JAVA SERVER PAGE
-JSP-

Master WISD

AU: 2018/2019

Introduction

- JSP est une technologie qui permet d'écrire très facilement des pages web dynamiques en insérant des portions du code Java dans une page HTML.
- Une page JSP permet de définir le design d'une page (en HTML) et d'y intégrer les données (grâce aux potion du code Java). Cette technologie favorise la séparation de la présentation et du contenu.
- Les pages JSP sont associées à une application web. Cette dernière est une collection de pages JSP, HTML, images, etc groupées ensemble.

JSP Exécution - Conditions

- Serveur Web HTTP (Apache, Netscape Enterprise Server …).
- Conteneur de JSP (Tomcat …).
- JDK (*Java Development Kit*) contient un "Java Runtime Environment" *(machine virtuelle), un compilateur* …

Création de pages JSP

Processus divisé en 3 phases:

- Création:

  Création du code source (code Java et HTML).
- Déploiement:

  Page JSP installée sur le serveur (serveur J2EE ou conteneur indépendant).
- Traduction et compilation:

  Traduction par le conteneur JSP du HTML et Java ➔ fichier source Java.

  Compilation du fichier en une classe Java.

  Exécution de la classe par le serveur (classe d'implémentation de la page JSP).

Cycle de vie de pages JSP

1°/ Chargement et instanciation:

Le serveur localise la classe Java correspondant à la page JSP.

Le serveur charge la page JSP dans la JVM

La JVM crée une ou plusieurs instances:

Immédiatement après le chargement

ou

Lorsque la première requête est reçue

2°/ Initialisation:

Page JSP initialisée.

Cycle de vie de pages JSP

3°/ Traitement des requêtes:

La page répond à une requête

Une fois le traitement terminé, la réponse est retournée au client

Cette réponse est constituée uniquement : du code HTML, des données.

Aucun code Java n'est retourné au client.

4°/ Fin du cycle de vie:

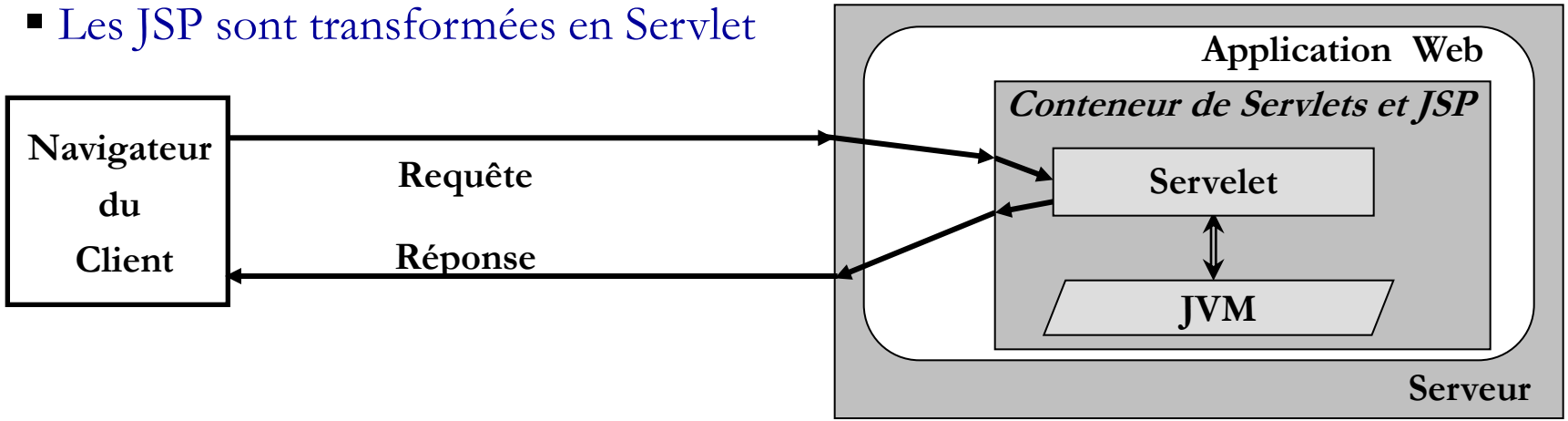
Le serveur cesse d'envoyer des requêtes à la page JSP.

Une fois l'exécution de toutes les requêtes terminée, destruction de toutes les instances représentant la page.

