Introduction à JSP

Bien débuter avec Java Server Page

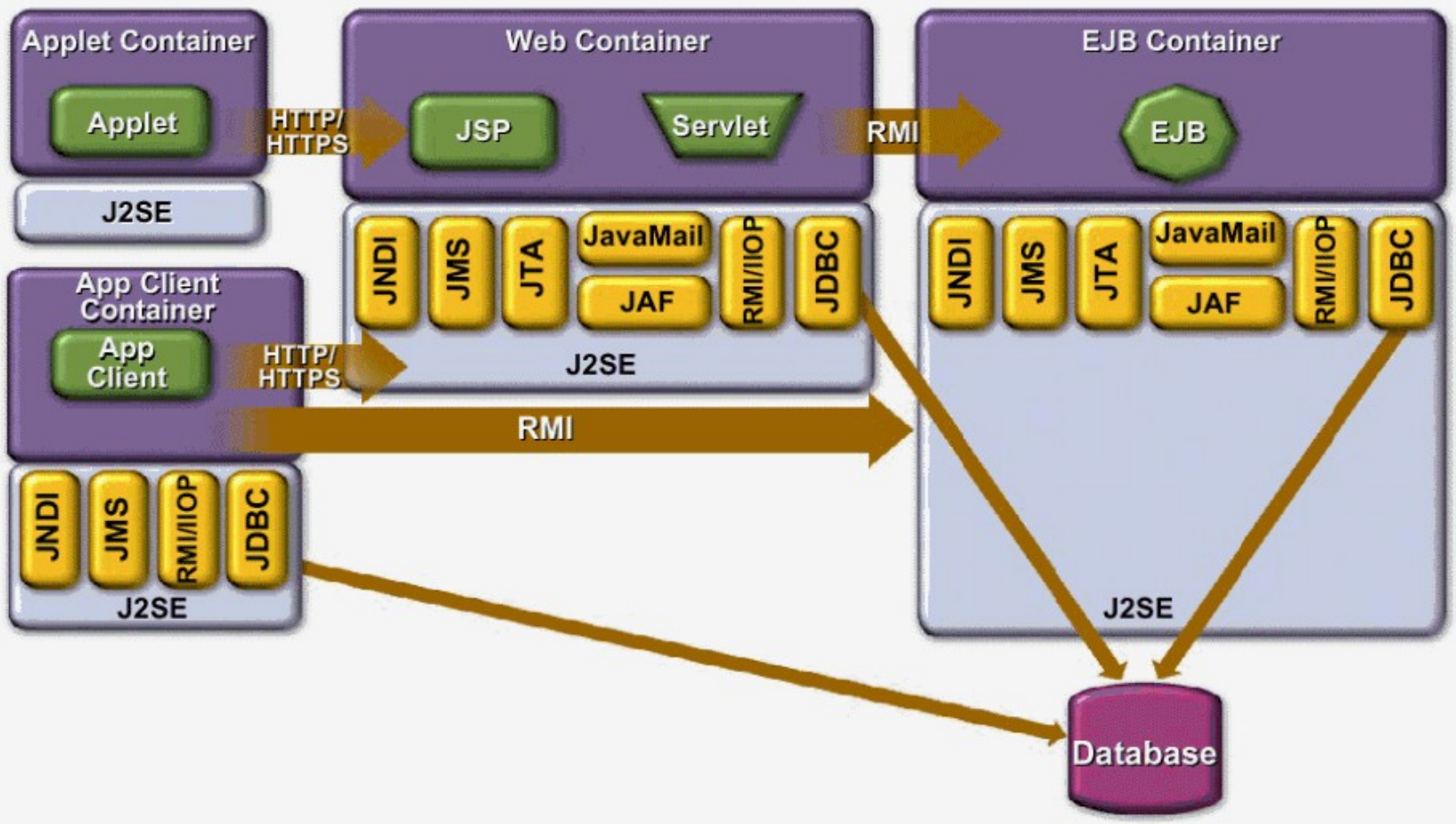
Mais qu'est ce que c'est ?

- Standard pour construire des applis Web
  - Portable
  - En mélangeant le HTML et des directives
  - Basé sur Java
- Equivalent PHP et ASP
  - Plus "propre" que PHP
  - Plus indépendant que ASP
- Plusieurs spécifications
  - JSP 2.0 = la dernière
- De nombreuses librairies : les TagLibs

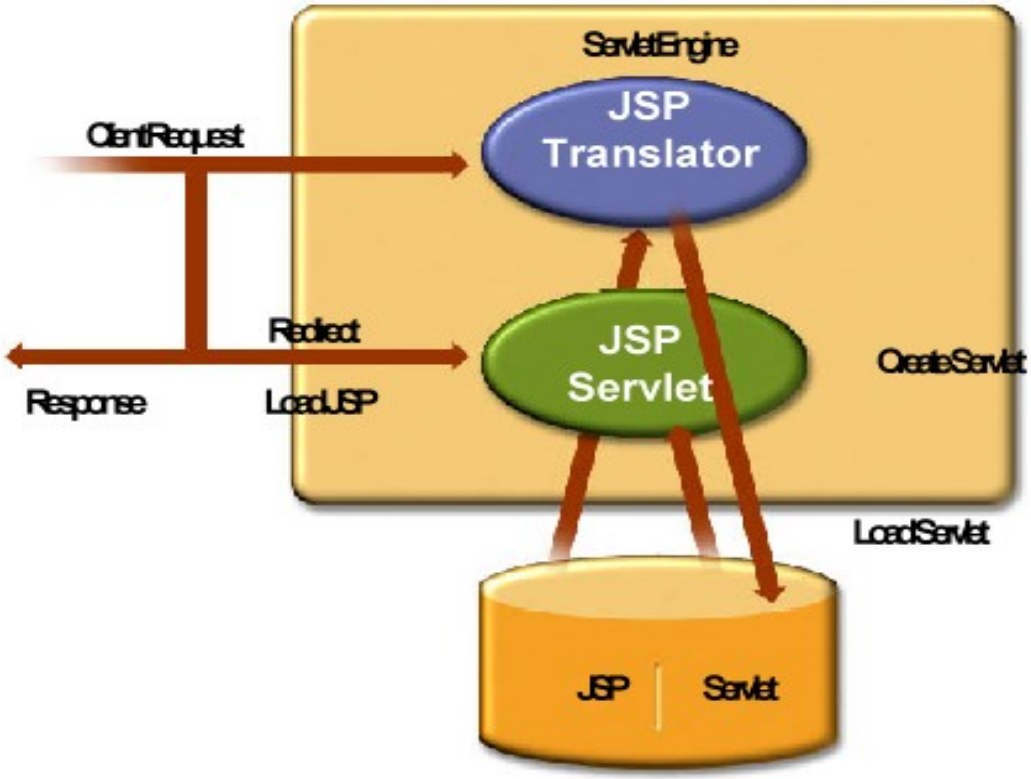
JSP et Servlets ?

- Les deux sont basés sur Java
- Les Servlets sont peu adapté à la génération de contenu
- Les JSP sont peu adaptés à l'extension de fonctions du serveur
- Note : Le JSP engine qui interprète les pages JSP est un Servlet
- Les JSP sont un moyen de construire des Servlet de façon déclarative
  - Un page JSP est transformée en un programme java (servlet) puis compilée et exécutée

