JSP Servlet

GUINKO Tonguim Ferdinand

9 octobre 2011

Sommaire

Page web statique

Caractéristiques d'une page web statique

- Page web apparaîssant sur le navigateur de l'utilisateur telle qu'elle a été créée ;
- Son contenu est invariable et est le même pour tous les visiteurs qui visitent la page ;
- Généralement conçue par le langage de balisage de texte HTML, associé ou non au langage de formatage de document CSS et/ou au langage de script JavaScript ;
- Son contenu change seulement par l'action d'un agent humain et non par un procédé automatique géré par un serveur.

Page web statique

Remarque

la présence d'objets animés tel que des vidéos, des images animés, des contenus de type flash ne font pas nécessairement d'une page web une page web dynamique.

Quelques inconvénients

- lorsqu'elles présentent un important flot de donnée, la recherche d'information y est très peu effiscience ;
- plus il y'a de pages web statiques, plus leur maintenance devient hardue.

Page web dynamique

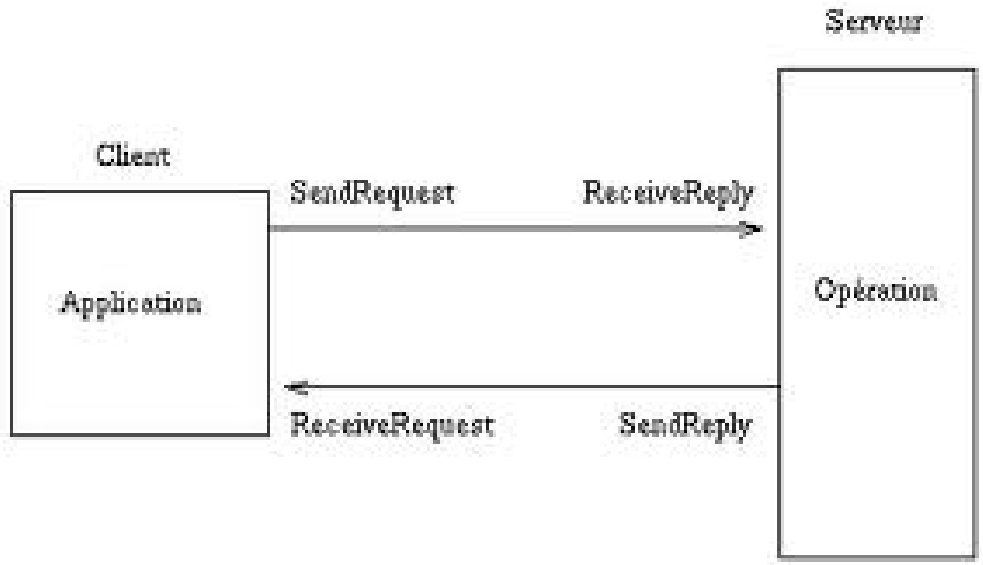
Caractéristiques d'une page web dynamique

- Page web qui contient des éléments qui interagissent avec l'utilisateur ;
- Son contenu varie donc et est généré en fonction de critères spécifiés par l'utilisateur : le serveur web rassemble les informations dont les critères sont spécifiés par l'utilisateur et les lui renvoie à travers un formulaire ;
- Il est nécessaire de disposer d'un serveur web pour visualiser une telle page ;
- Le contenu d'une page web dynamique est généralement issu d'une base de données.

Inconvénient : Les moteurs de recherche présentent souvent des difficultés pour l'indexation des pages web dynamiques car leur

Applications web orienté serveur

Plusieurs technologies existent pour rémédier aux inconvénients des pages web statiques et créer des pages web dynamiques.



