JSP - Introduction

JSP, Java Server Pages et Servlet – Introduction

1-Rappels modèles de développement Client/Serveur

Le modèle de développement d'applications Client-Serveur (C/S) permet de construire des applications légères faisant appel à des services proposés par des fournisseurs (serveurs) contrairement aux applications monolithiques où tout est réalisé dans un même programme sur le poste client..

On peut distinguer 2 grandes formes d'applications Client-Serveur :

- **Client lourd** : l'application sur le poste client gère l'interface avec l'utilisateur, des traitements, et fait appel au serveur pour interroger les données d'une base de données, par exemple.
- **Client léger** : l'application (en général un navigateur Web) sur le poste client gère l'interface utilisateur, et fait appel au serveur pour gérer les traitements et accès aux données

Ces nouvelles architectures font appel à des services réalisés par des programmes tiers, on parle d'architecture :

- 3 tiers : client (interface), serveur Web (traitement), serveur de base de données (données)
- n tiers : client (interface), serveur Web (traitement), serveur d'applications (moteur de servlet, serveur d'EJB, etc.), serveur de base de données (données) ou autres sources provenant d'autres applications ou ordinateurs.

En pratique les distinctions ne sont pas toujours aussi radicales, et la répartition entre client et serveur peut être un peu différente.
De nouvelles technologies ont redonné une place à la réalisation de certains traitements sur le client (exemple AJAX avec le développement de code Javascript sur le client et l'appel de ressources serveur via le protocole HTTP)

2-Les différents outils du client léger : Javascript / CGI, ASP, PHP, JSP/Servlet, etc.

Après le succès du Web avec les pages statiques en HTML, différentes technologies ont été développées afin d'en enrichir le contenu et arriver au web dynamique.

A- Exécution sur le poste client

- Langage de script : Javascript, VBScript,
- Objets : ActiveX, applets Java (nécessite la présence d'une machine virtuelle Java – JVM)

B- Exécution sur le serveur

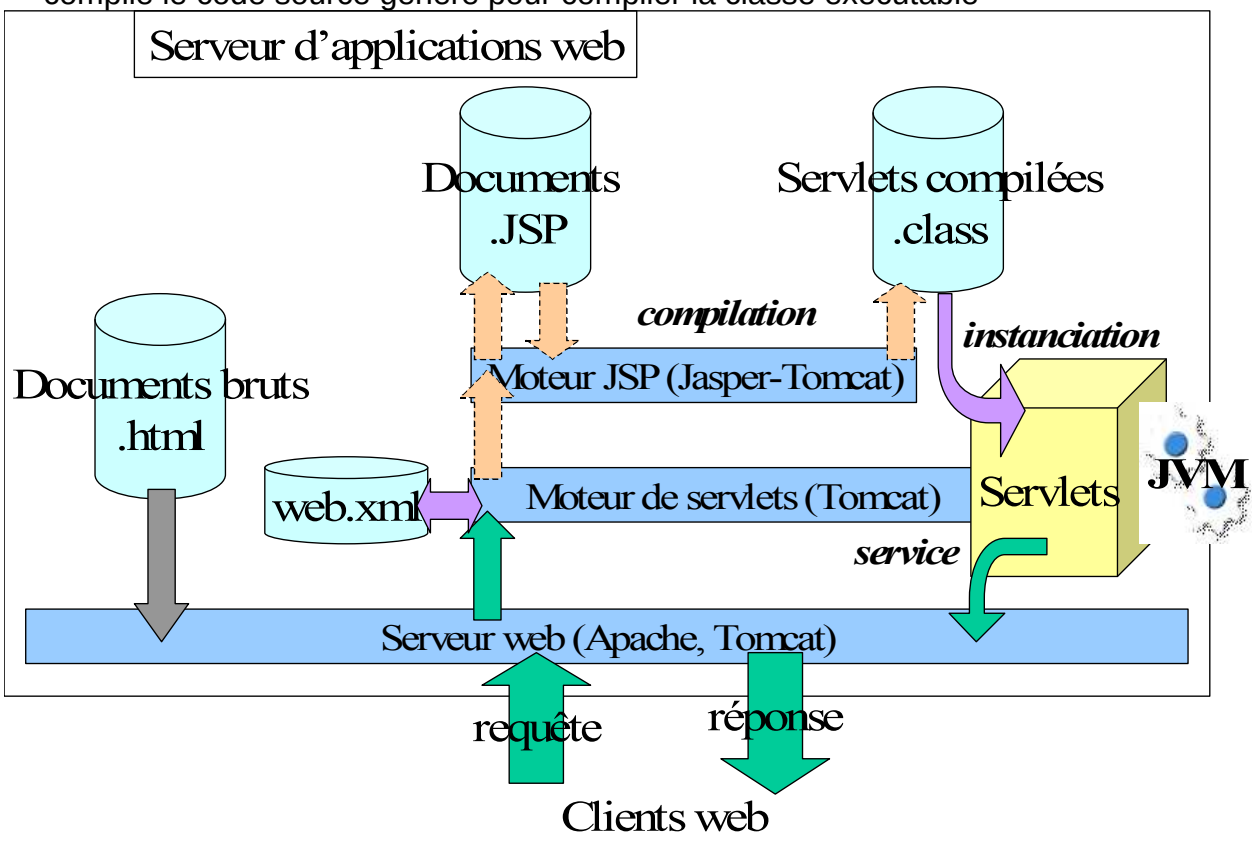
- CGI : Common Gateway Interface (Perl, Python, C++, Pascal Delphi, ...)
- ASP : Active Server Pages, associé au serveur HTTP Microsoft IIS
- PHP : associé le plus souvent au serveur HTTP Apache
- JSP/Servlet : Java Server Pages, associé à Tomcat (moteur de servlet, en lien avec la JVM)

