

LA TECHNOLOGIE JSP(JAVASERVER PAGES)

Les JSP (Java Server Pages) sont une technologie basée sur Java qui permettent la génération de pages web dynamiques.

La technologie JSP permet de séparer la présentation sous forme de code HTML et les traitements (logique applicative des pages Web) sous formes de classes java définissant un bean ou une servlet. Ceci est d'autant plus facile que les JSP définissent une syntaxe particulière permettant d'appeler un bean et d'insérer le résultat de son traitement dans la page HTML dynamiquement.

1. Présentation des JSP

Les JSP permettent d'introduire du code java dans des tags prédéfinis à l'intérieur d'une page HTML. La technologie JSP mélange la puissance de java côté serveur et la facilité de mise en page d'HTML côté client.

Concrètement, les JSP sont basées sur les servlets. Au premier appel de la page JSP, le moteur de JSP crée et compile automatiquement une servlet qui permet la génération de la page web. Le code HTML est repris intégralement dans la servlet. Le code java est inséré dans la servlet.

La servlet est sauvegardée puis elle est exécutée. Les appels suivants de la JSP sont beaucoup plus rapides car la servlet, conservée par le serveur, est directement exécutée.

JSP a donc un fonctionnement lourd qui se déroule en 4 temps :

- **La requête est reçu par le serveur**
- **La page demandé est traduite en servlets**
- **Puis elle est compilée**

Enfin elle est exécutée et la page html est transmise au client

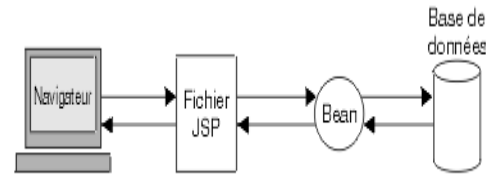
Modèles d'accès à JSP

Il est possible d'accéder aux fichiers JSP de de la manière suivante :

- Le navigateur envoie une demande de fichier JSP.

Le fichier JSP accède à des beans ou à d'autres composants qui génèrent du contenu dynamique envoyé au navigateur. Le schéma suivant illustre ce modèle d'accès pour JSP.

Demande de fichier JSP



Lorsque le serveur Web reçoit une requête pour un fichier JSP, le serveur envoie cette requête au serveur d'application. Celui-ci analyse le fichier JSP et génère une source Java qui est compilée et exécutée en tant que servlet. La génération et la compilation de la source Java s'effectue uniquement au premier appel du servlet à moins que le fichier JSP d'origine ait été mis à jour. Dans ce cas, le serveur détecte ce changement et régénère et compile le servlet avant de l'exécuter.

Il y a plusieurs manières de combiner les technologies JSP, les beans/EJB et les servlets en fonction des besoins pour développer des applications web.

Cette approche possède plusieurs avantages :

- l'utilisation de Java par les JSP permet une indépendance de la plate-forme d'exécution mais aussi du serveur web utilisé.
- la séparation des traitements et de la présentation : la page web peut être écrite par un designer et les tags Java peuvent être ajoutés ensuite par le développeur. Les traitements peuvent être réalisés par des composants réutilisables (des Java beans). alors que dans une servlet, les traitements et la présentation sont regroupés
- les JSP sont basées sur les servlets : tout ce qui est fait par une servlet pour la génération de pages dynamiques peut être fait avec une JSP.

Les JSP et les technologies concurrentes

Il existe plusieurs technologies dont le but est similaire aux JSP notamment ASP, PHP et ASP.Net. Chacune de ces technologies possèdent des avantages et des inconvénients dont voici une liste non exhaustive.

	JSP	PHP	ASP	ASP.Net
langage	Java	PHP	VBScript ou JScript	Tous les langages supportés par .Net (C#, VB.Net, Delphi, ...)
mode d'exécution	Compilé en pseudo code (byte code)	Interprété	Interprété	Compilé en pseudo code (MSIL)
principaux avantages	Repose sur la plateforme Java dont elle hérite des avantages	Open source Nombreuses bibliothèques et sources d'applications libres disponibles Facile à mettre en oeuvre	Facile à mettre en oeuvre	Repose sur la plateforme .Net dont elle hérite des avantages Wysiwyg et événementiel Code behind pour séparation affichage / traitements
principaux inconvénients	Débogage assez fastidieux Beaucoup de code à écrire	Débogage assez fastidieux Beaucoup de code à écrire support partiel de la POO en attendant la version 5	Débogage assez fastidieux Beaucoup de code à écrire Fonctionne essentiellement sur plateformes Windows Pas de POO, objet métier encapsulé dans des objets COM lourd à mettre en oeuvre	Fonctionne essentiellement sur plateformes Windows. (Voir le projet Mono pour le support d'autres plateformes)

2. Les outils nécessaires

Dans un premier temps, Sun a fourni un kit de développement pour les JSP : le Java Server Web Development Kit (JSWDK). Actuellement, Sun a chargé le projet Apache de développer l'implémentation officielle d'un moteur de JSP. Ce projet se nomme Tomcat.

En fonction des versions des API utilisées, il faut choisir un produit différent. Le tableau ci dessous résume le produit à utiliser en fonction de la version des API utilisée.

Produit	Version	Version de l'API servlet implémentée	Version de l'API JSP implémentée
JSWDK	1.0.1	2.1	1.0
Tomcat	3.2	2.2	1.1
Tomcat	4.0	2.3	1.2

Ces produits sont librement téléchargeables sur le site de Sun à l'adresse suivante :
<http://java.sun.com/products/jsp/download.html>

Configuration à l'aide des variables d'environnement

JAVA_HOME = c:\jdk // repertoire d'installation du jdk
TOMCAT_HOME = c:\tomcat // repertoire d'installation de Tomcat
CATALINA_HOME = c:\tomcat // repertoire d'installation de Tomcat

3. Les Tags JSP

Il existe trois types de tags :

- tags de directives : ils permettent de contrôler la structure de la servlet générée
- tags de scripting: il permettent d'insérer du code java dans la servlet
- tags d'actions: ils facilitent l'utilisation de composants



Attention : Les tags sont sensibles à la casse.

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

Les spécifications des JSP définissent trois directives :

- **page** : permet de définir des options de configuration
- **include** : permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet
- **taglib** : permet de définir des tags personnalisés

Leur syntaxe est la suivante :

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

3.2. La directive page

Cette directive doit être utilisée dans toutes les pages JSP : elle permet de définir des options qui s'applique 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.

Exemple :

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

Les options sont :

- **autoFlush="true|false"**

Cette option indique si le flux en sortie de la servlet doit être vidé quand le tampon est plein. Si la valeur est false, une exception est levée dès que le tampon est plein. On ne peut pas mettre false si la valeur de buffer est none.

- **buffer="none|8kb|sizekb"**

Cette option permet de préciser la taille du buffer des données générées contenues par l'objet out de type JspWriter.

- **contentType="mimeType [; charset=characterSet]" | "text/html; charset=ISO-8859-1"**

Cette option permet de préciser le type MIME des données générées.

Cette option est équivalente à <% response.setContentType("mimeType"); %>

Les options définies par cette directive sont de la forme option=valeur.

Option	Valeur	Valeur par défaut	Autre valeur possible
autoFlush	Une chaîne	«true»	«false»
buffer	Une chaîne	«8kb»	«none» ou «nnnkb» (nnn indiquant la valeur)
contentType	Une chaîne contenant le type mime		
errorPage	Une chaîne contenant une URL		
extends	Une classe		
import	Une classe ou un package.*		
info	Une chaîne		
isErrorPage	Une chaîne	«false»	«true»
isThreadSafe	Une chaîne	«true»	«false»
language	Une chaîne	«java»	
session	Une chaîne	«true»	«false»

- **errorPage="relativeURL"**

Cette option permet de préciser la JSP appelée au cas où une exception est levée

Si l'URL commence pas un '/', alors l'URL est relative au répertoire principale du serveur web sinon elle est relative au répertoire qui contient la JSP

- **extends="package.class"**

Cette option permet de préciser la classe qui sera la super classe de l'objet java créé à partir de la JSP.

- **import= "{ package.class | package.* }, ..."**

Cette option permet d'importer des classes contenues dans des packages utilisées dans le code de la JSP. Cette option s'utilise comme l'instruction import dans un source java.

Chaque classe ou package est séparée par une virgule.

Cette option peut être présente dans plusieurs directives page.

- **info="text"**

Cette option permet de préciser un petit

descriptif de la JSP. Le texte fourni sera renvoyé par la méthode `getServletInfo()` de la servlet générée.

- `isErrorPage="true|false"`

Cette option permet de préciser si la JSP génère un page d'erreur. La valeur `true` permet d'utiliser l'objet `Exception` dans la JSP

- `isThreadSafe="true|false"`

Cette option indique si la servlet générée sera multithread : dans ce cas, une même instance de la servlet peut gérer plusieurs requêtes simultanément. En contre partie, elle doit gérer correctement les accès concurrents aux ressources. La valeur `false` impose à la servlet générée d'implémenter l'interface `SingleThreadModel`.

- `language="java"`

Cette option définit le langage utilisé pour écrire le code dans la JSP. La seule valeur autorisée actuellement est «`java`».

- `session="true|false"`

Cette option permet de préciser si la JSP est incluse dans une session ou non. La valeur par défaut (`true`) permet l'utilisation d'un objet session de type `HttpSession` qui permet de gérer une session.

