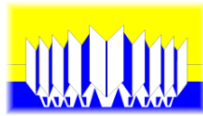


1

Faculté des Sciences de Tunis, Tunisie

Conception et programmation orientée objet



Présenté par : Nourhène ALAYA
2012-2013

2

Chapitre I Composants Web de J2EE: Servlet/JSP

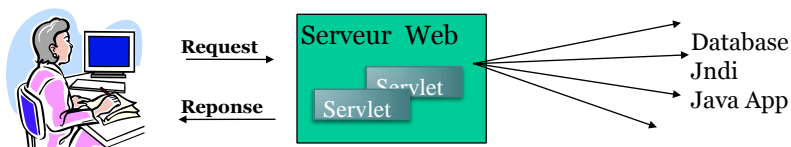
Les Servlets

Servlet : Server-side applet

- Permettre la programmation d'applications coté serveur
- Permettre l'extension d'un serveur Web en java
- Permettre la construction d'application Web dynamique
- Equivalent en java des CGI
- Un serveur web héberge des classes Java Servlets qui sont exécutées à l'intérieur du **conteneur web**. Le serveur web associe une ou plusieurs URLs à chaque Servlet.

Fonctionnement d'un servlet

- Un servlet lit les données envoyées par un client Web (transmises par le serveur)
 - Données explicites (Formulaire)
 - Données implicites (Request Header)
- Il génère un résultat
- Il envoie le résultat au client
 - Données explicites (Page HTML)
 - Données implicites (Response Header, Status code)

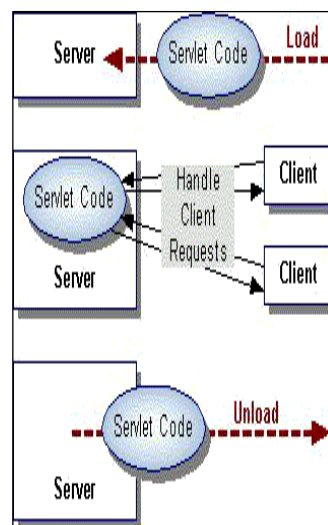


Servlet : Avantages coté serveur

- S'exécutent dans un moteur de Servlet (ou conteneur de Servlet) utilisé pour établir le lien entre la Servlet et le serveur web.
 - On ne se soucie plus de détails techniques tels que la connexion au réseau, la mise en forme de la réponse à la norme HTTP, ...
- Une servlet, peut utiliser toutes les API Java afin de communiquer avec des applications extérieures, se connecter à des bases de données, accéder aux entrées-sorties (fichiers par exemple), ...

Le cycle de vie d'un servlet

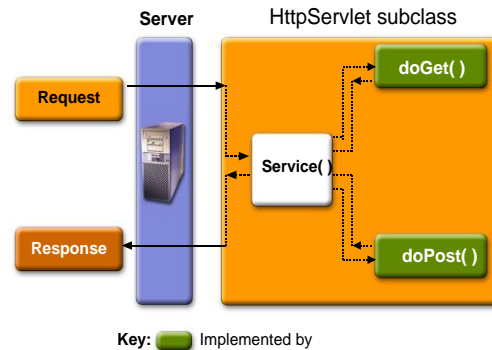
- Tous les servlets ont le même cycle de vie
 - Le conteneur charge la Servlet
 - Appel à la fonction: `init()`
 - La Servlet répond aux requêtes des clients
 - recevoir et répondre aux requêtes des clients par la fonction `service()`
 - Le conteneur détruit la Servlet
 - La fonction `destroy()` est appelée (fermeture des connections)
- Les servlets suivent un modèle de programmation requête-service-réponse
 - Requête : objet
`javax.servlet.ServletException`
 - contient les informations nécessaires pour une communication du client vers le serveur
 - Service : méthode `service()` est invoquée
 - Réponse : objet
`javax.servlet.ServletResponse`
 - contient les informations nécessaires pour une communication du serveur vers le client



Modèle de programmation Web

- Une Servlet Web étend la classe **javax.servlet.http.HttpServlet** (elle implémente **javax.servlet.Servlet**)
- Plusieurs méthodes spécifiques au **protocole HTTP remplacent la méthode service()**, qui appelle la méthode correspondant au type de requête.
- Une Servlet Web doit au moins redéfinir une de ces méthodes.

Méthode	Type de requête HTTP
doGet()	GET
doPost()	POST
doPut()	PUT
doDelete()	DELETE
doHead()	HEAD
doOptions()	OPTIONS
doTrace()	TRACE

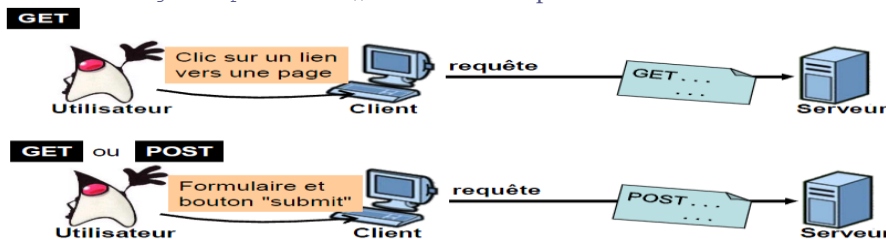


1. La classe HttpServlet



La classe HttpServlet

- Utiliser les objets `HttpServletRequest` et `HttpServletResponse` passés en paramètres des méthodes `doGet()` ou `doPost()` pour implémenter le service
 - `HttpServletRequest` contient les renseignements sur le formulaire HTML initial (utile pour `doPost()`)
 - `HttpServletResponse` contient le flux de sortie pour la génération de la page HTML résultat
 - ce flux de sortie est obtenu par les méthodes :
 - `getWriter()` : recommandé pour retourner du texte
 - `getOutputStream()` : recommandé pour des données binaires



Exemple d'une classe Servlet

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

public class SimpleServlet extends HttpServlet {

    public void init(ServletConfig c)
        throws ServletException {
        // phase d'initialisation
    }

    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException {
        // phase de traitement des requêtes HTTP GET
    }

    public void destroy() {
        // phase d'arrêt
    }

    public String getServletInfo() {
        // délivre des informations sur la servlet
    }

