

IFT 6800

Atelier en Technologies d'information

Chapitre 5 Introduction aux Servelets & JSP

Qu'est ce qu'une servlet

- Servlet: **Server**-side **applet**
- Une servlet est un programme Java utilisé pour étendre les fonctionnalités d'un serveur Web et pour accéder à des systèmes déjà existants.
- C'est une application :
 - côté serveur
 - Utilisée pour générer du contenu dynamique
 - Chargée dynamiquement quand elle est demandée.

Avantages / Inconvénients

- **Avantages**

- Indépendance vis à vis des plateformes.
- Modèle de sécurité hérité du serveur Web
- Support dans la plupart des serveurs Web
- Exploite toute l'API Java (API JDBC, protocoles, ...)

- **Les servlets vs CGI sont:**

- **Efficaces:** Les CGI entraînaient la création d'un processus pour chaque requête. Avec des servlets, chaque demande est manipulée par un thread Java.
De plus la servlet est chargée au démarrage du serveur ou lors de la première requête
- **Pratiques:** On utilise Java et on n'a pas besoin d'autres langages (Perl, shell, ...).

Avantages / Inconvénients

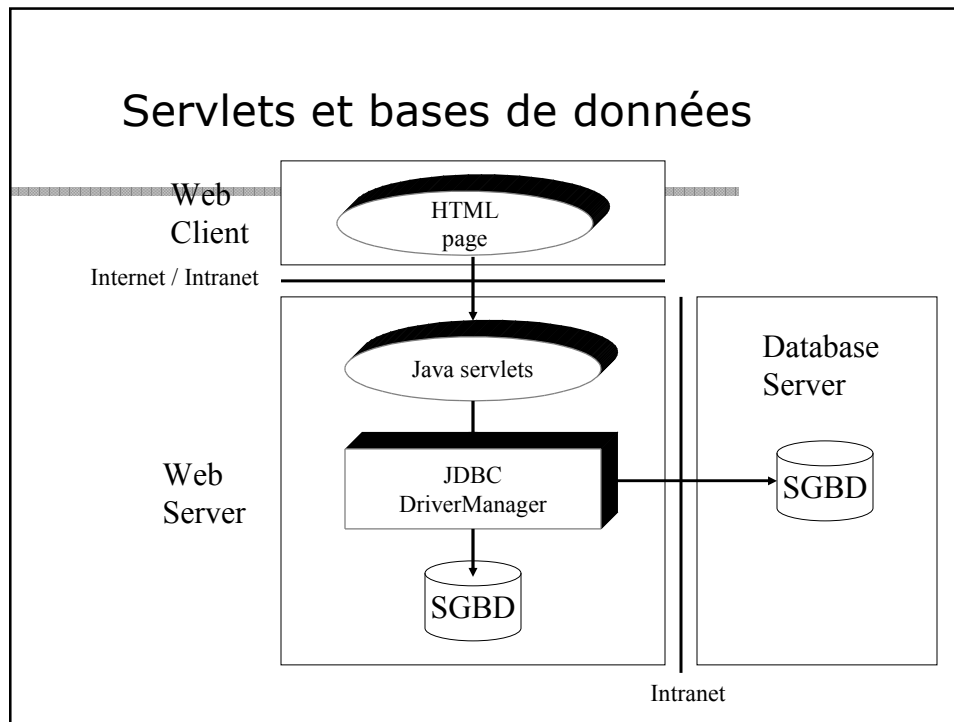
- **Puissantes:** Les servlets de Java permettent des fonctions que les CGI rendent difficiles (ex. parler à un serveur web).
- **Portables:** Par l'utilisation de Java.
- « Problèmes »
 - Apprendre java
 - Certains serveurs Web supportent mal la charge

Applets vs. servlets

- Les *servlets* sont le pendant des *applets* côté serveur
 - mais sans interface graphique utilisateur ...
 - elle est limitée à la puissance du langage HTML ...
 - par contre, elles ne sont pas astreintes aux mêmes règles de sécurité que les *applets*
 - peuvent établir une connexion avec d'autres clients (RMI, ...)
 - peuvent faire des appels système (utilisation pont JDBC-ODBC)
 - manipuler des ressources locales (sur le serveur), ...
 - l'approche *servlet* est donc plus sécurisée

Servlets et Serveurs Web

- Tous les serveurs Web n'intègrent pas la possibilité d'exécuter des *servlets*
 - la plupart nécessite un *patch* additionnel (cas de Apache)
- A noter 2 serveurs (http/servlets) gratuits et 100% Java :
 - JigSaw (W3C)
 - Java Web Server (JWS) de SUN



Fonctionnement

- Lorsqu'une servlet est appelée par un client, la méthode `service()` est exécutée. Celle-ci est le principal point d'entrée de toute servlet et accepte deux objets en paramètres:
 - l'objet *ServletRequest* encapsulant la requête du client, c'est-à-dire qu'il contient l'ensemble des paramètres passés à la servlet (informations sur l'environnement du client, cookies du client, URL demandée, ...)
 - l'objet *ServletResponse* permettant de renvoyer une réponse au client (envoyer des informations au navigateur). Il est ainsi possible de créer des en-têtes HTTP (headers), d'envoyer des cookies au navigateur du client, ...