Active Server Pages (ASP)

- ASP désigne une technologie fondée sur une suite logicielle, et appartenant à la compagnie Microsoft pour la conception de sites web dynamiques ;
- Basé sur 2 langages de scripts que sont le VBScript et le JScript qui sont des langages interprétés ;
- Fonctionne sur des plateformes Windows à travers le logiciel de serveur de services Web, Internet Information Services (IIS), de Microsoft ;
- Une implémentation du serveur web Apache, `Apache::ASP` permet de faire fonctionner des pages web ASP sur des plateformes de type GNU/Linux.

PHP : Hypertext Preprocessor (PHP)

- Langage de script s'exécutant sur un serveur et permettant de créer des pages web dynamiques ;
- Renvoie du code HTML au navigateur ;
- Les script PHP sont insérés dans une page HTML ; de telles pages portent l'extension `.php`
- PHP emprunte des éléments de syntaxe aux langages C, Perl et Java.

Common Gateway Interface (CGI)

- Interface standardisée par des normes et permettant à des programmes, d'interagir avec des serveurs web ;
- CGI ne désigne pas un langage de script en particulier : un CGI peut être écrit dans n'importe quel langage, par exemple PERL, Visual Basic, Python, Shell Unix pouvant s'écuter sur un serveur web ;
- Le langage de script le plus utilisé pour l'écriture de CGI est le langage PERL ;
- Un script CGI récupère les critères spécifiés par l'utilisateur et retourne la page générée.

JEE, JSP, Servlet

Java Server Pages (JSP)

- L'expression Java Server Pages (JSP) désigne une technologie basée sur le langage de programmation Java, permettant de créer des pages web dynamiques et s'exécutant sur un serveur ;
- Les pages web dynamiques créées suivant cette technologie portent l'extension `.jsp` ;
- JSP est conçue pour faciliter la génération de pages web dynamiques s'appuyant sur le langage de programmation Java.

JEE, JSP, Servlet

Java Server Pages (JSP)

- Lorsqu'un client web sollicite une page JSP située dans une application web particulière, le conteneur qui gère cette page JSP capture et analyse la requête. Si cette page n'a jamais été appelée ou si son code a été modifié, le conteneur la traduit sous la forme d'une servlet, qui est à son tour compilée puis exécutée pour fournir une réponse à la requête du client web ;
- JSP est directement associé à la notion de servlet.

C'est cette technologie qui fera l'objet de la suite du cours.

Java Enterprise Edition (JEE)

- technologie basée sur le langage Java
- ensemble de librairies qui permettent de résoudre des problèmes spécifiques aux entreprises (applications web, applications distribuées) ;