}
```

Initialisation

Service

Destruction


Exemple de Servlet (code complet)

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

public class HelloWorld extends HttpServlet {
    public void init() { }
    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(« Hello World !!!");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

→ **Compilation** : HelloWorld.class installé dans l'arborescence de Tomcat

→ **Chargement via une URL de type**

 http://localhost:8084/CoursWeb/hello

→ **Son exécution affiche**

Hello World

Exemple de servlet: *Explication du code*

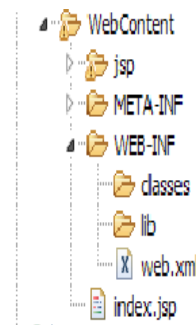
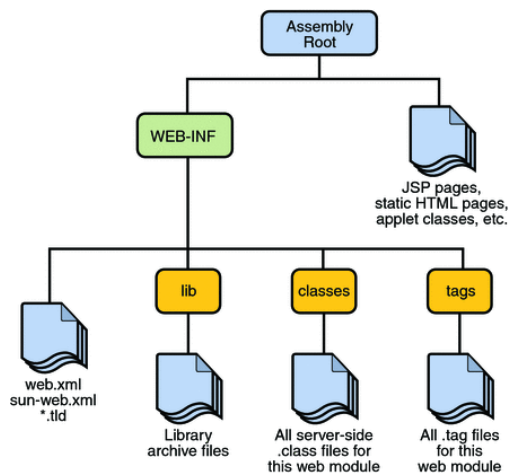
- La classe `HttpServlet` a été étendue : `public class HelloWorld extends HttpServlet {}`
- Lorsque la servlet est instanciée, il peut être intéressant d'effectuer des opérations qui seront utiles tout au long du cycle de vie de la servlet (se connecter à une base de données, ouvrir un fichier, ...). Pour ce faire, il s'agit de surcharger la méthode `init()` de la servlet. `public void init() {}`
- La méthode `doGet()` a été surchargée afin de traiter la requête utilisateur de type `Get` : `public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {}`
- L'objet `HttpServletResponse` permet de renvoyer une page à l'utilisateur:
 1. définir le type de données qui vont être envoyées au client : la méthode `setContentType()` de l'objet `HttpServletResponse`
 2. créer un objet `PrintWriter` grâce à la méthode `getWriter()` : → envoyer du texte formaté au navigateur `PrintWriter out = res.getWriter();`
 3. utiliser la méthode `println()` de l'objet `PrintWriter` afin d'envoyer les données textuelles au navigateur,
 4. fermer l'objet `PrintWriter` lorsqu'il n'est plus utile avec sa méthode `close()`

Déploiement d'une classe Servlet

- Un servlet ne peut pas être déployé directement dans un conteneur, elle doit faire partie d'un **module Web**.
- **Un module Web** est un ensemble de bibliothèques, de fichiers de configurations, de code Java (bytecode des servlets...), ...
- Pour déployer une application dans un conteneur, il faut lui fournir deux éléments :
 - **L'application** avec tous les composants (classes compilées, ressources ...) regroupée dans une archive ou module. Chaque conteneur possède son propre format d'archive.
 - **Un fichier descripteur** de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application

Packaging d'une application Web

- Arborescence d'un module Web



Packaging d'une application Web

- Web Component
 - Une application Web (*.html, *.jsp, servlets, ...) packagée dans un archive (.war) et est paramétrée dans le fichier WEB-INF/web.xml
- Structure d'une *Web Application Archive (.war)*
 - Fichiers Web de l'application (HTML, JSP, js, css...)
 - Répertoire « META-INF » : fichiers de configuration
 - MANIFEST.MF : informations sur le zip
 - Répertoire « WEB-INF » : contenu de l'application
 - **WEB-INF/web.xml**
 - Fichier de déploiement
 - Paramétrage des servlets, types MIME additionnels, ...
 - **WEB-INF/classes/**
 - .class des servlets et des classes (JavaBean, ...) associées
 - ressources additionnelles (localstring.properties, ...)
 - **WEB-INF/lib/**
 - .jar additionnels provenant de tierce parties (comme des drivers JDBC, TagLib (jsf, ...), ...)
 - **WEB-INF/tlds/**
 - .tld décrivant les TagLibs

Déploiement d'une application Web Java

- Consiste à permettre à un conteneur Web d'exécuter l'application
- Dépôt dans un répertoire *ad hoc du serveur*
 - Exemple : répertoire « webapps » de Tomcat
- Lecture des fichiers « war » au (re)démarrage du serveur
- Analyse du fichier « war » et des paramètres de configuration du descripteur de déploiement
- Création du répertoire correspondant dans webapps
- Mapping des URL de l'application vers le répertoire créé

Déploiement d'une application Web Java

- Tomcat : le conteneur Web le plus utilisé
 - Moteur d'exécution de servlets et de jsp
 - Projet soutenu conjointement par Apache et Sun
 - Codé en Java (nécessite une machine virtuelle Java)
- 2 modes de fonctionnement
 - Autonome (standalone)
 - Tomcat est aussi un serveur Web
 - Capable de servir des pages HTML, d'exécuter des servlets et des JSP
 - Collaboratif (in-process et out-of-process)
 - Extension d'un serveur Web (Apache, Microsoft IIS ou Netscape NetServer) meilleures performances pour le service des pages HTML

Le fichier descripteur Web.xml

- Déclaration d'une Servlet au sein d'une application Web se fait dans le fichier descripteur Web.xml

Web.xml
<pre> <web-app> <servlet> <servlet-name>.....</servlet-name> <servlet-class>.....</servlet-class> <init-param> < param-name> </param-name> < param-value> </param-value> </init-param> </servlet> <servlet-mapping> <servlet-name>.....</servlet-name> <url-pattern>.....</url-pattern> </servlet-mapping> </web-app> </pre>

Le fichier descripteur Web.xml

- Le fichier web.xml donne des instructions sur le déploiement du servlet dans le conteneur
 - **<web-app> </web-app>:**
 - encapsule l'ensemble des éléments servant à la configuration de l'application.
 - **<servlet> </servlet>:**
 - encapsule l'ensemble des éléments servant à la configuration de chaque Servlet.
 - **<servlet-name> </servlet-name>:**
 - contient une chaîne de caractère identifiant la Servlet au sein de l'application.
 - **<servlet-class> </servlet-class>:**
 - contient le nom complet de la classe de Servlet (package compris).
 - **<init-param> </init-param> :**
 - Encapsule les paramètres d'initialisation de la Servlet.
 - Chaque élément **<init-param> </init-param>** correspond à un paramètre représenté par une paire nom/valeur avec les éléments :
 - **<param-name> </param-name>**,
 - **<param-value> </param-value>**.
 - **<servlet-mapping> </servlet-mapping>:**
 - contient des informations permettant de définir la relation entre les URL et les servlets.
 - **<url-pattern>.....</url-pattern>**
 - définit comment une Servlet est invoquée

Le fichier descripteur Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi=
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>exemple.HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Le fichier descripteur Web.xml

- **Les paramètres d'initialisation**
- Les servlets peuvent avoir des paramètres d'initialisation.
- Ces paramètres peuvent être changés sans avoir à recompiler l'application
 - `<init-param>`
 - `<param-name>testValue</param-name>`
 - `<param-value>12</param-value>`
 - `<description>une valeur quelconque</description>`
 - `</init-param>`
- Ces paramètres peuvent être utilisés avec la méthode
 - `javax.servlet.getInitParameter()`

Le fichier descripteur Web.xml

- Exemple de paramètres d'initialisation

```
out.println("<h1>Servlet InitServlet at " + request.getContextPath () + "</h1>");
out.println("valeur = "+this.getInitParameter("valeur1"));
out.println("nombre = "+this.getInitParameter("nombre"));
```

```
<servlet>
  <servlet-name>InitServlet</servlet-name>
  <servlet-class>exemple.InitServlet</servlet-class>
  <init-param>
    <param-name>valeur1</param-name>
    <param-value>PAR DEFAULT</param-value>
  </init-param>
  <init-param>
    <param-name>nombre</param-name>
    <param-value>12</param-value>
  </init-param>
</servlet>
```

Servlet InitServlet at /CoursWeb

valeur = PAR DEFAULT nombre = 12

- Il existe d'autres propriétés qui peuvent être utilisé dans le fichier Web.xml comme:
 - `<welcome-file-list>`: pour indiquer le fichier d'accueil 'index.jsp'
 - `<error-page>`: indique la page à invoqué en cas d'erreur.

L'interface HttpServletRequest

- Fournit les informations sur la requête du client au serveur
- Principales méthodes (pour l'instant):
 - **String getParameter(String name)**
 - Permet de retourner le valeur d'un champ dont on a passé le nom (sensibles à la casse) en argument (Si le champ n'existe pas, la valeur *null* est retournée).
 - **public String[] getParameterValues(String Key)**
 - à utiliser Lorsqu'un champ d'un formulaire peut avoir plusieurs valeurs (liste à choix multiples, cases à cocher,... Retourne un tableau contenant l'ensemble des valeurs affectées à la clé spécifiée en paramètre
 - **Enumeration getParameterNames()**
 - Pour connaître l'ensemble des noms des champs du formulaire passé à la servlet. Retourne un objet *Enumeration*, contenant la liste des champs du formulaire. Il est possible de transformer chaque entrée en chaîne, puis de la traiter avec la méthode *getParameter()* afin de récupérer sa valeur
 - **String getHeader(String name)**
 - **Enumeration getHeaderNames()**
 - Retourne une énumération de tous les noms des propriétés du header
 - **String[] getHeaderValues()**
 - Retourne un tableau de toutes les valeurs du header

Exemple avec les request headers

Servlet

```
for (Enumeration e=request.getHeaderNames(); e.hasMoreElements(); ) {
    String name=(String) e.nextElement();
    out.println(name+" = "+request.getHeader(name)+"<br>");
}
```

Résultats

```
host=localhost:8084
user-agent=Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11
accept=text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
accept-language=fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
accept-encoding=gzip,deflate
accept-charset=ISO-8859-1,utf-8;q=0.7,*;q=0.7
keep-alive=300
connection=keep-alive
cookie=moncookie="mon password", JSESSIONID=AB4130236FA5B9E054232D468ED9F070
```

Exemple de sendRedirect()

```
public class WrongDestination extends HttpServlet{

    public void doGet (HttpServletRequest request ,
        HttpServletResponse response)
        throws ServletException, IOException{
        String userAgent = request.getHeader("User-Agent");
        if((userAgent != null) && ( userAgent.contains("MSIE ")){
            response.sendRedirect("http://home.netscape.com");
        }else{
            response.sendRedirect("http://www.microsoft.com");
        }
    }
}
```

L'interface HttpServletResponse

- Objet fournissant les services pour envoyer la réponse à un client.
- Les principales méthodes sont:
 - **java.io.PrintWriter getWriter()**
 - Pour récupérer un printWriter qui permet d'envoyer du texte au client
 - **public void setContentLength(int len)**
 - Positionne le Content-Length Header
 - **public void setContentType(java.lang.String type)**
 - Position le Content-Type header (exemple text/html)
 - **public void sendError(int sc, java.lang.String msg) throws java.io.IOException**
 - Envoi un message d'erreur au client (cf code dans l'API)
 - **public void setHeader(java.lang.String name, java.lang.String value)**
 - Ajoute une propriété à la requête

Les éléments du chemin de requête

- **ContextPath** : le chemin du contexte de déploiement
- **ServletPath** : la section du chemin qui a déclenché le mapping
- **PathInfo** : la partie de la requête qui n'est ni le ContextPath ni le ServletPath
- Exemple