Fonctionnement

- Afin de développer une servlet fonctionnant avec le protocole HTTP, il suffit de créer une classe étendant HttpServlet (qui implémente elle-même l'interface Servlet).
- La classe HttpServlet (dérivant de GenericServlet) permet de fournir une implémentation de l'interface Servlet spécifique à HTTP. La classe HttpServlet surcharge la méthode service en lisant la méthode HTTP utilisée par le client, puis en redirigeant la requête vers une méthode appropriée.

Développer une servlet

- Les deux principales méthodes du protocole HTTP étant GET et POST, il suffit de surcharger la méthode adéquate afin de traiter la requête; Ainsi donc:
 - Si la méthode utilisée est GET, il suffit de redéfinir la méthode public void doGet (HttpServletRequest req, HttpServletResponse res);
 - Si la méthode utilisée est POST, il suffit de redéfinir la méthode public void doPost(HttpServletRequest req, HttpServletResponse res);

Cycle de vie d'une servlet

- Le cycle de vie d'une servlet est assuré par le conteneur de servlets. Ainsi afin d'être à même de fournir la requête à la servlet, récupérer la réponse ou bien tout simplement démarrer/arrêter la servlet, cette dernière doit posséder
- Une interface déterminée par le JSDK afin de suivre le cycle de vie suivant :
 1. le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête
 2. La servlet est chargée au démarrage du serveur ou lors de la première requête

Cycle de vie d'une servlet (2)

La servlet est instanciée par le serveur

- La méthode `init()` est invoquée par le conteneur
- Lors de la première requête, le conteneur crée les objets `Request` et `Response` spécifiques à la requête
- La méthode `service()` est appelée à chaque requête dans une nouvelle thread. Les objets `Request` et `Response` lui sont passés en paramètre
- Grâce à l'objet `Request`, la méthode `service()` va pouvoir analyser les informations en provenance du client

Cycle de vie d'une servlet (3)

- Grâce à l'objet Response, la méthode service() va pouvoir fournir une réponse au client
- La méthode destroy() est appelée lors du déchargement de la servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La servlet est alors signalée au garbage collector.

Un exemple

- Voici un exemple simple de servlet dont le seul but est d'afficher du texte sur le navigateur du client :

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*
public class PremiereServlet extends HttpServlet {
public void init() {
    }
}
```

Exemple (suite 1)

```
public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE> Titre
    </TITLE></HEAD>");
    out.println("<BODY>");
    out.println("Ma première servlet");
    out.println("</BODY>");
```

Suite 2

- `out.println("</HTML>");`
- `out.close(); } }`
- La première étape consiste à importer les packages nécessaires à la création de la servlet, il faut donc importer `javax.servlet`, `javax.servlet.http` et `javax.io`

Suite 3

- Afin de mettre en place l'interface Servlet nécessaire au conteneur de servlet, la classe HttpServlet a été étendue

```
public class PremiereServlet extends  
HttpServlet {  
    }  
}
```

Lorsque la servlet est instanciée, il peut être intéressant d'effectuer des opérations qui seront utiles tout au long du cycle de vie de la servlet (par exemple se connecter à une base de données, ouvrir un fichier, ...).

Suite 4

- Pour ce faire, il s'agit de surcharger la méthode `init()` de la servlet et de définir les opérations d'initialisation.

```
public void init() {}
```
- A chaque requête, la méthode `service()` est invoquée. Celle-ci détermine le type de requête dont il s'agit, puis transmet la requête et la réponse à la méthode adéquate (`doGet()` ou `doPost()`). dans notre cas, on ne s'intéresse qu'à la méthode GET, c'est la raison pour laquelle la méthode `doGet()` a été surchargée

Suite 5

- `public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { }`
- L'objet `HttpServletRequest` permet de connaître les éventuels paramètres passés à la servlet (dans le cas d'un formulaire HTML par exemple), mais l'exemple ci-dessus n'en a pas l'utilité.
- Par contre l'objet `HttpServletResponse` permet de renvoyer une page à l'utilisateur.

Suite 5

- La première étape consiste à définir le type de données qui vont être envoyées au client. Généralement il s'agit d'une page HTML, la méthode `setContentType()` de l'objet `HttpServletResponse` doit donc prendre comme paramètre le type MIME associé au format HTML (`text/html`) :