JSP et Servlets



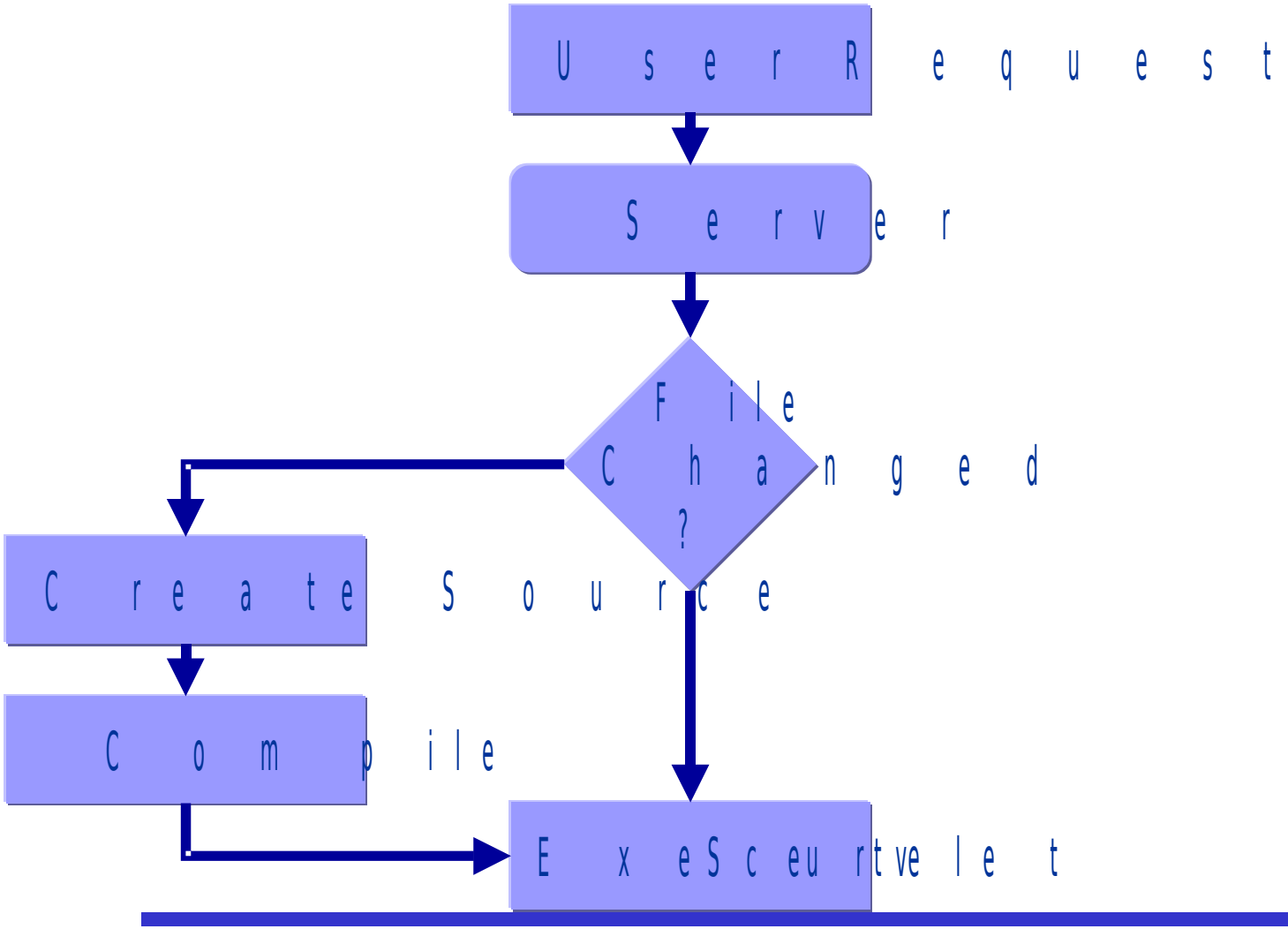
Architecture



Le cycle de vie d'un JSP

- JspInit()
- JspService
  - Accept request
  - Generate response
- JspDestroy()

Cycle de vie


User Request
Server
File Changed ?
Create Source
Compile
Execute servlet

La syntaxe de base

- Les directives
  - Instruction pour le moteur JSP
  - Encadrées par <%@ %>
- Les déclarations
  - Déclarations de variables ou de méthodes utilisables dans la page
  - Encadrées par <%! %>
- Les expressions
  - Un expression est évaluée, transformée en chaine et incluse dans la page
  - Encadrées par <%= %>
- Les scriplets
  - Morceau de code java exécuté dans la page
  - Encadrés par <% %>

Les expressions

- Une expression est une expression java dont la valeur est calculée, tranformée en String et insérée dans la page.

```
<body>
    <h2>Hello World!</h2>
    Aujourd'hui on est le
    <%=new java.util.Date(System.currentTimeMillis()).toString()%>
</body>
```

Hello World!

Aujourd'hui on est le Mon Feb 11 10:32:29 CET 2008

```
    out.write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"\n");
    out.write("   \"http://www.w3.org/TR/html4/loose.dtd\">\n");
    out.write("\n");
    out.write("<html>\n");
    out.write("    <head>\n");
    out.write("        <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\n");
    out.write("        <title>JSP Page</title>\n");
    out.write("    </head>\n");
    out.write("    <body>\n");
    out.write("        <h2>Hello World!</h2>\n");
    out.write("        Aujourd'hui on est le \n");
    out.write("        ");
    out.print(new java.util.Date(System.currentTimeMillis()).toString());
    out.write("\n");
    out.write("    </body>\n");
    out.write("</html>\n");
  } catch (Throwable t) {
    if (!(t instanceof SkipPageException)){
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        try { out.clearBuffer(); } catch (java.io.IOException e) {}
      if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
    }
  } finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
  }
```

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {

  PageContext pageContext = null;
  HttpSession session = null;
  ServletContext application = null;
  ServletConfig config = null;
  JspWriter out = null;
  Object page = this;
  JspWriter _jspx_out = null;
  PageContext _jspx_page_context = null;


  try {
    response.setContentType("text/html;charset=UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("\n");
    out.write("\n");
    out.write("\n");
    out.write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"\n");
    out.write("   \"http://www.w3.org/TR/html4/loose.dtd\">\n");
    out.write("\n");
```

Les déclarations

- Déclarations de variables et de fonctions locales à la page (en java)

```
<%! int i; %>
<%!
    int i=0;
    String text;
%>
<%!
  boolean f(int i)
  {return (i>0);}
%>
```

Les scriplets

- Peuvent contenir des fragments de code valide en java
- Ils sont exécutés quand la requête est évaluée

```
<% if (compte>0) { %>
Tu as encore des sous… <BR>
<% } else { %>
Tu es dans le rouge cette fois <BR>
<% } %>
```

J'oubliais les commentaires

- Les commentaires sont les mêmes qu'en XML
  - <%-- tagada --%>

Les objets implicites

- Objets utilisables sans déclaration dans les expressions et les scriplets
  - request (request scope) : HttpServletRequest
  - response (page scope) : HttpServletResponse
  - pageContext (page scope) : PageContext
    - L'objet représentant le contexte de la page
  - session (session scope) : HttpSession
    - L'objet représentant le contexte de la session
  - page (=this) (page scope) : HttpJSPPage
  - out (page scope) : JspWriter
  - application (application scope) ServletContext
  - config (page scope) : ServletConfig
  - exception (page scope) : Throwable
    - Définit dans les pages de traitement des erreurs

Exemple

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h2>Parameters !</h2>
        <%=request.getParameter("p1")%><br>
        <%=request.getParameter("p2")%>
    </body>
</html>
```

Parameters !

test

param

L'interface de PageContext

- Fournit des méthodes pour retrouver des attributs en suivant la visibilité
  - Object findAttribute(String name)
    - Recherche un attribut en utilisant l'ordre suivant : page, request, session, application
  - Object getAttribute(String name)
    - Retourne un attribut dans le contexte de la page
  - Object getAttribute(String name, in ctx)
    - Retourne un attribut dans le contexte donné (APPLICATION_SCOPE, PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE)
  - Int getAttributeScope(String name)
    - Retourne le scope d'un attribut
- Fournit aussi des méthodes pour retrouver toutes les informations du contexte d'exécution de la page (session, jspWriter,request, response,…)
  - Cf PageContext API pour les détails

Syntaxe XML

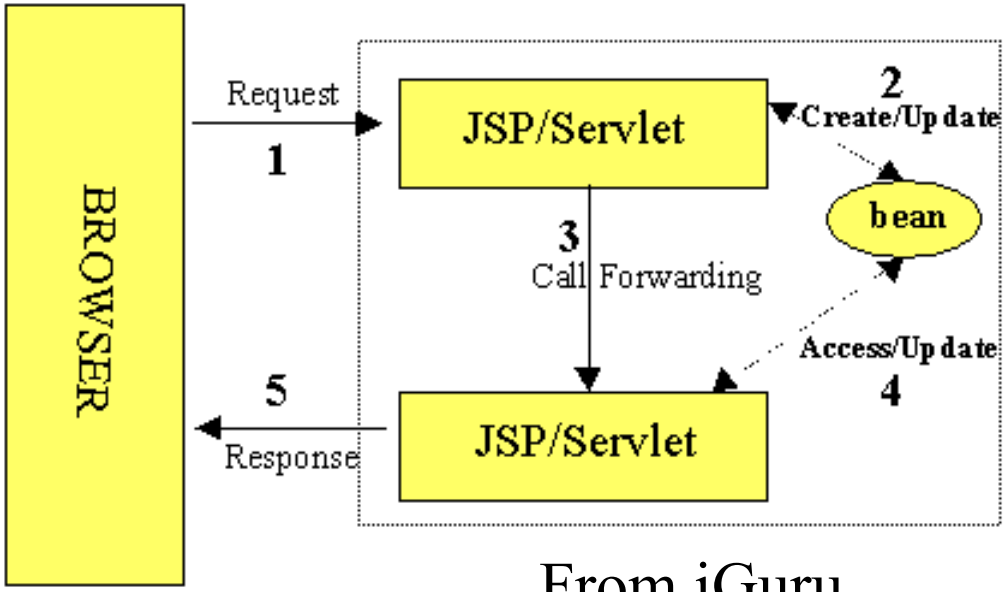
- Depuis JSP 1.2
  - <jsp:expression>Expression</jsp:expression>
  - <jsp:scriptlet>
  - <jsp:declaration>
- Permet
  - La validation
  - Les transformations
  - L’édition

Exemple

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">

    <jsp:directive.page contentType="text/html" pageEncoding="UTF-8"/>

    <!-- any content can be specified here, e.g.: -->
    <jsp:element name="text">
        <jsp:attribute name="lang">EN</jsp:attribute>
        <jsp:body>Hello World!<br/>
        <jsp:expression>new java.util.Date(System.currentTimeMillis()).toString()</jsp:ex
        </jsp:body>
    </jsp:element>

