nom est fourni en paramètre**
`void setAttribute(String name ,Object obj, int scope)`	**Place dans la portée définie par scope (**page, request, session, application**) un objet dont le nom est fourni en paramètre**
`void removeAttribute(String name,int scope)`	**Supprime de la portée définie par scope l'attribut associé au nom fourni en paramètre**
...	

Tag Library Descriptor

- Fichier au format XML décrivant une bibliothèque de Tags personnalisés
  - *Informations générales sur la bibliothèque*
  - *Description de chacun des tags personnalisés*

- Fichier utilisé par le conteneur Web à la compilation de la page JSP
  - *Pour valider les tags personnalisés*
  - *Pour générer le code Java correspondant*

- Fichier de description (Tag Library Descriptor) doit
  - *toujours avoir* ***`.tld`*** *comme extension*
  - *être placé dans le répertoire* ***`WEB-INF`*** *de l'application ou un de ses sous répertoires*
    - *En général dans un répertoire* ***`WEB-INF/tlds`***

Tag Library Descriptor

- Structure d'un fichier TLD

Schema XML auquel se conforme le fichier TLD

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-jsptaglibrary_2_0.xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>mataglib</short-name>
  <uri>/WEB-INF/tlds/MaTagLib</uri>
  ...
  <tag>
     ...
  </tag>
  <tag>
     ...
  </tag>

</taglib>
```

Caractéristiques générale de la bibliothèque

Caractéristiques d'un tag de la bibliothèque

Tag Library Descriptor

```
<tag>
    <name>monTag</name>
    <tag-class>tags.MonTagHandler</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>attribut1</name>
      <rtexprvalue>false</rtexprvalue>
      <type>java.lang.String</type>
    </attribute>
    <attribute>
      <name>attribut2</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
      <type>int</type>
    </attribute>
</tag>
```

Nom du tag (doit être unique dans la bibliothèque)

Nom entièrement qualifié du Tag Handler

Type du corps du Tag

Description d'un attribut du Tag

Nom de l'attribut

Présence obligatoire ou non

Indique si la valeur de l'attribut doit être évaluée lors de l'exécution.
Doit être à true, si la valeur de l'attribut est fournie avec un TAG JSP d'expression <%= %>

Type de l'attribut

Utilisation d'une Tag Lib

- Pour pouvoir être utilisée dans une page JSP, une bibliothèque de Tags doit être déclarée avec la directive **<%@ taglib>**

```
<%@ taglib  uri="/WEB-INF/tlds/MaTagLib.tld" prefix="maTagLib %>
```

Spécifie l'identité du fichier de description de la bibliothèque (fichier `.tld`)

Préfixe qui servira d'espace de noms pour les tags de le bibliothèque

Fichier `tld` désigné directement par son chemin relatif

Fichier `tld` désigné indirectement par un nom logique

```
<%@ taglib  uri="/laTagLib" prefix="maTagLib %>
```

Dans ce cas, la bibliothèque de tags personnalisés doit être enregistrée dans le fichier de déploiement de l'application (/WEB-INF/web.xml)

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

Utilisation d'une Tag Lib

- Appel d'un tag depuis la page JSP

```
<%@ taglib  uri="/WEB-INF/tlds/MaTagLib.tld" prefix="maTagLib %>
```

Le préfixe défini dans la directive `taglib`

```
<maTagLib:tag1/>
```
← Tag sans corps

```
<maTagLib:tag2>
   ...
<maTagLib:tag2/>
```
← Tag avec corps

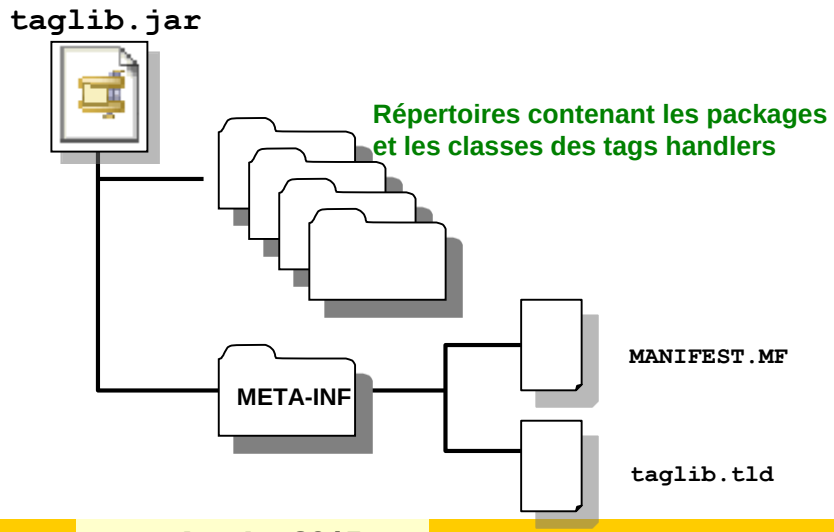
Corps : code HTML, code JSP, autre tag personnalisé