- Valeurs possibles des options de la directive page :

- `<%@ page language="java"`
- `<%@ page import="java.util.*, java.net.*" %>`
- `<%@ page contentType="text/plain" %>`
- `<%@ page session="true|false" %>`
- `<%@ page errorPage="pathToErrorPage" %>`
- `<%@ page isErrorPage="true|false" %>`
- `<%@ page ...`

Exemple de directive page

```
<%@ page language="Java"
import="java.rmi.*,java.util.*"
session="true" buffer="12kb" autoFlush="true"
info="Ma directive page" errorPage="error.jsp"
isErrorPage="false" isThreadSafe="false" %>
```

```
<html>
<head>
```

```
<title>Page de test de la directive page</title>
</head>
<body>
<h1> Page de test de la directive page</h1>
    Bla bla bla...
</body>
</html>
```

3.3. La directive include

Cette directive permet d'inclure un fichier dans le 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. Si le fichier inclus est un fichier HTML, celui-ci ne doit pas contenir de tag `<HTML>`, `</HTML>`, `<BODY>` ou `</BODY>` qui ferait double emploi avec ceux présents dans le fichier JSP. Ceci impose d'écrire des fichiers HTML particuliers uniquement pour être inclus dans les JSP : ils ne pourront pas être utilisés seuls.

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.

Exemple :

bonjour.htm :

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

Test1.jsp :

```
<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.htm"%>
<p align="center">fin</p>
</BODY>
</HTML>
```

Pour visualiser la JSP, il faut saisir l'URL
<http://localhost:8080/examples/jsp/test/Test1.jsp>
dans un navigateur.



Attention : un changement dans le fichier inclus ne provoque pas une régénération et une compilation de la servlet correspondant à la JSP.
Pour insérer un fichier dynamiquement à l'exécution de la servlet il faut utiliser le tag **<jsp:include>**.

3.4. Les tags de scripting

Ces tags permettent d'insérer du code java qui sera inclus dans la servlet générée à partir de la JSP. Il existe trois tags pour insérer du code java :

- le tag de déclaration : le code java est inclus dans le corps de la servlet générée. Ce code peut être la déclaration de variables d'instances ou de classes ou la déclaration de méthodes.
- le tag d'expression : évalue une expression et insère le résultat sous forme de chaîne de caractères dans la page web générée.
- le tag de scriptlets : par défaut, le code java est inclus dans la méthode service() de la servlet.

Il est possible d'utiliser dans ces tags plusieurs objets définis par les JSP.

3.4.1. 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 %>
```

Exemple :

```
<%! int i = 0; %>
<%! dateDuJour = new java.util.Date(); %>
```

Les variables ainsi déclarées peuvent être utilisées dans les tags d'expressions et de scriptlets.

Il est possible de déclarer plusieurs variables dans le même tag en les séparant avec des ';'.

Ce tag permet aussi d'insérer des méthodes dans le corps de la servlet.

Exemple :

```
<HTML>
<HEAD><TITLE>Test</TITLE>
</HEAD>
<BODY>
<%!
```

```
int minimum(int val1, int val2) {
    if (val1 < val2) return val1;
    else return val2;
}
%>
<% int petit = minimum(5,3);%>
<p>Le plus petit de 5 et 3 est <%= petit %></p>
</BODY>
</HTML>
```

3.4.2. 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 %>

Le signe '=' doit être collé au signe '%'



Attention : il ne faut pas mettre de ';' à la fin de l'expression

Exemple : Insertion de la date dans la page HTML

```
<%@ page import="java.util.*" %>
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<p align="center">Date du jour :
<%= new Date() %>
</p>
</BODY>
</HTML>
```

Résultat :

Date du jour : Thu Feb 15 11:15:24 CET 2001

L'expression est évaluée et convertie en chaîne avec un appel à la méthode toString(). Cette chaîne est insérée dans la page HTML en remplacement du tag. Il est ainsi possible que le résultat soit une partie ou la totalité d'un tag HTML ou même JSP.

Exemple :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<%= "<H1>" %>
Bonjour
<%= "</H1>" %>
</BODY>
</HTML>
```

Résultat : code HTML généré

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<H1>
Bonjour
</H1>
</BODY>
</HTML>
```

3.4.3. Les variables implicites

Les spécifications des JSP définissent plusieurs objets utilisables dans le code dont les plus utiles sont :

Object	Classe	Role
out	javax.servlet.jsp.JspWriter	Flux en sortie de la page HTML générée
request	javax.servlet.http.HttpServletRequest	Contient la requête
response	javax.servlet.http.HttpServletResponse	Contient la réponse
session	javax.servlet.http.HttpSession	Gère la session

3.4.4. Le tag des scriptlets <% ... %>

Ce tag contient du code java : un scriptlet.

La syntaxe est la suivante :

<% code java %>

Exemple :

```
<%@ page import="java.util.Date"%>
<html>
<body>
<%! Date dateDuJour; %>
<% dateDuJour = new Date();%>
Date du jour : <%= dateDuJour %><BR>
</body>
</html>
```

Par défaut, le code inclus dans le tag est inséré dans la méthode service de la servlet générée à partir de la JSP.

Ce tag ne peut pas contenir autre chose que du code java : il ne pas par exemple contenir de tags HTML ou JSP. Pour faire cela, il faut fermer le tag du scriptlet, mettre le tag HTML ou JSP puis de nouveau commencer un tag de scriptlet pour continuer le code.

Exemple :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<% for (int i=0; i<10; i++) { %>
<%= i %> <br>
<% }%>
</BODY>
</HTML>
```

Résultat : la page HTML générée

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
0 <br>
1 <br>
2 <br>
3 <br>
4 <br>
5 <br>
6 <br>
7 <br>
8 <br>
9 <br>
</BODY>
</HTML>
```

3.5. Les tags de commentaires

Il existe deux types de commentaires avec les JSP :

- les commentaires visibles dans le code HTML
- les commentaires invisibles dans le code HTML

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

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

La syntaxe est la suivante :

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

Exemple :

```
<%@ page import="java.util.*" %>
```

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<!-- Cette page a ete generee le <%= new Date() %> -->
<p>Bonjour</p>
</BODY>
</HTML>
```

Résultat :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<!-- Cette page a ete generee le Thu Feb 15
11:44:25 CET 2001 -->
<p>Bonjour</p>
</BODY>
</HTML>
```

Le contenu d'une expression incluse dans des commentaires est dynamique : sa valeur peut changer à chaque génération de la page en fonction de son contenu.

3.5.2 . Les commentaires cachés <%-- ... --%>

Les commentaires cachés sont utilisés pour documenter la page JSP. Leur contenu est ignoré par le moteur de JSP et ne sont donc pas reconduit dans les données HTML générées.

La syntaxe est la suivante :

<%-- commentaires --%>

Exemple :

```
<HTML>
<HEAD>
<TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<%-- Commentaires de la page JSP --%>
<p>Bonjour</p>
</BODY>
</HTML>
```

Résultat :

```
<HTML>
<HEAD><TITLE>Essai de page JSP</TITLE>
</HEAD>
<BODY>
<p>Bonjour</p>
</BODY>
</HTML>
```

Ce tag peut être utile pour éviter l'exécution de code lors de la phase de débogage.

EXEMPLE FINAL

```
<%@ page import = "java.util.Date, java.sql.*" %>
<HTML>
<BODY>
<%
// Déclaration
! int a, b,s ;
Date d ;
int produit (int a, int b)
{ return a*b ;}
%>
<%
//Scripplet
a=2;
b=5;
s=a+b;
d=new Date();
int p = produit (a,b);
System.out.println("<B> Produit = "+ p+ "</B>");
%>
<!-- Tague d'expression -->
Somme = <%= s %> <BR>
Nous sommes le <%=d%>
</BODY>
</HTML>
```

Envoi des données à partir d'un formulaire**Page1.html**

```
<FORM METHOD="POST/GET" ACTION = "page2.jsp" >
Nom : <input type=text name="nom">
Prenom : <input type=text name="prenom">
<input type=submit value="Valider">
</FORM>
```

ou bien

```
<a href = "page2.jsp" ? nom="rabe"& prenom ="koto" >
Envoyer </a>
```

page2.jsp

```
<HTML>
<BODY>
<% String nom = request.getParameter("nom");
String prenom = request.getParameter("prenom");
System.out.println("<B> Anarana= "+ nom+ "</B><BR>");
System.out.println("<B> Fanampin'anarana= "+ prenom+
"</B>");%>
<H2>Bonjour <%= nom %></H2>
</BODY>
</HTML>
```

Exemple 2 : Formulaire HTML

```
<form action="calcul.jsp"
method=post>
```

```
Nom : <input type=text name="nom"> <br>
Quelles sont vos connaissances de HTML ?
<input type="radio" name="choice" value="1"
checked>faibles
<input type="radio" name="choice" value="2">moyennes
<input type="radio" name="choice" value="3">bonnes
<br>
Indiquez votre expertise en programmation:
<input type="radio" name="choice2" value="1"
checked>absente
<input type="radio" name="choice2" value="2">moyenne
<input type="radio" name="choice2" value="3">bonne
<P>
<input type="submit"
value="Voir le résultat!">
</form>
```

```
JAVA: calcul.jsp
<%
// Parameters come as strings
String choice = request.getParameter("choice");
String choice2 = request.getParameter("choice2");
String nom = request.getParameter("nom");
out.println(nom + ", votre input était: question a=" +
choice + ", question b=" + choice2);