</jsp:root>
```

jsp:forward

- Jsp:forward permet de chainer les requêtes pour invoquer
  - Une autre page jsp
  - Un servlet
- <jsp:forward page="AutrePage.jsp"/>
- <jsp:forward page="/Servlet/control/>



From jGuru

Jsp:forward avec paramètres

- Il est possible d'ajouter des paramètres à la requête.
- Accessibles par request.getAttribute(name)

```
<jsp:forward page="chaine.jsp">
        <jsp:param name="name1" value="v1"/>
        <jsp:param name="name2" value="v2"/>
</jsp:forward>
```

Jsp:include

- Redirige la requête et inclut le résultat à l'endroit où se trouve l'action
  - <jsp:include page="checkBean.jsp"/>
  - Cette action est exécutée à chaque fois
- L'action exécutée ne peut pas modifier le Header de la page (pas de cookie ni de type de contenu)

Les tags JSP (ou actions)

- Les tags sont des actions incluses dans une page Web suivant la syntaxe XML
  - <mod:tag attr="value">
  - <mod:tag attr="value">body</mod:tag>
- Les actions de base font partie de la librairie jsp:
  - <jsp:useBean>
  - <jsp:setProperty>
  - <jsp:getProperty>
  - <jsp:include>
  - <jsp:forward>
  - <jsp:text>

Un JavaBean

- Composant simple. Respecte des conventions d'écriture

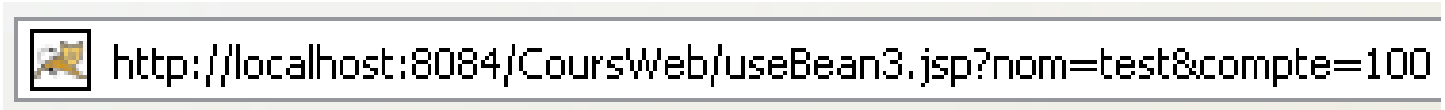
```
public class MyBean {
    private String nom;
    private int compte;
    private Date date;

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public int getCompte() {
        return compte;
    }

    public void setCompte(int compte) {
        this.compte = compte;
    }

    public Date getDate() {
        return date;
    }
```

Exemple UseBean

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
    <h2>Use Bean</h2>
    <jsp:useBean id="mybean" scope="page" class="exemple.MyBean" />
    <jsp:setProperty name="mybean" property="nom" value="Test" />
    <jsp:setProperty name="mybean" property="compte" value="0" />

    <jsp:getProperty name="mybean" property="nom"/>
    </body>
</html>
```

Autre exemple

```
<body>
    <h2>Use Bean 2</h2>
    <jsp:useBean id="bean2" scope="page" class="exemple.MyBean" />
    <jsp:setProperty name="bean2" property="nom" param="nom" />
    Valeur du paramètre = <jsp:getProperty name="bean2" property="nom" />
</body>
<body>
    <h2>Use Bean 3</h2>
    <jsp:useBean id="mybean" scope="request" class="exemple.MyBean" />
    <jsp:setProperty name="mybean" property="*" />
    Nom= <jsp:getProperty name="mybean" property="nom" /> <br>
    Compte = <jsp:getProperty name="mybean" property="compte" />
</body>
```



useBean et scope

```
<body>
    <h2>Scope 1</h2>
    <jsp:useBean id="aBean" scope="session" class="exemple.MyBean" />
    <jsp:setProperty name="aBean" property="*"/>

</body>
```

```
<body>
    <h2>Scope 2</h2>
    <jsp:useBean id="aBean" scope="session" class="exemple.MyBean" />
    <jsp:getProperty name="aBean" property="nom"/>
</body>
```

Scope 2

test

Code généré

```
synchronized (session) {
  aBean = (exemple.MyBean) _jspx_page_context.getAttribute("aBean", PageContext.SESSION_SCOPE);
  if (aBean == null){
    aBean = new exemple.MyBean();
    _jspx_page_context.setAttribute("aBean", aBean, PageContext.SESSION_SCOPE);
  }
}
```

JavaBean et JSP

- Les action useBean, setProperty et getProperty permettent de manipuler des JavaBean sans programmation
  - jsp:usebean pour nommer, créer ou désigner un bean
  - jsp:getProperty pour récupérer une propriété d'un bean
  - jsp:setProperty pour changer la valeur d'une propriété.

Les directives

- <%@ directive {attr="value"}* %>
  - Messages passés au moteur de JSP
- Page : propriétés de la page
  - **extends**="*ClassName"*
    - La page est une sous-classe de la classe indiquée
  - **import**="javax.jms.*,cour.test.util"
    - import des types ou des packages
    - Les packages lang, servlet et jsp sont importés par défaut
  - **session**="true" ou "false" (defaut=true)
    - La page participe à une session
  - **isThreadSafe**
  - **buffer**=16k
    - Taille du buffer par défaut pour le PrintWriter.
  - **autoFlush**="true" ou "false"
    - Vide le buffer automatiquement quand le buffer est plein

Les directives (2)

- Page -la suite
  - isErrorPage="true" ou "false"
    - La page est une page d'erreur
  - errorPage="/error.jsp"
    - Page chargée en cas d'exception
  - contentType="text/html;charset=UTF-8"
    - Type du contenu de la page et jeu de caractères
- Include
  - File="relative URL"
    - Insère le fichier correspondant dans le fichier JSP
    - Le fichier est évalué et compilé
  - page="test.html”
    - Inclut la ressource sans l'interpréter

Les propriétés

- El-enabled : permet l'utilisation du langae d'expression
- Scripting-enabled : permet l'utilisation du langage de script
- Page-encoding :
- Include-prelude : ajoute un header à toutes les pages
- Include-code : ajoute un footer après la génération de la page
- Is-xml : permet la validation XML des pages JSP

La JSTL

- But : Simplifier l'écriture des pages
- Langage pour les auteurs de pages
- Inclut un langage d'expression pour éviter le code java :
  - Langage d'expression pour calculer des valeurs
  - Contrôle du flot d'exécution
  - Validateur de librairie de tags
    - Permet des contrôler le style et les librairies utilisables dans une page

La JSTL

- JSTL= Java Standard Tag Library
  - Core tags
    - Sortie, gestion des variables, logique, boucles, importation de texte
  - XML tags
    - Parsing, sélection, XSLT, flot de contrôle
  - Database tags
    - Connection, accès, requêtes, mises à jour et transactions
  - Formattage
    - Localisation, internationalisation

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

Deux types de librairie

- Les librairies Expression Language (EL)
  - Dans le langage d'expression, les expressions sont encadrées par ${}
  - Les variables peuvent être dynamiques
- Request Time Langage (RTL)
  - Ici ce sont des expressions java encadrées par <%= %>
- Les deux types d'expressions peuvent être mixées
- La RTL garantit la type safety et les perfs sont meilleures (compilation)
- L'EL est plus simple et plus facile à lire

Le langage d'expression

- Le langage d'expression est invoqué par
  - ***${expr}***
- Des objets implicites permettent d'accéder aux contextes de l'environnement
  - **pageScope**, **requestScope**, **sessionScope**, **applicationScope** donnent accès aux variables de l'environnement
  - **param** et **paramValues** donnent accès aux paramètres d'une requête
    - ***${param["nom"]}*** donne la valeur du paramètre nom d'une requête http
    - ***${paramValues["nom"]}*** donne toutes les valeurs associées au paramètre nom de la requête http dans un tableau
  - **pageContext** donne accès aux valeurs associées au contexte
  - **initParams** donne accès aux paramètres d'initialisation
  - **headerValues** donne accès aux propriétés du header

Exemple : accès aux paramètres

```
<body>
    <h2>Param JSP ${param.nom}</h2>
    <c:forEach var="aParam" items="${paramValues}">
        param : ${aParam.key}<br>
        <c:forEach var="value" items="${aParam.value}">
            value : ${value}<br>
        </c:forEach>
    </c:forEach>
</body>
```

http://localhost:8084/CoursWeb/params.jsp?nom=test&a=1&a=2&a=3&b=true

Param JSP test

param : b
value : true
param : a
value : 1
value : 2
value : 3
param : nom
value : test

Exemple avec conditionnelle

```
<body>
    <h2>Hello</h2>
    <c:if test="${empty param.nom}" var="res">
        Saisissez votre nom.
    </c:if>
    ${param.nom}
</body>
```

Beans et Collection

- L'accès aux propriétés d'un bean se fait par un notation pointée
  - ${cd.titre}
  - ${personne.adresse.ville}
- Pour les collections (Map, List, Array) on peut utiliser la notation
  - mesCD[i] si mesCD est un tableau
  - mesCD[cd.titre] si c'est une Map avec le titre comme clé

Les opérateurs

- On dispose de tous les opérateurs classiques
  - +,-,*,/, mod,
  - == ou eq, != ou ne, < ou lt, > ou gt (pour éviter les soucis avec XML
  - empty pour vérifier si une variable est nulle ou vide
- Les conversions de types sont automatiques dans les expressions en fonction des besoins