JSP - Introduction

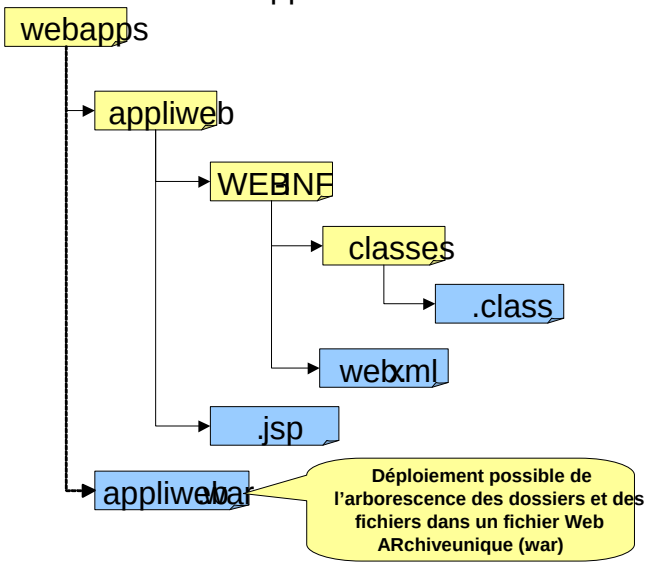
3- JSP et Servlet - Environnement d'exécution Tomcat

Le framework Java (J2EE) offre des classes permettant de développer des applications Web en Java. La **servlet** est la brique élémentaire permettant de construire des sites dans ce langage. Une application particulière, le moteur de servlet, va être chargé d'assurer l'exécution de ces servlets pour répondre aux requêtes transmises par le serveur web.
Le moteur de servlet (par exemple TOMCAT) :

- permet d'une part l'exécution des servlets Java (.class) en lien avec la JVM ; utilise le multithread pour instancier autant de servlets que nécessaires (en fonction de la charge)
- associé à l'interpréteur Jasper qui traduit le langage JSP en source servlet Java (.java), il compile le code source généré pour compiler la classe exécutable



Les serveurs d'applications Web nécessitent une définition précise de l'arborescence de répertoire de déploiement des fichiers de l'application.



JSP - Introduction

4- Servlets

La classe HttpServlet du framework Java permet le traitement des requêtes HTTP. Son exécution nécessite sa prise en charge par un moteur de servlet.

Cycle de vie d'une servlet

Une fois que la servlet a été déployée dans l'arborescence, le moteur de servlet la prend en charge :

- Si aucune instance de la servlet n'existe :
  - Charge la classe
  - Crée une instance de cette classe (un objet)
  - Appelle ma méthode **INIT** de la servlet (celle-ci peut être utilisée pour initialiser certains paramètres, etc.)
- Il invoque la méthode qui va 'servir' la requête (doXXX, où XXX = Get, Post) ; la fonction d'une méthode de service est d'extraire des informations de la requête, d'accéder à des ressources externes pour effectuer un traitement puis de constituer un message de réponse basé sur des informations récupées(préparer l'ent^te http de réponse, écrire le flux)
- Lorsque le conteneur doit décharger une servlet, il invoque sa méthode **DESTROY**

Exemples d'application :

A- Helloworld

Objectif : afficher le texte 'hello world' sur une page web

```
/**
  * Servlet Helloworld
  *
  * Exemple de servlet permettant l'affichage du texte helloworld
  *
  * @author P.Dezécache
  * @since dec 2006
  * @version 1
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

/** implémentation de la méthode doGet
 *
 * @param HttpServletRequest request : requête HTTP
 * @param HttpServletResponse response: reponse HTTP
 */
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

      /**
        * définir le type de contenu du flux de sortie (ce qu'on va renvoyer )
      */
```