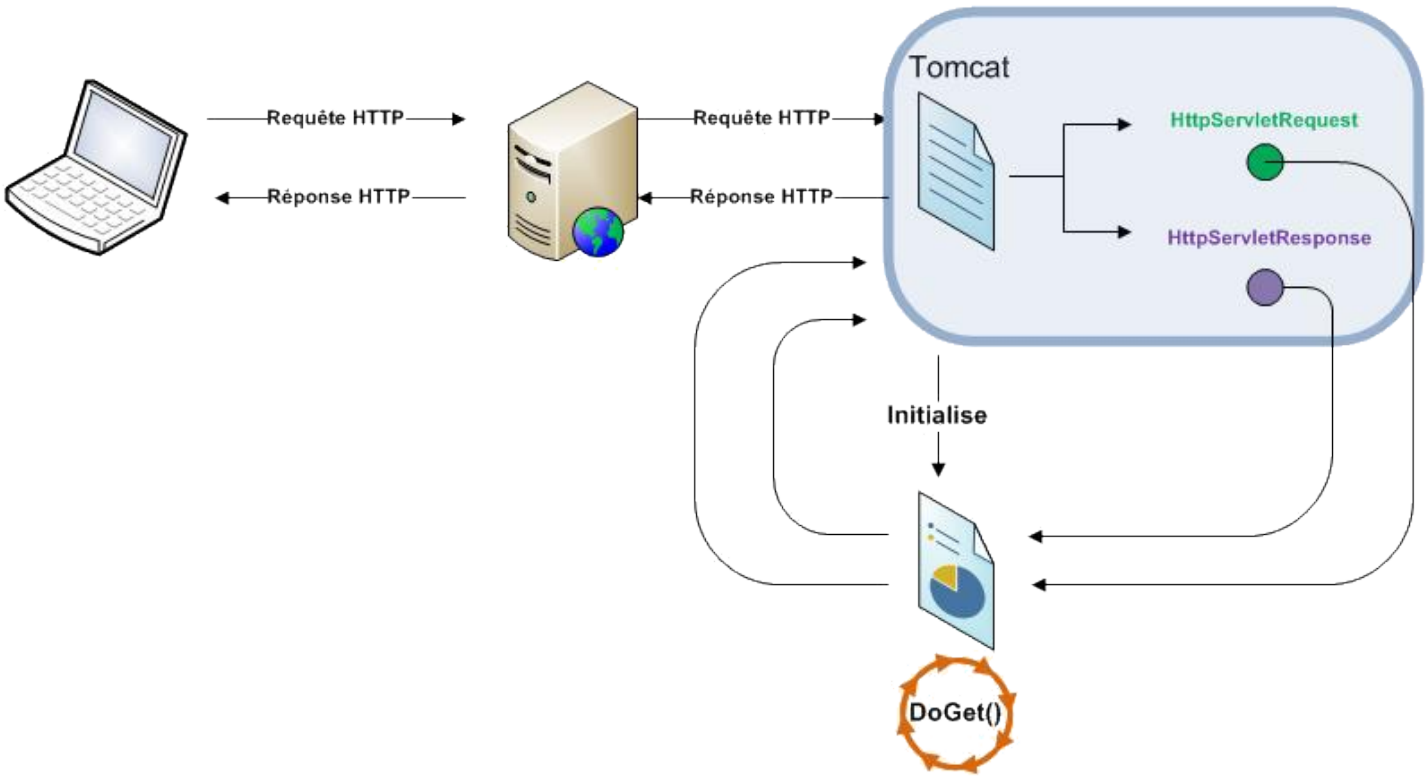
Java Enterprise Edition (JEE)

- Spécification ou norme décrivant les éléments constituant et intervenant dans le fonctionnement d'une application distribuée.
  - Comment doivent être développés les différents composants d'une application (servlet, pages JSP, etc.) ;
  - Comment ces différents composants peuvent communiquer entre eux ou avec d'autres applications ;
  - Comment doivent être organisés ces composants pour constituer une application ;
  - Contraintes que doivent respecter les serveurs d'hébergement ;
  - etc.

Servlet : généralités

Classe Java qui s'éxécute sur le serveur Tomcat ; Invoqué à travers une URL ; Cette classe s'exécute sur la machine virtuelle qui est sur le serveur. Le servlet n'a pas de méthode `main`.

Fonctionnement



Classes

La plupart des classes et des interfaces proviennent des 2 packages que sont :

1. `javax.servlet` : contient particulièrement les classes indépendantes du protocole HTTP ; un servlet indépendant du protocole HTTP utilisera la classe `GenericServlet` qui permet d'implémenter l'interface générique des servlets ;
2. `javax.servlet.http` : contient particulièrement les classes en rapport direct avec le protocole HTTP ; un servlet utilisant le protocole HTTP utilisera la classe `HttpServlet`.

Classes

Ces 2 packages sont importés dans la plupart des codes sources des servlets. D'autres packages additionnels peuvent être ajoutés en fonction de ce que l'on veut réaliser avec le servlet.

Contrairement à une classe Java normale, un servlet ne possède pas de classe `main()`. Lorsque le serveur web sollicite un servlet, il fait appel à sa méthode `service`.