Les actions générales

- c:out pour générer du texte (équivallent du <%= %>
  - <c:out value=${param.nom} default="N/A"/>
  - Affiche la valeur du paramèter ou N/A si elle est vide
- c:set affecte la valeur d'une variable dans le contexte courant
  - <c:set var="test" value="ca marche ?" scope="request"/>
  - La valeur peut être aussi calculée dans le body

```
<c:set var="test2>
    <ex:calc param1="pouet"/>
</c:set>
```

  - Peut être aussi utilisé pour changer la valeur d'une propriété d'un bean
  - <c:set target=${cd} property="titre" value="Mezzanine"/>
  - Peut être aussi utilisé pour ajouter un élément dans une collection
  - <c:set target=${myCD} property=${cd.titre} value=${cd}/>

Les actions générales (suite)

- <c:remove/> permet de supprimer une variable
  - <c:remove var="test" scope="page"/>
  - Supprime la variable test de la page
- <c:catch/> permet de capturer les exceptions dans une page

```
<c:catch var="exception">
    … some code
</c:catch>
<c:if test=”${exception != null}”>
   Il y a eu un souci
</c:if>
```

  - Attention ça ne remplace pas le traitement normal des erreurs

Les conditionnelles

- Les conditions simples
  - <c:if test="${cd.annee<1990}">
  - C'est un vieux disque
  - </c:if>
- Les conditions exclusives

```
<c:choose>
    <c:when test="${cd.auteur=='bob'}">
        Quelle daube
    </c:when>
    <c:when test=="${cd.auteur=='bill'}">
        Ca swing
    </c:when>
    …
    <c:otherwise>
        Beuh ?
    </c:otherwise>
</c:choose>
```

Les Iterateurs

- c:forEach répéte une action pour tous les éléments d'une collection

```
<table>
  <c:forEach var=”cd” items=”${myCD}”>
 <tr><td><c:out value=”${cd.titre}”/></td></tr>
</c:forEach>
</table>
```

- On peut aussi définir un interval

```
<c:forEach var=”i” begin=”100” end=”110”>
<c:out value=”${i}”/>
</c:forEach>
```

Encore des itérateurs

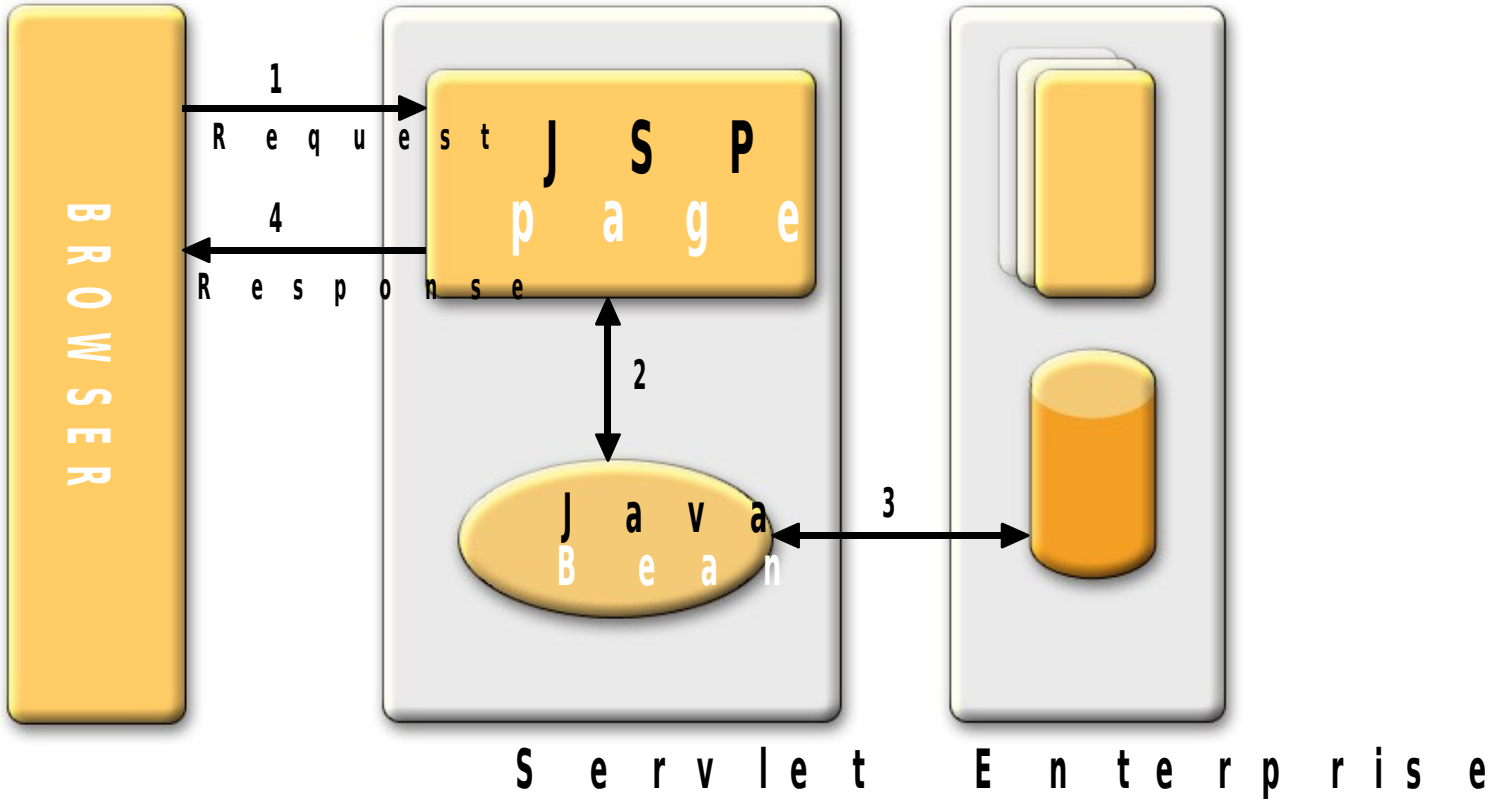
- c:forTokens permet de parcourir une chaine contenant des délimiteurs

```
<c:set var="liste" value="a,b,c,d"/>
<c:forTokens var="item" items="${liste}" delims=",">
    <c:out value="${item}"/>
</c:forTokens>
```

Architecture d’une application Web

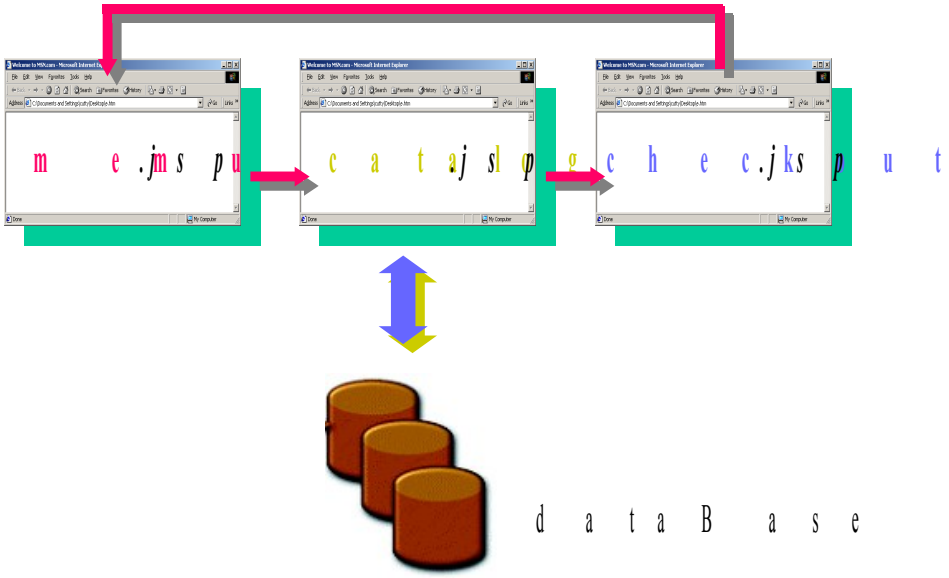
- Pas de MVC
- MVC Model 1 (Centré page)
- MVC Model 2 (Centré servlet)
- Web app framework (struts)
- Standard framework (JSF)

