

06) La présentation avec les JSP

420-109-GG

LOTFI DERBALI

Introduction

Les servlets ne sont pas adaptées pour gérer efficacement l'affichage comme vous avez pu le constater.

La plateforme Java EE propose une solution nommée **JSP** (Java Server Page).

- Cette technologie permet de créer facilement un contenu dynamique au format HTML ou XML.
- Elle correspond au **V** (vue) de l'architecture **MVC**.

La **servlet** s'occupe de faire **le traitement métier** et lorsque celui-ci est terminé, elle délègue l'affichage à une **JSP**.

Qu'est ce qu'une JSP

Les **JSP** sont, tout simplement, des pages HTML d'extension **.jsp** dans lesquelles il est possible **d'ajouter différents types de contenus** (non HTML) qui seront traités par le conteneur de servlets pour générer un rendu spécifique lié au contexte d'exécution de la requête.

Ces types de contenus peuvent être :

- des scripts sous la forme de **code Java**,
- des scripts sous la forme d'**EL** (Expression Language),
- des actions standards,
- des tags standards (**JSTL** - JSP Standard Tag Library),
- ou des **tags personnalisés**.

Les **JSP** sont décrites dans la **JSR 245**.

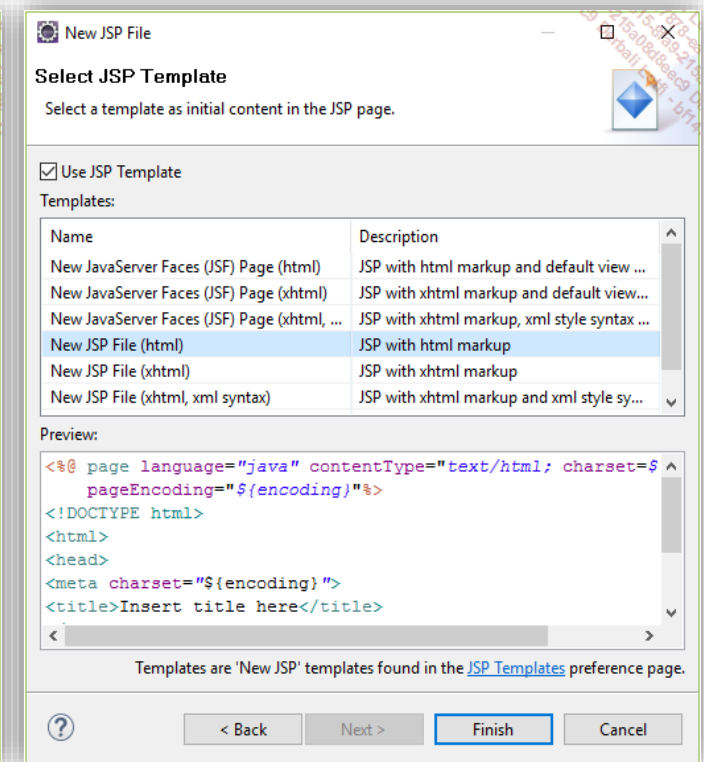
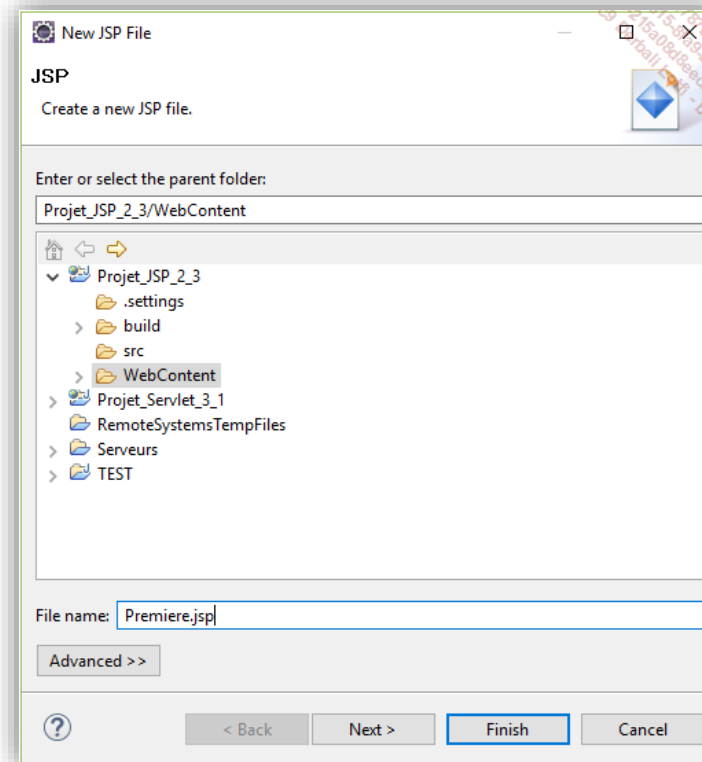
L'**EL** est décrit dans la **JSR 341**.

