

Java Server Page Servlet

Objectif !

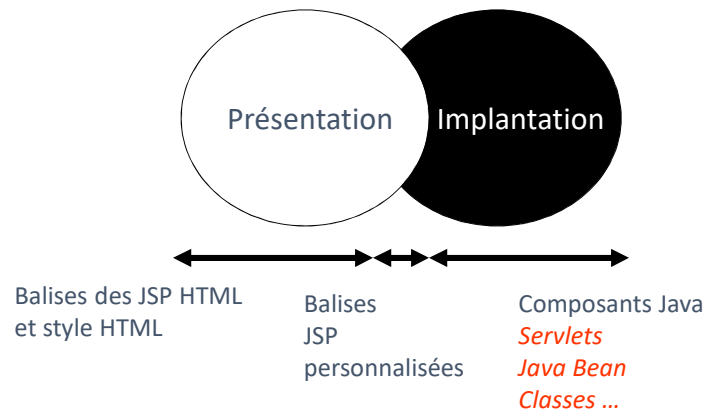
- Interaction avec l'utilisateur final.
- Connexions avec des Bases de données et d'autres applications.

Solution!

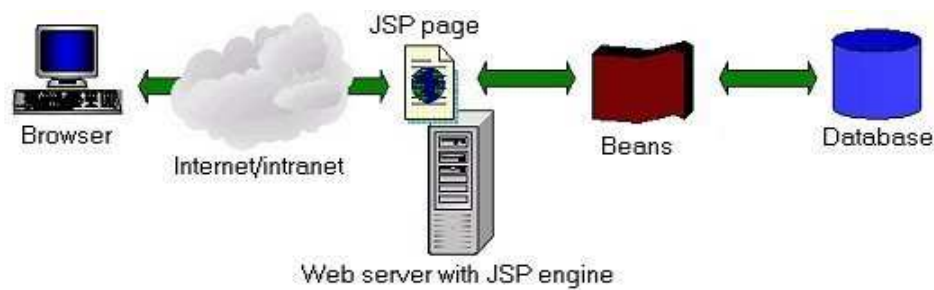
- JSP (Java Server Page).
- Technologie Servlet.

Solution

- Séparation de la présentation et de l'implémentation



Présentation



Qu'est ce qu'une Servlet ?

1. Une servlet est un programme qui s'exécute côté serveur.
2. Elle reçoit une requête du client, elle effectue des traitements et renvoie le résultat.
3. La liaison entre la servlet et le client peut être directe ou passer par un intermédiaire comme par exemple un serveur http.

Qu'est ce qu'une Servlet ?

- Les servlets Java :
 - Une technologie pour générer des pages web dynamiques (comme PHP, ASP, ASP.NET, ...)
 - Tournent dans des serveurs dédiés dits « serveurs de servlets ».

Pourquoi utiliser des Servlets ?

- Java
 - portable... c'est au coeur de la partie HTTP de Java EE
- Puissance
 - Utilisation de toutes les APIs Java.
 - Puissant mécanisme d'annotations.
- Efficace
 - Code compilé
 - Jusqu'à 100x plus rapide que PHP, même que C++.
<http://blog.dhananjaynene.com/2008/07/performance-comparison-c-java-python-ruby-jython-jruby-groovy/>

Pourquoi utiliser des Servlets (2)

- Sécurité
 - Typage fort,
 - Gestion de la mémoire efficace.
- Intégration forte avec le serveur
 - Via des variables "contexte" échanges forts entre le serveur et les servlets : configuration, partage de ressources (connexions BD) etc.
- Extensibilité & Flexibilité
 - Puissants mécanismes de "filtres" et de "chainage" de traitements.

