Nathanaël COTTIN

Java Enterprise Edition

Concepts et pratique

Version 0.0.2

www.ncottin.net

Table des matières

1. Introduction

La technologie Java est aujourd'hui incontournable, tant pour le développement d'applications autonomes que pour l'utilisation des outils orientés web. La technologie Java est ici employée afin de générer des pages HTML dynamiques (DHTML). Elles permettent l'intégration de l'ensemble des fonctionnalités de Java Enterprise Edition (XML, bases de données, services web, systèmes distribués CORBA, etc.), contrairement à d'autres langages web (tel que PHP par exemple).

Ce document présente l'essentiel des connaissances permettant de développer des applications web en s'appuyant sur JavaEE, des technologies les plus simples (servlets) aux plus évoluées (VisualJSF avec Hibernate par exemple).

L'objectif est de proposer un apprentissage progressif en s'appuyant sur une mise en situation concrète à l'aide de l'environnement de développement (EDI) NetBeans et du serveur d'applications Glassfish. Bien entendu, les concepts peuvent être aisément portés à d'autres EDIs et serveurs d'applications. Les questions de sécurité (authentification HTTPS, etc.) ne seront pas abordées dans ce document.

Ce document est organisé comme suit :

- Description rapide des diverses technologies JavaEE
- Besoins de serveurs d'applications (par rapport aux serveurs web notamment)
- Présentation de l'exemple qui sera implanté par les diverses technologies JavaEE présentées (à l'exception des servlets)
- Présentation des servlets et mise en œuvre d'un exemple simple de type « Hello World »
- Description des JSP, intérêt par rapport aux servlets et utilisation des JavaBeans et librairies de tags (taglibs)
- L'environnement Struts en tant que JSP amélioré par une meilleure intégration du concept MVC en se basant sur un développement évènementiel (réaction aux évènements générés par l'utilisateur)
- Une simplification de Struts avec JSF

- Rédiger du HTML uniquement en XML avec VisualJSF
- Spring
- L'accès aux bases de données avec l'aide d'Hibernate
- Les objets distribués avec le déploiement de services web et leur utilisation au sein d'une application JSP puis JSF
- Les composants EJB 3.0 permettant de développer des systèmes distribués

Bonne lecture !

2. Technologies JavaEE

3. Serveur web, serveur d'applications

Un serveur serveur web ou un serveur d'applications peuvent tous deux héberger des pages HTML statiques mais il ne s'agit que d'une conséquence de la possibilité de traiter des pages dynamiques.

Le traitement d'une demande émanent d'un client sur un serveur d'applications est comparable au traitement réalisé par un serveur web. Elle se résume comme suit :

1. Le serveur reçoit une requête
2. Il vérifie l'accès à la page demandée
3. En cas de succès, il active le moteur de traitement approprié (moteur PHP par exemple)
4. La page web générée est retournée au client

Nous parlons ici de page web au sens général, il peut s'agir de contenu autre : un flux vidéo, une image, un fichier pdf, etc.

Le serveur d'applications se différencie cependant d'un serveur web traditionnel notamment parce qu'il propose une gestion avancée de la montée en charge et de la disponibilité des pages hébergées.

4. Présentation des exemples suivis

L'illustration concrète des technologies présentées ici est réalisée par le biais de deux exemples communs (qui permettent de comparer lesdites technologies).

4.1. Informations d'identité

Cet exemple basique consiste à transmettre en HTTP POST un formulaire contenant les informations suivantes : nom, prénom et date de naissance. La page recevant la soumission du formulaire affiche les informations envoyées ainsi que l'âge de la personne.