Exemple

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

La classe HttpServlet

```
public class DateTimeServlet extends HttpServlet
{
```

- Nous n'utilisons pas l'interface basique `javax.servlet.Servlet.` car nous voulons ≪*parler*≫ `HTTP`. Nous devons donc hériter de la classe `javax.servlet.http.HttpServlet`. En effet, par défaut, les servlets sont indépendants du protocole `HTTP`.
- `javax.servlet. Servlet` est une interface basique
- `javax.servlet.http.HttpServlet` est une classe abstraite qui implémente la classe de base pour permettre la communication via le protocole `HTTP`

```
}
```

La classe HttpServlet : ses méthodes

```
public class javax.servlet.http.HttpServlet implements javax.servlet.Servlet
{
    ...
    //appelé lors du traitement d'une requete GET
    protected void doGet(HttpServletRequest requete,
                         HttpServletResponse reponse);

    //appelé lors du traitement d'une requete POST
    protected void doPost(HttpServletRequest requete,
                         HttpServletResponse reponse);

    //appelé lors du traitement d'une requete PUT
    protected void doPut(HttpServletRequest requete,
                         HttpServletResponse reponse);
    ...
}
```

La classe HttpServlet : ses méthodes : leurs arguments

```
public void doGet (HttpServletRequest requete, HttpServletResponse reponse)
    throws IOException, ServletException
```

Les methodes `doGet`, `doPost`, `doPut` contiennent 2 objets en guise d'arguments `HttpServletRequest` et `HttpServletResponse`.

- L'objet `requete` encapsule la requête formulée par le client au servlet, tandis que
- L'objet `reponse` encapsule la réponse envoyée par le servlet au client.

Classe HttpServlet : ses méthodes : HttpServletRequest()

Cette méthode encapsule les données de la requête, notamment les données suivantes :

- Paramètres de formulaire : `<form> </form>` ;
- Attributs ;
- Cookies ;
- Session.

```
public interface javax.servlet.ServletRequest
{
    public String getParameter (String key);
    public Enumeration getParameterNames ();
    public String[] getParameterValues (String key);
    public String getContentType ();
    public ServletInputStream getInputStream ();
}
```

Classe HttpServlet : ses méthodes : HttpServletRequest()

La méthode `getParameter` et les autres permettent d'accéder aux paramètres de la requête HTTP, paramètres transportés dans :

- L'URL dans le cas de la méthode `GET`
  Exemple :http ://mon-Site.org/MonServlet ?name=Exemple&course=TechnologieInforoute
- Le corps de la requête dans le cas de la méthode `POST`.

Classe HttpServlet : ses méthodes : HttpServletRequest()

Exemple :

```
//récupère les champs "nom" et "prenoms" d'un formulaire
String nom=reponse.getParameter("nom").trim();
String prenoms=reponse.getParameter("prenoms").trim();
//en cas d'erreur
if (nom.isEmpty() || prenom.isEmpty())
{
    request.setAttribute("error msg",
    "Les champs nom et prenoms ne doivent pas etre vides");
    RequestDispatcher disp=request.getRequestDispatcher("ajoute personne.jsp");
    disp.forward(request, response);
}
```

Classe HttpServlet : ses méthodes : HttpServletResponse()

```
public interface javax.servlet.ServletResponse
{
    public void setContentType(String type); //specifie le format de sortie
    public void setContentLength(int size);
    public ServletOutputStream getOutputStream();
    public PrintWriter getWriter(); //Pour obtenir le flux de sortie
    public void sendRedirect (String URL);
}
```

Les méthodes ci-dessus permettent d'écrire des données de type texte au sein du navigateur.