La **JSTL** est décrite dans la **JSR 52**.

La création de la première JSP

Pour créer une JSP, veuillez suivre les étapes suivantes :

- Faites un clic droit sur le répertoire **WebContent** de votre projet puis cliquez sur le menu **New - JSP File**.
- Donnez un nom à votre JSP (par exemple premiere.jsp) dans l'exemple et cliquez sur **Next** afin de pouvoir sélectionner le **template** à utiliser pour la création
- Sélectionnez le template **New JSP File (html)**.
- Cliquez sur **Finish**.



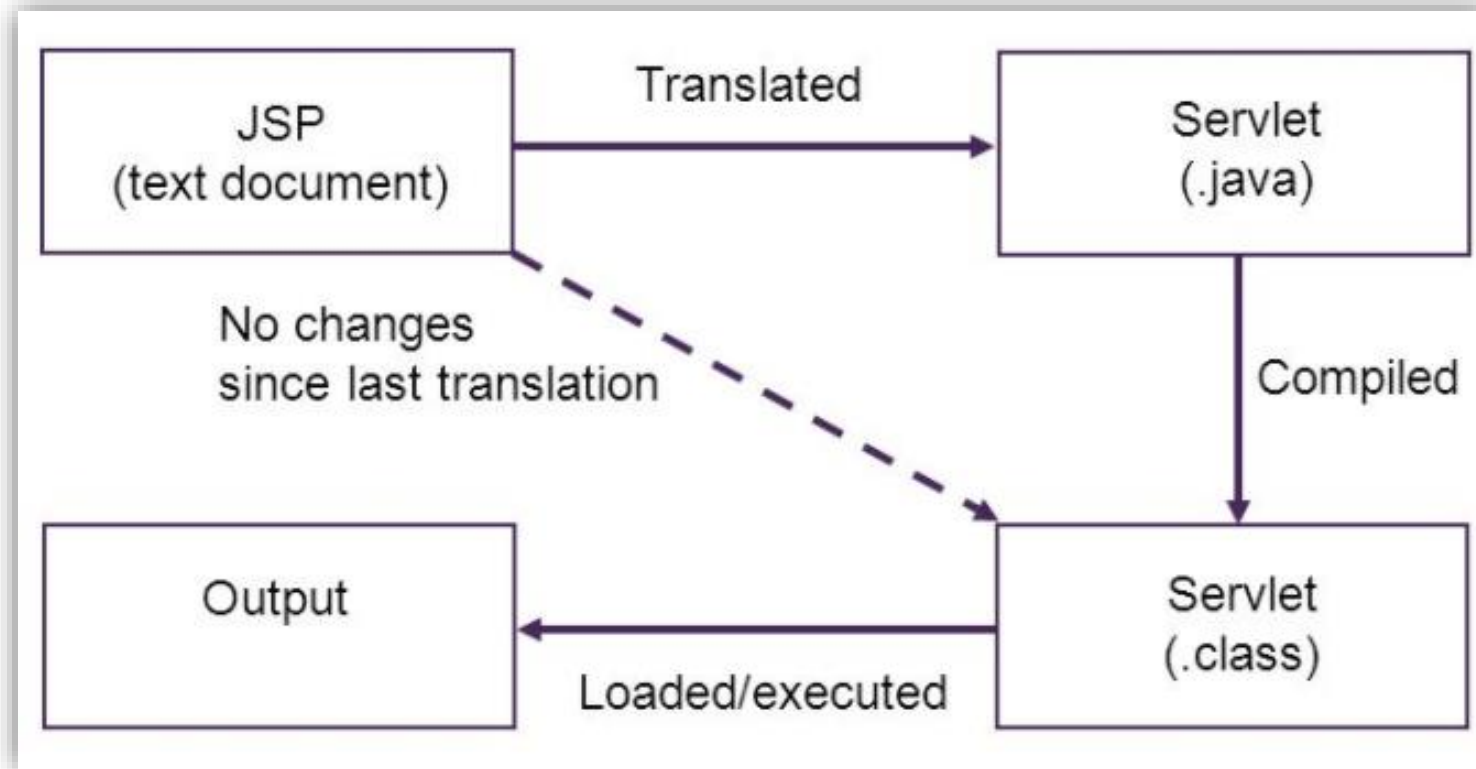
Le principe d'exécution

Les **JSP** existent pour simplifier la **création d'un contenu dynamique** car les servlets ne sont pas adaptées pour cette tâche.