// Integer.parseInt() translates a string to an Integer
int score = Integer.parseInt(choice) +
Integer.parseInt(choice2);

out.println("<h3>Votre score est de " + score + "</h3>");

if (score < 3) {
out.print("<p>Vous êtes un débutant</p>");
} else if (score < 5) {
out.print("<p>Vous avez un niveau moyen</p>");
} else {
out.print("<p>Vous êtes un expert !</p>");
}
%>
```

3.6. Les tags d'actions

3.6.1. Le tag <jsp:useBean>

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. Le bean peut notamment assurer l'accès à une base de données. L'utilisation des beans permet de simplifier les traitements inclus dans la JSP. Lors de l'instanciation d'un bean, on précise la portée du bean. Si le bean demandé est déjà instancié pour la portée précisée alors il n'y pas de nouvelle instance du bean qui est créée mais sa référence est simplement renvoyée : le tag <jsp:useBean> n'instancie pas obligatoirement un objet. La syntaxe est la suivante :

```
<jsp:useBean
id="beanInstanceName"
scope="page|request|session|application"
{ class="package.class" |
type="package.class" |
class="package.class" type="package.class" |
beanName="{package.class | <%= expression %>}"
type="package.class"
}
/> |
> ...
</jsp:useBean>
}
```

L'attribut id permet de donner un nom à la variable qui va contenir la référence sur le bean.

L'attribut scope permet de définir la portée durant laquelle le bean est défini et utilisable. La valeurs de cette attribut détermine la manière dont le tag localise ou instancie le bean. Les valeurs possibles sont :

Valeur	Rôle
page	c'est la valeur par défaut. Le bean est utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus.
Request	Le bean est accessible durant la durée de vie de la requête. La méthode getAttribute de l'objet request permet d'obtenir une référence sur le bean.
session	le bean est utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instancié le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent. La JSP qui créé le bean doit avoir l'attribut session = « true » dans sa directive page.
application	le bean est utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instancié le bean. Le bean n'est instancié que lors du rechargement de l'application.

L'attribut class permet d'indiquer la classe du bean

L'attribut type permet de préciser le type de la variable qui va contenir la référence du bean. La valeur indiquée doit obligatoirement être une super classe du bean ou une interface implémentée par le bean (directement ou par héritage)

L'attribut beanName permet d'instancier le bean grâce à la méthode instanciate() de la classe Beans.

Exemple :

```
<jsp:useBean id="monBean" scope="session"
class="test.MonBean" />
```

Dans cet exemple, une instance de MonBean est instancié un seul est unique fois lors de la session. Dans la même session, l'appel du tag <jsp:useBean> avec le même bean et la même portée ne feront que renvoyer

l'instance créée. Le bean est accessible durant toute la session.

Le tag `<jsp:useBean>` recherche si une instance du bean existe avec le nom et la portée précisée. Si elle n'existe pas, alors une instance est créée. Si il y a instantiation du bean, alors les tags `<jsp:setProperty>` inclus dans le tag sont utilisés pour initialiser les propriétés du bean sinon ils sont ignorés. Les tags inclus entre les tags `<jsp:useBean>` et `</jsp:useBean>` ne sont exécutés que si le bean est instancié.

Exemple :

```
<jsp:useBean id="monBean" scope="session"
class="test.MonBean" >
<jsp:setProperty name="monBean" property="*" />
</jsp:useBean>
```

Cet exemple a le même effet que le précédent avec une initialisation des propriétés du bean lors de son instantiation avec les valeur des paramètres correspondants.

3.6.2 . Le tag `<jsp:setProperty>`

Le tag `<jsp:setProperty>` permet de mettre à jour la valeur d'un ou plusieurs attributs d'un Bean. Le tag utilise le setter (méthode `getXXX()` ou `XXX` est le nom de la propriété) pour mettre à jour la valeur. Le bean doit exister grâce à un appel au tag `<jsp:useBean>`.

Il existe trois façon de mettre à jour les propriétés soit à partir des paramètres de la requête soit avec une valeur :

- alimenter automatiquement toutes les propriétés avec les paramètres correspondants de la requête
- alimenter automatiquement une propriété avec le paramètre de la requête correspondant
- alimenter une propriété avec la valeur précisée

La syntaxe est la suivante :

```
<jsp:setProperty name="beanInstanceName"
{ property="*" |
property="propertyName" [ param=" parameterName" ] |
property="propertyName" value="{string | <%=
expression%>}"
}
/>
```

L'attribut `name` doit contenir le nom de la variable qui contient la référence du bean. Cette valeur doit être identique à celle de l'attribut `id` du tag `<jsp:useBean>` utilisé pour instancier le bean.

L'attribut `property="*"` permet d'alimenter automatiquement les propriétés du bean avec les paramètres correspondants contenus dans la requête. Le nom des propriétés et le nom des paramètres doivent être identiques.

Comme les paramètres de la requête sont toujours fournis sous forme de String, une conversion est réalisée en utilisant la méthode `valueOf()` du wrapper du type de la propriété.

Exemple :

```
<jsp:setProperty name="monBean"
property="*" />
```

L'attribut `property="propertyName"`

[`param="parameterName"`] permet de mettre à jour un attribut du bean. Par défaut, l'alimentation est faite automatiquement avec le paramètre correspondant dans la requête. Si le nom de la propriété et du paramètre sont différents, il faut préciser l'attribut `property` et l'attribut `param` qui doit contenir le nom du paramètre qui va alimenter la propriété du bean.

Exemple :

```
<jsp:setProperty name="monBean"
property="nom"/>
```

L'attribut `property="propertyName" value="{string | <%= expression %>}"` permet d'alimenter la propriété du bean avec une valeur particulière.

Exemple :

```
<jsp:setProperty name="monBean"
property="nom" value="toto" />
```

Il n'est pas possible d'utiliser `param` et `value` dans le même tag.

Ce tag peut être utilisé entre les tags `<jsp:useBean>` et `</jsp:useBean>` pour initialiser les propriétés du bean lors de son instantiation.

EXEMPLE FINAL DE USEBEAN

Bean.java (à placer dans un repertoire Essai du repertoire classes de webInf)

```
package essai;
public class Bean{
// PROPRIETES
String login;
String password;
Bean(){

// METHODES OBLIGATOIRES
public void setLogin (String login ) {this.login=login;}
public void setPassword (String p ) {password=p;}

public String getLogin ( ) { return login;}
public String getPassword (String p ) { return
password;}}
```

```
// AUTRES METHODES
```

```
public void connectBase() {.....}
}
```

page1.jsp

```
<a href = "page2.jsp" ? login="rabe"& password ="rabe"
> Cliquer ici </a>
```

ou bien

```
<FORM METHOD = « POST » ACTION
= « page2.jsp » >
Nom : <input type=text name="login"> <br>
Mot de passe : <input type=password
name="password"> <br>
<input type=submit value="Valider">
</FORM>
```

page2.jsp

```
<jsp :useBean id = « Bean » class = »essai.Bean « />
<!-- Les noms des parameters de la requête sont
identiques à ceux des propriétés du bean -->
<jsp:setProperty name="Bean" property ="*" />

<jsp:setProperty name="Bean" property ="login" />
<jsp:setProperty name="Bean" property ="password" />

<% Bean.connectBase(); %>

<!-- Initialisation des propriétés du bean -->

<jsp:setProperty name="Bean" property ="login"
value="koto" />
<jsp:setProperty name="Bean" property ="password"
value="koto" />

<!-- Les noms des parameters de la requête sont
différents de ceux des propriétés du bean -->
<!-- <a href = "page2.jsp" ? nom="rabe"& motdepass
="rabe" > Cliquer ici </a> -->

<jsp:setProperty name="Bean" property ="login"
param="nom" />

<jsp:setProperty name="Bean" property ="password"
param="motdepass" />
```

3.6.3. Le tag <jsp:getProperty>

Le tag <jsp:getProperty> permet d'obtenir la valeur d'un attribut d'un Bean. Le tag utilise le getter (méthode getXXX() ou XXX est le nom de la propriété) pour obtenir la valeur et l'insérer dans la page HTML généré. Le bean doit exister grâce à un appel au tag <jsp:useBean>.

La syntaxe est la suivante :

```
<jsp:getProperty name="beanInstanceName"
property=" propertyName" />
```

L'attribut name indique le nom du bean tel qu'il a été déclaré dans le tag <jsp:useBean>.

L'attribut property indique le nom de la propriété dont on veut la valeur.

Remarque : avec Tomcat 3.1, l'utilisation du tag <jsp:getProperty> sur un attribut dont la valeur est null n'affiche rien alors que l'utilisation d'un tag d'expression retourne « null ».

Exemple :

TestBean.jsp

```
<html>
<HEAD>
<TITLE>Essai d'instanciation d'un bean dans une
JSP</TITLE>
</HEAD>
<body><p>Test d'utilisation d'un Bean dans une JSP
</p>
<jsp:useBean id="personne" scope="request"
class="test.Personne" />
<p>nom initial = <%=personne.getNom() %></p>
<p>nom initial = <jsp:getProperty name="personne"
property="nom" /></p>
<jsp:setProperty name="personne" property="nom"
value="mon nom est DUPONT" />
<p>nom mise à jour = <jsp:getProperty
name="personne" property="nom" /></p>
<% personne.setNom("mon nom est RABE");%>
<p>nom mise à jour = <%= personne.getNom() %></p>
<% personne.setNom(null);%>
<p>nom mis à jour = <jsp:getProperty name="personne"
property="nom" /></p>
<p>nom mis à jour = <%= personne.getNom() %></p>
</body>
</html>
```

Personne.java