Exemple :

```
PrintWriter out = response.getWriter();
response.setContentType("text/html");
out.println ("Bonjour les amis!");
```

Écrire la réponse HTML

```
out.println ("<html>");
out.println ("<head>");
out.println ("<title>Exemple de la date et de l'heure}</title>");
out.println ("</head>");
...
out.println ("<p>La date d'aujourd'hui est " + new Date () + ".</p>");
...
```

Cycle de vie du servlet

Le cycle de vie du servlet est controllé par le container qui le contient. Ce cycle de vie comprend les étapes suivantes, depuis l'initialisation du servlet jusqu'à sa destruction :

1. Instantiation
2. Initialisation par la méthode `init` : lorsque le server charge un servlet, en réalité il exécute la méthode `init` du servlet.
3. Service
4. Desctruction par la méthode `destroy`
5. Purge de la mémoire

Cycle de vie du servlet :

Le container fera appel à l'une ou plusieurs des méthodes suivantes :

```
public interface javax.servlet.Servlet
{
    public void init (ServletConfig config);
    public void service (ServletRequest request, ServletResponse response);
    public void destroy ();
    public String getServletInfo ();
}
```

Cycle de vie du servlet : méthodes init(), destroy() et service()

- `init()` initialise le servlet. Tandis que le container invoque la méthode `init()`, il lui passe en paramètre l'objet `ServletConfig` par lequel il transmet des informations de configuration au servlet. La méthode `init()` est invoquée une seule fois durant le cycle de vie d'une servlet ; `init()` joue le même rôle que le constructeur d'une classe ;
- `destroy()` est invoquée avant que la servlet ne soit collecté par le `ramasse-miettes` *(garbage collector)*. Il enclenche le processus de destruction des servlets. Après l'appel de la méthode `destroy()`, la servlet n'est plus disponible à répondre aux requêtes ;
- `service()` traite toutes les requêtes `HTTP` ; `service()` est appelée à chaque réception d'une requête sur la servlet

Exercice

Ecrire un servlet qui récupère les informations d'un formulaire (nom, prenom) et les affiche dans une nouvelle page html.

Intérêt des JSP

Avantage des servlets

- Lecture des requêtes HTTP ;
- Création des témoins de connexion (cookies) et suivit de session ;
- Partage de données entre servlets ;

Inconvénient des servlets

- Difficile de concevoir une page HTML : il faut, pour chaque instruction faire usage de l'instruction `println` ;
- Maintenance du code HTML difficile.

Qu'est ce qu'une page JSP

- Equivalent d'une page PHP mais avec code en Java ;
- Les JSP sont traduites en Servlet par le Serveur Tomcat.

Le contenu des pages JSP est organisé en 2 parties :

- Le contenu statique (HTML/XML, les images, javascript, etc.)
- Le contenu dynamique envoyé par le serveur, et qui peut varier à chaque chargement de la page. La page JSP doit donc contenir des instructions qui permettront la connection avec le serveur.