Centré page

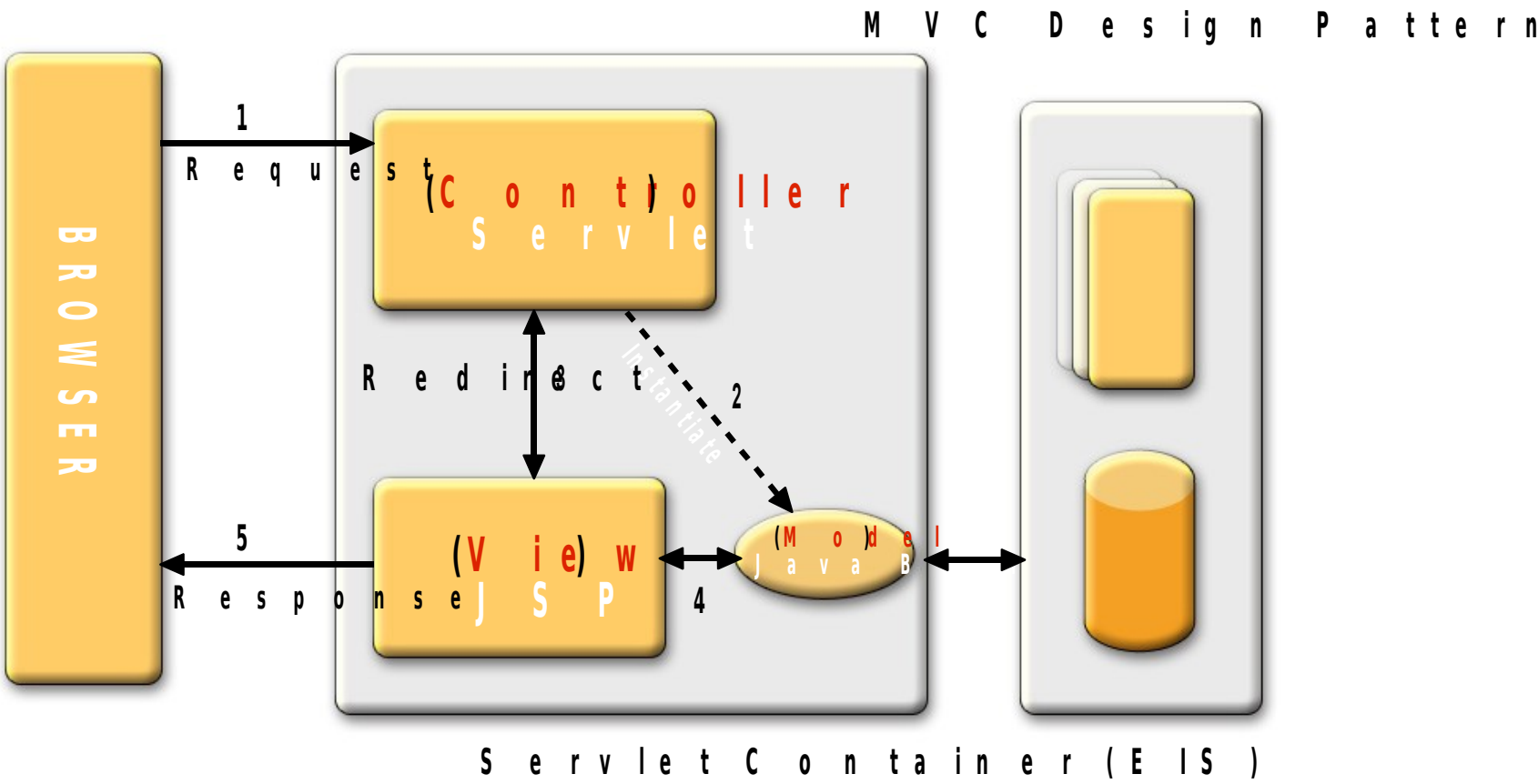

BROWSER
1
Request
4
Response
JSP
page
2
Java
Bean
3
Servlet
Enterprise

Centré page

- La logique de l'application est dans les pages JSP
- L'enchainement des pages est lui même codé dans les pages.



MVC Model 2 (Servlet)



Model 2

- Le controleur prend en charge la requête
- Il effectue les contrôles et les actions
- Il sélectionne la vue
- Le servlet sert de
  - Filtre (erreur, sécurité)
  - Controleur (dispatching, actions, redirection)

Exemple simple

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    String pathInfo=request.getPathInfo();
    if (pathInfo.equals("/add")) {
        String nom=request.getParameter("nom");
        String password=request.getParameter("passwd");
        if (nom==null || password==null) {
            request.setAttribute("message","Un champ n'a pas été rempli");
            request.setAttribute("nom",nom);
            RequestDispatcher rd=getServletContext().getRequestDispatcher("/newUser.jsp");
            rd.forward(request, response);
        } else {
            User u=new User();
            u.setNom(nom);
            u.setPasswd(password);
            UserBase ub=(UserBase) getServletContext().getAttribute("base");
            ub.addUser(u);
            RequestDispatcher rd=getServletContext().getRequestDispatcher("/listUser.jsp");
            rd.forward(request, response);
        }
    } else if (pathInfo.equals("/list")) {
            RequestDispatcher rd=getServletContext().getRequestDispatcher("/listUser.jsp");
            rd.forward(request, response);
```

Exemple suite

```
} else if (pathInfo.equals("/delete")) {
        String nom=request.getParameter("nom");
        UserBase ub=(UserBase) getServletContext().getAttribute("base");
        ub.remove(nom);
        RequestDispatcher rd=getServletContext().getRequestDispatcher("/listUser.jsp");
        rd.forward(request, response);
} else if (pathInfo.equals("/change")) {
        String nom=request.getParameter("nom");
        UserBase ub=(UserBase) getServletContext().getAttribute("base");
        User u=ub.getUser(nom);
        request.setAttribute("user", u);
        RequestDispatcher rd=getServletContext().getRequestDispatcher("/changeUser.jsp");
        rd.forward(request, response);
} else if (pathInfo.equals("/update")) {
        String nom=request.getParameter("nom");
        String oldpass=request.getParameter("oldpasswd");
        String newpass=request.getParameter("passwd");
        UserBase ub=(UserBase) getServletContext().getAttribute("base");
        User u=ub.getUser(nom);
        if (oldpass.equals(u.getPasswd())) {
            u.setPasswd(newpass);
        }
        RequestDispatcher rd=getServletContext().getRequestDispatcher("/newUser.jsp");
        rd.forward(request, response);
}
```

Demo

Liste Users

Nom	Password		
rototo	rototo	del	change
titi	titi	del	change
sd	w	del	change

new user ?

Accès aux bases de données

- La JSTL fournit une alternative à des appels JDBC depuis un servlet pour utiliser un SGBD
  - <sql:query>
  - <sql:update>
  - <sql:transaction>
  - <sql:setDatasource>
- Attention : il n'est pas toujours judicieux de mélanger présentation et traitements
- La librairie à insérer:
  - <%@ taglib uri="/tld/sql.tld" prefix="sql" %>

La source de donnée

- Une source de donnée permet d'établir une connection vers la base qu'elle représente.
  - Elle peut être désignée par un nom dans un serveur JNDI si le container le permet
  - Elle peut être désignée par une URL transmise au driver JDBC

```
<sql:setDataSource
{dataSource="dataSource" |
url="jdbcUrl"
[driver="driverClassName"]
[user="userName"]
[password="password"]}
[var="varName"]
[scope=”{page|request|session|application}”]/>
```

Les requêtes

Requête sans arguments

```
<sql:query sql="sqlQuery"
var="varName" [scope="{page|request|session|
application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]/>
```

Requête avec arguments dans le corps

```
<sql:query sql="sqlQuery"
var="varName" [scope="{page|request|session|
application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]>
<sql:param> actions
</sql:query>
```

Exemple de requête

```
<sql:query var="customers" dataSource="${dataSource}">
   SELECT * FROM customers
   WHERE country = 'China'
   ORDER BY lastname
</sql:query>
<table>
  <c:forEach var="row" items="${customers.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>
```

Jsp:update

```
<sql:update sql="sqlUpdate"
  [dataSource=”dataSource”]
  [var="varName"]
  [scope=”{page|request|session|application}”]/>
```

```
<sql:update sql="sqlUpdate"
  [dataSource=”dataSource”]
  [var="varName"]
  [scope=”{page|request|session|application}”]>
    <sql:param> actions
</sql:update>
```

Sql:transaction

- Le tag transaction permet d'encadrer un séquence d'accès à la base dans une transaction
  - Différents niveaux d'isolation possibles

```
<sql:transaction [dataSource=”dataSource”]
[isolation=isolationLevel]>
<sql:query> and <sql:update> statements
</sql:transaction>
isolationLevel ::= "read_committed"
| "read_uncommitted"
| "repeatable_read"
| "serializable"
```

Update exemple

```
<sql:transaction dataSource="${dataSource}">
  <sql:update>
    UPDATE account
    SET Balance = Balance - ?
    WHERE accountNo = ?
    <sql:param value="${transferAmount}"/>
    <sql:param value="${accountFrom}"/>
 </sql:update>

  <sql:update>
    UPDATE account
    SET Balance = Balance + ?
    WHERE accountNo = ?
    <sql:param value="${transferAmount}"/>
    <sql:param value="${accountTo}"/>
  </sql:update>
</sql:transaction>
```

Un dernier truc: <sql:dateParam>

- Un tag pour convertir un objet java.util.Date dans
  - java.sql.date
  - java.sql.time
  - java.sql.timestamp
- S'utilise comme <sql:param>