Lorsqu'une requête HTTP implique l'exécution d'une JSP:

- Si la JSP n'a encore jamais servi, celle-ci est transformée en classe Java (***Translation phase***) avant d'être compilée.
 - La transformation est réalisée par le moteur **Jasper 2** et la compilation est réalisée par défaut par le compilateur Java **Eclipse JDT**
 - Cette classe doit implémenter l'interface `javax.servlet.Servlet`.
 - **Une JSP n'est donc ni plus ni moins qu'une servlet.**
- Autres détails:
 - La classe doit aussi implémenter l'interface `javax.servlet.jsp.HttpJspPage`.
 - Cette interface force l'écriture de la méthode `_jspService(...)`. Cette méthode est l'équivalent des méthodes `doXXX(...)` des servlets. Elle prend en paramètre un objet de type `HttpServletRequest` et un objet de type `HttpServletResponse`. Elle a pour rôle la création d'une réponse à l'utilisateur.

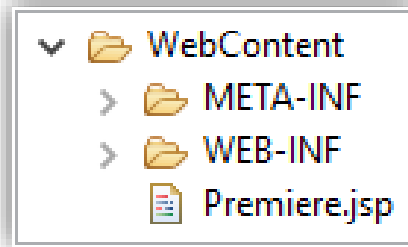
Le principe d'exécution



Le paramétrage d'une JSP

Une **JSP** se trouve naturellement dans le répertoire **WebContent** du projet.

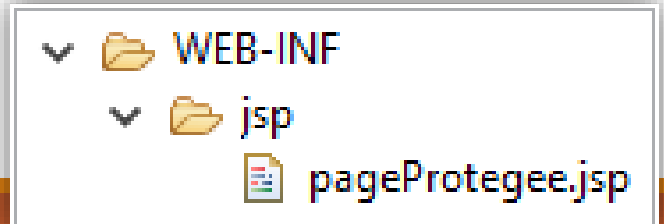
- Elle est ainsi accessible au travers d'une URL depuis le navigateur d'un client.



Cependant, si on souhaite être strict dans l'architecture MVC, il ne faut pas qu'une JSP soit accessible directement au travers d'une URL.