4.2. Réalisation d'un sondage de popularité

Cet exemple plus complexe propose de réaliser une application web permettant de voter parmi une liste de candidats afin d'élire le candidat le plus populaire. L'application doit pouvoir afficher en temps réel le nombre de voix attribuées à chacun des candidats. Les fonctionnalités proposées sont par conséquent :

- Le vote : l'application n'intègre aucune sécurité, il est donc permis à un utilisateur de voter plusieurs fois (c'est le principe mis en œuvre par certaines chaînes de télévision réalisant un bénéfice pour chaque appel téléphonique...)
- La consultation des résultats : le pourcentage de voix (par rapport au nombre total de votes ayant eu lieu) pour chaque candidat participant au sondage de popularité

Cet exemple suit le schéma de navigation suivant :

5. Les servlets

Ou « un peu d'HTML dans beaucoup de Java... »

5.1. Présentation

5.1.1. Définition

La technologie la plus basique afin de réaliser des applications web en Java est sans conteste l'utilisation des servlets. D'ailleurs, quel que soit le langage (framework) employé, les traitements automatisés (et donc masqués au développeur et à l'utilisateur) ont pour objectif principal de générer automatiquement, et généralement dynamiquement, des servlets.

La servlet, la réponse de Sun aux programmes CGI (« Common Gateway Interface »), est par conséquent le composant unitaire qu'un serveur d'applications instancie lorsqu'il déploie une application web écrite en Java. Exécutée sur un serveur d'applications, une servlet intercepte les requêtes des navigateurs web et génère des réponses (généralement en HTML) construites dynamiquement à l'aide du langage Java.

Avec l'avènement de langages plus évolués, les servlets manipulées par les développeurs occupent aujourd'hui un rôle de séparation entre présentation (pages JSP par exemple) et traitements (déportés sur les servlets).

Une servlet est concrètement une classe Java destinée à recevoir et transmettre des flux. Elle implémente l'interface « `javax.servlet.Servlet` » présente dans JavaEE. Lorsque ces flux répondent au protocole HTTP, une servlet est construite par héritage de la classe (non abstraite) « `javax.servlet.http.HttpServlet` ».

Cette dernière propose une mise en œuvre vide des diverses opérations « `do...()` » (telles que `doGet()` ou `doPost()` pour ne citer que les deux plus connues) et déclare l'opération « `service()` » comme chargée de rediriger la requête vers l'opération « `do...()` » correspondant à la méthode HTTP souhaitée par le demandeur.

5.1.2. Cycle de vie d'une servlet

Le conteneur de servlets du serveur d'applications hôte gère leur cycle de vie conformément à la figure 5.1.

Les méthodes « init() » et « `destroy()` » ne sont appelées qu'une seule fois par le conteneur de servlets, respectivement lors du premier appel de la servlet (première instanciation) et lorsque le serveur d'applications termine son exécution. Les autres méthodes sont appelées à chaque

invocation de la servlet.

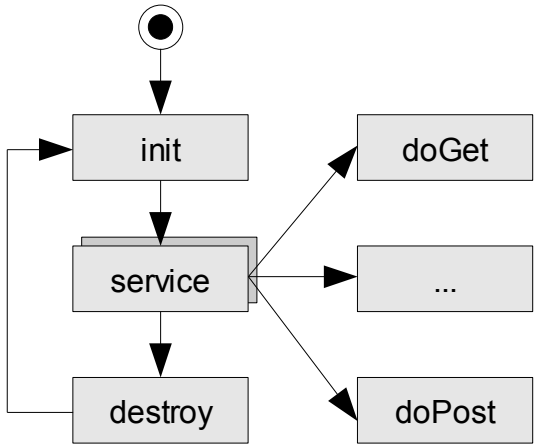

Fig. 5.1: Cycle de vie d'une servlet

Les deux opérations généralement redéfinies sont « `doGet()` » et « `doPost()` », bien qu'il soit possible de redéfnir l'opération « service() ». Le protocole HTTP comprend également les méthodes PUT, DELETE, HEAD, INFO et TRACE pour lesquelles sont définies respectivement les opérations « `doPut()` », « `doDelete()` », « `doHead()` », « `doInfo()` » et « `doTrace()` ».

Dans le cas où seules les opérations « `doGet()` » et « `doPost()` » sont redéfinies, l'exécution de l'opération « `doDelete()` » (en réponse à une requête HTTP de type DELETE) produira l'erreur HTTP 405 (méthode non supportée par l'URL).