```
<maTagLib:tag3 attribut1="valeur"/>
```
← Tag avec un attribut constant

```
<maTagLib:tag4 attribut1="<%=uneVariable%>"/>
```
← Tag avec un attribut évalué dynamiquement lors de l'exécution de la page

Déploiement d'une bibliothèque de Tags

- Deux manières de déployer des bibliothèques de Tags :
  - *Sans les packager*
    - *Le fichier .tld de description doit se trouver dans **/WEB-INF** où un de ses sous répertoire (`/WEB-INF/tlds`)*
    - *Les classes (bytecode) des tag handlers doivent se trouver dans **/WEB-INF/classes***
  - *En les "packageant" dans un fichier jar*
    - *Le fichier jar doit être placé dans **/WEB-INF/lib***
    - *Il doit avoir la structure suivante*



Exemple : un Tag simple sans attribut

Page JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="/WEB-INF/tlds/mesTags.tld" prefix="mesTags"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemple d'une Page JSP avec un tag personalisé</title>
    </head>
    <body>
        <h2>Appel du TAG personalisé</h2>
        <H1><mesTags:HelloTag/></H1>
    </body>
</html>
```

Fichier TLD

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-jsptaglibrary_2_0.xsd">
    <tlib-version>1.0</tlib-version>
    <short-name>mestags</short-name>
    <tag>
        <name>HelloTag</name>
        <tag-class>mestags.HelloTagHandler1</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

Tag Handler

```
package mestags;

import java.io.IOException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class HelloTagHandler1 extends SimpleTagSupport {

    @Override
    public void doTag() throws JspException, IOException {

        JspWriter out = getJspContext().getOut();
        out.println("<strong> BONJOUR MONDE CRUEL !</strong>");

    }
}
```

Exemple : un Tag simple avec attribut

Page JSP

```jsp
<%@ taglib uri="/WEB-INF/tlds/mesTags.tld" prefix="mesTags"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemple d'une Page JSP avec un tag personalisé</title>
    </head>
    <body>
        <h2>Appel du TAG personalisé</h2>
        <mesTags:helloTag2 name="TOTO"/>
    </body>
</html>
```

Fichier TLD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi=
  <tlib-version>1.0</tlib-version>
  <short-name>mestags</short-name>
  ...
  <tag>
    <name>HelloTag2</name>
    <tag-class>mestags.HelloTagHandler2</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>name</name>
      <type>java.lang.String</type>
    </attribute>
  </tag>
</taglib>
```

Tag Handler

```java
package mestags;

import java.io.IOException;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.JspException;

public class HelloTagHandler2 extends SimpleTagSupport {

    private String name = "Monde cruel";

    public void doTag() throws JspException, IOException {

        JspWriter out = getJspContext().getOut();
        out.println("<strong> BONJOUR " + name + "</strong>");

    }

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

Exemple : un Tag simple avec corps

Page JSP

```
<mesTags:HelloAvecBodyTag name="TITI">
    Ceci est un texte pour  le corps (body) du TAG<BR>
    pour montrer comment cela marche
</mesTags:HelloAvecBodyTag>
...
```

Fichier TLD

```
...
<tag>
  <name>HelloAvecBodyTag</name>
  <tag-class>mestags.HelloAvecBodyTagHandler</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>name</name>
    <rtexprvalue>true</rtexprvalue>
    <type>java.lang.String</type>
  </attribute>
</tag>
</taglib>
```

Le corps d'une balise personnalisée ne supporte pas de code de scriptlet JSP <% ... %>

Tag Handler

```
package mestags;

import java.io.IOException;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.JspException;

public class HelloAvecBodyTagHandler extends SimpleTagSupport {

    private String name;

    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();

        out.println("<H2>HELLO " + name + "</H2>");
        out.println("    <BR>");

        JspFragment f = getJspBody();
        if (f != null) {
            f.invoke(out);
        }
        out.println("    <BR>");
        out.println("<H2>BYE " + name + "</H2>");
    }

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

`getJspBody()` permet de récupérer le `JSPFragment` qui correspond au corps du TAG

`invoke(Writer)` évalue le `JSFragment` et écrit le résultat dans le `Writer` spécifié

Exemple : Tags imbriqués

Page JSP

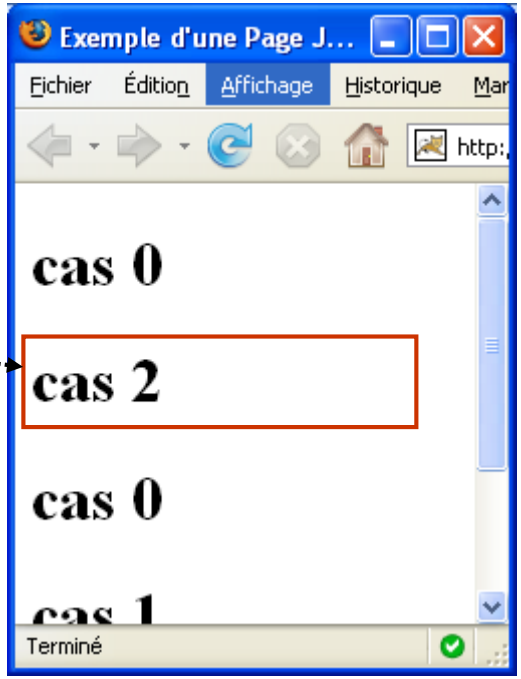
```
<% for (int i = 0; i < 5; i++) {
    Integer valTest = (int) (Math.random() * 3);%>
<Hl>
    <mesTags:switchtag test="<%=valTest.toString()%>">
        <mesTags:casetag value="0">cas 0</mesTags:casetag>
        <mesTags:casetag value="1">cas 1</mesTags:casetag>
        <mesTags:casetag value="2">cas 2</mesTags:casetag>
    </mesTags:switchtag>
</Hl>
<% }%>
```

Fichier TLD