- Il faut passer par une servlet qui délègue, le moment venu, la génération de la réponse à une JSP.
- Pour cela, il est possible de déplacer la JSP dans le répertoire WEB-INF, répertoire inaccessible au travers d'une URL.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/jsp/pageProtegee.jsp");
    rd.forward(request, response);
}
```



Les directives

Les **directives** sont des messages dirigés vers le conteneur Web afin de lui donner des indications sur l'étape de transformation.

- Les directives ne produisent pas de contenu.
- La syntaxe d'une directive est la suivante : `<%@ nom_de_la_directive {attr="value"}* %>`
- Une directive peut posséder plusieurs attributs.

Il existe trois directives :

- La directive **page**,
- La directive **taglib**,
- La directive **include**.

La directive **page**

La **directive page** permet de définir des caractéristiques spécifiques à la page.

- Ces caractéristiques sont à destination du conteneur Web pour la phase de transformation.

Il peut y avoir **plusieurs directives page dans une JSP**.

- Ces directives peuvent être positionnées n'importe où dans la page.
 - Il existe une exception pour les attributs `pageEncoding` et `contentType` qui doivent se trouver dans une directive page en haut de la JSP.
- Chaque attribut ne doit être présent, au maximum, qu'une seule fois.
 - Il existe une exception pour l'attribut `import` pour permettre l'import de plusieurs packages ou classes dans la JSP.

La directive **page**

Voici les attributs disponibles sur la **directive page** :

- **language** : cet attribut permet de définir le langage utilisé pour les scripts écrits dans la JSP devant s'exécuter côté serveur. Jusqu'à maintenant, la seule valeur possible est **java**.
- **extends** : cet attribut permet de définir le nom de la classe (complètement qualifié) dont dérive la JSP. Cet attribut est à utiliser avec précaution. En l'absence de cet attribut, le conteneur utilise la classe par défaut. Dans l'environnement Tomcat, c'est la classe **org.apache.jasper.runtime.HttpJspBase**.
- **import** : cet attribut permet de définir les imports des packages et des classes nécessaires au fonctionnement des scripts écrits en Java dans la JSP.
- **session** : cet attribut attend un booléen pour indiquer si l'utilisation de la session HTTP est possible dans la JSP. Si la valeur est **true**, alors une variable nommée session de type javax.servlet.http.HttpSession est disponible. La valeur par défaut est **true**.
- **buffer** : cet attribut permet de définir la taille du tampon de l'objet JspWriter permettant l'écriture de la réponse HTTP. Si la valeur est none, alors les informations sont directement écrites vers l'objet de type PrintWriter de l'objet de type ServletResponse. La valeur par défaut est **8kb**.

La directive **page**

Les attributs disponibles sur la **directive page** (suite):

- **autoFlush** : cet attribut attend un booléen pour spécifier si le tampon du JspWriter doit être vidé automatiquement lorsque celui-ci est plein. Si la valeur est false et que le tampon est plein, une exception est levée. **La valeur par défaut** est **true**.
- **isThreadSafe** : cet attribut attend un booléen pour indiquer si plusieurs requêtes HTTP peuvent être traitées simultanément (dans autant de threads) par une JSP.
- **info** : cet attribut permet de fournir des informations indéterminées sur la JSP. Ces informations sont accessibles au travers de la méthode `getServletInfo()` au niveau de la JSP.
- **errorPage** : cet attribut permet de définir une URL correspondant à la page d'erreur à appeler en cas de levée d'une exception (non traitée par un bloc try/catch) lors de l'exécution de la JSP.
- **isErrorPage** : cet attribut permet de définir si une page est une page d'erreur. Si **la valeur est true**, alors l'erreur est accessible au travers d'une variable nommée **exception** au niveau de la JSP. **La valeur par défaut** est **false**.
- **contentType** : cet attribut permet de définir le type de média (MIME) et éventuellement l'encodage de la réponse HTTP.
- **pageEncoding** : cet attribut permet de définir l'encodage de la réponse http.
- **isELIgnored** : cet attribut permet de définir si l'EL est ignorée (true) ou pas (false) dans la JSP.

La directive **page**

Exemple:

```
Test.java First.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ page import="java.text.SimpleDateFormat"%>
4 <%@ page import="java.util.Date"%>
5 <!DOCTYPE html>
6 <html>
7 <head>
8   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9   <title>Sample</title>
10 </head>
11 <body>
12   <%
13       SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
14       Date date = new Date();
15   %>
16   <h1>
17       Today is
18       <%= dateFormat.format(date) %>
19       !
20   </h1>
21 </body>
22 </html>
```

La directive **taglib**

La **directive taglib** permet de référencer des bibliothèques de balises externes.

- Il existe des bibliothèques standards : les **JSTL**.
- Il existe aussi des bibliothèques personnalisées.
- Ces bibliothèques mettent à disposition des balises complémentaires pour mettre au point les JSP.

Voici les attributs disponibles sur la directive **taglib** :

- **uri** : cet attribut permet d'identifier de manière unique une bibliothèque de balises. Cet attribut est disponible dans le fichier .tld (Tag Library Descriptor) décrivant les balises de la bibliothèque packagées dans un jar.
- **tagdir** : cet attribut permet de définir le répertoire dans lequel sont présentes les balises. Ce répertoire est forcément un sous-répertoire du répertoire /WEB_INF/tags.
- **prefix** : cet attribut permet de définir le préfixe pour utiliser les balises de la bibliothèque.

La directive **include**

La **directive include** permet d'inclure des ressources externes à la JSP pendant la phase de transformation.

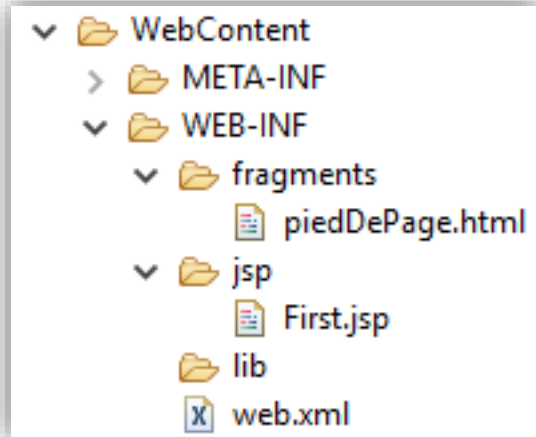
- Ces ressources sont physiquement ajoutées dans la classe Java issue de la transformation.
- Ces **ressources** peuvent être du **texte statique** (HTML, XML,...) ou une **JSP**.
- Il **n'est pas possible d'inclure une servlet**.
- L'inclusion peut permettre d'ajouter un en-tête de page, un pied de page, un menu par exemple.

La directive **include** possède un **unique attribut** nommé **file**.

- Cet attribut permet de définir le chemin d'accès relatif à la page JSP dans le système de fichiers.

La directive **include**

Exemple:



```
Test.java First.jsp *piedDePage.html
1
2 <div style="background-color: navy; width: 100%; color: white">
3     JavaEE - &copy;Lotfi Derbali
4 </div>
```

```
Test.java First.jsp *piedDePage.html
1 <%@ page language="java" contentType="text/html; charset=
2     pageEncoding="UTF-8"%>
3 <%@ page import="java.text.SimpleDateFormat"%>
4 <%@ page import="java.util.Date"%>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta http-equiv="Content-Type" content="text/html; cha
9 <title>Sample</title>
10 </head>
11 <body>
12     <%
13         SimpleDateFormat dateFormat = new SimpleDat
14         Date date = new Date();
15     %>
16     <h1>
17         Today is
18         <%= dateFormat.format(date) %>
19         !
20     </h1>
21     <%@ include file="../fragments/piedDePage.html" %>
22 </body>
23 </html>
```

Les éléments de script

Les **éléments de script** sont très utilisés dans une page **JSP**. Ils permettent de rendre le **contenu dynamique**. Il y a deux types de scripts principaux :

- Les scripts écrits en **Java**
- Les scripts écrits en **EL**

Les éléments de scripts permettent de faire un mélange de code, classiquement un mélange Java/HTML

Les éléments de script

Les déclarations

- Les déclarations sont des éléments de script permettant de déclarer des variables membres et des méthodes dans la classe Java générée à partir de la page JSP. Le code doit être écrit dans la page JSP entre les balises suivantes : **<%! Vos variables membres et vos méthodes %>**

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3
4 <%!
5     //Déclarations des variables membres et des méthodes
6     private int compteur = 0;
7
8     private void incrementer()
9     {
10         this.compteur++;
11     }
12 %>
13 <!DOCTYPE html>
14 <html>
15 <!-- ... -->
16 </html>
```

Les éléments de script

Les scriptlets

- Il est possible d'y déclarer une variable locale à la méthode. La variable est utilisable dans les scriptlets et les éléments scripts de type expression dans le fichier. Mais la variable n'est pas accessible dans les éléments de scripts de type déclaration.
- Il est aussi possible de conditionner des traitements, des affichages...
- Le code doit être écrit dans la page JSP entre les balises suivantes : **<% votre code %>**

```
16 <body>
17 <%
18     this.compteur=0;
19     Random rd = new Random();
20     int nombreAppels = rd.nextInt(21);
21     for(int i= 0;i< nombreAppels; i++)
22     {
23         this.incrementer();
24     }
25 %>
26
27 <%
28     if(this.compteur>10)
29     {
30 %>
31 <p>Le compteur dépasse 10</p>
32 <%
33     }
34     else
35     {
36 %>
37 <p>Le compteur ne dépasse pas 10</p>
38 <%
39     }
40 %>
41 </body>
```

Les éléments de script

Les expressions

- Les expressions sont des éléments de script permettant de simplifier l'écriture des sorties dynamiques dans la réponse HTTP.
- Le code doit être écrit dans la page JSP entre les balises suivantes : **<%= le contenu à afficher sur une ligne %>**
- Ce contenu correspond au paramètre de la méthode `write(...)` de l'objet de type `JspWriter` disponible dans la méthode `_jspService(...)`
- Comme ce contenu correspond au paramètre de la méthode, il ne faut pas mettre de point-virgule à la fin de l'instruction.

```
<p>Le compteur dépasse 10. Il vaut <%=compteur %></p>
```

Les éléments de script

Les commentaires

- Les commentaires sont des éléments de script qui sont totalement ignorés lors de la transformation de la page JSP en classe Java.
- Ils s'écrivent ainsi :

```
<%--  
    Cette page permet de comprendre  
    le fonctionnement des éléments de script  
--%>
```

Les objets disponibles dans une JSP

L'objet `request` de type `HttpServletRequest` : cet objet a le même rôle que dans une servlet.