```
out.println("<h1>Servlet URLServlet at " + request.getContextPath () + "</h1>");
out.println("context path= "+request.getContextPath()+"<br>");
out.println("servlet path= "+request.getServletPath()+"<br>");
out.println("path info= "+request.getPathInfo()+"<br>");
```

```
<servlet-mapping>
  <servlet-name>URLServlet</servlet-name>
  <url-pattern>/url/*</url-pattern>
</servlet-mapping>
```

 http://localhost:8084/CoursWeb/url/servlet

```
context path= /CoursWeb
servlet path= /url
path info= /servlet
```

 http://localhost:8084/CoursWeb/url/request

```
context path= /CoursWeb
servlet path= /url
path info= /request
```

28

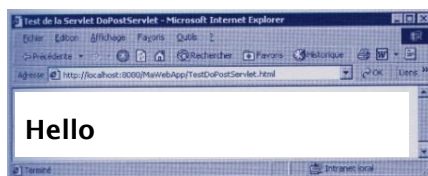
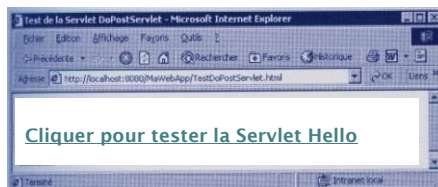
Invocation d'une Servlet à partir d'un navigateur Web

- Invocation de la méthode `doGet(...)`:
 - Cliquez sur un lien hypertexte qui pointe sur l'URL de la Servlet.
 - l'URL déclaré dans Web.xml (url-pattern)

Index.html

```
<HTML>
<HEAD>
  <TITLE>
    Test de la servlet Hello par clic sur lien
  </TITLE>
</HEAD>
<BODY>
  <P>
    <A href="/MaWebApp/servlet/Hello">
      Cliquez pour tester la Servlet Hello
    </A>
  </P>
</BODY>
</HTML>
```

URL du Servlet déclaré dans web.xml



Invocation d'une Servlet à partir d'un navigateur Web

- Invocation de la méthode `doPost(...)` : Formulaire

Index.html

```

<HTML>
<HEAD>
  <TITLE> Test de la servlet Hello </TITLE>
</HEAD>

<BODY>
  <FORM action = "/test/servlet/Hello" method = "post">
    <P>
      Prenom : <INPUT type = "text" name = "prenom">
    <BR>
      Nom : <INPUT type = "text" name = "nom">
    <BR>
    <INPUT type = "submit" value = "Valider">
  </FORM>
</BODY>
</HTML>

```

URL du Servlet déclaré dans web.xml

➤ Indiquez l'url de la classe Servlet comme étant l'action d'une balise <form> (formulaire)

Affichage

Prenom :
 Nom :

Invocation d'une Servlet à partir d'un navigateur Web

- Récupération des paramètres d'un formulaire dans `doPost(...)` :

Hello.java

```

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

public class Hello extends HttpServlet{

    public void doPost(HttpServletRequest req,
        HttpServletResponse res) throws ServletException, IOException
    {
        String prenom = req.getParameter("prenom");
        String nom = req.getParameter("nom");

        res.setContentType("text/html");

        PrintWriter out = res.getWriter();
        out.println("<HTML><BODY>");
        out.println("<H1>Bonjour" + prenom + " " + nom + "." + "</H1>");
        out.println("</HTML><BODY> ");
    }
}

```

Récupération requête

Génération réponse

Affichage

Bonjour Mohamed Salah.

Le ServletContext

- Les servlets peuvent s'échanger des informations par une « mémoire partagée »
- **Servlet.getServletContext()**
 - C'est un espace commun partagé dans une webapp
- Les principales méthodes de ServletContext
 - **Object getAttribute(String name)**
 - Retourne un attribut du contexte
 - **Void setAttribute(String name, Object value)**
 - Ajoute ou remplace un objet dans le contexte
 - **String getInitParameter(String name)**
 - Retourne un paramètre d'initialisation de l'application
 - **Void Log(String msg)**
 - Ajoute un message dans le log file du servlet container

```
out.println("Context 1 "+getServletContext().getInitParameter("context1"));
out.println("Context 2 "+getServletContext().getInitParameter("context2"));
```

```
<context-param>
  <param-name>context1</param-name>
  <param-value>ma valeur 1</param-value>
</context-param>
<context-param>
  <param-name>context2</param-name>
  <param-value>ma valeur 2</param-value>
</context-param>
```

Servlet WebContextServlet

Context 1 ma valeur 1 Context 2 ma valeur 2

Le RequestDispatcher

- L'objet RequestDispatcher est utilisé pour
 - Transférer la requête à un autre programme (jsp, servlet)
 - Inclure la réponse d'un autre programme
- Pour obtenir le request dispatcher

```
RequestDispatcher rd=getServletContext().getRequestDispatcher("/index.jsp");
rd.forward(request, response);
```

- La requête peut être transférée
 - `rd.forward(request,response)`
 - L'utilisation classique consiste à avoir un servlet contrôleur qui transmet les commandes à des servlets spécialisés
 - Le contrôle ne revient jamais au servlet
- La réponse peut être incluse dans la réponse en cours de construction
 - `rd.include(request,response)`
 - Ici, on peut implanter un équivalent de server side include.

Exemple d'utilisation de RequestDispatcher

```
RequestDispatcher rd;
String path=request.getPathInfo();
if (path.equals("/test")) {
    rd=getServletContext().getRequestDispatcher("/index.jsp");
} else {
    rd=getServletContext().getRequestDispatcher("/error.jsp");
}
rd.forward(request, response);
out.println("</body>");
out.println("</html>");
```

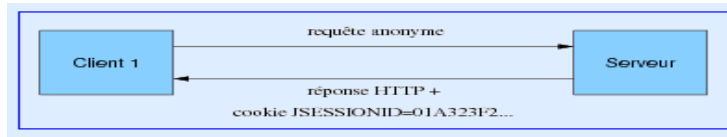
Les Sessions

- HTTP est stateless
 - Les informations ont une durée de vie d'un aller ou un retour au serveur.
- Certaines applications WEB (E-Commerce) ont besoin de maintenir une mémoire entre deux requêtes:
 - *Exemple* : le chariot (panier) de supermarché mis en place pour effectuer des achats sur le WEB
 - Pour ce faire : concept de « suivi de sessions »
- Comment maintenir l'état d'un client au cours d'une série de requêtes d'un même utilisateur pendant un temps donné ?
- Comment identifier le client ?

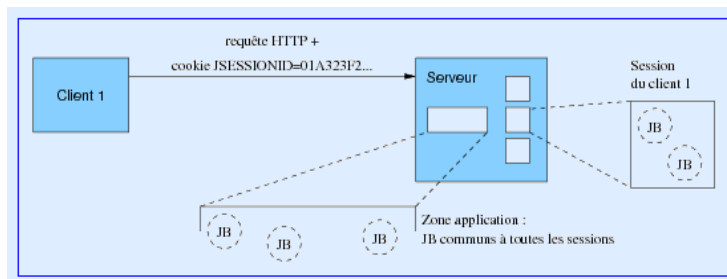
- ⇒ La gestion des sessions utilise les techniques classiques
 - url rewriting, Cookies, Champs cachés dans les formulaires
- ⇒ L'API HttpServlet fournit des fonctions pour gérer les sessions
- ⇒ L'implémentation est transparente pour l'utilisateur

Les Sessions

- **Principe** : pour identifier le client, le serveur renvoi, dans la réponse à la première requête, un *cookie* (JSESSIONID) :



- Les cookies sont tirés au hasard.
- Lors des requêtes suivantes, le client est repéré et le serveur peut lui associer une session :



Session & HttpServletRequest

- C'est le paramètre **request** qui maintient les informations sur la session
- Les méthodes sont
 - **HttpSession session = request.getSession(boolean flag)**
 - **Flag=true** : retourne l'objet session courant ou en crée un s'il n'y en a pas.
 - **Flag=false** : Retourne l'objet session courant ou null s'il n'y en a pas
 - **isRequestedSessionIdValid()**
 - Vrai si l'id de la session est valide dans le contexte courant
 - **isRequestedSessionIdFromCookie()**
 - Vrai si l'id de la session vient d'un cookie
 - **isRequestedSessionIdFromURL()**
 - Vrai si l'id de la session vient d'un URL

La classe HttpSession

- Exemple de création d'une nouvelle Session:
 - Partie de code de servlet:

```
out.println("<h1>Servlet NewSessionServlet at " + request.getContextPath () + "</h1>");
HttpSession sess=request.getSession();
out.println("Session is new ? "+sess.isNew()+"<br>");
out.println("Session id ? "+sess.getId()+"<br>");
sess.setAttribute("user", "john");
```

- La méthode isNew() de la classe HttpSession permet de vérifier si la session est nouvellement créée.
- Affichage résultat suite à l'appel de la servlet

Servlet NewSessionServlet at /CoursWeb