Il est possible d'ajouter à la page une méthode particulière qui sera appelée automatiquement à ce moment.

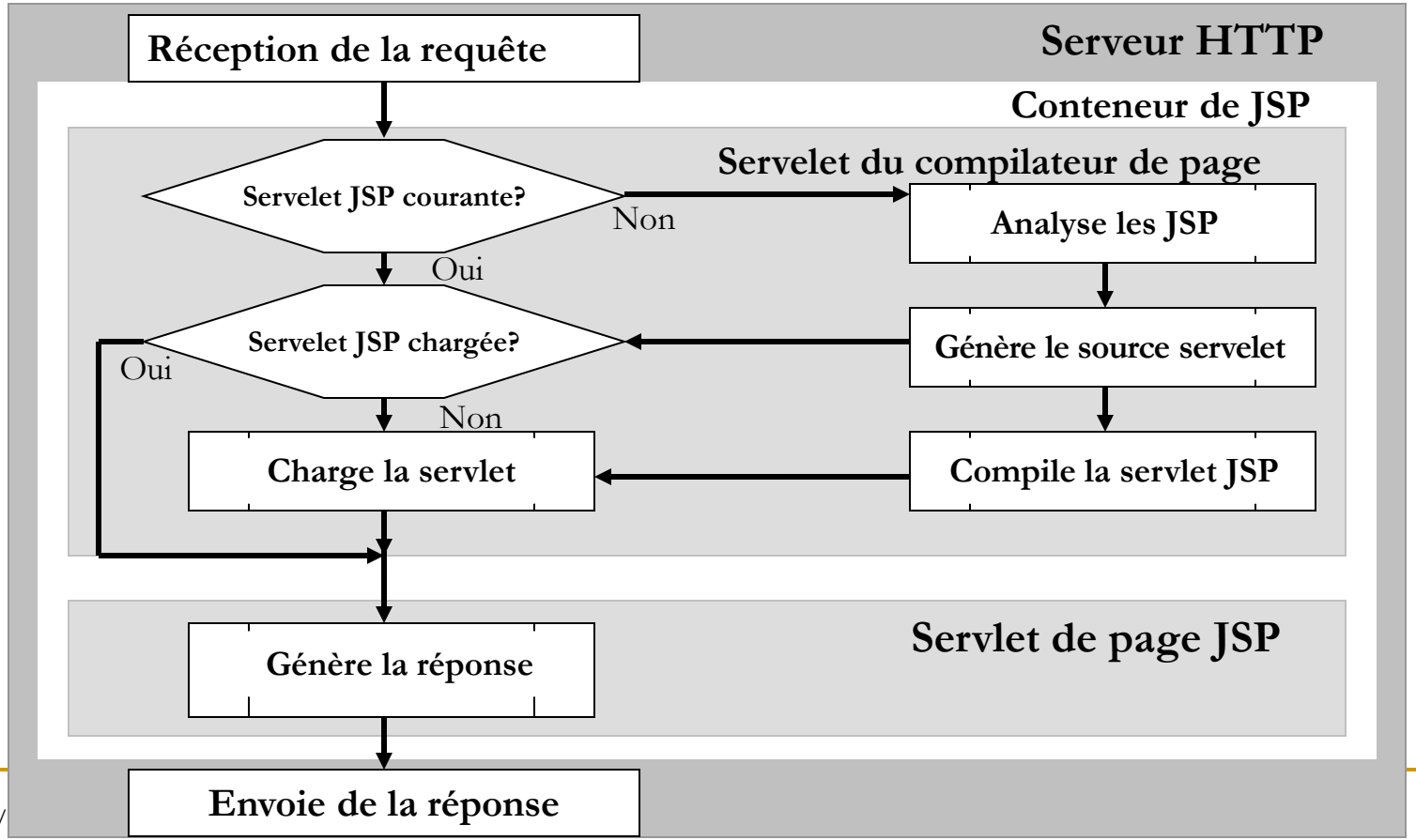
JSP Exécution - Servlets

- Programme exécuté sur le serveur Web
- Peut recevoir et émettre des requêtes HTTP
- Les JSP sont transformées en Servlet



- API Java "javax.servlet" et "javax.servlet.http"

JSP Exécution - Fonctionnement



Intégration du code Java dans HTML

On distingue 5 types de balises jsp:

1- Balise directive <%@ et %>

2- Balise expression <%= et %>

3- Balise déclaration <%! et %>

4- Balise scriplet <% et %>

5- Balise action <%jsp: et />

Balise directive

<%@ et %>: permet le contrôle de la structure de la page, l'inclusion du contenu d'un fichier (include), l'importation d'une classe java (import), …

→ Transmettre au conteneur de JSP des informations de traitement spécifiques à une page.