5.2. Création d'un projet

Sous NetBeans, créer un projet d'application web sans spécifier de framework et en indiquant Glassfish V2 (et non V3 !) comme serveur d'applications cible.

Une page d'index JSP est alors automatiquement créée par NetBeans.

5.3. Exemple de type « Hello World »

Créer un projet comme indiqué précédemment.

5.3.1. Rédaction de la servlet

Effectuer un clic droit sur le nom du projet dans l'explorateur de projets et

sélectionner « New » puis « Servlet » (au besoin, sélectionner le menu « Other », puis choisir « Servlet » dans le dossier « Web »).

Cliquer sur « Suivant » puis nommer la classe de cette servlet, par exemple « SimpleServlet ». Ne pas oublier d'indiquer un nom de paquetage (tel que «simpleservlet»).

Cliquer à nouveau sur « Suivant » : la boîte affichée permet de modifier le nom attribué à la servlet ainsi que l'adresse à laquelle a servlet sera contactée. Cliquer sur « Terminer ».

Le canevas de code source suivant est alors généré par NetBeans :

```
package simpleservlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author ncottin
 */
public class SimpleServlet extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code>
     * and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet SimpleServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet SimpleServlet at "
                        + request.getContextPath () + "</h1>");
            out.println("</body>");
            out.println("</html>");
```

```
            */
        } finally {
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed"
    // desc="HttpServlet methods. Click on the + sign on the left
    // to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Compléter le code précédent pour obtenir :

```
package simpleservlet;

import java.io.IOException;
```

```java
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author ncottin
 */
public class SimpleServlet extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code>
     * and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
           String title = "Simple Servlet";
           out.println("<html>");
           out.println("    <head>");
           out.println("      <title>" + title + "</title>");
           out.println("    </head>");
           out.println("<body>");
           out.println("<h1>" + title + "</h1>");
           out.println("La servlet est en cours d'exécution");
           out.println("</body>");
           out.println("</html>");
           out.flush();
        } finally {
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed"
    // desc="HttpServlet methods. Click on the + sign on the left
    // to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(
```

```
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Exemple simple d'exécution d'une servlet";
    }// </editor-fold>
}
```

5.3.2. Configuration du projet avant exécution

Par défaut, la page « index.jsp » est indiquée comme page d'accueil. Cependant, nous souhaitons exécuter la servlet. Un clic droit sur le nom du projet, puis « Properties » permet, dans la catégorie « Run » ,de spécifier l'URL de la servlet, à savoir « SimpleServlet ». ce nom correspond au nom indiqué dans le dernier écran affiché lors de la demande de création de la servlet.

5.4. Exécution d'un projet

Un projet de type « Servets » est exécuté à l'aide d'un clic droit sur le nom du projet puis en sélectionnant « Run ». Cette opération construit le projet, exécute le serveur d'applications et affiche le navigateur web par défaut. Si tout s'est déroulé correctement, la page HTML correspondant à l'exécution de la servlet s'affiche.

Afin d'afficher la page adéquate, penser à effectuer les opérations

de configuration de la page d'accueil du projet (afin d'éviter d'afficher la page « index.jsp »...

5.5. Instanciation et initialisation

Une servlet étant une classe Java avant tout, l'instanciation de celle-ci par le conteneur de servlets du serveur d'applications exécute le constructeur sans argument qui lui est associé. Ce constructeur peut être défini implicitement (non mentionné dans le code source de la servlet, comme c'est le cas dans l'exemple précédent) ou explicitement.

Lors de la phase d'instanciation, la servlet ne connait pas son environnement (contexte d'exécution) de sorte que le constructeur de la servlet est généralement vierge. Dans le cas de l'exemple précédent, le constructeur n'est pas mentionné dans le code source.

5.6. Envoi et réception de paramètres

5.6.1. Envoi de paramètres

Les paramètres HTTP peuvent être déclarés de différentes manières, à savoir :

- Par réécriture d'URL : `http://.../servlet?p1=v1&p2=v2`
- Par le biais d'un formulaire où la valeur « action » indique le nom de la servlet destination (il s'agit ici de « servlet »)

Les paramètres sont transmis lors de la soumission de l'URL. L'emploi d'un formulaire supporte les requêtes HTTP aux formats GET et POST alors que la réécriture d'URL[1] se limite à la méthode HTTP GET uniquement.

5.6.2. Réception de paramètres

Les paramètres soumis sont reçus par la servlet indiquée en destination. Cette dernière dispose, via l'instance « request » de « HttpServetRequest », de l'opération « getParameter(String) » lui permettant de demander la valeur associée à un nom de paramètre :