```
package test;
public class Personne {
    private String nom;
    private String prenom;
    public Personne() {
        this.nom = "nom par default";
        this.prenom = "prenom par default";
    }
    public void setNom (String nom) {
        this.nom = nom;
    }
    public String getNom() { return (this.nom);
    }
    public void setPrenom (String prenom) {
        this.prenom = prenom;
    }
    public String getPrenom () {
        return (this.prenom);
    }
}
```

Remarques

Code java

```
<%@ page import="test.Personne" %>
<html>
<body>
<%
Personne personne = new Personne ();
personne.setNom(request.getParameter("nom");
personne.setPrenom(request.getParameter("prenom");
%>
<p>Votre nom = <%= personne.getNom() %></p>
<p>Votre prenom = <%= personne.getPrenom()
%></p>
</body>
</html>
```

Tags JSP (équivalent)

```
<jsp:useBean id="personne" scope="request"
class="test.Personne" />
<jsp:setProperty name="personne" property="nom"
value="*" />
<p>Votre nom = <jsp:getProperty name="personne"
property="nom" /></p>
<p>Votre nom = <jsp:getProperty name="personne"
property="prenom" /></p>
```

Pour tester cette JSP avec Tomcat, il faut compiler le bean `Personne` dans le répertoire `\webapps\examples\web-inf\classes\test` et placer le fichier `TestBean.jsp` dans le répertoire `t\webapps\examples\jsp\test`.

3.6. 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>
```

L'option `page` doit contenir la valeur de l'URL de la ressource vers laquelle la requête va être redirigée.

Cette URL est absolue si elle commence par un `'/'` ou relative à la JSP sinon. Dans le cas d'une URL absolue,

c'est le serveur web qui détermine la localisation de la ressource.

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

Exemple :

```
Test8.jsp
<html>
<body>
<p>Page initiale appelée</p>
<jsp:forward page="forward.htm"/>
</body>
</html>

forward.htm
<HTML>
<HEAD>
<TITLE>Page HTML</TITLE>
</HEAD>
<BODY>
<p><table border="1" cellpadding="4"
cellspacing="0" width="30%" align=center >
<tr bgcolor="#A6A5C2">
<td align="center">Page HTML forwardée</Td>
</tr>
</table></p>
</BODY>
</HTML>
```

Le fichier `forward.htm` doit être dans le même répertoire que la JSP. Lors de l'appel à la JSP, c'est le page HTML qui est affichée. Le contenu généré par la page JSP n'est pas affichée.

Exemple : Utilisation des paramètres

```
Page1.jsp
<html>
<body>
<p>Page initiale appelée</p>
<jsp:forward page="page2.jsp">
<jsp:param name="nom" value="rabe" />
</jsp:forward>
</body>
</html>
```

```
Page2.jsp
<HTML>
<BODY>
<%
String npm = request.getParameter("nom");
out.print( nom);
%>
</BODY>
</HTML>
```

3.7. 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` 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 et 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.

Exemple :

```
<html>
<body>
  <jsp:include page="bandeau.jsp"/>
  <H1>Bonjour</H1>
  <jsp:include page="pied.jsp"/>
</body>
</html>
```

Il est possible de fournir des paramètres à la servlet ou à la JSP appelée en utilisant le tag `<jsp:param>`.

3.8. Le tag `<jsp:plugin>`

Ce tag permet la génération du code HTML nécessaire à l'exécution d'une applet en fonction du navigateur : un tag HTML `<Object>` ou `<Embed>` est généré en fonction de l'attribut `User-Agent` de la requête.

Le tag `<jsp:plugin>` possède trois attributs obligatoires :

Attribut	Rôle
<code>code</code>	permet de préciser le nom de classe
<code>codebase</code>	contient une URL précisant le chemin absolu ou relatif du répertoire contenant la classe ou l'archive
<code>type</code>	les valeurs possibles sont <code>applet</code> ou <code>bean</code>

Il possède aussi plusieurs autres attributs optionnels dont les plus utilisés sont :

Attribut	Rôle
<code>align</code>	permet de préciser l'alignement de l'applet : les valeurs possibles sont <code>bottom</code> , <code>middle</code> ou <code>top</code>
<code>archive</code>	permet de préciser un ensemble de ressources

	(bibliothèques jar, classes, ...) qui seront automatiquement chargées. Le chemin de ces ressources tient compte de l'attribut <code>codebase</code>
<code>Height</code>	précise la hauteur de l'applet en pixel ou en pourcentage
<code>hspace</code>	précise le nombre de pixels insérés à gauche et à droite de l'applet
<code>jrversion</code>	précise la version minimale du jre à utiliser pour faire fonctionner l'applet
<code>Name</code>	précise le nom de l'applet
<code>vspace</code>	précise le nombre de pixels insérés en haut et en bas de l'applet
<code>Width</code>	précise la longueur de l'applet en pixel ou en pourcentage

Pour fournir un ou plusieurs paramètres, il faut utiliser dans le corps du tag `<jsp:plugin>` le tag `<jsp:params>`. Chaque paramètre sera alors défini dans un tag `<jsp:param>`.

Exemple :

```
<jsp:plugin type="applet" code="MonApplet.class"
  codebase="applets"
  jrversion="1.1" width="200"
  height="200" >
  <jsp:params>
    <jsp:param name="couleur" value="eeeeee" />
  </jsp:params>
</jsp:plugin>
```

Le tag `<jsp:fallback>` dans le corps du tag `<jsp:plugin>` permet de préciser un message qui sera affiché dans les navigateurs ne supportant pas le tag HTML `<Object>` ou `<Embed>`.

4. La gestion des erreurs

Lors de l'exécution d'une page JSP, des erreurs peuvent survenir. Chaque erreur se traduit par la levée d'une exception. Si cette exception est capturée dans un bloc `try/catch` de la JSP, celle-ci est traitée. Si l'exception n'est pas capturée dans la page, il y a deux possibilités selon qu'une page d'erreur soit associée à la page JSP :

- sans page d'erreur associée, la pile d'exécution de l'exception est affichée
- avec une page d'erreur associée, une redirection est effectuée vers cette JSP

La définition d'une page d'erreur permet de la préciser dans l'attribut **errorPage** de la directive `page` des autres JSP de l'application. Si une exception est levée dans les traitements d'une de ces pages, la JSP va automatiquement rediriger l'utilisateur vers la page d'erreur précisée.

La valeur de l'attribut `errorPage` de la directive `page` doit contenir l'URL de la page d'erreur. Le plus simple est de définir cette page à la racine de l'application web et de faire précéder le nom de la page par un caractère '/' dans l'url.

Exemple :

```
<%@ page errorPage="/mapagederreur.jsp" %>
```

5. La définition d'une page d'erreur

Une page d'erreur est une JSP dont l'attribut **`isErrorPage` est égal à `true` dans la directive `page`**. Il est possible dans une telle page d'afficher un message d'erreur personnalisé mais aussi d'inclure des traitements liés à la gestion de l'exception : ajouter l'exception dans un journal, envoi d'un mail pour son traitement, ...

Exemple :

```
<%@ page language="java"
contentType="text/html" %>
%@ page isErrorPage="true" %>
<html>
<body>
<h1>Une erreur est survenue lors des
traitements</h1>
<p><%= exception.getMessage() %></p>
</body>
</html>
```

Exemple final :

Exemple :

Page1.jsp

```
<%@ page errorPage="pageerreur.jsp" %>
<html>
<body>
Calcul division par zero <%=10/0 %>
</body>
```

Pageerreur.jsp

```
@ page isErrorPage="true" %>
<html>
<body>
<h1>Tentative de division par zéro</h1>
</body>
</html>
```

5. Les cookies en JSP

- Un cookie est une information envoyée par un serveur web à un navigateur et sauvegardée par celui-ci sur le disque de sa machine

- Le navigateur retourne cette information (inchangée) lorsqu'il visite de nouveau le même site (URL).
- Ayant lu cette information, le serveur peut identifier le client et lui fournir un certain nombre de facilités :
 - ✓ achat en ligne
 - ✓ authentification
 - ✓ personnalisation de portail
 - ✓ diriger les recherches d'un moteur
 - ✓ Identification des utilisateurs (e-commerce)
- Durée de vie réglable
- Pas de menace sécuritaire (Jamais interprété ou exécuté : pas de virus)
- Un cookie est limité à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque
- Bien pour rendre privées des données non sensibles
 - nom, adresse,
- Eviter la saisie d'informations à répétition
 - login, password, adresse, téléphone...