Exemple1

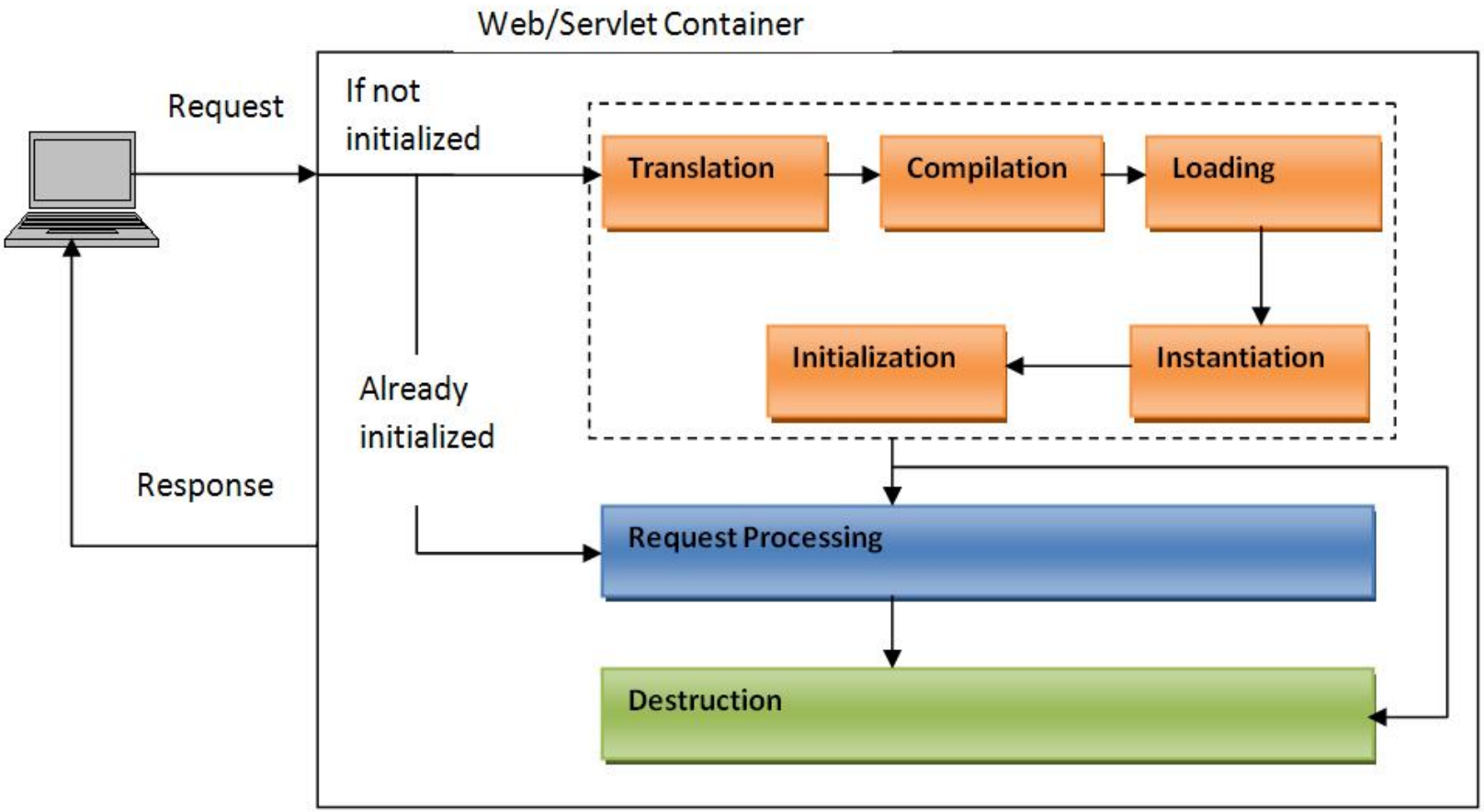
```
<html>
    <head>
        <title>Ma première page JSP </title>
    </head>
    <body>
        <p>Bonjour les amis; nous sommes: <%= new java.util.Date() %></p>
    </body>
</html>
```

La partie dynamique, encadrée par le symbole % est interprétée par le moteur JSP qui renvoie du code HTML au navigateur.

Exemple2

```
<html>
    <head>
        <title>Une page JSP</title>
        <meta name="author" content="GUINKO T. Ferdinand">
        <meta name="keywords" content="JSP, servlets, JavaServer, Pages">
        <meta name="description" content="Exemple d'une page JSP.">
    </head>
    <body>
        <h1>Une page JSP</h1>
        <ul>
            <li>Heure: <%= new java.util.Date() %>
            <li>Nom d'hôte: <%= request.getRemoteHost() %>
            <li>Numéro de session: <%= session.getId() %>
        </ul>
    </body>
</html>
```

Cycle de vie des JSP : illustration



Cycle de vie des JSP

Lorsque le container web reçoit la requête JSP qui lui a été transmise par le navigateur web, il vérifie l'existence du servlet correspondant. Si aucune le servlet de la page n'est pas trouvé, alors le container web la créée en suivant les étapes suivantes :

`Traduction` → `Compilation` → `Chargement` → `Instantiation` → `Initialisation`.

Cycle de vie des JSP : Traduction

Pendant la phase de traduction, le container web traduit le code JSP en une classe servlet. Le servlet obtenu a la même architecture que celle vue dans la section précédente.