```
Session is new ? true
Session id ? 4077AFBCF13A4C2F94B821207A1147D6
```

Configuration des sessions

- Il est possible de fixer la durée d'une session par application (en minutes) dans le Web.xml
 - <session-config>
 - <session-timeout>30</session-timeout>
 - </session-config>

Terminer une session

- Pour terminer une session, soit la laisser s'expirer seule, soit mettre fin à sa durée de vie en utilisant une méthode de l'objet HttpSession, il s'agit de **invalidate()** :

```
HttpSession session = request.getSession(true);
session.invalidate();
```

Traitement des objets HttpSession

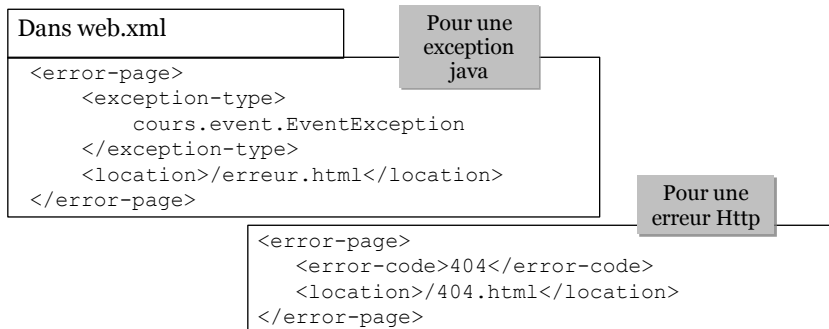
- L'information contenue dans un objet HttpSession est de la forme *(attribut : String, valeur : Object)(attribut,valeur)(attribut,valeur)...*
- La méthode `getAttribute (String)`
 - permet d'extraire la valeur correspondante à un attribut donné dans l'objet session.
 - Elle retourne un Objet de type `Object` ⇒ il est nécessaire de faire le *casting* pour manipuler la valeur retournée.
 - Exemple :

```
Panier p=(Panier) session.getAttribute("Schopping");
```
 - Cette méthode retourne `null` si l'attribut n'existe pas.
- La méthode `setAttribute (String, Object)`
 - Permet d'insérer une paire *(attribut, valeur)* dans un objet de type HttpSession
 - Si l'attribut existe déjà dans l'objet session sa valeur sera mise à jour.
 - Exemple :

```
Panier p = new Panier();
session.settAttribute("Schopping", p);
```
- La méthode `removeAttribute (String)`
 - Permet de supprimer un pair *(attribut, valeur)* dans un objet de type HttpSession
 - Exemple : `session.removeValue("Schopping");`

La gestion des erreurs

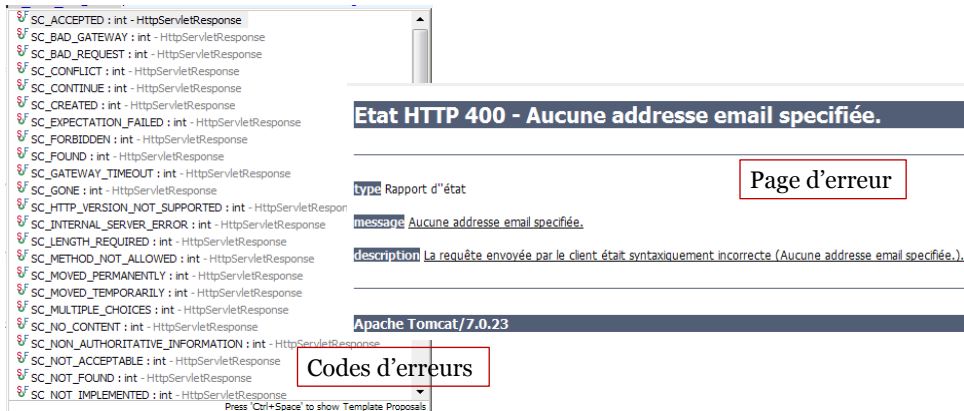
- Il est possible de définir les pages à afficher
 - En fonction d'erreurs http
 - En fonction d'exceptions java



La gestion des erreurs

- Il est possible aussi de rediriger la requête vers la sortie d'erreur du serveur, en précisant un message par la méthode `sendError()` :

```
response.sendError(int code, String msg);
response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Aucune
adresse email spécifiée.");
```
- SC_BAD_REQUEST** est un code d'erreur prédéfini dans la classe `HttpServletResponse`



42

Servlet : Conclusion

- Servlets : étendent le comportement des serveurs Web avec des programme Java
 - Portabilité, facilité d'écriture (**Java**)
 - Définition du code, du packaging, du déploiement
 - Persistance des données** dans les servlets
 - Servlet chargée et instanciée **une seule fois**
 - Exécutée en // avec des processus légers (**threads**)
- Mais :
 - Difficile d'écrire du code HTML dans du code Java
 ⇒ Introduction de la technologie Java Server Pages (JSP)
 - Pas de mécanisme intégré de distribution
 ⇒ Introduction de la technologie Enterprise Java Beans (EJB)

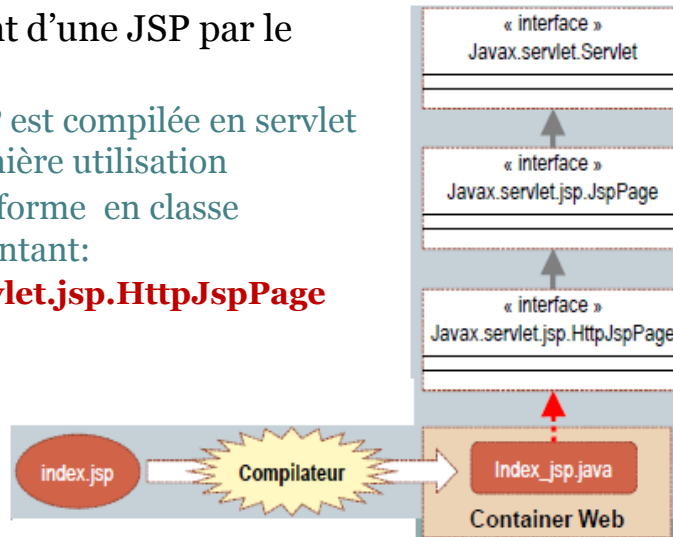
Composants Web de J2EE : JSP

JSP: Java Server Pages

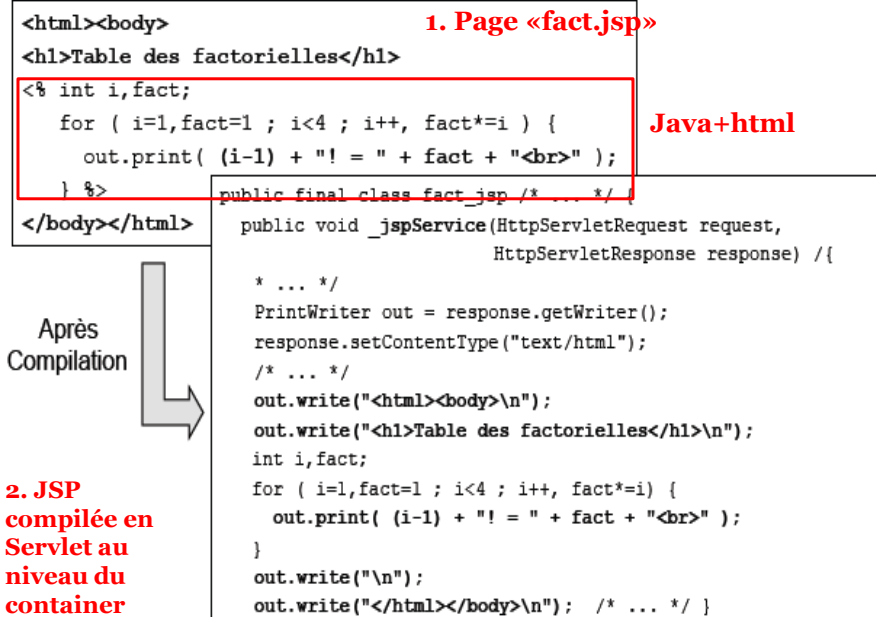
- **Servlet** : programme "*autonome*" stockés dans un fichier **.class** sur le serveur
- **Java Server Pages** : programme source Java "embarqué" dans une page .html
 - Insertion *syntaxe Java dans les pages HTML*
- JSP ont les même fonctionnalités que HttpServlet
 - *Implémentation du mécanisme requête/réponse*
 - *Accéder aux même données/objets qu'une servlet*
 - *Inclure ou rediriger la requête vers une autre servlet/JSP*
 - *Spécifique à HTTP*
 - *Génération de différents types de contenus : HTML, XML, SVG...*

JSP : Compilation

- Traitement d'une JSP par le serveur
 - Une JSP est compilée en servlet à la première utilisation
 - Se transforme en classe implémentant:
javax.servlet.jsp.HttpJspPage

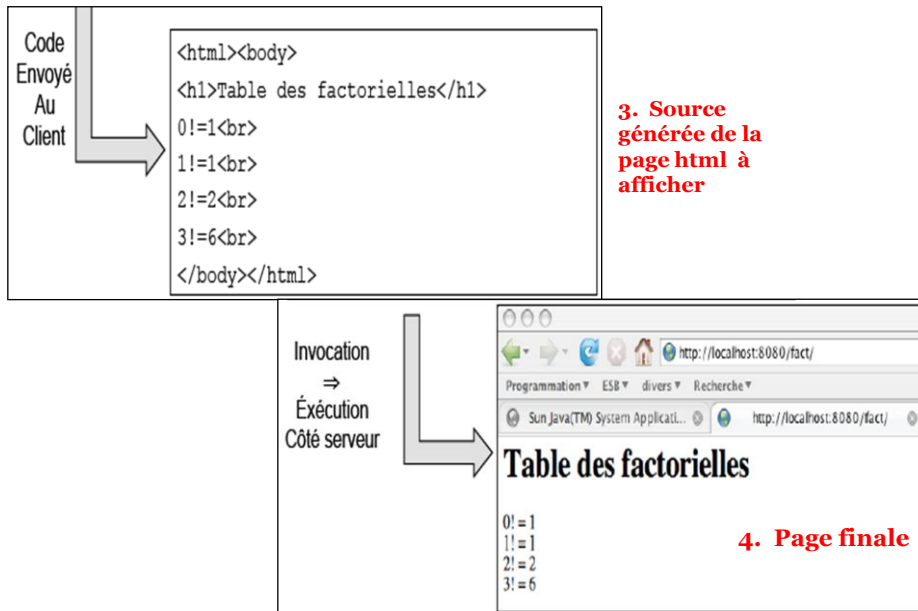


JSP: illustration du fonctionnements



47

JSP: illustration du fonctionnement



48

JSP : Mécanismes mis en œuvre

- Plusieurs zones `<% ... %>` peuvent cohabiter dans une même JSP. Lors du premier chargement d'une JSP (ou après modification), le moteur :
 - rassemble tous les fragments `<% ... %>` de la JSP dans une classe
 - la compile
 - l'instancie
- ☐ Puis, ou lors des chargements suivants, le moteur:
 - exécute le code dans un thread
 - délai d'attente lors de la 1ère invocation dû à la compilation
 - en cas d'erreur de syntaxe dans le code Java de la JSP : le message récupéré dans le navigateur

JSP: Syntaxe

- 3 parties d'une JSP
- scriptlets `<% %>`
- déclarations `<%! %>`
- expressions `<%= %>`

JSP: Scriptlets `<% ... %>`

- contient du code Java
- insérer dans `_jspService()` de la servlet, donc peut utiliser **out**, **request**, **response**, etc.

index.jsp

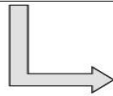
```
<html> <body>
<%
    String[] langages = {"Java", "C++", "Smalltalk", "Simula
67"};
    out.println("<h3>Principaux langages orientés objets :
</h3>");
    for (int i=0; i < langages.length; i++) {
        out.println("<p>" + langages[i] + "</p>")
    }
%>
</body> </html>
```



JSP: Expressions `<%= ..%>`

- La directive `<%= expr %>` génère l'affichage d'une valeur de l'expression **expr**
 - expr** : expression en Java qui renvoie un objet *String* ou un type *primitif*.
 - `<%= expr %>` est un raccourci pour `<% out.println(...); %>`
 - `<%= XXX %>` \approx `<% out.println(XXX); %>` **équivalence**
 - attention pas de `';` à la fin de l'expression