Utilisation de cookies en JSP

En JSP, un cookie est un objet de la classe **`javax.servlet.http.Cookie`** utilisée pour créer un cookie ayant pour Constructeur :

```
public Cookie(String name,String value)
```

- Création d'un cookie

```
Cookie unCookie = new Cookie("nom","martin");
response.addCookie(unCookie);
```

- Récupération d'un cookie

```
Cookie[] cookies = request.getCookies();
if (cookies != null)
for ( int i=0;i<cookies.length;i++ ){
if (cookies[i].getName().equals("nom")){
String valeur = cookies[i].getValue();
break;
}
}
}
```

• Afficher la liste des cookies

```
Cookie cookie = null;
Cookie[] cookieArray = request.getCookies();
if(cookieArray != null)
{
%>
<p><b>Cookies Information</b></p>
<%
```

```

for(int i=0; i<cookieArray.length; i++)
{
    cookie = cookieArray[i];
    out.println("CookieName :" + cookie.getName());
    out.println("CookieValue :" + cookie.getValue());
}
}
else
{
    out.println("Cookies is not added in the response");
}
%>

```

Attributs des cookies

Un cookie est défini par son nom auquel est associée une valeur.
 Il est possible d'y ajouter certains attributs et de les lire avec les méthodes **cookie.setXxx(...)**
cookie.getXxx(...) suivantes :

Méthode	Description
getComment ()	Retourne le commentaire décrivant le but de ce cookie, ou null si aucun commentaire n'a été définie.
getMaxAge ()	Renvoie l'âge maximal spécifié du cookie.
getName ()	Retourne le nom du cookie.
getPath ()	Renvoie le préfixe de toutes les URL pour lesquelles ce cookie est ciblé.
getValue ()	Retourne la valeur du cookie.
setComment (String)	Si un navigateur Web présente ce cookie à un utilisateur, le but de l'cookie sera décrit en utilisant ce commentaire.
setMaxAge (int)	Définit l'âge maximum du cookie (durée de vie dans le disque) . Le cookie expirera après que plusieurs secondes se sont écoulées. Les valeurs négatives indiquent le comportement par défaut: le cookie n'est pas stocké persistante, et sera supprimé lorsque le navigateur Web de l'utilisateur sort. Une valeur de zéro provoque le cookie à supprimer
setPath (String uri)	- définit le chemin dans le domaine pour lequel le cookie est valide - par défaut, le cookie est valide pour la page qui le définit et toutes les pages du répertoire du dessous

	- si uri="/", le cookie est valide pour toutes les pages du serveur
setValue (String)	Définit la valeur du cookie. Valeurs avec différents caractères spéciaux (espace blanc, supports et parenthèses, le signe égal, virgule, guillemet, barres obliques, des points d'interrogation, le signe «à», et point-virgule) doit être évitée. Les valeurs vides ne peuvent pas se comporter de la même manière sur tous les navigateurs.

Exemple d'utilisation de cookies

Nous allons écrire du code dans le fichier JSP pour définir et afficher le cookie.

Créer un formulaire

Voici le code du formulaire (**cookieform.jsp**) qui invite l'utilisateur à entrer son nom.

```

<% @ Page language = "java"%>
<html>
<head>
    Formulaire d'entrée <title> Cookie </ title>
</ Head>
<body>
    <form method="post" action="setcookie.jsp">
    <p> <b> Entrez Votre Nom: </ b> <input type="text"
    name="username"> Photos
    <input type="submit" value="Submit">

</ Form>

</ Body>

```

Voici le code du fichier de **setcookie.jsp** qui permet de créer le cookie:

```

<% @ Page language = "java" import = "java.util.
**"%>
<%
String username = request.getParameter
("username");
if (username == null) username = "";

Date now = new Date ();
String timestamp = now.toString ();
Cookie cookie = new Cookie ("username",
username);
cookie.setMaxAge (365 * 24 * 60 * 60);
response.addCookie (cookie);
%>

<html>
<head>
<title> Cookie enregistré </ title>
</ Head>
<body>

```

```
<p> <a href="showcookievalue.jsp"> Page
suivante pour afficher la valeur du cookie </ a> <p>
</ Body>
```

Voici le code de la page qui affiche la cookie
(**showcookievalue.jsp**)

```
<% @ Page language = "java"%>
<%
String cookieName = "username";
Cookie cookies [] = request.getCookies ();
Cookie myCookie = null;
if (cookies != null)
{
for (int i = 0; i < cookies.length; i++)
{
if (cookies [i].getName().equals (cookieName))
{
myCookie = cookies[i];
break;
}
}
}
%>
<html>
<head>
<title> Afficher les fichiers Cookie </ title>
</ Head>
<body>
<%
if (myCookie == null) {
%>
Pas de Cookie trouvé avec le nom <% = cookieName%>
<%
} else {
%>
<p> Bienvenue: <% = myCookie.getValue ()%>.
<%
}
%>
</ Body> </html>
```

6. Les sessions en JSP

Exemple 1 :

FICHER session.jsp

```
<%@ page session="true" %>
<html>
<head>
<title> Page du debut de la session </title>
</head>
<body>
<form method="post" action="enregistreSession.jsp">
Quell est votre nom? <input type="text"
name="nomUsager" size="20" />
<input type="submit" value="Soumettre"/>
</form>
</body>
</html>
```

ENREGISTREMENT DE VARIABLE DANS LA SESSION

FICHER enregistreSession.jsp

```
<%@ page session="true" %>
<%
String nomForm =
request.getParameter( "nomUsager" );
session.setAttribute( "nom", nomForm );
%>
<html>
<head>
<title> Exemple de session - page d'enregistrement de
variable </title>
</head>
<body>
<a href="recupereSession.jsp">Continue</a>
</body>
</html>
```

RECUPERATION DE VARIABLE DANS LA SESSION

FICHER recupereSession.jsp

```
<%@ page session="true" %>
<% String nomUsager = (String)
session.getAttribute("nom"); %>
<html>
<head>
<title> Exemple de session - page de recuperation de
variable </title>
</head>
<body>
Salut, <%= nomUsager %>
</body>
</html>
```

Exemple 2 : VARIABLE DE TYPE PRIMITIF STOCKEE DANS UNE SESSION

```
<%@ page session="true" %>
<%
int compteur;
Integer compteurInteger = (Integer)
session.getAttribute("compteur");
compteur = (compteurInteger != null) ?
compteurInteger.intValue(): 0;
compteur++;
session.setAttribute("compteur",new
Integer(compteur));
%>
<html>
<head>
<title> Exemple de session </title>
</head>
<body>
<table>
<th align="left" >Nombre de fois que cette page a
&eacute;t&eacute; visit&eacute;e </th>
<td> <%= compteur %> </td>
</tr>
</table>
</body>
```

```
</html>
```

Exemple3 :

```
T ACTION="page2jsp">
```

```
ME="nom">
```

```
ME="solde">
```

```
VALUE="SUBMIT">
```

```
)
```

```
omme");
```

```
tSolde="0" ;
```

```
de">
```

```
getParameter("solde"));
```

```
om,solde) dans une table  
soldes
```

```
omme",TotSolde) ;
```

```
ge1.jsp">
```

1) Exemple Bean1.jsp

```
<html>
```

```
<body bgcolor="white">
```

```
<jsp:useBean id='clock' scope='page'
```

```
class='essai.Bean1' type="essai.Bean1" />
```

```
<font size=4>
```

```
<ul>
```

```
<li> NOM: is <jsp:getProperty name="clock"
```

```
property="nom"/>
```

```
<li> Year: is <jsp:getProperty name="clock"
```

```
property="year"/>
```

```
<li> Month: is <jsp:getProperty name="clock"
```

```
property="month"/>
```

```
<li> Time: is <jsp:getProperty name="clock"
```

```
property="time"/>
```

```
<li> Date: is <jsp:getProperty name="clock"
```

```
property="date"/>
```

```
</ul>
```

```
</font>
```

```
</body>
```

```
</html>
```

Bean1.java :

```
package essai;
```

```
import java.text.DateFormat;
```

```
import java.util.*;
```

```
public class Bean1 {
```

```
    Calendar calendar = null;
```

```
    public Bean1() {
```

```
        calendar = Calendar.getInstance();
```

```
        Date daty = new Date();
```

```
        calendar.setTime(daty);
```

```
    }
```

```
    public int getNom() {
```

```
        return "RABE JEAN";
```

```
    }
```

```
    public int getYear() {
```

```
        return calendar.get(Calendar.YEAR);
```

```
    }
```

```
    public String getMonth() {
```

```
        int m = getMonthInt();
```

```
String[] months = new String [] { "January", "February",  
"March", "April", "May", "June",
```

```
"July", "August", "September",
```

```
"October", "November", "December" };
```

```
if (m > 12)
```

```
    return "Unknown to Man";
```

```
    return months[m - 1];
```

```
    }
```

```
    public int getMonthInt() {
```

```
        return 1 + calendar.get(Calendar.MONTH);
```

```
    }
```

```
    public String getDate() {
```

```
return getMonthInt() + "/" + getDayOfMonth() + "/" +
```

```
getYear();
```

```
    }
```

```
    public String getTime() {
```

```
return getHour() + ":" + getMinute() + ":" + getSecond();
```

```
    }
```

```
    public int getDayOfMonth() {
```

```
        return
```

```
calendar.get(Calendar.DAY_OF_MONTH);
```

```
    }
```

```
    public int getDayOfYear() {
```

```
        return calendar.get(Calendar.DAY_OF_YEAR);
```

```
    }
```

```
    public int getHour() {
```

```
        return calendar.get(Calendar.HOUR_OF_DAY);
```

```
    }
```

```
    public int getMinute() {
```



```

        return calendar.get(Calendar.MINUTE);
    }
    public int getSecond() {
        return calendar.get(Calendar.SECOND);
    }
}

