Bienvenue !

Pierre-Julien VILLOUD

pjvilloud@protonmail.com
https://github.com/pjvilloud
https://linkedin.com/pjvilloud
http://pjvilloud.github.io

Introduction

Nous avons vu dans le cours précédent comment générer dynamiquement des pages HTML avec les servlets. Pour l'instant nous avons gérer la partie HTML dans le contrôleur. Afin de respecter mieux coller au principe du *MVC*, nous allons voir les JSP.

Une page JSP est une page HTML dans laquelle on place des bouts de code Java (de plusieurs manières).

Les scriptlets

Un **scriptlet** est un bout de code java placé entre `<% %>`. On peut placer une scriptlet dans une page à partir du moment ou elle porte l'extension `.jsp` et qu'elle est hébergée sur un serveur d'application (Tomcat par exemple).

```
<html lang="en">
  <head>
      <title>JSP</title>
  </head>
  <body>
  <h1>

   <% out.println("Hello !"); %>

  </h1>
  </body>
</html>
```

Une bonne pratique consisste à, dans la mesure du possible, ne mettre que les éléments qui ne sont pas du HTML pour éviter par exemple `<% out.println("<h1>Hello !</h1>"); %>`.

Les objets JSP

ℹ Différents objets sont accessibles depuis une JSP.

Nom	Type	Description
`request`	`HttpServletRequest`	Objet contenant tous les données de la requêtes HTTP initiale (paramètres...)
`response`	`HttpServletResponse`	Objet contenant toutes les données de la réponse HTTP du serveur (statut HTTP...)
`out`	`JspWriter`	Objet utilisé pour afficher des données au niveau du client avec `print`
`session`	`HttpSession`	Objet gardant les informations de la session utilisateur. Il est possible de manipuler des données entre les différentes pages avec cet objet.
`application`	`ServletContext`	Objet gardant les informations au niveau de l'application globale.
`config`	`ServletConfig`	Objet contenant les informations d'initialisation de la servlet.
`page`	`this`	Objet représentant la page, équivalent du `this` de la servlet.

Les paramètres (avec `GET` ou `POST`)

ℹ L'objet `request` permet de récupérer les éventuels paramètres passés dans l'URL (dans le cas de `GET`) ou dans la requête (dans le cas de `POST`).

```
<html lang="en">
  <head>
      <title>JSP</title>
  </head>
  <body>
  <h1>

   <%
    //On peut utiliser ici n'importe quel code Java !
    String nom = request.getParameter("name");
    if(nom == null){
      out.println("Hello " + nom + " !");
    } else {
      out.println("Hello !");
    }
   %>

  </h1>
  </body>
</html>
```

ℹ L'objet `request` est de la classe `HttpServletRequest` déjà vue dans le cours précédent.

Les expressions

ℹ La notation en utilisant `out.print` à chaque fois est assez lourde ! Lorsqu'on vu uniquement afficher directement la valeur d'une variable, on peut utiliser les *expressions*

```
<html lang="en">
  <head>
      <title>JSP</title>
  </head>
  <body>
  <h1>

    Hello <%= request.getParameter("nom") %>

  </h1>
  </body>
</html>
```

ℹ Attention à tout de même tester les variables car si `nom` n'est pas renseigné, `null` sera affiché

Les déclarations

Tous les scriptlets d'une JSP sont inclus à la compilation dans une seule méthode. Ainsi, lorqu'on déclare une variable avec les balises `<% %>`, la variable est locale à la méthode (et donc réinitialisée à chaque chargement de page). Il est possible de déclarer des variables et des méthodes avec les balises de déclarations `<%! %>`. Dans ce cas, les variables deviennent des attributs de classe et ont la même durée de vie que la classe.

```
<html lang="en">
  <head>
      <title>JSP</title>
  </head>
  <body>

  <%! int compteur = 0; %>
  <%  int compteur2 = 0; %>
  <%! int add(int c, int c2) {
        return c + c2;
    }%>

<%= ++compteur %> <%= ++compteur2 %>
    <%= add(compteur,compteur2) %>

  </body>
</html>
```

Au premier chargement de la page, la page affichera *1 1 2*, au deuxième *2 1 3* ...

Les directives

ℹ Les directives JSP permettent d'aller plus loin que de simples scriptlets.

Inclure d'autres pages JSP
Importer de classes issues d'autres packages
Ajouter des *bibliothèques* de balises
Définir des propriétés sur la page JSP

ℹ Pour définir une directive, on utilise les balises `<%@ %>` et on doit les placer en haut de la page JSP (sauf la directive d'inclusion de page).

Directive `include`

ℹ Il est courant d'avoir le même code sur plusieurs pages HTML (menu, footer...). Pour éviter le copier-coller, il est possible d'inclure des pages JSP au sein de votre page avec la directive `include`.

```
<%@ include file="header.jsp" %>
...
<%@ include file="footer.jsp" %>
```

⚠ Attention, l'inclusion est *basique*, c'est-à-dire que le code contenu dans la JSP est simplement ajouté au reste de votre page. Si vous aviez des éléments manipulés par un contrôleur, ceux-ci ne seront pas définis.