```
<html> <body>
<% int aleat = (int) (Math.random() * 5); %>
<h1> Nombre aléatoire : <%= aleat %> </h1>
</body> </html>
```



```
<html><body>
La somme est: <%= (195 + 9 + 273) %>
Je vous réponds à l'adresse : <%= request.getParameter("email_address")
%>
</html></body>
```

JSP: Déclarations `<%! .. %>`

- Permet de définir des méthodes ou des variables d'instances qui peuvent être associées à une JSP

Page.jsp

```

<HTML> <BODY>
<H1> Compteur</ H1>
<%!
int cpt = 0;

int getCpt() {
return cpt++;
}
%>
<H1> <%= getCpt() %> </ H1>
</ BODY> </ HTML>
```

Variable d'instance
- initialisée à l'instanciation de la JSP
- **persiste** entre 2 invocations tant que la JSP ne change pas

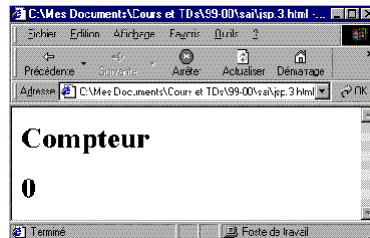
Méthode d'instance
- attachée à l'objet correspondant à la JSP

JSP: Déclarations `<%! .. %>`

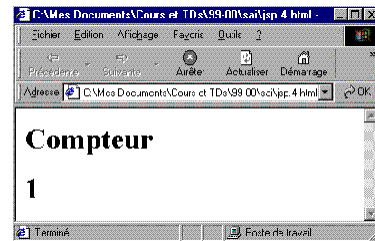
Attention !! `<%! int cpt = 0; %>` \neq `<% int cpt = 0; %>`

- variable d'instance de la JSP (**persiste**)
- ☐ variable locale à la JSP (**réinitialisée** à chaque invocation de la JSP)

Exemple avec la directive `<%! %>`



1ère invocation de la jsp



2ème invocation de la jsp

JSP : La directive `<%@code... %>`

- Fournit des informations globales relatives à la page. Trois types de directives:
 - **page** : modifier les données de la page (import de packages, spécification d'un type de contenu, gestion des sessions)
 - **Forme générale :**

```
<%@ page import="java.io.*;java.sql.*"
      session="true"
      isThreadSafe="true"
      errorPage="bug/erreur.html"
      isErrorPage="false"
      contentType="text/html; charset=UTF-8"
      pageEncoding="UTF-8"
      language="java" %>
```
 - il peut y avoir plusieurs directives page dans une JSP :


```
<%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
<%@ page isThreadSafe="true" %>
```
 - **include** : inclure des fichiers ou autres servlets/JSP, C'est comme si le contenu du fichier à inclure était directement copié dans le fichier courant.


```
<%@ include page="/monJSP.jsp" %>
```

 : inclusion statique (inclusion au niveau source : une seule servlet : Fusion)
 - **taglib** : utiliser des bibliothèques de balises personnalisées


```
<%@ taglib uri="..." prefix="..." %>
```

Les objets implicites dans une JSP

- Objets pré-déclarés utilisables dans le code Java des JSPs

out : le flux de sortie pour générer le code HTML

request : la requête qui a provoqué le chargement de la JSP

response : la réponse à la requête de chargement de la JSP

page : l'instance de servlet associée à la JSP courante (this)

exception : l'exception générée en cas d'erreur sur une page

session : suivi de session pour un même client

application : espace de données partagé entre toutes les JSP

Exemple1: d'une page JSP

```
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1> Hello
<%
String pname;                                // déclaration de variable
pname = request.getParameter("name"); // request : objet implicite
if (pname== null) { out.println("World"); } else {
%>
Mister <%=pname%>
<% } // fin du else %>
</H1>
</BODY></HTML>
```

Scriptlet
(source Java)

Expression (EL)

Exemple2

```

<HTML>
<BODY BGCOLOR="#ffffff">
<CENTER>
<% if (request.getParameter("nom")==null &&
    request.getParameter("email")== null) { %>
    <H2>Information utilisateur</H2>
    <FORM method="GET"
    action="/exemple2/web/process.jsp">
    <P>Votre nom: <input type="text" name="nom"
    size=26>
    <P>Votre email: <input type="text" name="email"
    size=26>
    <P><input type="submit" value="Envoyer">
    </FORM>
<% } else { %>
<%! String nom, email; %>
<%
    nom = request.getParameter("nom");
    email = request.getParameter("email");
    %>
    <P><B>Vous avez fourni les informations
    suivantes:</B>
    <P><B>Name</B>: <%= nom %>
    <P><B>Email</B>: <%= email %>
    <% } %>
</CENTER>
</BODY>
</HTML>

```

S'il s'agit d'une première invocation de l'URL, la jsp affiche un formulaire

Information utilisateur

Votre nom:

Votre email:

Si on accède à process.jsp après avoir rempli le formulaire et appuyer sur **Envoyer**, la JSP affiche

Vous avez fourni les informations suivantes:

Name: ali

Email: ali@ensi.mu.tn

58

Gestion d'erreur

- Une page JSP peut référencer une page erreur par `<%@ page errorPage="page.jsp"%>`
- Une page d'erreur est identifiée par l'entête `<%@ page isErrorPage="true"%>`
- Si une exception est levée le traitement est dérivé vers la page erreur qui peut repérer l'exception.

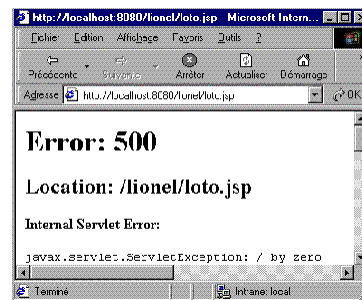
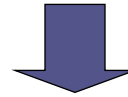
Exemple de gestion d'erreur

- Aucune page d'erreur n'est indiquée,