```

2) Exemple Bean2.jsp

```
<%@ page import = "essai.Bean2" %>
```

```

<jsp:useBean id="bean" class="essai.Bean2"
scope="session"/>
<jsp:setProperty name="bean" property="*/>

```

```

<html>
<head><title>Exemple Bean</title></head>
<body bgcolor="white">
<font size=4>
Le nom ...=<jsp:getProperty name="bean"
property="nom"/>
Le nom ...=<%= bean.getNom() %>
La note...=<jsp:getProperty name="bean"
property="note"/>

```

```
<% if (! bean.getSuccess()) { %>
```

```

    Vous êtes <%= bean.getResultat() %> <p>
<% }
else if (bean.getSuccess()) { %>

```

```

    Vous avez une note comprise entre 10 et 20<p>
    Vous êtes admis en classe supérieure<p>
<% } %>

```

```

<form method=get>
Votre nom <input type=text name=nom> <br>
Votre note<input type=text name=note>
<input type=submit value="Submit">
</form>
</font>
</body>
</html>

```

Bean2.java

```

package Bean2;

import java.util.*;

public class Bean2 {
    int note;
    String nom;

    public Bean2() {
        note=0;
    }

    public void setNote(int a) { this.note = a; }

    public int getNote() { return this.note ;}

    public String getNom() { return nom; }
}

```

```

public void setNom(String nom) { this.nom=nom; }

    public boolean getSuccess() {
        if (note<10) return false;
        else return true;
    }

    public String getResultat() {
        return " ECHEC";
    }
}

```

Exemple 3 d'utilisation de JavaBeans contenant les méthodes de manipulation d'une base de données (modèle MVC

VUE : FORMULAIRE

```

<html>
<head>
<title> Validation au serveur </title>
</head>
<body>

```

```
<form name="form" action="maj.jsp" method="post">
```

```

<b>Codecli:</b> <input type="text" name="codecli"
Value=""> <br>

```

```

<b>Nom :</b><input type="text" name="nom"
value="" /> <br>

```

```

<b>Adresse:</b><input type="text" name="adresse"
value="" /> <br>

```

```

<b>Solde :</b><input type="text" name="solde"
value="" >
<br>
<p> </p>

```

```

<input type="submit" name="BT" value="Ajouter" >
<input type="submit" name="BT" value="Supprimer" >
<input type="submit" name="BT" value="Modifier" >
<input type="submit" name="BT" value="Consulter" >
<input type="submit" name="BT" value="Lister" >

```

```

<br>
</form>
</body>
</html>

```

MODELE : Fichier JavaBeans : BeanClient.java

```

package TClient;
import java.sql.*;
import java.io.*;
import java.util.*;

```

```

import java.lang.*;

public class BeanClient{

    //variable client
    public String Codecli;
    public String Nom;
    public String Adresse;
    public int Solde;
    Connection con = null;
    Statement statement = null;
    Statement st = null;
    ResultSet res = null;
    ResultSet rescom = null;
    public BeanClient(){//debut constructeur
        this.Codecli = "";
        this.Nom = "";
        this.Adresse = "";
        this.Solde = 0;
    }//fin constructeur

    public void connection(){
        String url ="jdbc:odbc:Base_dsn";
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection(url,"","");
            statement = con.createStatement();
            st = con.createStatement();

        }//fin try
        catch(Exception e1)
        { System.out.println(e1.toString());}
    }//fin connection

    public void setCodecli(String _Codecli){
        this.Codecli = _Codecli;
    }

    public String getCodecli(){
        return(this.Codecli);
    }

    public void setNom(String _Nom){
        this.Nom = _Nom;
    }

    public String getNom(){ return(this.Nom);
    }

    public void setAdresse(String _Adresse){
        this.Adresse = _Adresse;
    }
    public String getAdresse(){
        return(this.Adresse);
    }

    public void setSolde(int Solde){ this.Solde = Solde;
    }
    public int getSolde(){ return(this.Solde);
    }

    // ajout client
    public String AjoutClient()
    {

```

```

        connection();
        try{
            rescom=statement.executeQuery("Insert into client
            values(' "+this.Codecli+" ' , ' "+this.Nom+" ' , ' "+
            this.Adresse+" ' , ' "+
            + this.Solde+" ' ) ");
            return"enregistrement client réussi";
        }
        catch(Exception e2){return"client déjà
        existant"+e2.toString();
        }
    }//fin ajoutClient

    public String ModifClient()
    {
        connection();
        try{

            res=statement.executeQuery(" update client set
            Nom=' ' "+this.Nom+
            " ' ,Adresse=' ' "+this.Adresse+" ' ,Solde='
            "+this.Solde+" ' where Codecli=' ' "+this.Codecli+" '
            ");

            return "modification client
            reussi";
        }//fin try
        catch(Exception e){
            return"erreur de
            modification"+e.toString();
        }
    }//fin modification client

    //fin resaka client

} //fin class BeanClient

```

CONTROLEUR : maj.jsp (PAGE DE RECUPERATION©)

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">

<html>
<head>    <title>modifClient</title>
</head>

<body>
<%
String Vcodecli = request.getParameter("Codecli");
String Nom1 = request.getParameter("Nom");
String Adresse1 = request.getParameter("Adresse");
String VSolde = request.getParameter("Solde");
String BT= request.getParameter("BT");
%>
<jsp:useBean id="obj_client" scope="request"
class="Tclient.BeanClient"/>

<%
try{
    if(BT.equals("Modifier")){
        int t= Integer.parseInt(VSolde);

```

```

        obj_client.setCodecli(Vcodecli);
        obj_client.setNom(Nom1);
        obj_client.setAdresse(Adresse1);
        obj_client.setSolde(t);
        out.println("");
    %>

    <font color="blue"><%=obj_client.ModifClient()
    %></font>
    <%
        }//fin if
    else
        If if(BT.equals("Ajouter")){ //code .....}
    else
        If if(BT.equals("Ajouter")){ // code .....}

    //fin try

    catch(Exception e){
        out.println("");
        out.println("erreur eto ve"+e.toString());
    }
    %>
    <jsp:forward page="AfficheClient.jsp" />
</body>
</html>

```

TP1 : VALIDATION COTE SERVEUR

FICHIER form.jsp

```

<jsp:useBean id="form" class="form.FormBean">
    <jsp:setProperty name="form" property=""/>
</jsp:useBean>
<html>
<head>
<title> Validation au serveur </title>
</head>
<body>

<%
    String[] errors = (String[])request.getAttribute("errors");
    if (errors != null && errors.length > 0) {
    %>
    <b> S'il vous pla&icirc;t, corrigez les erreurs
    suivantes</b>
    <ul>
    <% for (int i=0; i < errors.length; i++) { %>
    <li> <%= errors[i] %> </li>
    <% } %>
    </ul>
    <% } %>
    <form action="formHandler.jsp" method="post">
    <input type="text" name="name"
    value="<jsp:getProperty name="form"
    property="name"/>" />
    <b>Nom</b> (Nom de Famille, Pr&eacut;nom)</br>

    <input type="text" name="email"

```

```

value="<jsp:getProperty name="form"
property="email"/>" />
<b>Courriel</b> (user@host)</br>

```

```

    <input type="text" name="ssn"
    value="<jsp:getProperty name="form"
    property="ssn"/>" />
    <b>NAS</b> (123456789)</br>
    <p>
    <input type="submit" value="Soumettre" />
    </form>
</body>
</html>

```

REMARQUE :

A la place de value="<jsp:getProperty name="form" property="name"/>" , on peut écrire
value="<%= form.getName() %> "

FICHIER JAVA BEAN : FormBean.java

```

package form;

public class FormBean {
    private String name;
    private String email;
    private String ssn;

    public FormBean() {
        name = "";
        email = "";
        ssn = "";
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

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

    public String getEmail() {
        return email;
    }

    public void setSsn(String ssn) {
        this.ssn = ssn;
    }

    public String getSsn() {
        return ssn;
    }
}

```

FICHIER formHandler.jsp

```

<%@ page import="java.io.IOException,java.util.Vector"
%>
<%!

```

```

private boolean isValidSSN(String ssn) {
    // check for 9 characters, no dashes
    return (ssn.length() == 9 && ssn.indexOf("-") == -1);
}

private boolean isValidEmail(String email) {
    // check an "@" somewhere after the 1st character
    return (email.indexOf("@") > 0);
}

private boolean isValidName(String name) {
    // should be Last, First - check for the comma
    return (name.indexOf(",") != -1);
}
%>
<%
    Vector errors = new Vector();

    String name = request.getParameter("name");
    String ssn = request.getParameter("ssn");
    String email = request.getParameter("email");

    if (! isValidName(name))
        errors.add("Le nom doit &ecirc;tre sp&eacute;cifier
comme suit: Nom de Famille, Pr&eacute;nom");
    if (! isValidEmail(email))
        errors.add("Le courriel doit avoir le symbole @ ");
    if (! isValidSSN(ssn))
        errors.add("Le NAS ne doit pas avoir de tiret
et doit avoir 9 chiffres");

    String next;
    if (errors.size() == 0) {
        // data is OK
        // dispatch to thanks
        next = "thanks.jsp";
    }
    else {
        // data has errors, try again
        String[] errorArray = (String[])errors.toArray(new
String[0]);
        request.setAttribute("errors", errorArray);
        next = "form.jsp";
    }
%>
<jsp:forward page="<%= next %>" />

```

PAGE thanks.jsp