```
String paramValue = request.getParameter("parameterName");
```

1 Souvent utilisée par des individus malveillants d'ailleurs...

Le protocole HTTP supporte l'envoi de plusieurs valeurs associée à un même paramètre. Récupérer l'ensemble des valeurs d'un paramètre demande l'emploi de l'opération « getParameterValues(String) » :

```
String[] paramValues = request.getParameterValues("parameterName");
```

5.7. Gestion des informations de session

Les servlets permettent de mettre en œuvre un mode « connecté » permettant de manipuler des informations communes à différentes pages (un identifiant utilisateur par exemple) sans avoir recours aux cookies.

En effet, le protocole HTTP ne propose aucun suivi de session si ce n'est par l'utilisation de champs cachés (`<input type="hidden"...>`) dans les formulaires. Cette fonctionnalité est supportée par les servlets par le biais de l'instance de la classe « HttpSession » obtenue à l'aide de l'opération « `getSession()` » de l'objet « `request` ».

Les opérations « `putValue()` » et « `getValue()` » permettent respectivement d'attribuer (mutateur) et consulter (accesseur) la valeur du paramètre de session identifié par son nom.

5.8. Limites

Les servlets imposent des contraintes de création des pages web telles que :

- L'inclusion du code HTML au sein de chaînes de caractères (avec les problèmes liés à l'échappement des doubles quotes par exemple)
- L'impossibilité d'insérer du code commun aux différentes pages de l'application web, si ce n'est par le biais d'un héritage ou de la composition (au sens UML) d'une classe mettant en œuvre des opérations d'écriture des en-têtes et pieds de page

C'est pourquoi d'autres outils chargés de simplifier le développement des interfaces web ont vu le jour, notamment les JSP.

6. Les JSP

Ou « un peu de Java dans beaucoup d'HTML... »

6.1. Présentation

6.1.1. Définition

La technologie JSP (« Java Server Pages ») simplifie la conception des pages HTML en Java en palliant les problèmes de rédaction inhérents aux servlets[2]. Les JSP se rapprochent ainsi du MVC qui sépare :

- Les traitements communs
- La logique métier
- La présentation

Conceptuellement parlant, une page JSP est comparable à une page rédigée à l'aide du langage PHP. Ainsi, du point de vue du concepteur, une page JSP est une page HTML dans laquelle s'intègrent des blocs de code Java appelés « scriptlets ». Les blocs Java de même que les directives JSP sont encadrées par les séquences « `<%` » et « `%>` » ou écrites en XML à l'aide de l'espace de nommage identifié par le préfixe « jsp ».

Les pages JSP sont traduites en servlets par un moteur inclus dans le serveur d'applications[3] lors de leur premier appel.

Un abus de langage fréquent consiste à omettre l'étape de traduction et considérer qu'une page JSP produit directement du code HTML.

6.1.2. Appel d'une page JSP

Le processus de réponse à l'appel d'une page JSP se décompose comme suit :

1. S'il s'agit d'un premier appel :
    2. La servlet correspondante est générée à partir du code JSP à l'aide du moteur de traduction inclus dans le serveur d'applications

2 Ce qui ne constitue en rien un dénigrement des servlets. Ces deux techniques ont leurs avantages et inconvénients.

3 Il s'agit de « Catalina » dans le cas du serveur Tomcat.

3. Une instance est créée (l'opération « `init()` » de la servlet est appelée)

4. Un algorithme de partage de ressources est employé par le serveur afin de gérer les threads associés à l'instance de la servlet invoquée et déterminer le thread devant répondre à la demande

5. Le thread choisi effectue le traitement en appelant l'opération « `service()` » de l'instance de la servlet

Concrètement, l'ensemble du code HTML est intégré au sein de l'opération « `service()` » redéfinie à cette occasion par le moteur de traduction de JSP en servlet.

6.2. Création d'un projet

Créer un projet NetBeans de type application web en ne sélectionner aucun framework. Une page « index.jsp » est automatiquement créée.

Il est possible d'utiliser la fenêtre « Palette » de NetBeans (accessible via le raccourci Ctrl+Maj+8) pour glisser les composants au sein de la page JSP (formulaires, champs, buttons d'envoi, liens hypertextes, etc.).

6.3. Exemple d'informations d'identité

Créer un projet comme indiqué précédemment.

A FAIRE...

6.4. Exemple du sondage

Créer un second projet.

Ce projet va faire appel à 3 JavaBeans, chacun ayant une portée spécifique, à savoir : « request », « session » et « application ».

6.4.1. « include/header.inc »