Cycle de vie des JSP : Compilation

Le servlet obtenu est compilé. A la fin de la compilation l'on obtient du `byte code` qui sera exécuté par la `JVM`.

Cycle de vie des JSP : Chargement, Instantiation

Le reste du cycle de vie est le même que celui d'une servlet. Les métodes permettant d'initialiser, manipuler les données HTTP, et arrêter la servlet, seront respectivement :

`jspInit()`, `jspService()` et `jspDestroy()`

Les éléments de JSP

- Directives
- Scriptlets
- Actions
- Expressions
- Commentaires
- Déclarations

Les éléments de JSP : directives

Elles sont généralement placées au début de la page. On distingue 3 types de directives :

1. page
2. include
3. taglib

Les éléments de JSP : directives : page

Syntaxe :

```
<%@page attributename="value"%>
```

Liste des attributs de la directive page :

```
<%@ page
  [language="java"]
  [extends="package.class"]
  [import="{package.class | package.*}, ..."]
  [session="true | false"]
  [buffer="none | 8kb | sizekb"]
  [autoFlush="true | false"]
  [isThreadSafe="true | false"]
  [info="text"]
  [errorPage="relativeURL"]
  [contentType="mimeType [;charset=characterSet]" | "text/html; charset=ISO-8859-1"]
  [isErrorPage="true | false"]
%>
```

Les éléments de JSP : directives : page

Exemple 1 :

```
<%@page import="java.lang.io"%>
```

Les éléments de JSP : directives : page

Exemple 2 :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
import="ca.openbook.view.*"
import="com.openbook.persistence.*"
%>
<html>
    <head>
        <title>...</title>
    </head>
    <body>
        <jsp:useBean id="user" class="com.project.model.User" scope="session" />
        <jsp:useBean id="person" class="com.project.model.Person" scope="request" />
        <%
        if (Page.allowed(out,user)==1)
        {
        %>
            <h1><% out.println("add new person"); %></h1>
        <%
            AuthorView.to html table(out,jsp author list,jsp author page);
            AuthorView.new and back panel(out);
            Page.page end(out);
        }
        %>
    </body>
</html>
```

Les éléments de JSP : directive : include

Directive include

Elle permet d'inclure un autre fichier à l'intérieur du fichier JSP :

Syntaxe :

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

Exemple :

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

Les éléments de JSP : directive : taglib

Directive taglib

Elles sont utilisées pour permettre la création de balises personalisées :

Syntaxe :

```
<%taglib Uri="path" prefix="prefixToBeUsed"%>
```

Exemple :

```
<%taglib Uri="WEB-INF/mytaglib.tld" prefix="mytaglib"%>
```

Les éléments de JSP : scriptlets

Les `scriplets` sont les instructions du langage `Java` incluses dans une page `JSP`. Ils sont encadrés par les balises <% ... %>

Les éléments de JSP : actions

Une `action` est exécutée lorsqu'une requête atteint une page JSP. Une `action` est donc une requête traitant les instructions du container web ou du container servlet.

Exemples de container :

- `<jsp:include>` : inclusion dynamique d'une page ; la réponse de la page incluse est insérée dans la réponse finale ;
- `<jsp:forward>` : redirige une requête vers `n` différentes pages ;
- `<jsp:param>` : passage de paramètres à une autre page JSP ;
- `<jsp:useBean>` : integration de `java beans` dans une page JSP.

Les éléments de JSP : Expressions et commentaires

expressions

Elles sont encadrées par les balises <%=expression %>. Elles sont évaluées et leur valeur est renvoyée comme une réponse au client.

commentaires

On distingue 2 types de commentaires :

1. <%- commentaire -%> : ce commentaire est totalement ignoré par le contener web ;
2. <!- commentaire -> : le contener web traitera ce commentaire comme du code HTML et l'enverra au client.

Les éléments de JSP : Déclarations

Syntaxe :