```
<tag>
  <name>casetag</name>
  <tag-class>mestags.CaseTag</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>value</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
    <type>java.lang.String</type>
  </attribute>
</tag>
```

Tag Handler

```
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.JspException;

public class CaseTag extends SimpleTagSupport {

    private String value;

    @Override
    public void doTag() throws JspException, IOException {

        if (this.getParent() instanceof SwitchTag) {
            SwitchTag parent = (SwitchTag) this.getParent();
            if (parent.isValid(value)) {
                this.getJspBody().invoke(null);
                // identique à getJspBody().invoke(getJspContext().getOut());
            }
        } else {
            throw new JspException("Case doit être à l'intérieur du tag Switch");
        }
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```



Exemple : Tags imbriqués

Page JSP

```
<% for (int i = 0; i < 5; i++) {
    Integer valTest = (int) (Math.random() * 3);%>
<Hl>
    <mesTags:switchtag test="<%=valTest.toString()%>">
        <mesTags:casetag value="0">cas 0</mesTags:casetag>
        <mesTags:casetag value="1">cas 1</mesTags:casetag>
        <mesTags:casetag value="2">cas 2</mesTags:casetag>
    </mesTags:switchtag>
</Hl>
<% }%>
```

Fichier TLD

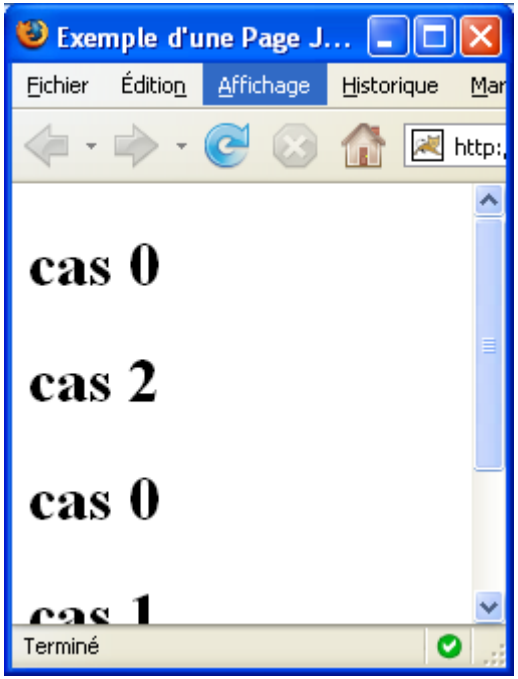
```
<tag>
  <name>switchtag</name>
  <tag-class>mestags.SwitchTag</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>test</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
    <type>java.lang.String</type>
  </attribute>
</tag>
```

Tag Handler

```
import javax.servlet.jsp.JspException;

public class SwitchTag extends SimpleTagSupport {

    private String test;

    public void doTag() throws JspException, IOException {

        JspWriter out = getJspContext().getOut();
        JspFragment f = getJspBody();
        if (f != null) {
            f.invoke(out);
        }

    }

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

    public boolean isValid(String caseValue) {
        if (test == null) {
            return false;
        }
        return test.equals(caseValue);
    }
}
```

le corps du tag est évalué

Vérifie que l'attribut `test` est le même que celui du *tag "enfant"*



Bibliothèques de Tags

- Nombreuses bibliothèques de tags existantes
  - *Libres*
  - *Commerciales*

- JSTL : Java Standard Tag Library for JavaServer Pages
  - *Bibliothèque standard développée par JSR 052*
    - *Tags de structure (itération, conditions)*
    - *Internationalisation*
    - *Requêtes SQL*
    - *…*
  - *Nécessite conteneur Web implémentant au moins API 2.3 des servlets et l'API JSP 1.2*