Directive `page`

Il est courant d'avoir le même code sur plusieurs pages HTML (menu, footer...). Pour éviter le copier-coller, il est possible d'inclure des pages JSP à la compilation au sein de votre page avec la directive `include`.

```
<%@ page contentType="text/html;charset=UTF-8" %>
...
<%@ page import "java.util.List" %>
```

Ici, nous avons défini l'encodage de la page avec `contentType` et importé la classe `List` qui pourra désormais être utilisée dans des scriptlets. Il est possible aussi d'utiliser `<jsp:include page="header.jsp" />` qui inclue dynamiquement à l'exécution le code.

Directive `taglib`

ℹ Cette directive permet d'inclure une *librairie de tags* qui pourront être utilisés dans la page.

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

ℹ Nous verrons quelques possibilités de la librairie JSTL plus tard

Les JavaBeans

Il est possible d'utiliser les objets des classes définies par Java (Integer...) mais il est aussi possible de manipuler nos propres objets (*JavaBeans*)

```java
public class Vehicule {
  private String marque;
  private String modele;

  private String getMarque(){
    return this.marque;
  }
  private void setMarque(String marque){
    this.marque = marque;
  }

  private String getModele(){
    return this.modele;
  }
  private void setModele(String modele){
    this.marque = marque;
  }
}
```

```jsp
//Appelle le constructeur par défaut
<jsp:useBean id="vehicule" class="com.ipiecoles.Vehicule" />

//Appelle le setter de marque avec la valeur Peugeot
<jsp:setProperty name="vehicule" property="marque" value="Peugeot"/>

//Appelle le setter de marque avec le paramètre marque
//dans l'URL pour GET ou dans la requête pour POST
<jsp:setProperty name="vehicule" property="marque" param="marque"/>

//Appelle le getter de marque qui retourne la valeur "Peugeot"
<jsp:getProperty name="vehicule" property="marque" />

//L'objet sera conservé pendant toute la durée de la session d'un même
//utilisateur et accessible par toutes les JSP qui l'auront déclaré
<jsp:useBean id="vehicule" class="com.ipiecoles.Vehicule" scope="session"/>
//Scope peut valoir request, page (défaut), session ou application
```

La portée *application* implique l'objet est partagé par toutes les pages JSP exécutées sur le serveur. La portée *request* implique la création d'un nouvel objet à chaque requête. La portée *page* implique que l'objet n'est visique que dans la servlet et a la même durée de vie que cette dernière.

Expression Language

L'utilisation des notations *JavaBeans* est assez lourde... Nous allons donc voir *Expression Language* ou *EL* qui propose une syntaxe beaucoup plus simple et beaucoup plus riche.

```
//Soit le véhicule { marque: "Peugeot", modele: "208", puissance: 90, prixHT:12000}
Véhicule : ${vehicule} //Affiche toString de vehicule

Marque : ${vehicule.marque} //Affiche "Peugeot"

Modèle : ${empty vehicule.modele ? "?" : vehicule.modele} //Affiche 208

Date de mise en circulation :
${!empty vehicule.dateMiseCirculation ? vehicule.dateMiseCirculation : "?"}
//Affiche ?

Puissance :
supérieure à 80ch ? ${vehicule.puissance 80} //Affiche true
inférieure à 50ch ? ${vehicule.puissance < 50} //Affiche false
entre 80ch et 100ch ? ${vehicule.puissance > 80 && vehicule.puissance < 100}
//Affiche true

Prix TTC : ${vehicule.prixHT * 1.2} //Affiche 14400
```

Cette syntaxe permet également d'effectuer des tests simples. A noter que `==` appelle `equals()` et `> ou <` utilise `compareTo()`. Il n'y a pas de `null` en *EL*, si un objet est `null`, rien ne sera affiché.

Collections

On peut également manipuler des collections avec *EL*

```
//Si vehicules est un Vehicule[]
${vehicules[0].marque} ${vehicules['0'].marque} ${vehicules["0"].marque}

//Si vehicules est une List<Vehicule>, idem ci-dessus +
${vehicules.get(0).marque}

//Si vehicules est une Map, la syntaxe est analogue à celle de la List
//mais on remplace l'indice par la clé de la map.

Modèle : ${empty vehicule.modele ? "?" : vehicule.modele} //Affiche 208

Date de mise en circulation :
${!empty vehicule.dateMiseCirculation ? vehicule.dateMiseCirculation : "?"}
//Affiche ?

Puissance :
supérieure à 80ch ? ${vehicule.puissance 80} //Affiche true
inférieure à 50ch ? ${vehicule.puissance < 50} //Affiche false
entre 80ch et 100ch ? ${vehicule.puissance > 80 && vehicule.puissance < 100}
//Affiche true

Prix TTC : ${vehicule.prixHT * 1.2} //Affiche 14400
```

Les objets *EL*

ℹ Tous les objets EL (sauf `pageContext`) sont des `Map` permettant l'accès aux propriétés avec `nomObjet.nomParametre`.