Les servlets héritent de la classe **HttpServlet**

Le type de contenu renvoyé (type MIME) est défini ici

```
      response.setContentType("text/html");

      /**
        * définir le flux de sortie associé à la reponse
      */
      PrintWriter out = response.getWriter();

      /**
        * écrire un texte dans le flux de sortie
      */
      out.println("<h1>Hello World</h1>");

      /**
        * vider le flux de sortie (envoi vers le navigateur)
      */
      out.flush();

      /**
        * fermer le flux de sortie
      */
      out.close();

  } // fin doGet

/**
 * implémentation de la méthode doPost
 * l'appel de la méthode doGet permet le traitement identique quelque soit
 * la méthode d'appel de la page
 *
 * @param HttpServletRequest request : requête HTTP
 * @param HttpServletResponse response: reponse HTTP
 */
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {

      /**
        * appeler la méthode doGet
        *
      */
      doGet(request, response);

  } // fin doPost


} // fin classe HelloWorld
```

L'écriture dans le flux de sortie (html, xml, etc.):
_println (avec saut de ligne)
_print (sans saut de ligne)

Fermeture du flux de sortie (libération des ressources)

Définition de la servlet dans le fichier WEB-INF\web.xml :

1. Définition d'un nom de servlet associé à une classe :

```
<servlet>
      <servlet-name>
          HelloWorldName
      </servlet-name>
      <servlet-class>
          HelloWorld
      </servlet-class>
</servlet>
```

2. Définition du mapping entre le nom de la servlet et l'URL d'appel :

```
    <servlet-mapping>
      <servlet-name>HelloWorldName</servlet-name>
      <url-pattern>/HelloWorldUrl</url-pattern>
    </servlet-mapping>
```

B- Login et Recup

Objectif : afficher un formulaire de saisie d'un code utilisateur et d'un mot de passe et envoi à un servlet qui va récupérer ces valeurs et les afficher

```
/**
  * Servlet Login
  *
  * Servlet permettant l'affichage d'un formulaire de login
  *
  * @author P.Dezécache
  * @since dec 2006
  * @version 1
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Login extends HttpServlet {

/*
      Implementation de la méthode doGET
*/
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      out.println("<form method=\"get\" action=\"VerifUrl\">");
      out.println("<input type=\"text\" name=\"user\"/>");
      out.println("<input type=\"password\" name=\"pass\" />");
      out.println("<input type=\"submit\" name=\"btn\" />");
      out.println("</form>");

      out.flush();

  } // fin doGet

/*
      Implementation de la méthode doPost
*/
  public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);

  } // fin doPost


} // fin classe Login
```

```
/**
  * Servlet Verif
  *
```

```
  * Servlet permettant la recupération de paramètres provenant d'un formulaire POST
ou GET
  * et affichage de ces paramètres
  *
  * @author P.Dezécache
  * @since dec 2006
  * @version 1
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Verif extends HttpServlet {

/*
      Implementation de la méthode doGet
*/
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

      response.setContentType("text/html");

      PrintWriter out = response.getWriter();

      out.println("<h1>OK</h1>");
      out.println("<h2>utilisateur = "+request.getParameter("util")+"</h2>" );
      out.println("<h2>password = "+request.getParameter("pass")+"</h2>" );

      out.flush();

  } // fin doGet

/*
      Implementation de la méthode doPost
*/
  public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);

  } // fin doPost

} // fin classe Verif
```

Cette technique d'écriture est lourde : l'insertion de balises HTML de présentation au sein d'instruction de traitement ne permet pas de faire évoluer simplement l'application Web.
Le langage JSP va nous apporter une simplification d'écriture pour la partie présentation.

5- JSP, Java Server Pages

Le langage JSP va permettre une écriture plus lisible en séparant les balises HTML du code Java au travers de blocs de scripts Java balisés, les scriplets :

- des **déclarations** permettant de définir des paramètres généraux relatifs à la page
- des **instructions Java** permettant le codage du traitement lui-même

- des **expressions** permettant l'affichage de contenu de variables afin de rendre le contenu dynamique

6- Taglib, librairie de balises pour JSP et langage EL

Afin d'apporter encore plus de cohérence au langage, une nouvelle norme de balisage pour les développement Web avec Java est apparue : **J**avaServerPages **S**tandard **T**ag **L**ibrary.
Elle définit un certain nombre de balises permettant la suppression du code Java dans les pages web au profit de balises :

- Fourniture de structures algorithmiques standard
- Traitement XML
- L'accès aux bases de données

7- Les Beans, classes Java réutilisables

Les beans sont des classes d'objets Java possédant certaines caractéristiques et permettant la simplification de l'écriture des applications en général. (à developper)

8- Du codage à l'utilisation de Frameworks d'applications

Le développement Web avec Java offre plusieurs possibilités d'écriture : il s'agira de tirer profit de l'intérêt de chacun des outils offerts :

- La servlet : permet de coder des traitement complexes d'une manière performante
- La JSP : permet de définir simplement les interactions avec l'utilisateur
- Les beans : capitalisation

Cependant, de nouvelles manières d'écrire des applications utilisent les **framework d'application**, modèles de structure d'applications, cadres d'application, qui simplifient les développements, les rendent plus sûrs et accroissent leur productivité.

JSP - Introduction

Sujet de TP