```

<jsp:useBean id="form" class="form.FormBean">
    <jsp:setProperty name="form" property=""/>
</jsp:useBean>
<html>
<body bgcolor="white">
<b>Merci beaucoup! Votre formulaire a
&eacute;t&eacute; bien re&ccedil; </b>
<ul>

<b>Nom: </b><jsp:getProperty name="form"
property="name"/><br>
<b>Courriel: </b><jsp:getProperty name="form"
property="email"/><br>

```

```

<b>NAS: </b><jsp:getProperty name="form"
property="ssn"/><br>
</ul>
</body>
</html>

```

TP2 :MANIPULATION D'UNE BASE DE DONNEES

FICHER GESTIONCLIENT.HTML

```

<html>
<head>
<title> Validation au serveur </title>
</head>
<body>

<form name="form"
action="/examples/formHandlerclients.jsp"
method="post">

<b>Codecli: </b><input type="text" name="codecli"
Value=""> <br>

<b>Nom :</b><input type="text" name="nom"
value="" /> <br>

<b>Adresse:</b><input type="text" name="adresse"
value="" /> <br>

<b>Solde :</b><input type="text" name="solde"
value="" >
<br>
<p> </p>

<input type="submit" name="BT" value="Ajouter" >
<input type="submit" name="BT" value="Supprimer" >
<input type="submit" name="BT" value="Modifier" >
<input type="submit" name="BT" value="Consulter" >
<input type="submit" name="BT" value="Lister" >

<br>
</form>
</body>
</html>

```

PAGE DE VALIDATION (formHandlerclients.jsp)

```

<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<%@ page session="true" %>

<html>
<head>
<title> Validation au serveur </title>
</head>
<body>

<%
    int nb ;

```

```

double s ;
Connection con=null;
Statement statement = null;
ResultSet res =null;
ResultSet rescom =null;
String requete ;

String mcodecli = (String)
request.getParameter("codecli");
String mnom = (String) request.getParameter("nom");
String madresse = (String)
request.getParameter("adresse");
String msolde = (String)
request.getParameter("solde");
if (msolde.equals("") || msolde==null)
    s=0;
else
    s= Double.parseDouble(msolde);
if (mcodecli.equals("") || mcodecli==null)
    out.println("LE CODE DU CLIENT EST NUL ");
else
{

String url ="jdbc:odbc:veno";
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con =DriverManager.getConnection(url,"","");
statement=con.createStatement();
}
catch(Exception e) {
out.println("ERROR CONNEXION"); return: }

String bt = request.getParameter("BT");
if (bt.equals("Ajouter"))
{
// AJOUTER
try {
requete ="INSERT INTO
client(codecli,nom,adresse,solde) VALUES( "+" '
"+mcodecli +" ' , ' " +mnom +" ' , ' " +madresse+
" ' , "+"s+" )";
nb =statement.executeUpdate(requete);
out.println("<h3> ENREGISTREMENT AJOUTE </h3>");
};
statement.close();
con.close();
}
catch(Exception e) {
out.println("<h3> ENREGISTREMENT DEJA
EXISTANT </h3> ");
}
}
else
if (bt.equals("Supprimer"))
{
// SUPPRIMER
try {
nb =statement.executeUpdate("DELETE FROM
commande where codecli = ' " + mcodecli + " ' " );
nb =statement.executeUpdate("DELETE FROM cCLIENT
where codecli = ' " + mcodecli + " ' ");
out.println("<h3> ENREGISTREMENT SUPPRIME
</h3> ");

```

```

statement.close();
con.close();
}
catch(Exception e) {
out.println("ERROR DELETE");
}
}
else
if (bt.equals("Modifier"))
{
// MODIFIER
try {
nb =statement.executeUpdate("UPDATE client set nom
=' " +mnom+" ' , ' " +adresse= ' " +madresse+" ' , "+"
"solde= " + s + " WHERE codecli= ' " +mcodecli+ " ' ");
out.println("<h3> ENREGISTREMENT MODIFIE </h3>");
};
statement.close();
con.close();
}
catch(Exception e) {
out.println("ERROR UPDATE");
}
}
else

if (bt.equals("Consulter"))
{
// CONSULTEUR
if (mcodecli.equals("") || mcodecli==null)
    out.println("Selectionnez un client svp ");
else{
try {
res =statement.executeQuery("select * from client where
codecli=' "+mcodecli+" ' ");
if (res==null)
out.println("<h3> CODE INCORRECT </h3> ");
else
{
res.next();
mnom= res.getString("nom");
madresse =res.getString("adresse");
msolde= (new
Double(res.getDouble("solde"))).toString();
out.println("<h3> CODECLI : " + mcodecli+"</h3>
<BR>");
out.println("<h3> NOM : " + mnom+"</h3> <BR>");
out.println("<h3> ADRESSE : " +
madresse+"</h3><BR>");
out.println("<H3>SOLDE : " + msolde+"</h3> <BR>");
}
res.close();
statement.close();
con.close();
}

catch(Exception e) {
out.println("<h3> CODE CLIENT INCORRECT </h3> ");
}
}
}
else
// LISTER

```

```

if (bt.equals("Lister"))
{
%>
<TABLE Border="1">
<tr>
<th><b><i>NOM</i></b></th>
<th><b><i>LIBELLE</i></b></th>
<th><b><i>PU</i></b></th>
<th><b><i>QTE</i></b></th>
<th><b><i>MONTANT</i></b></th>
</tr>

<%
try
{
rescom =statement.executeQuery("select
Client.Codecli,Nom,Adresse,Libelle,Pu,Qte,qte*Pu as
Montant from client,commande,produit where
client.codecli=commande.codecli and
commande.codepro=produit.codepro Order by nom");
while (rescom.next()) {
%>
<tr>
<td><b><i><%=rescom.getString("nom")
%></i></b></td>
<td><b><i><%=rescom.getString("Libelle")%></i></b></td>
<td><b><i><%=rescom.getInt("Pu")%></i></b></td>
<td><b><i><%=rescom.getInt("qte")%></i></b></td>
<td><b><i><%=rescom.getDouble("Montant")%></i>
</b></td>
</tr>

<%
}
%>
</TABLE>
<% rescom.close();statement.close();con.close();
}
catch(Exception e)
{ out.println("<h3> Error de lecture Commande </h3>
"); ; }
}
// Fin Lister
}
%>
</BODY>
</HTML>

```

TP 3 : UTILISATION D'UNE SESSION

```

<%@ page import ="java.sql.*" %>
<%@ page import ="java.io.*" %>
<%@ page session="true" %>

<html>
<head>
<title> Validation au serveur </title>
</head>
<body>

<%
String url ="jdbc:odbc:veno";
Connection con=null;
Statement statement = null;

```

```

ResultSet res =null;
ResultSet rescom =null;
ResultSetMetaData metaData;
double SumSolde=0;
double QteTot=0 ;
double MontantTot=0 ;
String message ;
// Connection à la base

String codecli = (String)request.getAttribute("codecli");
String nom = (String)request.getAttribute("nom");
String adresse =
(String)request.getAttribute("adresse");
String solde= (String)request.getAttribute("solde");
message= (String)request.getAttribute("message");
if (message==null) message="NO ERROR" ;
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con =DriverManager.getConnection(url,"","");
statement = con.createStatement();
}

catch (ClassNotFoundException ex) {
message="Cannot find the database driver
classes.";
}
catch (SQLException ex) {
message="Cannot connect to this database.";
}

if (con == null || statement == null) {
message="There is no database to execute the
query.";
return;
}
%>
<%
if (codecli==null) {
try {
res =statement.executeQuery("select sum(solde) as
SumSolde from Client ");
res.next();
SumSolde =res.getDouble("SumSolde");
res.close();
res =statement.executeQuery("select * from Client ");
res.next();
codecli =res.getString("codecli");
nom =res.getString("nom");
adresse =res.getString("adresse");
solde =res.getString("solde");
res.close();
}
catch(Exception e) { message="Error IO Client" ; }
}
%>

<form name="form" action="/examples/MAJCLI.jsp"
method="post">

<b>Selectionnez un client:</b>
<select name="liste" size="1">
<%

```

```

try {
res =statement.executeQuery("select codecli,nom from
Client order by nom ");
while (res.next()){ %>
<option value="<%=res.getString("codecli")%>">
<%= res.getString("nom")%>
</option>
<%
}
%>
</select>
<% res.close(); %>
<% }
catch(Exception e)
{ message="Error LISTE" ; }
%>

<br><br>

<b>Codecli:</b> <input type="text" name="codecli"
onkeypress="if (event.which==13 &&
event.keyCode==13) document.form.nom.focus();"
value="<%= codecli%>" /> <br>

<b>Nom :</b><input type="text" name="nom"
value="<%= nom %>" /> <br>

<b>Adresse:</b><input type="text" name="adresse"
value="<%= adresse %>" /> <br>
<b>Solde :</b><input type="text" name="solde"
value="<%=solde%>" />
<br>
<p> </p>
<input type="reset" name="nouveau"
value="Nouveau" />
<input type="submit" name="BT" value="Ajouter" />
<input type="submit" name="BT" value="Supprimer" />
<input type="submit" name="BT" value="Modifier" />
<input type="submit" name="BT" value="Consulter" />
<br>
<b>Message :</b><input type="text" size="40"
name="message" value="<%=message%>" /> <br>
</form>
<table border="2">
<%