Nom	Description
`pageContext`	Objet permettant d'accéder à plusieurs objets JSP (request, response...)
`pageScope`, `sessionScope`, `requestScope` et `applicationScope`	Map contenant les attributs de portée page, session, requête et application.
`header` et `headerValues`	Map contenant les valeurs des entêtes de la requête HTTP.
`param` et `paramValues`	Map contenant les valeurs des paramètres de la requête.
`initParam`	Map contenant les paramètres d'initialisation du fichier `web.xml`.
`cookie`	Map contenant les éventuels cookies.

JSTL

JSTL, pour *JSP Standard Tag Library* contient de nombreux tags permettant de faire des itérations, des conditions et autres fonctionnalités intéressantes pour éviter d'écrire directement du Java dans les JSP.

Les tags de base de `core`

ℹ Voici les tags de base permettant de manipuler des variables et d'effectuer des tests de conditions.

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

<c:out value="${vehicule.description}" default="Pas de description" escapeXml=false />

<c:set var = "vehiculeChoisi" scope = "session" value = "vehicule"/>
<c:remove var = "vehiculeChoisi" />

<c:if test = "${vehicule.puissance > 130}">
   Malus écologique
</c:if>

<c:choose>
  <c:when test = "${vehicule.puissance <= 50}">
    Catégorie 1
  </c:when>
  <c:when test = "${vehicule.puissance <= 100}">
    Catégorie 2
  </c:when>
  <c:otherwise>
    Catégorie 3
  </c:otherwise>
</c:choose>
```

Les tags de base de `core` (2)

Voici les tags de base permettant d'effectuer des boucles

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

<c:forEach var = "i" begin = "1" end = "5">
   Véhicule <c:out value = "${i}"/>
</c:forEach>
//Véhicule 1 Véhicule 2 Véhicule 3 Véhicule 4 Véhicule 5

<c:forTokens items = "Peugeot,Citroën,Renault" delims = "," var = "marque">
   <c:out value = "${marque}"/><p>
</c:forTokens>
//Peugeot Citroën Renault

<c:url value = "/recherche.jsp" var = "myURL">
   <c:param name = "marque" value = "Peugeot"/>
   <c:param name = "modele" value = "208"/>
</c:url>

<c:redirect url = "http://www.google.com"/>
```

Les tags de formatage `fmt`

ℹ La bibliothèque `fmt` permet d'effectuer des opération de formattage sur des nombres.

```
<%@ taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>
<c:set var = "prix" value = "12000.2309" />
<c:set var = "percentage" value = "0.156" />
<fmt:setLocale value = "fr_FR"/>
<fmt:formatNumber value = "${prix}" type = "currency" currencySymbol="€"/> 12 000,23 €
<fmt:formatNumber type = "number" maxFractionDigits = "3" value = "${prix}" /> 12 000,231
<fmt:formatNumber type = "number" groupingUsed = "false" value = "${prix}" /> 12000,231
<fmt:formatNumber type = "number" pattern = "###.###E0" value = "${prix}" /> 12,0002E3
<fmt:formatNumber type = "percent" maxIntegerDigits="3" value = "${percentage}" /> 16 %
<fmt:formatNumber type = "percent" minFractionDigits = "2" value = "${percentage}" /> 15,60 %
<fmt:formatNumber type = "percent" maxIntegerDigits = "1" value = "${percentage}" /> 6 %
```

ℹ Toutes les possibilités offertes par `fmt:formatNumber` tag sont répertoriées ici. il y a également ici les informations sur le tag `fmt:parseNumber` qui fait l'inverse.

Les tags de formatage (2)

ℹ La bibliothèque `fmt` permet d'effectuer des opération de formattage sur des dates.

```
<c:set var="now" value="<%=new java.util.Date()%>"/>
<fmt:setLocale value = "fr_FR"/>
<fmt:formatDate type = "time" value = "${now}" /> 17:36:55
<fmt:formatDate type = "date" value = "${now}" /> 2 févr. 2018
<fmt:formatDate type = "both" value = "${now}" /> 2 févr. 2018 17:36:55
<fmt:formatDate type = "both" dateStyle = "short" timeStyle = "short" value = "${now}" /> 02/02/18 17:36
<fmt:formatDate type = "both" dateStyle = "medium" timeStyle = "medium" value = "${now}" /> 2 févr. 2018 17:36:55
<fmt:formatDate type = "both" dateStyle = "long" timeStyle = "long" value = "${now}" /> 2 février 2018 17:36:55 CET
<fmt:formatDate pattern = "yyyy-MM-dd" value = "${now}" /> 2018-02-02
```

ℹ Toutes les possibilités offertes par `fmt:formatDate` tag sont répertoriées ici. il y a également ici les informations sur le tag `fmt:parseDate` qui fait l'inverse.