```
loto.jsp
<HTML>
<BODY>
<H1> Pourvu que le 0 ne sorte
pas!!</ H1>
<% int hasard =
(int) ( Math. random() * 5 );
%>
<H1> <%= 12 / hasard %> </ H1>
</ BODY>
</ HTML>
```



Si hasard = 0
page d'erreur par défaut



Exemple de gestion d'erreur:

- définition d'une page d'erreur

```
loto.jsp
<%@ page errorPage=" err.jsp" %>
<HTML> <BODY>
<H1> Pourvu ... !!</ H1>
<% int hasard = ... %>
<H1> <%= 12 / hasard %> </ H1>
</ BODY> </ HTML>
```

```
err.jsp
<HTML> <BODY>
<%@ page isErrorPage="true" %>
<h1> Le 0 est sorti !!</ h1>
Erreur :
<%= exception.getMessage() %>
</ BODY> </ HTML>
```

Si hasard = 0
page d'erreur err.jsp
Récupération de l'erreur via l'objet
prédéfini exception



JSP TagLib

- Un tag JSP est une simple balise XML à laquelle est associée une classe Java. A la compilation d'une page JSP, ces tags sont remplacés par l'utilisation de ces classes Java qui implémentent une interface particulière.


```
<prefix:nomDuTag attribut="valeur">
    Corps du tag
</prefix:nomDuTag>
```

 - Un **préfixe**, qui permet de distinguer les différentes taglibs utilisées.
 - Le **nom du tag** de la librairie.
 - D'un certain nombre de couple d'attribut/valeur.
 - D'un corps.
- Une JSP Taglib est une collection d'actions prédéfinies destinée à être utilisée dans une page JSP sous forme de tags. Elle se compose:
 - d'un descripteur de taglib (Tag Librarie Descriptor)
 - d'un ensemble de classes Java implémentant l'interface JspTag.
- Le descripteur de taglib (*.tld) est un document XML qui décrit les associations entre les balises et la classe Java.

Exemple de TagLib usuels de JSP (les actions standards)

- **jsp:useBean** associe une instance d'objet Java(Bean)
- **jsp:setProperty** positionne la valeur d'une propriété d'un bean
- **jsp:getProperty** récupère la valeur de la propriété d'un bean
- **jsp:include** inclut le contenu du page statique ou dynamique
- **jsp:invoke** invoque l'exécution d'un fragment (JSP, ...)
- **jsp:forward** redirige le traitement de la requête à un autre script
- **jsp:body**
- **jsp:doBody** appelle le traitement des sous-éléments (cf: TagFile)
- **jsp:element** insère un élément (XML) dans le document
- **jsp:attribute** attribut d'un élément inséré ou
- **jsp:output** configure le prélude du document XML
- **jsp:param** paramètre (key/value) pour jsp:forward et jsp:include
- **jsp:text** ajoute le texte à la sortie
- **jsp:plugin** insère un objet externe (dépendant de l'UA)
- **jsp:fallback** alternative en cas d'échec du jsp:plugin
- **jsp:params** liste les jsp:param dans un jsp:plugin

Les tags d'actions: <jsp:useBean>

- Les tags d'actions permettent de réaliser des traitements couramment utilisés.
- **Le tag <jsp:useBean>** permet de localiser une instance ou d'instancier un **bean** pour l'utiliser dans la JSP.
- L'utilisation d'un **bean** dans une JSP est très pratique car il peut encapsuler des traitements complexes et être réutilisable par d'autre JSP ou composants

JavaBean

- Classe Java
 - Encapsule des propriétés XXX uniquement accessible au travers de setXXX et getXXX
 - Constructeur sans argument
 - Support de la persistance en implémentant l'interface Serializable
- Exemple d'un JavaBean:

```
package clientele;
import java.io.*;

public class ClientBean implements
    Serializable {
    private String nom;
    private String email;
    public ClientBean() {
        nom = null;
        email = null;
    }
    public void setNom(String nom) { this.nom
        = nom; }
    public String getNom() { return nom; }

    public void setEmail(String email) {
        this.email = email; }

    public String getEmail() { return email; }
}
```


JSP et Java Beans

- JavaBean est utilisé afin d'avoir le moins de code Java possible dans une page JSP (HTML)
- Syntaxe générale :

```
<jsp:useBean id="nomInstanceJavaBean" class="nomClasseDuBean"
scope="request|session|application|page">
</jsp:useBean>
```
- Le bean est alors identifié et accessible par : "nomInstanceJavaBean"
- L'attribut **scope** indique la portée du bean.

valeur	Description
request	Le bean est valide pour cette requête. Il est utilisable dans les pages de redirection de la requête (<jsp:forward>). Il est détruit à la fin de la requête.
page	Similaire à request, mais le bean n'est pas transmis aux pages de redirection <jsp:forward>. C'est la portée par défaut
session	Le bean est valide pour la session courante. S'il n'existe pas encore dans la session courante, il est créé et placé dans la session du client. Il est réutilisé jusqu'à ce que la session soit invalidée
application	Le bean est valide pour l'application courante. Il est créé une fois et partagé par tous les clients des JSP.

JSP et Java Beans

<jsp:useBean>	Instanciation d'un bean <pre><jsp:useBean id="clientbean" class="cliente.ClientBean" /></pre>
<jsp:setProperty>	Modification d'une propriété <pre><%! String tonNom; %> <% tonNom = request.getParameter("nom"); %> <jsp:setProperty name="clientbean" property="nom" value="<%=tonNom %>" /></pre>
<jsp:getProperty>	Affiche une propriété

JSP et Java Beans

- Récupération des propriétés du bean:
 - Par appel de méthode `getXXX()` ou par la balise `<jsp:getProperty ...>`

```
<p> on repere le bean par le nom nomBean<br>
<jsp:useBean id="nomBean" class="SimpleBean" scope="session">
</jsp:useBean>
<p> On accede a une propriété avec une expression:
<br> compteur = <%= nomBean.getCompter() %>
<hr>
On incrémente le compteur <%= nomBean.increment(); %>
<p>On peut accéder à la propriété par une balise :<br>
<jsp:getProperty name="nomBean" property="compter" />
```

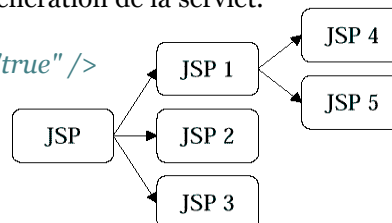
- Positionner les propriétés du bean dans une JSP
 - Par appel de méthode `setXXX(...)` ou par la balise `<jsp:setProperty ...>`

```
<p> on repere le bean par le nom nomBean<br>
<jsp:useBean id="nomBean" class="SimpleBean" scope="session">
</jsp:useBean>
<p> On positionne une propriété avec une expression:
<br> compteur = <%= nomBean.setCompter(6) %>
<p>ou par une balise :<br>
<jsp:setProperty name="noBean" property="compter" value="6" />
```

Le tag `<jsp:include>`

- Ce tag permet d'inclure le contenu généré par une JSP ou une servlet dynamiquement au moment où la JSP est **exécutée**.
- C'est la différence avec la directive `<@include page= >` avec laquelle le fichier est inséré dans la JSP avant la génération de la servlet.
- La syntaxe est la suivante

```
<jsp:include page="relativeURL" flush="true" />
```



- L'attribut `page` permet de préciser l'URL relative de l'élément à insérer.
- L'attribut `flush` permet d'indiquer si le tampon doit être envoyé au client est vidé.
 - Si la valeur de ce paramètre est `true`, il n'est pas possible d'utiliser certaines fonctionnalités dans la servlet ou la JSP appelée : il n'est pas possible de modifier l'entête de la réponse (header, cookies) ou renvoyer ou faire suivre vers une autre page.

Inclusion de JSP: Tag <jsp:include>:

agrégation des résultats fournis par plusieurs JSP

→ inclusion dynamique (délégation de servlets **deux servlets, l'un appelle l'autre**)

Exemple:

```
<HTML> <BODY>

<H1> JSP principale</ H1>

<jsp: include page=" inc. jsp" flush=" true" >
</ jsp: include>

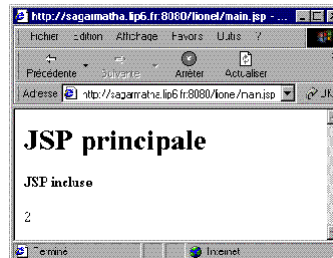
</ BODY> </ HTML>
```

Fichier **inc.jsp**

```
<B> JSP include</ B>
<P>
<%= (int) (Math. random() * 5)
%>
</P>
```

Résultat