L'objet `response` de type `HttpServletResponse` : cet objet a le même rôle que dans une servlet.

L'objet `pageContext` de type `javax.servlet.jsp.PageContext` : cet objet permet d'accéder différents contextes

- *`PageContext.APPLICATION_SCOPE`, `PageContext.REQUEST_SCOPE`, `PageContext.SESSION_SCOPE` et `PageContext.PAGE_SCOPE`.*
- Cet objet met à disposition des méthodes permettant de manipuler les attributs :
 - `getAttribute(String name, int scope)` : cette méthode permet d'obtenir la valeur d'un attribut dans un contexte donné.
 - `setAttribute(String name, int scope)` : cette méthode permet d'écrire la valeur d'un attribut dans un contexte donné.
 - `removeAttribute(String name, int scope)` : cette méthode permet de supprimer la valeur d'un attribut dans un contexte donné.
 - `findAttribute(String name)` : cette méthode permet d'obtenir la valeur d'un attribut en le cherchant dans les différents contextes dans l'ordre suivant : `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE`, `APPLICATION_SCOPE`.

Les objets disponibles dans une JSP

L'objet **session** de type `HttpSession` : cet objet a le même rôle que dans une servlet. Cet objet n'est pas disponible si la directive page possède l'attribut `session` avec la valeur `false`.

L'objet **application** de type `ServletContext` : cet objet a le même rôle que dans une servlet. Pour rappel, dans une servlet, cet objet est accessible au travers de la méthode `getServletContext()` de la classe `HttpServlet`.

L'objet **config** de type `ServletConfig` : cet objet a le même rôle que dans une servlet. Pour rappel, dans une servlet, cet objet est accessible au travers de la méthode `getServletConfig()` de la classe `HttpServlet`.

L'objet **out** de type `javax.servlet.jsp.JspWriter` : cet objet permet d'écrire le corps de la réponse HTTP.

L'objet **exception** de type `java.lang.Throwable` : cet objet permet d'accéder à l'exception en cours. Cet objet n'est disponible que si la directive page possède l'attribut `isErrorPage` positionné à `true`. La valeur par défaut est `false` donc dans les cas standards, cet objet n'est pas disponible.

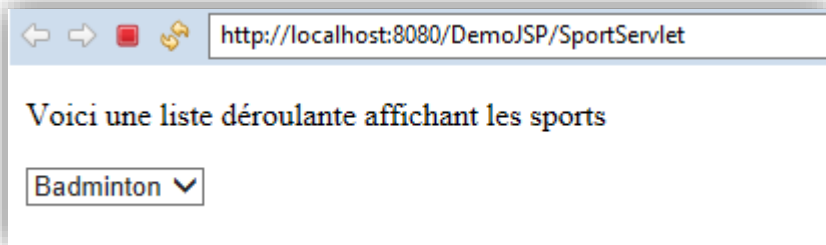
Les objets disponibles dans une JSP

Exemple: La servlet **SportServlet.java**

```
14 @WebServlet("/SportServlet")
15 public class SportServlet extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     public SportServlet() {
19         super();
20     }
21
22
23     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException {
24         List<String> sports = new ArrayList<String>();
25         sports.add("Badminton");
26         sports.add("Padel");
27         sports.add("Squash");
28         sports.add("Tennis");
29         //Mise en place de cette liste en attribut de requêtes
30         request.setAttribute("listeSports", sports);
31         //Délégation de l'affichage à la JSP nommée AffichageSport
32         RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/jsp/AffichageSport.jsp");
33         rd.forward(request, response);
34     }
35 }
```

Les objets disponibles dans une JSP

Exemple: La JSP **AffichageSport.jsp**



```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@page import="java.util.List"%>
4 <!DOCTYPE html>
5 <html>
6 <head>
7   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8   <title>Affichage Sport</title>
9 </head>
10 <body>
11   <p>Voici une liste déroulante affichant les sports</p>
12   <%
13     List<String> sports = (List<String>)request.getAttribute("listeSports");
14     %>
15   <%
16     if(sports!=null)
17     {
18     %>
19     <select>
20     <%
21       for(String sport:sports)
22       {
23       %>
24       <option>
25         <%=sport%>
26       </option>
27     %>
28       }
29     %>
30     </select>
31   <%
32   }
33   else
34   {
35   %>
36   <P>Pas de sport disponible</p>
37   <%
38   }
39   %>
40 </body>
41 </html>
```


La gestion des erreurs

L'origine des **erreurs à la transformation**:

- Un mauvais usage des directives,
- Un mauvais usage des éléments de script.