Date Servlet – Exemple

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

public class DateServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println( "<HTML>" );
        out.println( "<HEAD>" );
        out.println( "<TITLE>Page generee par une servlet</TITLE>" );
        out.println( "</HEAD>" );
        out.println( "<BODY>" );
        out.println( "<H1>Bonjour</H1>" );
        out.println("The time is:"+ new java.util.Date());
        out.println( "</BODY>" );
        out.println( "</HTML>" );
    }
}
```

L'API servlet

L'API servlet regroupe un ensemble de classes dans deux packages :

1. **javax.servlet** : contient les classes pour développer des servlets génériques indépendantes d'un protocole.
2. **javax.servlet.http** : contient les classes pour développer des servlets qui reposent sur le protocole http utilisé par les serveurs web.

L'API servlet

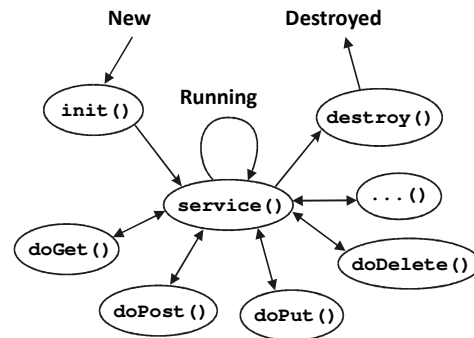
1. L'usage principal des servlets est la création de pages HTML dynamiques.
2. L'API fournit une classe qui encapsule une servlet utilisant le protocole http. Cette classe est la classe `HttpServlet`.
3. Ce type de servlet n'est pas utile seulement pour générer des pages HTML bien que cela soit son principal usage, elle peut aussi réaliser un ensemble de traitements tels que mettre à jour une base de données.

L'API servlet

La méthode `service()` héritée de `HttpServlet` appelle l'une ou l'autre de ces méthodes en fonction du type de la requête http :

1. une requête GET : c'est une requête qui permet au client de demander une ressource.
2. une requête POST : c'est une requête qui permet au client d'envoyer des informations issues par exemple d'un formulaire.

L'API servlet



- On implémente souvent la méthode `init()` (invoquée lors de la première exécution de la servlet) pour récupérer des ressources ou paramètres d'initialisation.

La méthode `init()`

- Appelée à la première instanciation,
- La spécification garantit qu'aucune requête ne sera envoyée avant que `init()` soit exécutée entièrement.
- On redéfinit `init()` quand :
 - On doit ouvrir des ressources (connexions JDBC par ex)
 - On doit initialiser l'état d'une Servlet,

Exemple de traitement de paramètre : Hello Servlet

- Servlet qui récupère le paramètre "nom" et qui répond : "Hello, <nom>"
- Formulaire HTML :

```
<form method="GET ou POST" action="URL relatif de
  la Servlet">
  <input type="text" name="nom">
</form>
```

- Récupération du paramètre par la méthode doGet() ou doPost() :

```
String nom= request.getParameter("nom");
```

Hello Servlet : exemple complet

HelloForm.html

```
<html><body>
<form method="GET" action="HelloServlet">
  Entrez votre nom :
  <input type="text" name="nom">
  <input type="submit" value="OK">
</form>
</body></html>
```

HelloServlet.java

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

public class HelloServlet extends HttpServlet {
```

Hello Servlet – Example

HelloServlet.java

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {

    response.setContentType("text/html");
    ServletOutputStream out = response.getOutputStream();
    String nom= request.getParameter("nom");
    out.println("<html><head>");
    out.println("\t<title>Hello Servlet</title>");
    out.println("</head><body>");
    out.println("\t<h1>Hello, " + nom + "</h1>");
    out.println("</body></html>");

}
```

Présentation de JSP

Présentation

- JSP : Standard Java permettant de développer des Applications Web interactives
- Intégrable au sein d'une page HTML
- Langage script exécuté du côté serveur
- Répartition des tâches de développement

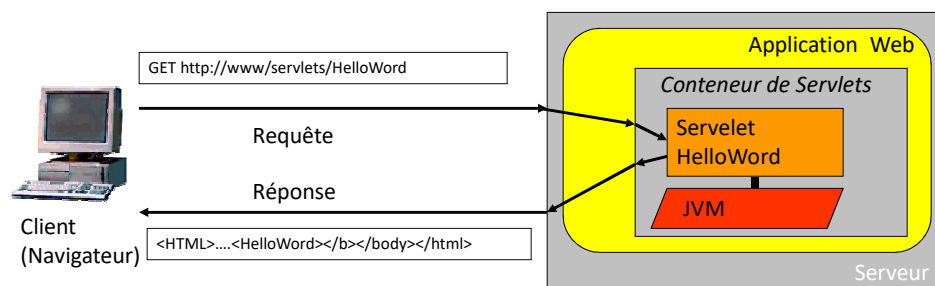
1. Il est préférable d'utiliser les JSP pour générer des pages web dynamiques.
2. L'usage des servlets est obligatoire si celles-ci doivent communiquer directement avec une applet ou une application et non plus avec un serveur web.

JSP Exécution

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

JSP Exécution

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



JSP Présentation

■ Exemple de code dans une page HTML

```
<html>
<body>
<% String visitor = request.getParameter("name");
    if (visitor == null) visitor = " World"; %>
    Hello, <%= visitor %>!
</body>
</html>
```

■ Affichage

"Hello, World !" pour une requête <http://serveur.fr/helloScript.jsp>

"Hello, Java !" pour une requête <http://serveur.fr/helloScript.jsp?name=Java>

Les tags de directives <%@ ... %>

Les directives permettent de préciser des informations globales sur la page JSP. Les spécifications des JSP définissent trois directives :

1. page : permet de définir des options de configuration.
2. include : permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet
3. taglib : permet de définir des tags personnalisés.

Leur syntaxe est la suivante :

```
<%@ directive attribut="valeur" ... %>
```

La directive page

- Cette directive doit être utilisée dans toutes les pages JSP, elle permet de définir des options qui s'appliquent à toute la JSP.
- Elle peut être placée n'importe où dans le source mais il est préférable de la mettre en début de fichier, avant même le tag <HTML>.
- Elle peut être utilisée plusieurs fois dans une même page mais elle ne doit définir la valeur d'une option qu'une seule fois, sauf pour l'option import.