```
<sql:dateParam value="value" type="[timestamp|time|date]"/>
```

Internationalisation

- Permettre de construire des pages tenant compte
  - De la langue,
  - Des différentes usages culturels
    - Format des nombres
    - Format de dates
- Deux façons de procéder
  - Construire une version de page par langue et un dispatcher qui transmet la requête en fonction de l'utilisateur
  - Utiliser des fichiers de ressources pour localiser chaque partie sensible de la page
  - La librairie d'internationalisation permet les deux approches.

Trois concepts clés

- Locale
  - Représente une région géographique, culturelle ou politique
  - Code langage à deux lettres en minuscule normalisé (iso-639)
  - Code pays à deux lettres en majuscule
- Ressource bundle
  - Contient des objets spécifiques à un context local (des messages)
  - Chaque objet est associé à une clé
- Basename
  - Permet d'identifier un ressource bundle
  - Associé à un code local, donne les ressources pour ce pays ou cette langue
  - Si Registration est un basename, Registration_fr est sa version française.

Le ressource bundle

- Il est déterminé à partir du basename et des suffixes régionaux et linguistiques
  - Ordres :
  - basename + "_" + language + "_" + country + "_" + variant
  - basename + "_" + language + "_" + country
  - basename + "_" + language
- Les préférences de l'utilisateurs sont positionnées dans son navigateur

<fmt:setLocale>

- Choix forcé du modèle "culturel"

```
<fmt:setLocale value="locale"
[variant="variant"]
[scope="{page|request|session|
  application}"]/>
```

- Si cette fonction est utilisée les choix de l'utilisateur sont annulés

<fmt:bundle>

- Construction d'une ressource linguistique spécifique.
- Crée un contexte et charge une ressource dans ce contexte

```
<fmt:bundle basename="basename" [prefix="prefix"]>
    body content
</fmt:bundle>

<fmt:bundle basename="Labels" prefix="com.acme.labels.">
<fmt:message key="firstName"/>
<fmt:message key="lastName"/>
</fmt:bundle>
```

<fmt:setBundle>

- Crée un contexte de localisation et le stocke dans la variable indiquée

```
<fmt:setBundle basename="basename"
   [var="varName"]
   [scope="{page|request|session|
  application}"]/>
```

Basename est le nom qualifié (avec package) de la ressource mais sans extension

Il peut correspondre à un fichier .properties disponible au déploiement de l'application (dans le classpath)

<fmt:message>

- Permet d'afficher un message à partir de sa clé

```
<fmt:message key="messageKey"
   [bundle="resourceBundle"]
   [var="varName"]
   [scope="{page|request|session|application}"]/>
```

– Avec des paramètres

```
<fmt:message key="messageKey"
   [bundle="resourceBundle"]
   [var="varName"]
   [scope="{page|request|session|application}"]>
   <fmt:param> subtags
</fmt:message>
```

Un fichier de propriétés

```
listeCD=Liste de CD
saisieCD=Saisie d'un nouveau CD
nom=nom
titre=titre
annee=année
parametre= il manque un paramètre. je ne conserve pas le CD
ajout= on ajoute le CD à la liste
affichage= affichage de la
nbCD= il y a {0} CD
```

MessageBundle.properties

Pour un paramètre

```
listeCD=CD List
saisieCD=CD entry
nom=name
titre=title
annee=year
parametre= A parameter is missing. CD is cancelled
ajout= CD is recorded
affichage= Listing of CDs
nbCD= there are {0} CDs
```

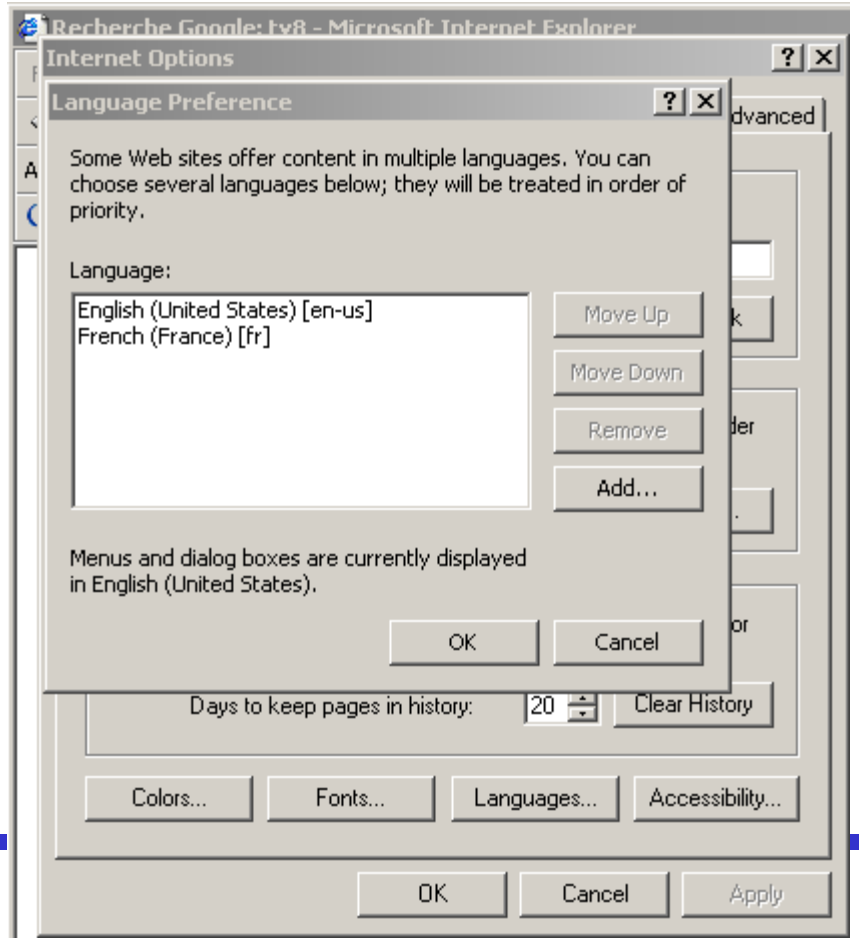
MessageBundle_en_US.properties

Exemple d'utilisation

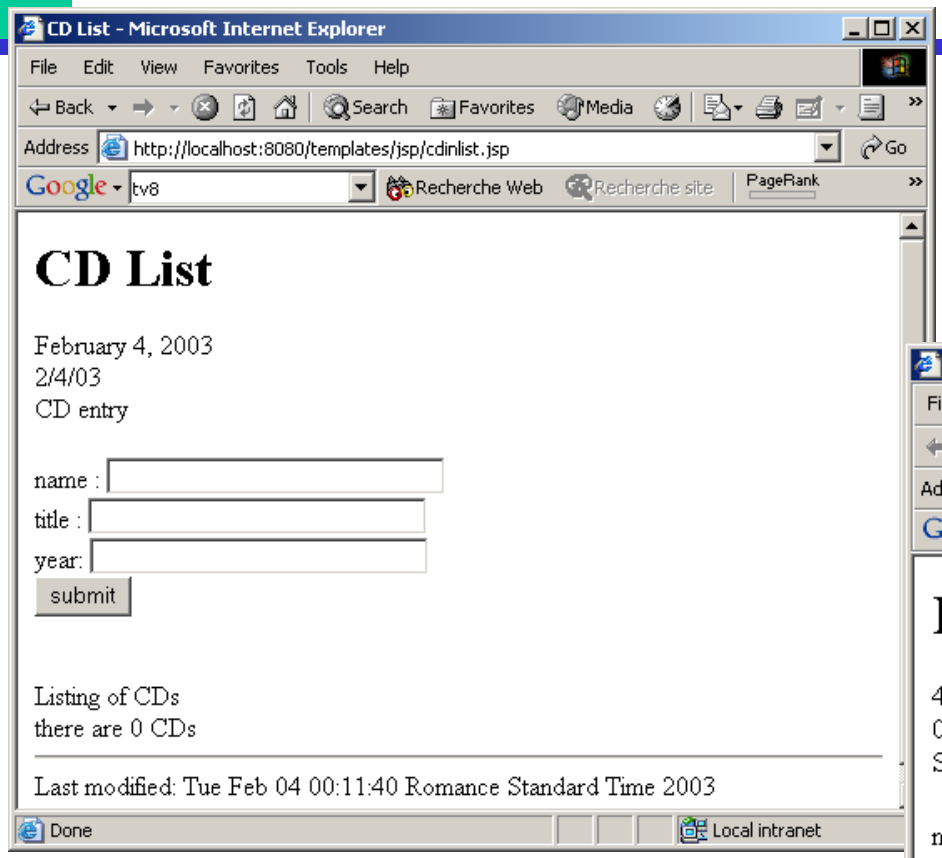
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<%@ taglib uri="/tld/c.tld" prefix="c" %>
<%@ taglib uri="/tld/fmt.tld" prefix="fmt" %>
<%@ page session="true" %>
<fmt:setBundle basename="MessageBundle"/>

<%! java.util.Map myCD=new java.util.Hashtable(); %>
<% session.setAttribute("myCD",myCD); %>

<html>
  <body>
    <h1><fmt:message key="listeCD"/></h1>
<fmt:message key="affichage"/>
<br>
<fmt:message key="nbCD">
  <fmt:param>
     <%=myCD.size()%>
  </fmt:param>
</fmt:message>
    <hr>
</body>
</html>
```