Exemple: Une erreur liée à l'application d'une mauvaise valeur dans l'attribut **buffer** de la directive **page**. Le développeur a écrit 10 au lieu de 10kb.

```
<%@page language="java" contentType="text/html; charset=UTF-8"  
    pageEncoding="UTF-8" buffer="10"%>
```



La gestion des erreurs

Les **erreurs à l'exécution** sont des erreurs qui peuvent arriver en production.

Afin de pouvoir traiter correctement ces erreurs, il est possible de:

- Utiliser de l'instruction **try/catch**
 - Les éléments de script de type déclaration et de type scriptlet peuvent accueillir très simplement la structure de code Java try/catch
- Utiliser des **pages d'erreur JSP**
 - L'utilisation des pages d'erreur est une autre solution pour gérer les erreurs.
 - Une page pouvant lever une erreur lors de son exécution doit déclarer dans la **directive page** un attribut **errorPage** faisant référence à la page à utiliser pour traiter l'erreur et fournir une réponse adaptée à l'utilisateur.

La gestion des erreurs

Les erreurs à l'exécution

- Exemple:

```
*PageAvecErreur.jsp Apache Tomcat/7.0.75 - Rapport d'erreur
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6   <meta charset="UTF-8">
7   <title>Cette page provoque une erreur</title>
8 </head>
9 <body>
10   <p>Une erreur non gérée par un try/catch est levée :</p>
11   <%
12     String chaine = null;
13     //permet de provoquer l'envoi d'une réponse au client
14     out.write(chaine.toUpperCase());
15   %>
16 </body>
17 </html>
```

```
http://localhost:8080/Demo/JSP/Test
Etat HTTP 500 - An exception occurred processing JSP page

type Rapport d'exception
message An exception occurred processing JSP page /WEB-INF/jsp/PageAvecErreur.jsp at line 14
description Le serveur a rencontré une erreur interne qui l'a empêché de satisfaire la requête.
exception
org.apache.jasper.JasperException: An exception occurred processing JSP page /WEB-

11:   <%
12:       String chaine = null;
13:       //permet de provoquer l'envoi d'une réponse au client
14:       out.write(chaine.toUpperCase());
15:   %>
16: </body>
17: </html>

Stacktrace:
org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletW
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:395)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:339)
```

La gestion des erreurs

Les erreurs à l'exécution

- Exemple: avec un page d'erreur JSP

```
PageAvecErreur.jsp PageTraitantLesErreurs.jsp Cette page traite les erreurs
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8" errorPage="./PageTraitantLesErreurs.jsp"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Cette page provoque une erreur</title>
8 </head>
9 <body>
10 <p>Une erreur non gérée par un try/catch est levée :</p>
11 <%
12     String chaine = null;
13     //permet de provoquer l'envoi d'une réponse au client
14     out.write(chaine.toUpperCase());
15 %>
16 </body>
17 </html>
```

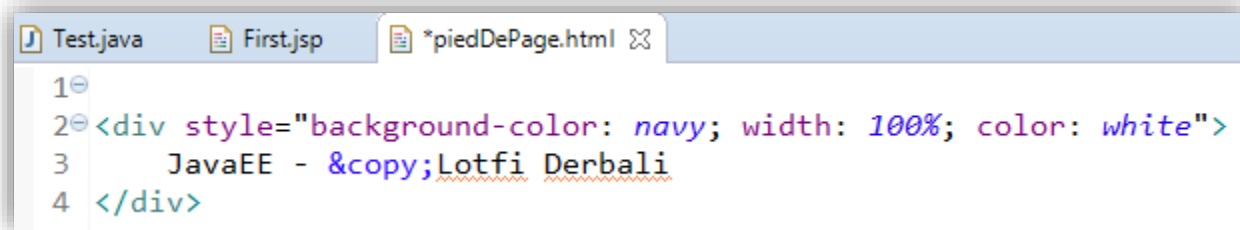


```
PageAvecErreur.jsp PageTraitantLesErreurs.jsp Cette page traite les erreurs
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8" isErrorPage="true"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Cette page traite les erreurs</title>
8 </head>
9 <body>
10 <h1>Une erreur est survenue</h1>
11 <p>
12     <%=exception.getClass().getName()%>
13 </p>
14 <p>
15     <%=exception.getMessage()%>
16 </p>
17 <p>
18     Veuillez cliquer sur ce lien pour continuer <a
19       href="<%=request.getContextPath()%>/index.html">Cliquez ici</a>
20 </p>
21 </body>
22 </html>
```