Exemple:

```
<%@ page language= "java" %>
<%@ page import  "java.util.*", "java.sql.*" %>
<%@ include file= "/entete.html" %>
<%@ page session= "true|false" %>
```

Balise expression

<%= et %>: cette balise permet d'intégrer des valeurs issues d'objet java dans le code html (ces valeurs sont converties en chaînes de caractères).

Exemple:

```
<%= new java.util.Date().toString() %>
<%= n1+n2 %>
<%= request.getRemoteHost() %>
```

Balise expression

- Les expressions sont utilisées pour renvoyer au client les valeurs d'expressions Java.
- Toutes les expressions Java valides peuvent être employées
- Une expression peut contenir :
  - une variable ou un appel de méthode
  - une expression littérale (avec même des mots clés Java)
  - une combinaison des trois
- Une expression ne doit pas comporter de « ; ».

Balise déclaration

<%! et %>: permet de déclarer des variables ou des méthodes de classes à utiliser dans toute la page.

Exemple:

```
<%! String noms[]={"Mohamed", "Fatima", "Omar"};
        private int i, n=10;
        private boolean estVide(String str) {
                return (str == null) || str.equals("");
          }
%>
```

Balise déclaration

- Les variables déclarées deviennent des variables d'instance dans la classe d'implémentation.
- Pas de protection contre les accès simultanés.
- Le serveur peut utiliser la même instance pour servir plusieurs requêtes.
- Les déclarations doivent se terminer par « ; ».
- Les déclarations sont présentes dans le code Java créé lors de la traduction de la page, mais elles n'existent pas dans le résultat renvoyé au client.
- Les variables déclarées dans une balise déclaration sont des variables d'instance, accessibles dans toutes les scriptlets de la page.

Balise scriplet

<% et %> : permet d'incorporer et d'intégrer une portion du code java dans la page. Il devient possible d'insérer toutes les instructions de type test, boucle et affichage décrites par le langage java.

Exemple:

```
<%  String host= request.getRemoteHost();
        out.println("La machine utilisée est: " + host);
%>
```

Balise scriplet

- Contiennent des instructions Java
- Apparaissent dans le code Java produit lors de la traduction des pages JSP.
- N'apparaissent pas dans les réponses envoyées au client.
- Peuvent contenir n'importe quel code Java valide.
- Doivent se terminer par « ; ».
- Les scriptlets ne peuvent pas être employées pour définir des méthodes.
- Les variables déclarées dans une scriptlet sont locales et ne sont visibles qu'à l'intérieur du bloc dans lequel elles sont définies.

Balise action

<jsp: et />: permet la manipulation dynamique des composants java ou autres.

Exemples:

<jsp:forward page="relativeUrl" /> : chaînage JSP/Servlet

<jsp: useBean id="nomBean" class="LaClasseBean" scope="session" /> : les javaBeans

Directives JSP : include

- Cette inclusion se fait au moment de la conversion.

```
<%@ include file="unFichier.jsp" %>
```

- Tout le contenu du fichier externe est inclus comme s'il était saisi directement dans la page JSP.

Exemple:

```
<%@ include file = "entetePage.html" %>
<%@ include file = "corpsPage.jsp" %>
<%@ include file = "piedPage.html" %>
```

Directives JSP : page

Attributs de la directive **page** :

- **import** : importe un paquetage Java. Cette directive résulte en une instruction import dans la Servlet. <%@ page import="java.util.*, java.text.*" %>
- **langage** : définit le langage de script utilisé dans la page
- **contentType** : définit le type de contenu de la page générée

  <%@ page contentType="text/plain" %>
- **errorPage** : définit la page d'erreurs à afficher si une exception se produit pendant l'exécution. <%@ page errorPage="erreurs.jsp" %>
- **isErrorPage** : vaut true si la page est une erreur et false pour une page normale. <%@ page isErrorPage=false %>
- session, info, …

Directives JSP: taglib

Cette directive permet de déclarer l'utilisation d'une bibliothèque de balises personnalisées (TLD: descripteur de bibliothèque de balises).