```
1.<%@ page import="java.util.*" %>  
2.<%@ page import="java.util.Vector" %>  
3.<%@ page info="Ma premiere JSP"%>
```

La directive include

Cette directive permet d'inclure un fichier dans le code source JSP. Le fichier inclus peut être un fragment de code JSP, HTML ou Java.

Le fichier est inclus dans la JSP avant que celle-ci ne soit interprétée par le moteur de JSP.

Ce tag est particulièrement utile pour insérer un élément commun à plusieurs pages tel qu'un en-tête ou un bas de page.

La syntaxe est la suivante :

```
<%@ include file="chemin relatif du fichier" %>
```

Si le chemin commence par un '/', alors le chemin est relatif au contexte de l'application, sinon il est relatif au fichier JSP.

```
bonjour.html :
<p>
<table border="1" cellpadding="4" cellspacing="0" width="30%" align
=center >
<tr bgcolor="#A6A5C2">
<td align="center">BONJOUR</td>
</tr>
</table></p>
```

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<p align="center">Test d'inclusion d'un fichier dans la JSP</p>
<%@ include file="bonjour.html"%>
<p align="center">fin</p>
</BODY>
</HTML>
```

Le tag de déclarations <%! ... %>

Ce tag permet de déclarer des variables ou des méthodes qui pourront être utilisées dans la JSP. Il ne génère aucun caractère dans le fichier HTML de sortie.

La syntaxe est la suivante : <%! declarations %>

```
01.<HTML>
02.<HEAD>
03.<TITLE>Test</TITLE>
04.</HEAD>
05.<BODY>
06.<%!
07.int minimum(int val1, int val2) {
08.if (val1 < val2) return val1;
09.else return val2;
10.}
11.%>
12.<% int petit = minimum(5,3);%>
13.<p>Le plus petit de 5 et 3 est <%= petit %></p>
14.</BODY>
15.</HTML>
```

Le tag d'expressions <%= ... %>

Le moteur de JSP remplace ce tag par le résultat de l'évaluation de l'expression présente dans le tag.

Ce résultat est toujours converti en une chaîne. Ce tag est un raccourci pour éviter de faire appel à la méthode `println()` lors de l'insertion de données dynamiques dans le fichier HTML.

La syntaxe est la suivante : <%= expression %>

```
01.<%@ page import="java.util.*" %>
02.<HTML>
03.<HEAD>
04.<TITLE>Essai de page JSP</TITLE>
05.</HEAD>
06.<BODY>
07.<p align="center">Date du jour :
08.<%= new Date() %>
09.</p>
10.</BODY>
11.</HTML>
```

Les commentaires HTML <!-- ... -->

Ces commentaires sont ceux définis par le format HTML. Ils sont intégralement reproduits dans le fichier HTML généré. Il est possible d'insérer, dans ce tag, un tag JSP de type expression qui sera exécuté.

La syntaxe est la suivante :

<!-- commentaires [<%= expression %>] -->

```
01.<%@ page import="java.util.*" %>
02.<HTML>
03.<HEAD>
04.<TITLE>Essai de page JSP</TITLE>
05.</HEAD>
06.<BODY>
07.<!-- Cette page a ete generee le <%= new
Date() %> -->
08.<p>Bonjour</p>
09.</BODY>
10.</HTML>
```

Le tag <jsp:useBean>: Le tag <jsp:useBean> permet de localiser une instance ou d'instancier un bean pour l'utiliser dans la JSP.

```
01.<html>
02.<HEAD>
03.<TITLE>Essai
d'instanciation d'un bean
dans une JSP</TITLE>
04.</HEAD>
05.<body>
06.<p>Test d'utilisation d'un
Bean dans une JSP </p>
07.<jsp:useBean id="personne"
scope="request" class="test.P
ersonne" />
08.<p>nom initial =
<%=personne.getNom() %></p>
09.<p>nom mis à jour = <%=
personne.getNom() %></p>
11.</body>
12.</html>
```

```
01.package test;
02.public class Personne {
03.private String nom;
04.private String prenom;
06.public Personne() {
07.this.nom = "nom par default";
08.this.prenom = "prenom par default »;}
10.
11.public void setNom (String nom) {
12.this.nom = nom;}
14.
15.public String getNom() {
16.return (this.nom);}
18.public void setPrenom (String prenom)
{this.prenom = prenom; }
22.public String getPrenom () {
24.return (this.prenom);}
26.}
```