```
res.setContentType("text/html");
```
- Ensuite la création d'un objet `PrintWriter` grâce à la méthode `getWriter()` de l'objet `HttpServletResponse`

Suite 6

- permet d'envoyer du texte formaté au navigateur (pour envoyer un flot de données, il faudrait utiliser la méthode `getOutputStream()`)

```
PrintWriter out = res.getWriter();
```

- Enfin il faut utiliser la méthode `println()` de l'objet `PrintWriter` afin d'envoyer les données textuelles au navigateur, puis fermer l'objet `PrintWriter` lorsqu'il n'est plus utile avec sa méthode `close()`

Suite 7

```
out.println("<HTML>");  
out.println("<HEAD><TITLE> Titre  
  </TITLE></HEAD>");  
out.println("<BODY>");  
out.println("Ma première servlet");  
out.println("</BODY>");  
out.println("</HTML>");  
out.close();
```

Une extension des servlets: Les Java Server Page (JSP)

JSP : Java Server Pages

- Extension de la technologie des Servlets créée pour aider à l'écriture de pages Web Elle
 - génère une page vers le client
 - est portable (Write Once, Run Everywhere)
 - met en avant l'approche par composants
 - permet la mise en œuvre facile des sites dynamiques
- Equivalents : ASP, PHP, PSP

Les JSP

- Séparent la présentation du contenu
- Une page JSP contient
 - moules (squelettes) contenant le texte fixe
 - Une partie qui génère les parties dynamiques.
- Permet la séparation du travail du concepteur de page de celui de développeur des fonctionnalités
 - Permet de se concentrer sur les fonctionnalités
 - Facilite les changements
 - Permet de développer plus vite (partage de tâches)
 - Les pages JSP peuvent accéder à des composants réutilisables (servlets, Java Beans).

Les JSP

- Les JSP sont intégrables au sein d'une page Web en HTML à l'aide de balises spéciales permettant au serveur Web de savoir que le code compris à l'intérieur de ces balises doit être interprété afin de renvoyer du code HTML au navigateur du client.
- Les Java Server Pages s'inscrivent dans une architecture 3-tier, c'est à dire qu'un serveur supportant les Java Server Pages peut servir d'intermédiaire (on parle de serveur applicatif) entre le navigateur du client et un serveur de données en permettant un accès transparent à celle-ci. JSP fournit ainsi les éléments nécessaires à la connexion au système de gestion de bases de données, à la manipulation des données grâce au langage SQL

Éléments des JSP

- JSP peut contenir en plus du code HTML quatre types d'éléments :
 - **des directives**: informations globales relatives à la page.
 - **des déclarations**: permettant de déclarer des méthodes et attributs.
 - **des scriptlets**: du code Java qui sera traduit en code dans la méthode service() de la servlet résultant.
 - **des expressions**: permettant d'envoyer facilement des chaînes créées dynamiquement vers le navigateur.

Les directives JSP

- Les directives JSP sont des instructions insérées dans des tags HTML spéciaux. La syntaxe des directives JSP est la suivante :

```
<%@ directive [attribut="valeur `` ] %>
```

Exemples:

```
<%@ page  
  [language="java"] [extends="package.class"]  
  [contentType="mimeType [charset=characterSet]" |  
  <"text/html; charset=ISO-8859-17quot; ]  
  %>
```

Les déclarations JSP

- Syntaxe : `<%%! declaration %>`

Exemple :

```
<%%! String Chaine = "bonjour";  
Int Numero = 10;  
  
public void jspInit() {  
// instructions;  
}  
%>
```

Les scriptlets JSP

- Une scriptlet JSP est un bloc de code Java compris entre les balises suivantes :

```
<% /* scriptlet */ %>
```

Le code Java présent entre les balises `<%` et `%>` devient le corps de la méthode `_jspService()` lors de la génération de la servlet (du moins si aucune directive de méthode n'est indiquée).

Les scriptlets JSP (2)

Les objets les plus utilisés dans les scriptlets sont l'objet request pour connaître les détails de la requête HTTP (notamment lors de l'utilisation de formulaires) et l'objet out permettant d'envoyer des données vers le navigateur du client. Par exemple :

```
<% String[] phrases = {« Adel", « Michel",  
« Yacoub", « Andre es-tu fais la"};  
    for (int i=0; i<phrases.length; i++) {  
        out.println(phrases[i]);  
    }  
    %>
```

Les expressions JSP

- Les expressions JSP permettent d'insérer simplement des chaînes de caractères générées dynamiquement dans la page HTML. La syntaxe d'une expression JSP est la suivante :

```
<%= Expression >
```

- L'expression suivante permet par exemple de retourner une chaîne contenant l'adresse IP du client :

```
<%= request.getRemoteAddr(); >
```

- Il s'agit ainsi d'un raccourci pour la scriptlet suivante :

```
<% out.println(request.getRemoteAddr()); >
```