```
<jsp:useBean id="nbVotesBean"
  scope="session" class="sondage.bean.NbVotesBean" />
<jsp:setProperty name="nbVotesBean" property="candidat" />

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <title>Sondage JSP</title>
    </head>
    <body>
    <p>Vous avez voté <%= nbVotesBean.getVotes() %> fois.</p>
```

Ce fichier sera inclus en en-tête de chaque page JSP du projet.

6.4.2. « include/footer.inc »

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

Ce fichier sera inséré à la fin de chaque page web JSP du projet.

6.4.3. JavaBean « CandidatsBean »

Ce bean est chargé d'une part de fournir la liste des candidats pour le vote ainsi que de comptabiliser le nombre de voix obtenues par chaque candidat[4].

```
package sondage.bean;

/**
 * @author ncottin
 */
public final class CandidatsBean {

    public static final String[] candidats = {
        "Titi",
        "Gros minet",
        "Riri",
        "Fifi",
        "Loulou"
    };

    private int[] votes;

    public CandidatsBean() {
        votes = new int[candidats.length];
        for(int i=0; i<votes.length; i++) {
            votes[i] = 0;
```

4 D'autres manières plus élégantes sont possibles (utilisation d'objets « Map » parle biais de tables de hachage par exemple. L'objectif ici est de simplifier au maximum le projet...

```
        }
    }

    public void setCandidat(String candidat) {
        for (int i=0; i<candidats.length; i++) {
            if (candidats[i].equals(candidat)) {
                votes[i]++;
            }
        }
    }

    public int getVotes(int index) {
        return votes[index];
    }
}
```

6.4.4. Bean « NbVotesBean »

Ce bean permet de comptabiliser le nombre de fois qu'un utilisateur a voté au cours d'une même session. Cette comptabilisation est remise à zéro lorsque l'utilisateur ferme son navigateur web.

```
package sondage.bean;

/**
 * @author ncottin
 */
public final class NbVotesBean {

    private int votesCount = 0;

    public int getVotes() {
        return votesCount;
    }

    public void setCandidat(String candidat) {
        votesCount++;
    }
}
```

6.4.5. Page « index.jsp »

Remplacer lecde source généré par défaut lors de la création du projet par celui-ci :

```
<%@include file="include/header.inc" %>
<h1>Votez pour votre favori</h1>
<form action="vote.jsp" method="POST">
    <%
```

```
    for (String cand : sondage.bean.CandidatsBean.candidats) {
    %>
            <input type="radio" name="candidat" value="<%= cand %>"
checked="checked" />
        <%= cand %><br/>
    <%
    }
    %>
    <input type="submit" value="Voter" />
</form>
<br/>
<a href="resultats.jsp">Consulter les résultats</a>
<%@include file="include/footer.inc" %>
```

6.4.6. Page « vote.sp »

Cette page est appelée lors de la soumission du formulaire par la page d'index.