Les options du navigateur

- En fonction de l'ordre, la page s'affichera dans une langue ou l'autre
- (cf cdinlist.jsp)



VO/VF


CD List - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/templates/jsp/cdinlist.jsp
Go
Google tv8
Recherche Web
Recherche site
PageRank
CD List
February 4, 2003
2/4/03
CD entry
name :
title :
year:
submit
Listing of CDs
there are 0 CDs
Last modified: Tue Feb 04 00:11:40 Romance Standard Time 2003
Done
Local intranet


Liste de CD - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/templates/jsp/cdinlist.jsp
Go
Google
Recherche Web
Recherche site
PageRank
Liste de CD
4 février 2003
04/02/03
Saisie d'un nouveau CD
nom :
titre :
année:
submit
affichage de la liste
il y a 0 CD
Last modified: Tue Feb 04 00:11:40 Romance Standard Time 2003
Done
Local intranet

<fmt:timeZone>

- Spécifie le fuseau horaire pour l'affichage du contenu (il peut différer entre le serveur et le client)
- Par défaut, GMT
- Voir java.util.TimeZone pour les time zone possibles

```
<fmt:timeZone value="timeZone">
body content
</fmt:timeZone>
```

<fmt:formatNumber>

- Formattage d'une valeur numérique ou monétaire
- La valeur numérique peut être un attribut ou dans le contenu

```
<fmt:formatNumber value=”numericValue”
  [type=”{number|currency|percent}”]
  [pattern=”customPattern”]
  [currencyCode=”currencyCode”]
  [currencySymbol=”currencySymbol”]
  [groupingUsed=”{true|false}”]
  [maxIntegerDigits=”maxIntegerDigits”]
  [minIntegerDigits=”minIntegerDigits”]
  [maxFractionDigits=”maxFractionDigits”]
  [minFractionDigits=”minFractionDigits”]
  [var=”varName”]
  [scope=”{page|request|session|application}”]/>
```

<fmt:formatDate>

- Formattage de la date (java.util.Date)
- Cf java.text.DateFormat pour les styles utilisables

```
<fmt:formatDate value="date"
  [type="{time|date|both}"]
  [dateStyle="{default|short|medium|long|full}"]
  [timeStyle="{default|short|medium|long|full}"]
  [pattern="customPattern"]
  [timeZone="timeZone"]
  [var="varName"]
  [scope="{page|request|session|application}"]/>
```

Les Custom Tags

- API de la spécification JSP permettant de créer ses propres tags JSP
- Un custom tag c'est une extension du langage JSP
  - Quand la page JSP est transformée en servlet, le custom tag est transformé en operations appelées sur un Tag Handler
  - Il peuvent recevoir des attributs
  - Accéder aux objets du contexte de la page
  - Modifier la réponse de la page
  - Communiquer entre eux
  - Etre imbriqués

Intérêt des custom tags

- Eviter d'avoir à mettre des scriptlets dans la page
- Séparation vue/logique
- Facile à utiliser pour des non programmeurs
- Portables et réutilisables

L'utilisation d'un tag

- Il faut déclarer la librairie contenant le tag
  - <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
  - <%@ taglib prefix="ex" uri="/WEB-INF/test-tag.tld" %>
  - <%@ taglib prefix="ex" uri="/test-tag" %>
    - Dans ce dernier cas il faut décrire le mapping dans le fichier web.xml

```
<taglib>
  <taglib-uri>/test-tag</taglib-uri>
  <taglib-location> /WEB-INF/test-tag.tld</taglib-location>
</taglib>
```

- Rendre son implantation disponible pour l'application Web
  - Il suffit de mettre la librairie implantant le tag et les librairies dépendantes dans le répertoire WEB-INF/lib de l'application

L'utilisation d'un tag

- Ensuite, on peut utiliser les tags définis dans la librairie:
  - <ex:mytag/>
- Les tags peuvent avoir des attributs
  - <ex:mytag myattr="kikou"/>
- Les tags peuvent encadrer du texte

```
<ex:mytag>
Du texte
</ex:mytag>
```

Un petit exemple (tout petit)

- Un tag doit implanter l'interface Tag ou BodyTag
- Pour un nouveau Handler on hérite de TagSupport ou BodyTagSupport qui sont des implantations de ces interfaces
- Il faut aussi écrire le Tag Library Descriptor qui décrit les tags définis dans la librairie et leurs propriétés

Le descripteur du tag

```
<?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>2.0</jspversion>
        <shortname>tagexemple</shortname>
        <info>Un petit exemple de tag</info>
  <tag>
    <name>test</name> <!-- le nom du tag -->
    <tagclass>cours.tag.TestTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <!-- Un attribut optionel  -->
    <attribute>
      <name>nom</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
  </tag>
</taglib>
```

La classe d'implantation

La description de l'attribut optionnel

L'implantation du Tag

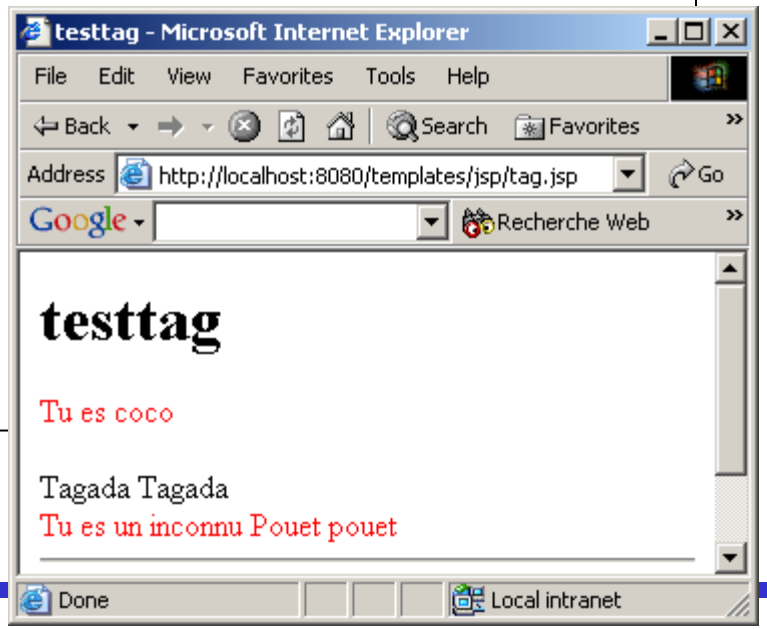
```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class TestTag extends BodyTagSupport {
    private String nom=null; // Attribut optionel du tag
    public String getNom(){return this.nom;}
    public void setNom(String nom){this.nom=nom;}

    // Appelée par le container JSP quand le tag est rencontré
    public int doStartTag() { // appelée au départ du tag
        try {
            JspWriter out=pageContext.getOut();
            out.println("<div><font color='red'>");
            if (this.getNom()==null) {
                out.println("Tu es un inconnu");
            } else {
                out.println("Tu es "+this.getNom());
            }
        } catch (Exception e) {
            throw new Error("Ca chie....");
        }
        return EVAL_BODY_INCLUDE; // on évalue le body
    } …
```

L'implantation du Tag (suite)

```
public int doEndTag() { // appelée après l'évaluation du body
        try {
            JspWriter out = pageContext.getOut();
            out.println("</font></div>");
        } catch (Exception ex){
            throw new Error("Ca chie.........");
        }
        return EVAL_PAGE; // on continue l'évaluation
    }
}
```

L'utilisation du tag

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/tagexemple.tld" prefix="ex" %>
<!-- déclaration de la library -->
<html>
<body>
    <h1>testtag</h1>

<ex:test nom="coco"/> <!-- avec attribut, sans body -->

<br>
Tagada Tagada
<ex:test> <!-- avec body sans attribut >
Pouet pouet
</ex:test>
</body>
</html>
```



Les Tag simple en JSP 2.0

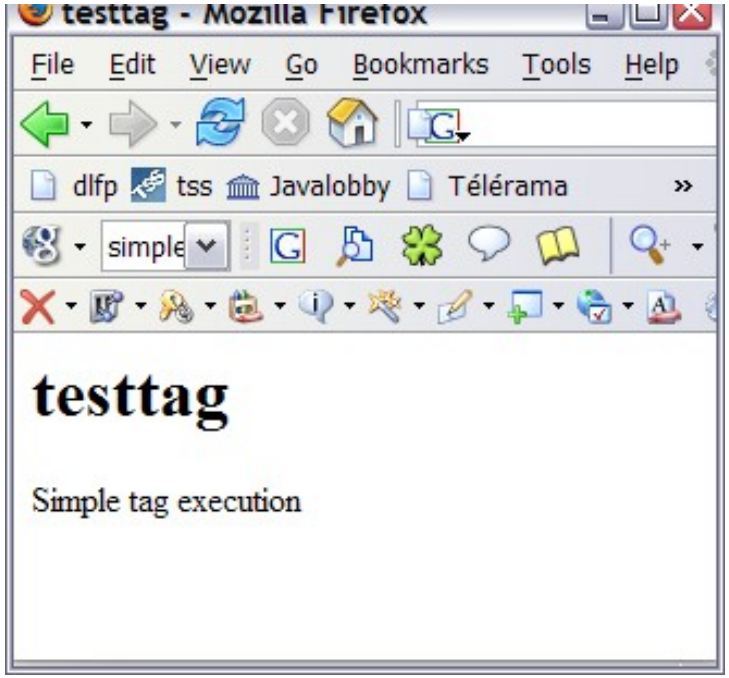
- Invocation plus simple que les tags classiques
- Classe implantant l'interface SimpleTag
- Une seule méthode à implanter
  - doTag()
- Cycle de vie :
  - Création d'une instance pour chaque invocation
  - Appel de setJspContext() et setParent()
  - Évaluation des attributs
  - S'il y a un body appel de setJspBody()
  - Appel de doTag()
  - En cas d'exception le reste de la page peut ne pas être évalué

Un tag tout simple