L'utilisation des fragments

L'inclusion statique

- La **directive include** permet d'inclure une ressource externe dans la JSP au moment de la transformation.
- Le code du fragment est copié dans chacune des classes Java issues des pages JSP ayant utilisé cette directive.
- Problème: en cas de **mise à jour du fragment**, **l'ensemble des pages JSP doit être transformé de nouveau** pour inclure les nouvelles modifications.



```
1
2 <div style="background-color: navy; width: 100%; color: white">
3     JavaEE - &copy;Lotfi Derbali
4 </div>
```

```
<%@ include file="../../../fragments/piedDePage.html" %>
```

L'utilisation des fragments

L'inclusion dynamique

- L'inclusion dynamique se fait au travers de l'utilisation de l'action standard `<jsp:include>`.
- Cette balise permet d'inclure dynamiquement une ressource comme cela peut être réalisé à l'aide de la méthode `include(...)` de l'interface `RequestDispatcher`.
- La **modification** de la ressource incluse n'entraîne pas une transformation des pages JSP l'incluant.

La balise `<jsp:include>` possède deux attributs :

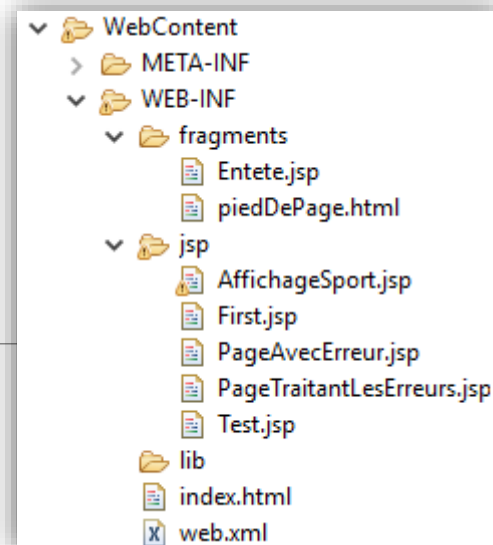
- **page** : cet attribut obligatoire permet de définir le chemin d'accès à la ressource à inclure. Ce chemin est relatif à la JSP courante.
- **flush** : cet attribut optionnel permet d'indiquer si le début de la réponse doit être renvoyé au client avant d'inclure la ressource ciblée par l'attribut page. Cet attribut est un booléen. Par défaut, la valeur est false.

L'utilisation des fragments

Exemple:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException {
    HttpSession session = request.getSession();
    session.setAttribute("utilisateurConnecte", "Lotfi Derbali");

    List<String> sports = new ArrayList<String>();
    sports.add("Badminton");
    sports.add("Padel");
    sports.add("Squash");
    sports.add("Tennis");
    //Mise en place de cette liste en attribut de requêtes
    request.setAttribute("listeSports", sports);
    //Délégation de l'affichage à la JSP nommée AffichageSport
    RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/jsp/AffichageSport.jsp");
    rd.forward(request, response);
}
```



```
PageAvecErreur.jsp *Entete.jsp AffichageSport.jsp SportServlet.java Affichage Sport
1 <header>
2     <h1>Bienvenue sur l'application</h1>
3     Bonjour <%=session.getAttribute("utilisateurConnecte")%>
4 </header>
```

```
PageAvecErreur.jsp Entete.jsp AffichageSport.jsp SportServlet.java Affichage Sport
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.util.List"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Affichage Sport</title>
</head>
<body>

    <jsp:include page="../fragments/Entete.jsp"></jsp:include>
    <%
        List<String> sports = (List<String>)request.getAttribute("listeSports");
    %>
    <%
```