TLD : document XML contenant des informations concernant les gestionnaires de balises disponibles dans une bibliothèque.

Attributs de taglib:

- prefix : espace de noms pour les tags de la bibliothèque dans la JSP
- uri : identifie de façon unique la bibliothèque.

Exemples :

```
<%@taglib prefix="c" uri="WEB-INF/tld/core.tld" %>
<%@ taglib uri="/WEB-INF/tld/testtaglib.tld" prefix="maTagLib" %>
```

Commentaires

- Les commentaires JSP ont leur propre syntaxe :

  <%-- Un commentaire JSP --%> →commentaire non transmis au navigateur
- Les commentaires en langage HTML s'écrivent comme suit :

  <!-- Commentaire HTML --> →commentaire transmis au navigateur client
- Les commentaires Java demeurent valides au sein des scriptlets.

  // Un commentaire java

  /* Un autre commentaire
  java */

Gestion des erreurs

- Contrôler la gestion des erreurs pendant l'exécution de l'application WEB.
- Les erreurs sont déclenchées par l'intermédiaire des exceptions et transmises à une page d'erreur.
- La définition de la page d'erreur se fait par la directive *page* et l'attribut *errorPage*.

Exemple: <%@ page errorPage="pageErreur.jsp" %>

... code JSP lançant l'erreur (exception)

- Une page JSP est définie comme une page erreur par l'attribut *isErrorPage* de la directive *page*.

Exemple: <%@ page isErrorPage=true %>

... code JSP traitant l'exception

Gestion des erreurs

- L'objet implicite exception (Exception) permet de manipuler l'exception lancée. Cet objet est accessible dans les pages d'erreurs.
- Référence à l'objet java.lang.Throwable qui a causé l'utilisation de la page.
- L'objet exception n'est utilisable que dans les balises Scriplet et Expression.
- La page erreur doit être obligatoirement une page JSP.
  - exception.getMessage(): retourne le message d'erreur décrivant l'exception (la cause de l'erreur).
  - exception.printStackTrace(): retourne la liste des appels des méthodes ayant conduit à l'exception.

Objets Implicites

- Les objets implicites sont accessibles sans déclaration ou initialisation. Ils sont accessibles dans les scriptlets et dans les expressions.
- Ces objets sont les objets présents dans la méthode service(…) de la Servlet.
  - Requête et réponse :

    request response out
  - Accès aux données :

    session application
  - Contexte et paramètres :

    pageContext config

Objets Implicites

Les objets implicites sont les objets présents dans la méthode service(…) de la Servlet. Ils sont identifiés par :

- **javax.servlet.http.HttpServletRequest request :** accès à la requête du client Web (infos sur la requête, le navigateur, le protocole et le client).
- **javax.servlet.http.HttpServletResponse response :** construire la réponse du serveur Web à son client (infos sur la réponse, le codage, les erreurs, le tampon de sortie).
- **javax.servlet.jsp.JspWriter out :** le flux de sortie, permet d'envoyer du code HTML au client (accès à l'impression formatée).
- **javax.servlet.http.HttpSession session :** informations sur la session courante.

Objets Implicites

- **javax.servlet.ServletContext application :** contient des méthodes log() permettant d'écrire des messages dans le journal du contenu (infos sur l'application).
- **javax.servlet.jsp.PageContext pageContext :** utilisé pour partager directement des variables entre des pages JSP et supportant les beans et les balises (infos sur la page : requête, réponse, session, sortie).
- **javax.servlet.ServletConfig config** : tout sur la configuration de la servlet : paramètres, nom, ...
- **exception :** disponible seulement dans les pages d'erreurs qui donnent détails sur les erreurs.

Objet request

- Représente une requête que la page doit traiter. La portée est limitée à la requête (càd jusqu'à ce que la réponse soit retournée au client).
- Cet objet permet de lire:
  - les entêtes de la requête,
  - les paramètres de la requête,
  - de nombreuses autres informations.
- Paramètres:
  - Les paramètres encodés (envoyés) dans l'URL.
  - Les paramètres de formulaire.