```
public class YoTag extends SimpleTagSupport {
        public void doTag() throws JspException, IOException {
                try {
                    JspWriter out= this.getJspContext().getOut();
                    out.println("Simple tag execution");
                } catch (Exception e) {
                    throw new Error("Ca marche pas....");
                }
        }
}
```

Utilisation du tag

```
<%@ taglib uri="/tagexemple.tld" prefix="ex" %>
<html>
  <head>
    <title>testtag</title>
  </head>
<body>
<h1>testtag</h1>
<ex:yo/>
</body>
</html>
```



Utilisation d'attributs

- Il faut implanter une setter méthode pour chaque attribut du tag
- Il faut déclarer l'attribut dans le fichier TLD
  - Attribute
    - Name : nom de l'attribut
    - Required : obligatoire ou non (defaut false)
    - Rtexprvalue : la valeur peut être calculée
    - Type : le type java (defaut String)
    - Description
    - Fragment : La valeur est un fragment de JSP

Tag avec un attribut

```
public class AttrTag extends SimpleTagSupport {
        private String nom;
        public void setNom(String nom) {
            this.nom=nom;
        }
        public void doTag() throws JspException, IOException {
                try {
                    JspWriter out= this.getJspContext().getOut();
                    out.println("Simple tag execution avec paramètre "+nom);
                } catch (Exception e) {
                    throw new Error("Ca marche pas....");
                }

        }
}
```

Evaluation du corps (body)

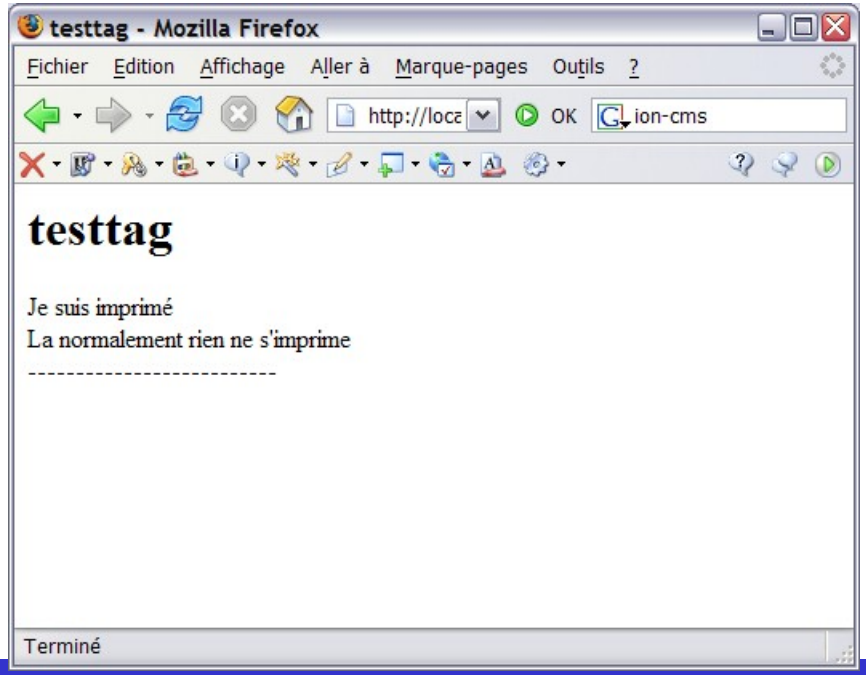
- getJspBody() retourne un JspFragment
- Un JspFragment contient du jsp sans scriptlet
  - Une classe est générée pour ces fragments et le fragment est interprété en utilisant la méthode invoke(PrintWriter)
  - Si on ne précise pas le printwriter, c'est la sortie par défaut qui est utilisée (vers le client)

Evaluation du body

```
public class BodyTag extends SimpleTagSupport {
        private String print="false";
        public void setPrint(String print) {
                        this.print=print;
        }

        public void doTag() throws JspException, IOException
        {
                if (print.equals("true")) {
                        // affiche le contenu du body
                        this.getJspBody().invoke(null);
                }
        }
}
```

Utilisation

```
<ex:body print="true">
Je suis imprimé
</ex:body>
<br>

La normalement rien ne s'imprime<br>
<ex:body>
Je suis pas imprimé
</ex:body>
---------------------------
<br>
```



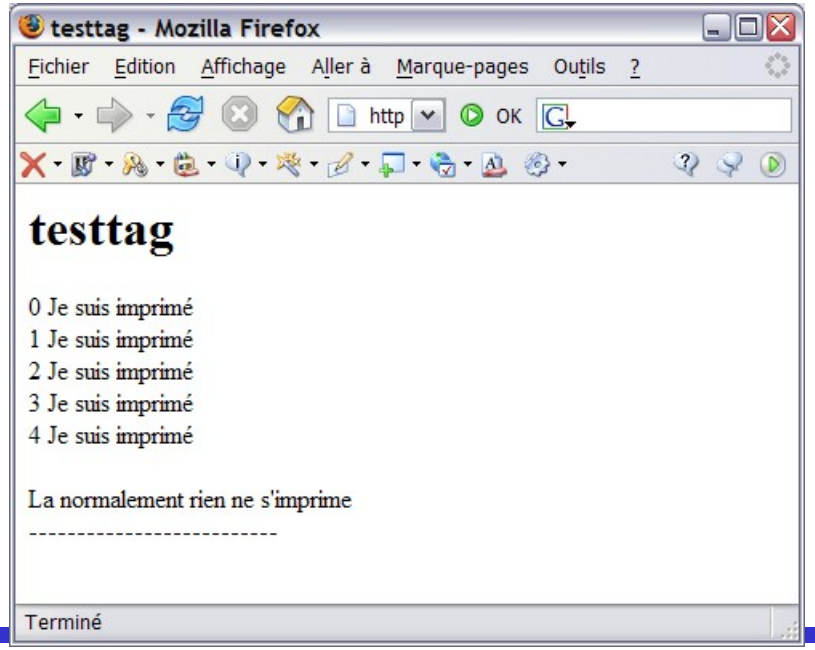
Les tag itératifs

- On peut contrôler l'affichage et l'interprétation du contenu
- Invoke peut être appelé plusieurs fois sur le même fragment

```
public class IterativeTag extends SimpleTagSupport {

        private String print="false";
        private int count=0;

        public void setPrint(String print) {
                        this.print=print;
        }
        public void setCount(int count) {
                this.count=count;
        }
        public void doTag() throws JspException, IOException {
                JspWriter out=this.getJspContext().getOut();
                if (print.equals("true")) {
                        for (int i=0;i<count;i++) {
                                out.println(i);
                                this.getJspBody().invoke(null);
                        }
                }
        }
}
```

Utilisation

```
<ex:iter print="true" count="5">
Je suis imprimé <br>
</ex:iter>
<br>

La normalement rien ne s'imprime<br>
<ex:iter>
Je suis pas imprimé
</ex:iter>
--------------------------
<br>
```



Tags en JSP

- Il est possible d'écrire des tags en JSP
  - Créer un répertoire tags dans WEBINF
  - Les fichiers dans ce répertoire sont nommés xxx.tag
  - Xxx est le nom du tag
- Nouvelles directives
  - Taglib
  - Tag
  - Attribute
  - Variable
  - Include

Tag directive

- Body-content : empty, tagdependent, scriptless
- Display-name
- Dynamic-attribute : true/false
- pageEncoding
- Import
- isELIgnore

Attribute directive

- Name
- Required
- Fragment
- Rtexprvalue
- Type
- description

Un exemple

Rollover.tag

```
<%@ tag isELIgnored="false" %>
<%@ attribute name="link" required="true" %>
<%@ attribute name="image" required="true" %>

<a href="${link}" onmouseover="${image}.src='images/${image}_on.gif';"
onmouseout="${image}.src='images/${image}_off.gif';" >
<img name="${image}" src="images/${image}_off.gif" border="0"/>
</a>
```

```
<body>
<h1>Utilisation du tag</h1>
<t:rollover link="index.jsp" image="nav1"/>

</body>
```