```
<jsp:useBean id="candidatsBean"
  scope="application" class="sondage.bean.CandidatsBean" />
<jsp:useBean id="voteBean"
  scope="request" class="sondage.bean.VoteBean" />
<jsp:setProperty name="candidatsBean" property="candidat" />
<jsp:setProperty name="voteBean" property="candidat" />

<%@include file="include/header.inc" %>
<h1>Merci d'avoir voté</h1>
Vous avez voté pour
  <jsp:getProperty name="voteBean" property="candidat" />.
<br/>
<a href="resultats.jsp">Consulter les résultats</a>
<a href="index.jsp">Voter</a>
<%@include file="include/footer.inc" %>
```

6.4.7. Page « resultats.jsp »

Cette page, indépendante, comptabilise le nombre de votes our chaque candidat.

```
<jsp:useBean id="candidatsBean"
  scope="application" class="sondage.bean.CandidatsBean" />

<%@include file="include/header.inc" %>
<h1>Résultats des votes</h1>
<table border="1" cellpadding="2">
    <thead>
        <tr>
            <th>Candidat</th>
            <th>Votes</th>
```

```
        </tr>
    </thead>
    <tbody>
        <%
        for (int i=0; i<candidatsBean.candidats.length; i++) {
        %>
        <tr>
            <td><%= candidatsBean.candidats[i] %></td>
            <td><%= candidatsBean.getVotes(i) %></td>
        </tr>
        <%
        }
        %>
    </tbody>
</table>
<br/>
<a href="resultats.jsp">Actualiser les résultats</a>
<a href="index.jsp">Voter</a>
<%@include file="include/footer.inc" %>
```

6.5. Résumé des directives JSP

Les directives JSP s'écrivent sous l'une des deux formes suivantes:

```
<%@nom_directive attributs %>
<jsp:nom_directive attributs />
```

Elles sont généralement placées en en-tête des sources JSP et disposées de telle sorte qu'une ligne ne concerne qu'une seule directive. Une dérogation à cette règle est cependant permise dans l'un des deux cas suivants :

- Le code HTML demande une en-tête XML en tant que première ligne du flux renvoyé par la page (une indication de conformité à la recommandation XHTML Strict du W3C par exemple)
- Le type MIME de la page générée est autre que HTML.

Dans ces deux cas, le problème est lié à l'insertion (intempestive) de retours à la ligne (considérés comme des espaces en HTML) lorsque les directives JSP sont rédigées sur les lignes différentes.

6.5.1. Directive « page »

La directive de type « page » fournit des informations relatives à la page JSP courante. Elle s'écrit sous la forme suivante :

```
<%@page nom_attribut="valeur_attribut" %>
```

Les différents attributs pouvant être déclarés sont répertoriés dans le tableau ci-après.

Attribut	Définition	Exemple	
autoFlush	Permet de vider le buffer automatiquement (par défaut)	true	false
buffer	Taille du buffer (8Kb par défaut)	4096kb	none
contentType	Type MIME et jeu de caractères du contenu transmis (par défaut, text/html;charset=ISO-8859-1)	application/pdf	
errorPage	Page d'erreur sur laquelle est renvoyée le demandeur en cas de génération d'une exception	error.jsp	
extends	Implémente l'interface du paquetage choisi	pkg.classe	
import	Liste les paquetages et classes à importer	pkg1.*, pkg2.classe	
info	Information ajoutée dans le fichier compilé	Description de la page	
isErrorPage	Indique si la page JSP est une page d'erreur (faux par défaut)	true	false
isThreadSafe	Indique au serveur si la page JSP supporte ou non l'accès concurrent (vrai par défaut)	true	false
language	Language utilisé (Java par défaut)	Java[5]	
pageEncoding	Jeu de caractères à employer (par défaut, ISO-8859-1)	UTF-8	
session	Accorde l'utilisation des sessions (vrai par défaut; faux indique que les valeurs de session sont inaccessibles)	true	false

6.5.2. Directive « include »

Cette directive permet d'inclure un fichier local au sein de la page JSP. Cette inclusion s'opère préalablement à la phase de traduction de la page

5 Peut-il s'agir d'un autre langage que Java ?

JSP de sorte que le fichier inclus peut contenir du code[6] dynamique en Java.

Cette directive est généralement employée afin d'inclure une en-tête et un pied de page communs à l'ensemble des pages composant une application.

L'inclusion de fichier s'écrit sous la forme suivante :