```
<%! String name = "Tutoriel Jsp"; %>
```

Variables prédéfinies

On les appelle aussi *objets implicites*. Ces variables sont déjà prédéfinies :

- request (HttpServletRequest)
- response (HttpServletResponse)
- out (PrintWriter)
- session (HttpSession)
- application (ServletContext)
- page (Object)
- etc.

Variables prédéfinies : request

`request` est une instance de la classe `javax.servlet.HttpServletRequest`. Elle permet d'extraire des paramètres d'une requête envoyée par un client :

- le type de la requête (GET, POST, PUT, etc.) ;
- les entêtes HTTP entrantes.

Exemple 1 : Supposons que la requête suivante ait été envoyée par un navigateur web :

```
http://hostname.com/monAppWeb/maPageWeb.jsp?name=Jean+Tremblay
```

Alors l'instruction suivante :

```
Bonjour M. <b><%= request.getParameter("name") %></b>!
```

retournera comme résultat :

```
Bonjour M. <b>Jean Tremblay</b>!
```

Variables prédéfinies : request

Exemple 2 :

```
<%
    if (request.getParameter("name") == null)
    {
        out.println("Veuillez entrer votre nom s'il vous plaît: ");
    }
    else
    {
        out.println("Bonjour <b>"+request.getParameter(i)+"</b>!");
    }
%>
```

Variables prédéfinies : response

`response` est une instance de la classe `javax.servlet.HttpServletResponse`. Elle représente la réponse envoyée par le servlet au client.

Exemple :

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class RedirectServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.sendRedirect(response.encodeRedirectURL("http://www.uqar.ca"));
    }
}
```

Variables prédéfinies : out

`out` est une instance de la classe `javax.servlet.jsp.JspWriter`. Elle est utilisée pour envoyer des données au client. Certaines de ses méthodes sont `print`, `println`, `flush`, etc.

Exemple :

```
//Dans cet exemple, si le fichier dans lequel l'on tente d'écrire n'existe pas, il est automatiquement cre
try
{
    BufferedWriter out = new BufferedWriter(new FileWriter("Exemple.txt"));
    out.write("Première ligne du fichier");
    out.close();
} catch (IOException e){}
```

Variables prédéfinies : session

`session` est une instance de la classe `HttpSession`. Elle représente la session courante établie entre le client et le servlet (serveur).

Variables prédéfinies : application

`application` est une instance de la classe `ServletContext`. Elle est utilisée pour partager des données entre les servlets, les pages JSP, et les pages HTML dans une application.

Le concept de javaBeans

Définition

Composant réutilisable manipulable visuellement.

Les javaBeans sont les classes habituelles Java qui répondent aux critères suivants :

- doit posséder un constructeur sans arguments
- ils n'ont pas d'instances publiques ; les attributs privés sont accessibles publiquement via les méthodes `get()` ou `set()`.

Les bibliothèques graphiques de Java utilisent les conventions JavaBeans pour leurs composants.

Le concept de javaBeans : exemple

Exemple :

```
class Personne
{
    protected String nom;
    protected String prenom;
    public Personne() {}
    public void setNom(String nom) {this.nom=nom;}
    public void setPrenom(String nom) {this.prenom=prenom;}
    public String getNom() {return this.nom;}
    public String getPrenom() {return this.prenom;}
}
```

Le concept de javaBeans : appel d'un javaBeans

Pour faire appel et utiliser un `javaBeans` dans une page JSP, il faut définir les attributs suivants :

- `id` : nom de l'instance
- `class` : type
- `scope` : portee (page, request, session, application)

et utiliser la `directive de page jsp:useBean`.

Exemple :

```
//Cet exemple fait un appel au javaBeans précédemment créé.
<jsp:useBean id="MadMax" class="Personne" score="request" />
<%
out.println("Hello "+MadMax.getPrenom()+" "+MadMax.getNom());
%>
```