Objet request

Quelques méthodes de l'objet:

- String request.getParameter(String nom): retourne la valeur du paramètre dont le nom correspond à la chaîne utilisée en argument.
- String[] request.getParameterValues(String nom): retourne un tableau de chaînes de caractères.
- Enumeration request.getParameterNames(): retourne une énumération des noms des paramètres présents dans la requête.
- Map getParameterMap(): retourne une Map de couples nom/valeur.
- String request.getPathInfo(): retourne les informations ajoutées après le nom de la ressource web.

Objet request

- request.getRemoteAddr(); retourne l'adresse IP du client.
- request.getProtocol(); retourne le protocole accepté par le client
- request.getServerName(); retourne le nom du serveur.
- request.getServerPort(); retourne le port du serveur.
- request.isSecure(): retourne un booléen. la connexion est-elle sécurisée ?
- request.getCookies(): récupère les objets Cookie sous forme de tableau de variable.
- request.getHeader(): récupère l'entête de la requête utilisée.
- request.getMethod(): récupère le nom de la méthode utilisée.

Objet out

L'objet out référence au stream de sortie utilisé par la réponse.

- Instance de la classe javax.jsp.JspWriter
- Portée limitée à la page.
- Renvoie au client exactement le même code généralement écrit selon la syntaxe html:
- Peut-être employé dans une scriptlet :
  - Permet d'écrire des données dans la réponse envoyée au client.

Objet out

L'objet out référence au stream de sortie utilisé par la réponse.

- Instance de la classe javax.jsp.JspWriter
- Portée limitée à la page.
- Renvoie au client exactement le même code généralement écrit selon la syntaxe html.
- Peut-être employé dans une scriptlet :
  - Permet d'écrire des données dans la réponse envoyée au client.

Exemple:

```
out.print("Bonjour <B>coucou</B><BR>");
```

Objet response

Cet objet contient la réponse renvoyée au client de l'application. Sa portée est limitée à la page.
Il permet de:

- Modifier les entêtes:

  public void addHeader(String nom, String valeur)

- Envoyer un cookies au client:

  public void addCookie(Cookie cookie)

- Rediriger la réponse:

  public void sendRedirect(String destination)

Objet session

- Identifie l'ensemble d'une conversation entre un client et le serveur.
- Les composants JSP d'une application Web participent automatiquement à une session, sans nécessiter aucune intervention.
- Si une page JSP utilise la directive page pour donner à l'attribut session la valeur false, alors cette page n'aura plus accès à l'objet session, et ne pourra donc pas participer à la session.
- L'objet session permet de stocker des informations à propos du client ou de l'état de la conversation avec celui-ci, comme elles pourraient l'être dans une liste Hashtable ou dans une HashMap.

Objet session

- Une session ne peut stocker que des objet, et non pas des primitives Java (utilisation des classes enveloppes)
- Portée limitée à la session, ainsi que tous les objets qu'il contient.
- Méthodes utilisées:
  - Object setAttribute(String nom, Object valeur)
  - Object getAttribute(String nom)
  - Enumeration getAttributeNames()
  - void removeAttribute(String nom).

Utilisation de JSTL

- JSP Standard Tag Library met à disposition du développeur des balises pour accomplir la plupart des taches qui doivent être réalisées avec les JSP.
- La JSTL est une implémentation de Sun qui décrit plusieurs actions basiques pour les applications web JEE. Elle propose ainsi un ensemble de librairies de tags pour le développement de pages JSP.
- JSTL offre plusieurs balises pour :
  - la lecture et le traitement de documents XML,
  - les structures alternatives et répétitives,
  - le traitement des requêtes SQL,
  - l'internationalisation.

Utilisation de JSTL

- Librairies de la JSTL: l'URL de base http://java.sun.com
  - core : /jsp/jstl/core
  - Format : /jsp/jstl/fmt
  - XML : /jsp/jstl/xml
  - SQL : /jsp/jstl/sql
  - Functions : /jsp/jstl/functions
- Récupérer une version JSTL pour tomcat :
  http://tomcat.apache.org/taglibs/standard/

Exemple de déclaration :

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

JSTL - Core

- Utiliser JSTL Core:

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

- **Core: c**
  - remove, set
  - choose, forEach, forTokens, if
  - import, redirect, url
  - catch, out

  http://java.sun.com/products/jsp/syntax/2.0/syntaxref207.html#1010522

JSTL - Core

- Core - set/remove: définir des variables et des propriétés

Attributs : *value* : expression / valeur
var : nom de l'attribut qui va contenir la valeur
scope : indique la portée qui peut être page, request, session ou application
target : l'objet ayant la propriété à modifier
property : nom de la propriété modifiée

```
<c:set var="userId" scope="session" value="${param.textId}" />
<c:set var="langProg" scope="page" value="Java" />
```

Supprimer une variable de scope:

```
<c:remove var="userId" scope="session" />
```

JSTL - Core

- Core - catch: attraper des exceptions

Cette balise permet d'ignorer les exceptions lancées entre les tags « catch » et enregistrer l'exception dans une variable

Exemple:

```
<c:catch var="uneException">
        <c:set target="${param.personne}" property="age" value="18"/>
 </c:catch>
```

JSTL - Core

- Core - URL: gestion des URLs

Exemple 1:

Le script JSP : <a href="<% response.encodeURL("page.jsp") %>">lien

est équivalent au JSTL Core: <a href="<c:url value='page.jsp' />">lien

Exemple 2:

```
<% Produit p = (Produit) session.getAttribute("unProduit");
      String url = "page.jsp?prRef=" + p.getRef();  %>
<a href="<% response.encodeURL(url) %>">afficher</a>
```

Equivalent JSTL :

```
<a href="<c:url value='page.jsp?prRef=${unProduit.ref}'/>">afficher</a>
```

JSTL - Core

- Core - redirect: redirection

Le reste de la page ne sera pas interprété

Exemple:

```
response.sendRedirect("http://fsdmfes.ac.ma");
```

Equivalent JSTL :

```
<c:redirect url="http://fsdmfes.ac.ma"/>
```

JSTL - Core

- Core - forEach: effectuer simplement des itérations.

Exemple: parcourir une liste des produits :

```
<c:forEach var="prd" items="${jspCommande.produits}">
    <tr>
        <td> ${prd.ref} </td>
        <td> ${prd.descr} </td>
        <td> ${prd.prix} </td>
         <td> ${prd.qte} </td>
    </tr>
</c:forEach>
```

JSTL - Core

Attributs de core – forEach:

Permettent une gestion plus fine de la boucle :

- begin : indice de début
- end : indice de fin
- step : incrément
- varStatus : informations sur la boucle (first, last, index, count)

JSTL - Core

Afficher seulement les 6 premiers produits

```
<c:forEach var="elem" items="${jspCommande.produits}"  begin="0" end="5">
        ${elem.ref}<br/>
</c:forEach>
```

Afficher les nombres de 1 à 10

```
<c:forEach var="valeur" begin="1" end="10">
        ${valeur}<br/>
</c:forEach>
```

Afficher les paramètres, avec leurs valeurs, de la requête

```
<c:forEach var="elem" items="${param}">
        Le paramètre ${elem.key} vaut ${elem.value}<br/>
</c:forEach>
```

JSTL - Core

- Core – if : faire un test conditionnel

Exemple:

```
<c:if test="${produit.qte > 0}" >
    <p>Produit existe : quantité
        <c:out value="${produit.qte}" />
    </p>
</c:if>
```

L'attribut ***test*** indique le test (condition) à réaliser

JSTL - Core

- Core – choose: traitement conditionnel exclusif (sorte de switch/if)
  - when : permet de décrire chaque condition
  - otherwise : correspond au default de switch

Exemple:

```
<c:choose>
    <c:when test="${param['n']==1}">Une fois</c:when>
    <c:when test="${param['n']==2}">Deux fois</c:when>
    <c:otherwise>Aucune</c:otherwise>
</c:choose>
```

JSTL - Core

Exemple:

```
<c:choose>
    <c:when test="${empty param.textNom}">
        Saisir votre nom SVP..
    </c:when>
    <c:otherwise>
        Bonjour
        <b> <c:out value="${param.textNom}" /> </b>
    </c:otherwise>
</c:choose>
```

JSTL - Core

- Core – out : afficher une expression

Exemple:

Afficher le paramètre nom de la requête http :

```
<c:out value="${param['nom']}" default="Inconnu"/>
```

Afficher le contenu d'une variable

```
<=!  int annee=2018; %>
<c:out value="${annee}" />
```

JSTL - Core

- Core – import: récupérer des ressources

Ressemble aux balises <jsp: include/> et <%@ include %>

Exemple:

```
<c:import url="entetePage.html" %>
<c:import url="corpsPage.jsp" %>
<c:importurl="ftp://server.com/path/fiche.txt" var="file"scope="page"/>
```

Utilisation des Expressions Languages

Une expression en EL débute par **${** et se termine par **}**.