```
<%@include file="chemin_fichier" %>
```

Le chemin indiqué peut être absolu (déconseillé) ou relatif. Les fichiers inclus portent généralement l'extension « jsp », « htm », « html », ou « inc ».

6.5.3. Autres directives

La directive « taglib » permet de définir des balises personnalisées dans des bibliothèques de balises :

```
<%@taglib uri="tagLibrairie" prefix="tagPrefix" %>
```

La commande « forward » permet de rediriger la page courante (JSP) vers une seconde page tout en conservant les informations de session collectées jusqu'alors (par les beans notamment) :

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

Ou alternativement (en mettant à jour la valeur d'un paramètre) :

```
<jsp:forward page="autre_page.jsp" />
  <jsp:param name="paramName" value="paramValue" />
</jsp:forward>
```

6 Certains serveurs d'applications ne permettent pas l'inclusion en cascade de fichiers (un fichier inclus mentionnant l'inclusion d'un autre fichier).

7. Les services web

7.1. Rappels sur les services web

Un service web est un composant distribué autonome, sans état[7], interrogeable à distance via Internet. Les communications entre un service web et ses clients sont réalisées à l'aide du protocole HTTP et peuvent utiliser (entre autres) le format XML SOAP.

A COMPLETER...

7.2. Création d'un service web simple

Cette section propose de concevoir un service web dont le rôle est de déterminer l'âge d'une personne à l'aide de sa date de naissance (et de l'horloge du système). Les concepts sont similaires au tutoriel de NetBeans relatif aux services web[8].

7.2.1. Définition de l'interface

Ce service propose une opération baptisée « `calculAge` ». Sa signature est la suivante :

```
long calculAge(String dateNaissance)
```

Pour des raisons de simplification, cette opération ne lève aucune erreur relative au paramètre (date non valide, postérieure à la date courante, etc.).

La date de naissance est ici supposée au format français `JJ/MM/AAAA`.

7.2.2. Déclaration du service web

Créer un projet de type « Web Application » sans indiquer de framework et en spécifiant l'emploi du serveur d'applications Glassfish V2 (la version 3 ne permet pas de déployer correctement les services web). Pour la suite, ce projet sera nommé « IdentiteWebService ».

7 On utilise souvent le terme anglais « stateless ».

8 Il s'agit du service « Calculator » dont le rôle est le réaliser l'addition de deux entiers donnés.

Créer le service web en procédant comme suit :

1. Opérer un clic droit sur le projet
2. Sélectionner « New » pis « Other... »
3. Choisir « Web Service » dans le dossier « Web Services »
4. Cliquer sur « Suivant »
5. Nommer le service web en « IdentiteWebService » et spécifier un paquetage (tel que « identite » par exemple)
6. Terminer la procédure de création du service web

Le code source suivante est alors généré :

```
package identite;

import javax.jws.WebService;

/**
 * @author ncottin
 */
@WebService()
public class IdentiteWebService {

}
```

Ce dernier ne compile pas car NetBeans requiert que ce service propose au moins une opération publique.

7.2.3. Ajout de l'opération

L'opération de calcul d'âge est ajoutée au service web en procédant à un clic droit au sein du code source et en sélectionnant « Web Service » puis « Add Operation... » et enfin, dans la boîte qui s'affiche alors :

1. Attribuer « `calculAge` » au nom de cette opération
2. Changer le tye de retour en « `long` »
3. Ajouter le paramètre « `dateNaissance` » de type « `String` »
4. Terminer la définition de la signature de cette opération

La déclaration finale est alors :