```
<HTML> <BODY>
<H1> JSP principale</ H1>
<B> JSP include</ B>
<P>
<%= (int) (Math.
random() * 5) %>
</ P>
</ BODY> </ HTML>
```



Le tag de redirection <jsp:forward>

- Le tag <jsp:forward> permet de rediriger la requête vers une autre URL pointant vers un fichier HTML, JSP ou un servlet.
- Dès que le moteur de JSP rencontre ce tag, il redirige la requête vers l'URL précisée et ignore le reste de la JSP courante (tout ce qui a été généré par la JSP est perdu).

- La syntaxe est la suivante :

```
<jsp:forward page="{relativeURL | <%= expression %>}" />
```

ou

```
<jsp:forward page="{relativeURL | <%= expression %>}" >
```

```
<jsp:param name="parameterName" value="{ parameterValue |
<%= expression %>}" /> +
```

```
</jsp:forward>
```

- Il est possible de passer un ou plusieurs paramètres vers la ressource appelée grâce au tag <jsp:param>.

Enchaîner les pages JSP

- Exemple:

lanceForward.jsp

```
<% String repUtilisateur = request.getParameter("repTextField");
int rep = Integer.parseInt(repUtilisateur);
if ((rep % 2) == 0) {
%>
<jsp:forward page="gagne.jsp"/>
<% } else { %>
<jsp:forward page="perdu.jsp"/>
<% } %>
On n'affiche jamais cela
```

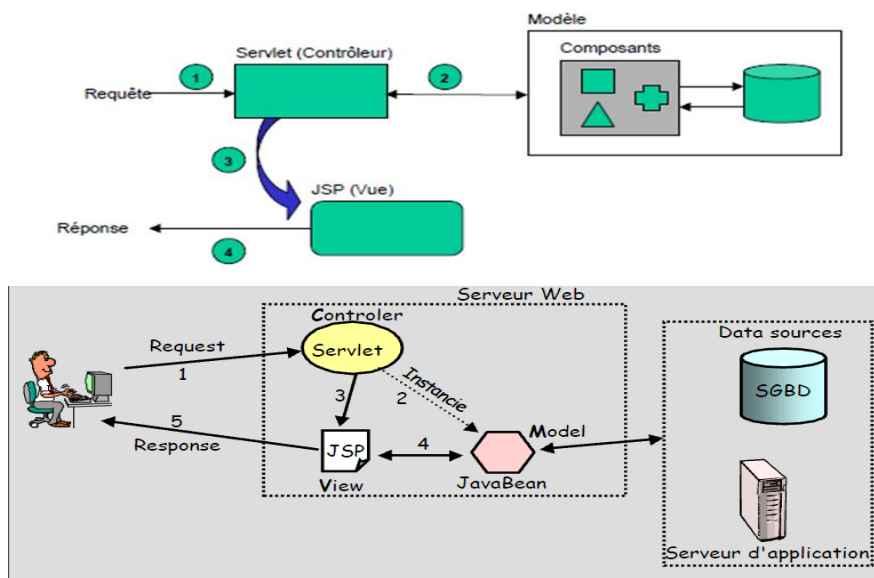
Exemple des JSP TagLib

```
<%@ include file="header.jsp" %>
<jsp:useBean id="mydate" scope="session" class="my.Date" />
<jsp:setProperty name="mydate" property="date" param="{param.pdate}" />
<c:if test="{mydate.date eq 0}" >
    <jsp:forward page="error.jsp" />
</c:if>
<jsp:plugin type="applet" code="Clock.class" width="160" height="150" >
    <jsp:params>
        <jsp:param name="adate" value="{mydate.date}" />
    </jsp:params>
    <jsp:fallback>
        Plugin tag OBJECT or EMBED not supported by browser.
    </jsp:fallback>
</jsp:plugin>
<jsp:include page="footer.jsp" flush="true"/>
```

MVC : JSP/Servlet

Exemple d'application Web J2EE

Communication Servlet/JSP: Le modèle MVC



Communication Servlet/JSP: Le modèle MVC

- La servlet peut invoquer une JSP en utilisant:
 - `response.sendRedirect(« /url-jsp »)` : redirection vers une jsp.
Le Servlet ne peut pas envoyer des paramètres aux JSP.
 - la classe `RequestDispatcher`. Le Servlet transfère les paramètres du request et du response vers la JSP.

```
public void doPost(HttpServletRequest request, HttpServletResponse response) {
    ServletContext context = getServletContext(); // héritée de GenericServlet
    RequestDispatcher dispatcher =
        context.getRequestDispatcher("/maPageMiseEnForme.jsp");
    dispatcher.forward(request, response);
}
```

- Une JSP peut invoquer une classe Servlet d'une manière classique:
 - Lien hypertexte : URL du Servlet
 - Formulaire : `<form action="url-servlet" >`

Communication Servlet/JSP: MVC

- La classe Servlet peut passer des valeurs à la JSP grâce aux méthodes `setParameter(String, String)` ou `setAttribute(String, Object)` de la classe `HttpServletRequest`

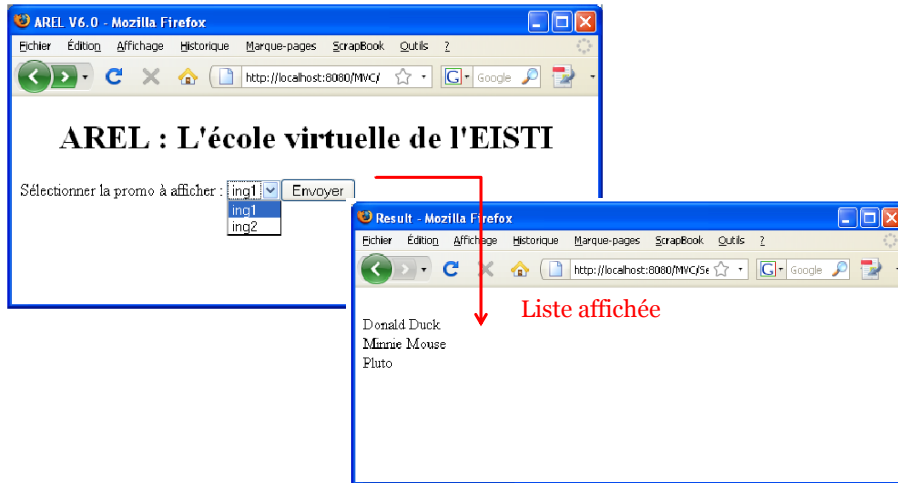
```
public void doPost(HttpServletRequest request, HttpServletResponse response) {
    // appelle les méthodes sur les objets métiers
    ArrayList theList = // un objet à passer
    // ajoute à la requête
    request.setAttribute("nomDelObjet", theList);
    ServletContext context = getServletContext();
    RequestDispatcher dispatcher = context.getRequestDispatcher("/jspAppeler.jsp");
    dispatcher.forward(request, response);
}
```

- La JSP extrait des paramètres de request grâce `String` `getParameter(String)` ou `Object` `getAttribute(String)` :

```
<% ArrayList theList = (ArrayList)
    request.getAttribute("nomDelObjet");
// maintenant, utiliser l'ArrayList
%>
```

Exemple projet JSP/Servlet : MVC

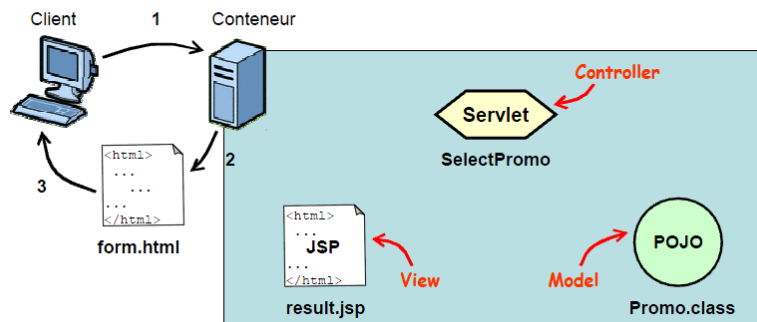
Ex : AREL V6 – liste des promos



Exemple MVC avec JSP/Servlet

MVC : étape 1

Le client récupère un formulaire (form.html)
pour passer une requête avec paramètres (1, 2, puis 3)



Exemple MVC avec JSP/Servlet

Formulaire : form.html

```
<!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=ISO-8859-1">
<title>AREL V6.0</title>
</head>
<body>
  <h1 align="center">AREL: L'école virtuelle de l'EISTI</h1>

  <form method="GET" action="http://localhost:8080/MVC/SelectPromo">
    Sélectionner la promo à afficher:

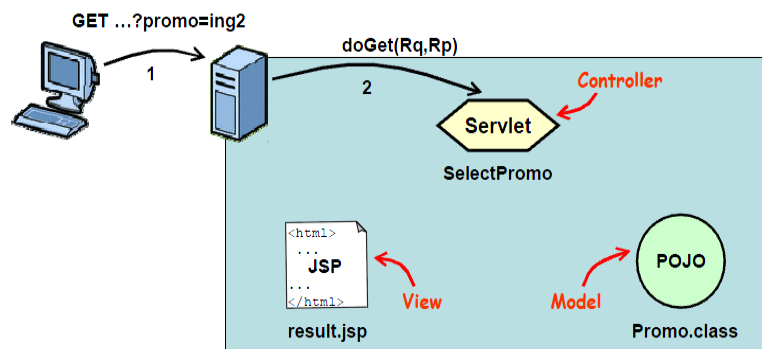
    <select name="promo" size="1">
      <option>ing1</option>
      <option>ing2</option>
    </select><input type="SUBMIT" />

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

Exemple MVC avec JSP/Servlet

MVC : étape 2

1. Le client envoie son formulaire (GET/POST avec paramètres)
2. Le conteneur transmet au servlet correspondant (le *controller*)



Exemple MVC avec JSP/Servlet

- *Controler le servlet: **SelectPromo.java***

```
package arel;

import ...;

public class SelectPromo extends javax.servlet.http.HttpServlet
    implements javax.servlet.Servlet{

    //...
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException{

        String promoName = request.getParameter( "promo" );

        //...
    }
}
```

Exemple MVC avec JSP/Servlet

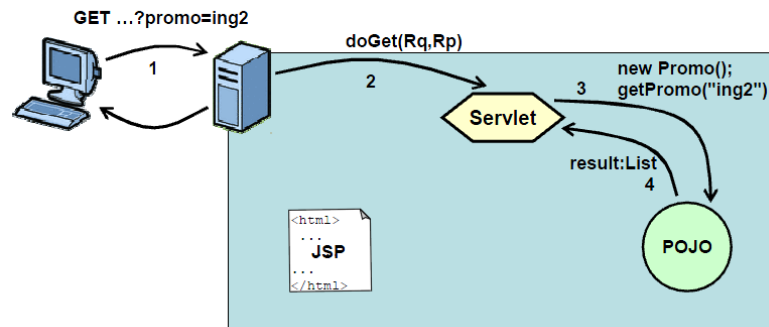
Configuration : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>MVC</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <description></description>
        <display-name>SelectPromo</display-name>
        <servlet-name>SelectPromo</servlet-name>
        <servlet-class>arel.SelectPromo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SelectPromo</servlet-name>
        <url-pattern>/SelectPromo</url-pattern>
    </servlet-mapping>
</web-app>
```

Exemple MVC avec JSP/Servlet

MVC : étape3

3. Le servlet *controller* interroge le *model* sur « ing2 »
4. Le *model* retourne au *controller* le résultat correspondant



Exemple MVC avec JSP/Servlet

Model : Promo.java

```

package arel;
import ...;

public class Promo{

    public List<String> getPromo(String promo){
        List<String> promoList = new ArrayList<String>();
        if(promo.equals("ing1")){
            promoList.add("Donald Duck");
            promoList.add("Minnie Mouse");
            promoList.add("Pluto"); //...
        } else if (promo.equals("ing2")){
            promoList.add("Mickey Mouse");
            promoList.add("Daisy Duck");
            promoList.add("Goofy"); //...
        } else{ return null;}

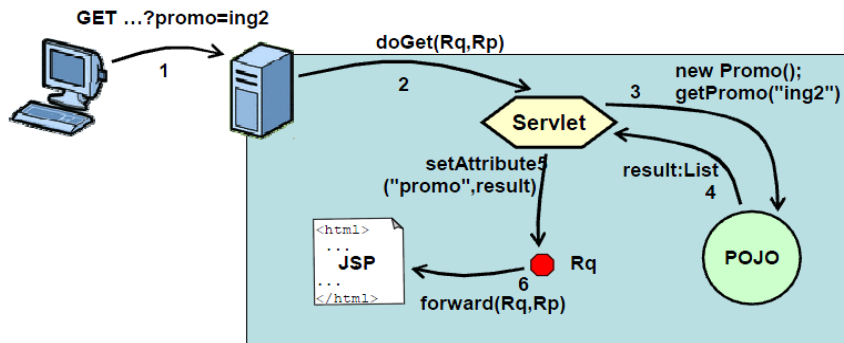
        return promoList;
    }
}

```

Exemple MVC avec JSP/Servlet

MVC : étape 4

5. Le *controller* utilise les données du *model* pour sa réponse
6. Le *controller* transmet sa réponse à la *view* (JSP)



Exemple MVC avec JSP/Servlet

Controller : SelectPromo.java

```

package arel;

import ...;

public class SelectPromo extends javax.servlet.http.HttpServlet
    implements javax.servlet.Servlet {
    //...
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String promoName = request.getParameter("promo");
        Promo promo = new Promo();
        List<String> result = promo.getPromo(promoName);
        request.setAttribute("promo", result);
        RequestDispatcher view =
            request.getRequestDispatcher("result.jsp");
        view.forward(request, response);
    }
}

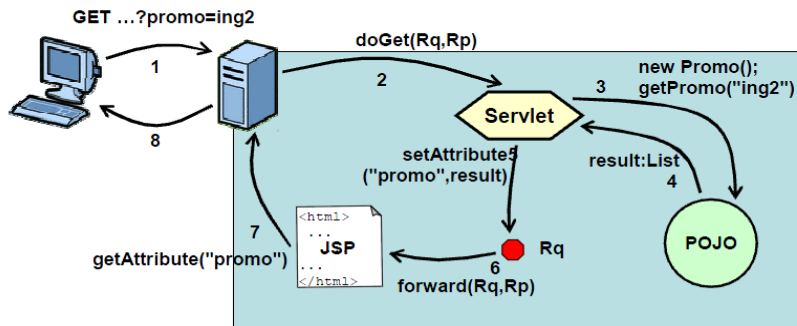
```

Exemple MVC avec JSP/Servlet



MVC : étape 5

7. La JSP (view) traite la réponse transmise par le *controller*
8. La page HTML résultante est reçue par le client



Exemple MVC avec JSP/Servlet

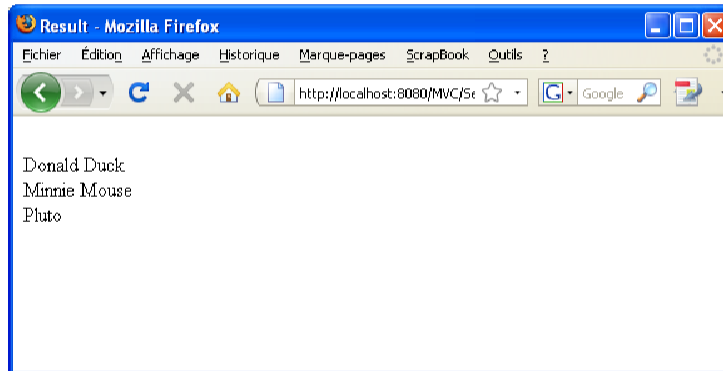
View : result.jsp

```

<%@ page import="java.util.*" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Result</title>
</head>
<body>
<%
    List<String> promoList=(List<String>)request.getAttribute("promo");
    Iterator it= promoList.iterator();
    while(it.hasNext()){
        out.print("<br />" + it.next());
    }
%>
</body>
</html>
  
```

Exemple MVC avec JSP/Servlet

Ex : AREL V6 - liste des promos



Conclusion sur JSP / Servlets

- Quoi mettre où ? (travailler en servlets ou en JSP ?)
 - Dépend de la quantité de code Java / HTML
 - Dépend de la couche dans laquelle on se trouve
- Plus objectivement
 - Servlets
 - Orientation des données
 - Accès aux autres méthodes que celles de service
 - Définition d'autres méthodes
 - JSP
 - Inclusions d'autres JSP / intégration de composants
 - Appel à des JavaBeans
 - Utilisation de taglibs

Annexe A : les formulaires

- Définition d'un formulaire : balise **<form>**
 - **Attribut method :**
 - get : méthode HTTP GET, les paramètres sont passés via l'URI
 - post : méthode HTTP POST, la page HTML avec le contenu des paramètres est envoyée au serveur
 - **Attribut action :**
 - URI du composant Web qui va recevoir le formulaire
 - Peut contenir n'importe quelle sous balise HTML valide
 - Sous-balise `<input>` : définition d'un paramètre
 - **Attribut type :**
 - Définit le type du paramètre (text, password, reset, submit...)
 - **Attribut name :**
 - Définit le nom du paramètre
 - **Attributs optionnels (size...)**

Annexe A: les formulaires

- Définition d'un formulaire en HTML


```

<html> <body>
<h3>Hello, What's your name?</h3>
<form method="post" action="url-servlet">
Name: <input type="text" name="username" size="25">
Password: <input type="password" name="password"
size="25">
<p></p>
<input type="submit"
value="Submit">
<input type="reset"
value="Reset">
</form>
</body> </html>

```