Syntaxe :

${une expression}

Exemple:

${1+2} affiche 3

${(2*3)+4} affiche 10

${"une expression …"} affiche le message indiqué

${annee} affiche le contenu de la variable annee

Utilisation des Expressions Languages

- Il est possible d'indiquer au conteneur JSP 2.0 de ne pas interpréter les EL d'une page. Il suffit pour cela d'utiliser l'attribut *isELIgnored* de la directive page :

```
<%@ page isELIgnored="true" %>
```

- Il est également possible de ne pas interpréter une EL en particulier en la protégeant avec un anti-slash (\):

```
\${ ceci ne sera pas interprété comme une EL }
```

Utilisation des Expressions Languages

Gestion des Exceptions : les EL gèrent un certains nombres d'exceptions afin de ne pas avoir à les traiter dans les JSP.

- Si la donnée à afficher est valide alors on l'affiche, sinon on n'affiche rien.
- Cela permet de se passer d'effectuer plusieurs vérifications au sein des JSP avant d'afficher les données.
- Les propriétés d'un élément qui ne sont pas renseignées sont considérées null. Ainsi l'expression prend la valeur null mais aucune exception n'est levée.
- l'utilisation d'un indice incorrect d'un tableau ou d'une List ne provoque aucune d'exception mais renvoie la valeur null.
- Lors de l'affichage d'une page JSP, toutes les valeurs null sont remplacées par des chaînes vides → pour ne pas afficher le mot « null ».

Utilisation des Expressions Languages

Les objets implicites :

- pageContext : accès à l'objet PageContext de la page JSP.
- pageScope : Map permettant d'accéder aux différents attributs du scope *page*.
- requestScope : accéder aux différents attributs du scope *request*.
- sessionScope : accéder aux différents attributs du scope *session*.
- applicationScope : accéder aux différents attributs du scope *application*.
- header : accéder aux valeurs du Header HTTP sous forme de String.
- param : accéder aux paramètres de la requête HTTP sous forme de String.
- paramValues : accéder aux paramètres de la requête HTTP (tableau de String).
- headerValues : accéder aux valeurs du Header HTTP (tableau de String).
- cookie : Map permettant d'accéder aux différents Cookies.
- initParam : Map permettant d'accéder aux init-params du web.xml.

Utilisation des Expressions Languages

Expression EL

${nomVariable}

Correspond à : <%= pageContext.findAttribute("nomVariable"); %>

Effectue une recherche successive dans les différents scopes de l'application. Ordre : *page*, *request*, *session* puis *application*.

Risque de conflits, utiliser les objets implicites lorsque vous ne recherchez pas dans le scope page : requestScope[" nomVariable"]

Utilisation des Expressions Languages

Méthodes d'accès aux propriétés des beans standards (des objets) :

- Opérateur . (point): ${personne.nom}
- Opérateur [] (crochet): ${personne["nom"]}

Equivalent scriptlet :

```
<%@ page import="package.MonBean" %>
<% Personne personne = (Personne) pageContext.findAttribute ("personne");
if (personne!= null) out.print ( personne.getNnom() ); %>
```

- Avantages:
  - Simplicité, lisibilité,
  - Gestion des exceptions,
  - Pas de cast donc pas besoin d'import.

Utilisation des Expressions Languages

Propriétés des List et tableaux :

Utilisation de l'opérateur crochet []:

${list[0]} ou ${list["0"]} → Conversion automatique en entier.

Fonctionnement :

Utilisation de la méthode get(int) de l'interface List

Ou

Accès au éléments du tableau par son index.

Utilisation des Expressions Languages

- Propriétés des Map

Utilisation de l'opérateur crochet []

${map ["cle"]}

Ou

${map['cle']}

Fonctionnement :

Utilisation de la méthode get(Object) de l'interface Map.

Utilisation des Expressions Languages

Les opérateurs

- Opérateurs arithmétiques, logique et de comparaison
- Empty
- ()
- ?: condition ? valeurTrue : valeurFalse