```
package identite;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 * @author ncottin
 */
@WebService()
public class IdentiteWebService {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "operation")
    public long operation(@WebParam(name = "dateNaissance")
      String dateNaissance) {
        //TODO write your implementation code here:
        DateFormat fd = new SimpleDateFormat("dd/MM/yyyy");
        try {
           Date naiss = fd.parse(dateNaissance);
           Calendar cal = GregorianCalendar.getInstance();
           return cal.getTimeInMillis() - naiss.getTime();
        } catch(Exception e) {
           return -1;
        }
    }
}
```

7.3. Appel d'un service web

Quelle que soit la méthode employée, l'appel à un service web via NetBeans suppose que le projet déclarant ce service est préalablement correctement déployé. Dans le cas contraire, une erreur HTTP 404 est générée par le navigateur web (ou par la portion de code chargée d'appeler le service web).

7.3.1. Test d'un service web

La première phase consiste à déployer le projet à l'aide d'un clic droit sur le nom du projet puis « Deploy ».

Lorsque cette étape est achevée, opérer un clic droit sur le service web dans la section intitulée « Web Services » (et non « Source Packages ») puis sélectionner « Test Web Service ».

Le navigateur web défini par défaut est alors démarré et une page contenant la liste des opérations déclarées par le service web est affichée. Il suffit alors de remplir correctement les champs et de cliquer sur le bouton nommant l'opération souhaitée pour réaliser l'appel du service web.

La page résultante propose le résultat de l'appel de cette opération ainsi que le détail des informations transmises (requête et réponse SOAP par exemple).

7.3.2. Appel au sein d'une page JSP

Tout en conservant déployé le projet contenant le service web, créer un projet JSP. Un clic droit sur le nom du projet permet de sélectionner « New » puis « Web Service Client ». Dans la boîte de dialogue, sélectionner le projet contenant le service web à l'aide du bouton « Browse » située sur la ligne relative aux projets (cette ligne est sélectionnée par défaut). Choisir un projet puis le service web à utiliser. Appuyer enfin sur « Terminer ».

Le service web[9] est alors accessible depuis le dossier « Web Services References » du projet JSP. La technique consiste alors à sélectionner une opération du service web (en déroulant les divers éléments du service web) puis à la glisser-déposer au sein de la page JSP (par exemple sous le titre de niveau 1). Le code d'appel est automatiquement généré par NetBeans. La page JSP ressemble alors à la page suivante :

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
      <meta  http-equiv="Content-Type"
             content="text/html; charset=UTF-8">
```

9 Il s'agit plus exactement d'un proxy client sur le service web...

```
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Appel d'un service web</h1>
    <%-- start web service invocation --%><hr/>
    <%
    try {
        calculator.CalculatorWebServiceService service =
          new calculator.CalculatorWebServiceService();
        calculator.CalculatorWebService port =
          service.getCalculatorWebServicePort();
         // TODO initialize WS operation arguments here
        int arg1 = 0;
        int arg2 = 0;
        // TODO process result here
        int result = port.add(arg1, arg2);
        out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
    %>
    <%-- end web service invocation --%><hr/>
    </body>
</html>
```

Il suffit alors de modifier (« en dur » pour le moment) les valeurs des variables « arg1 » et « arg2 » pour obtenir le résultat de leur addition.

7.3.3. Appel au sein d'un bean

Le modèle MVC implique la séparation entre présentation et traitements. De fait, l'appel d'un service web directement au sein d'une page JSP n'est pas recommandé. On préférera ajouter une fonctionnalité à un bean dont le rôle sera de réaliser l'appel au service web.

Pour ce faire, créer un projet JSP puis

```
ceci
est
du
code
```

Texte

texte

texte

Ceci est une note. Ceci est une note. Ceci est une note. Ceci est une note. Ceci est une note. Ceci est une note. Ceci est une note. Ceci est une note. Ceci est une note.

Ceci est un message important. Ceci est un message important. Ceci est un message important. Ceci est un message important. Ceci est un message important. Ceci est un message important.

gcz

Ceci est un théorème. Ceci est un théorème. Ceci est un théorème. Ceci est un théorème. Ceci est un théorème. Ceci est un théorème. Ceci est un théorème. Ceci est un théorème.

cg

zvzcg