// Parcours de la requete Commande
try {

rescom =statement.executeQuery("select
Client.Codecli,Nom,Adresse,Libelle,Pu,Qte,qte*Pu as
Montant from client,commande,produit where
client.codecli=commande.codecli and
commande.codepro=produit.codepro Order by libelle");
MontantTot=0;
QteTot=0;
while (rescom.next()){
if (codecli.equals(rescom.getString("codecli")))
{MontantTot+=rescom.getInt("Pu")*rescom.getInt("Qte");
// QteTot+=rescom.getInt("Qte");
}
}
rescom.close();

```

```

rescom =statement.executeQuery("select
Client.Codecli,Nom,Adresse,Libelle,Pu,Qte,qte*Pu as
Montant from client,commande,produit where
client.codecli=commande.codecli and
commande.codepro=produit.codepro Order by nom");
QteTot=0;
while (rescom.next()){
if (codecli.equals(rescom.getString("codecli")))
{QteTot+=rescom.getInt("Qte");
}
}
rescom.close();
%>

<tr>
<th><b><i>NOM</i></b></th>
<th><b><i>LIBELLE</i></b></th>
<th><b><i>PU</i></b></th>
<th><b><i>QTE</i></b></th>
<th><b><i>MONTANT</i></b></th>
</tr>

<%
rescom =statement.executeQuery("select
Client.Codecli,Nom,Adresse,Libelle,Pu,Qte,qte*Pu as
Montant from client,commande,produit where
client.codecli=commande.codecli and
commande.codepro=produit.codepro Order by nom");
while (rescom.next()){
if (codecli.equals(rescom.getString("codecli")))
{
%>
<tr>
<td><b><i><%=rescom.getString("nom")%></i></b>
</td><td><b><i><%=rescom.getString("Libelle")
%></i></b></td>
<td><b><i><%=rescom.getInt("Pu")%></i></b></td>
<td><b><i><%=rescom.getInt("qte")%></i></b></td>
<td><b><i><%=rescom.getDouble("Montant")%></i>
</b></td>
</tr>
<%
}
%>
<%
}
%>

<%
rescom.close();
statement.close();
con.close();
}

catch(Exception e)
{ message="Error de lecture Commande" ; }
%>

<br>
<tr>
<td>&nbsp;</td>

```

```

<td>&nbsp;</td>
<td>MONTANT TOTAL </td>
<td>&nbsp;</td>
<td><%= MontantTot+" Fmg" %></td>
</tr>

<tr>
<td>&nbsp;</td>
<td>&nbsp;</td>
<td>QTE TOTAL </td>
<td><%= QteTot%></td>
<td>&nbsp;</td>

</tr>
</TABLE>
</body>
</html>

```

FICHER MAJCLI.JSP

```

<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<%@ page session="true" %>

<html>
<head>
<title> Validation au serveur </title>
</head>
<body>
<%
    int nb ;    double s ;
    Connection con=null; Statement statement = null;
    ResultSet res =null;
    String requete ;

    String mliste = (String) request.getParameter("liste");
    String mcodecli =(String)
    request.getParameter("codecli");
    String mnom = (String) request.getParameter("nom");
    String madresse = (String)
    request.getParameter("adresse");
    String msolde = (String) request.getParameter("solde");
    String messages ="OPERATION REUSSIE" ;
    if (msolde.equals("") || msolde==null)
        s=0;
    else
        s= Double.parseDouble(msolde);
    if (mcodecli.equals("") || mcodecli==null)
        messages="LE CODE DU CLIENT EST NUL ";
    else
    {
        String url ="jdbc:odbc:DsnEssai";
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con =DriverManager.getConnection(url,"","");
            statement=con.createStatement();
        }
        catch(Exception e) {
            messages="ERROR CONNEXION";
        }
        String bt = request.getParameter("BT");
        if (bt.equals("Ajouter")){

```

```

try {
    requete ="INSERT INTO
    client(codecli,nom,adresse,solde) VALUES("+
    ""+mcodecli+"",""+mnom+"",""+madresse+"','"+s+"");
    nb =statement.executeUpdate(requete);
    messages="ENREGISTREMENT AJOUTE";
    statement.close();
    con.close();
}
catch(Exception e) {
    messages="ENREGISTREMENT DEJA EXISTANT";
}
}
else
if (bt.equals("Supprimer"))
{
    try {
        nb =statement.executeUpdate("DELETE FROM
        commande where codecli = " + mcodecli + "");
        nb =statement.executeUpdate("DELETE FROM cCLIENT
        where codecli = " + mcodecli + "");
        messages="ENREGISTREMENT SUPPRIME";
        statement.close();
        con.close();
    }
    catch(Exception e) { messages="ERROR DELETE";}
}
else
if (bt.equals("Modifier")){
    try {
        nb =statement.executeUpdate("UPDATE client set nom
        ='"+mnom+"','"+adresse+"'+madresse+'','"+s+" WHERE codecli='"+mcodecli+" '");
        messages="ENREGISTREMENT MODIFIE";
        statement.close();
        con.close();
    }
    catch(Exception e) {
        messages="ERROR UPDATE";
    }
}
else
if (bt.equals("Consulter"))
{
    if (mliste.equals("") || mliste==null)
    { messages="Selectionnez un client svp ";
    }
    else
    {
        try {
            res =statement.executeQuery("select * from client where
            codecli='"+mliste+"'");
            res.next();
            mcodecli= res.getString("codecli");
            mnom= res.getString("nom");
            madresse =res.getString("adresse");
            msolde= (new
            Double(res.getDouble("solde"))).toString();
            res.close();
            statement.close();
            con.close();
        }
        catch(Exception e) {

```



```

messages="ERROR CONSULTER";
}
}
}
// data is OK
// dispatch to thanks
request.setAttribute("codecli", mcodecli);
request.setAttribute("nom", mnom);
request.setAttribute("adresse", madresse);
request.setAttribute("solde", msolde );
request.setAttribute("message", messages );
%>
<jsp:forward page="gestcli.jsp" />
</BODY>
</HTML>

```

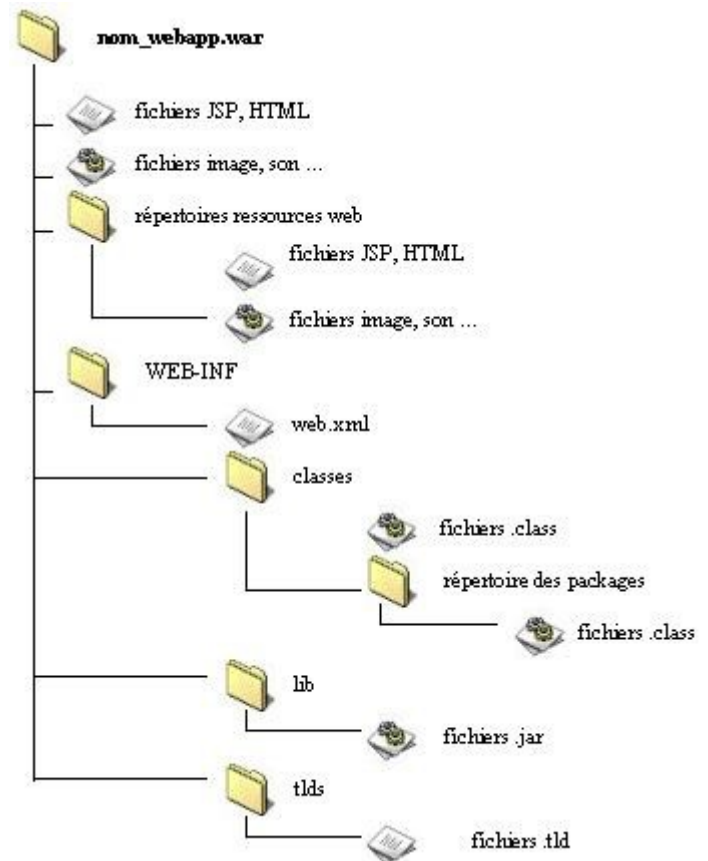
7. Packager une application web

Le format war (Web Application Archive) indépendant de toute plate-forme et exploitable par tous les conteneurs web ,permet de regrouper en un seul fichier tous les éléments d'une application web que ce soit pour le côté serveur (servlets, JSP, classes java, ...) ou pour le côté client (ressources HTML, images, son ...).

Le but principal est de simplifier le déploiement d'une application web et d'uniformiser cette action quel que soit le conteneur web utilisé.

7.1. La structure d'un fichier .war

Comme les fichiers jar, les fichiers war possèdent une structure particulière qui est incluse dans un fichier compressé de type "zip" possédant comme extension ".war".



Le nom du fichier .war est important car ce nom sera automatiquement associé dans l'url pour l'accès à l'application en concaténant le nom du domaine, un slash et le nom du fichier war. Par exemple, pour un serveur web sur le poste local avec un fichier **test.war** déployé sur le serveur d'application, l'url pour accéder à l'application web sera **http://localhost:8080/test/**

Le répertoire WEB-INF et le fichier web.xml qu'il contient doivent obligatoirement être présents dans l'archive. Le fichier web.xml est le descripteur de déploiement de l'application web.

Le répertoire **WEB-INF/classes** est automatiquement ajouté par le conteneur au CLASSPATH lors du déploiement de l'application web.

L'archive web peut être créée avec l'outil jar fourni avec le JDK ou avec un outil commercial. Avec l'outil jar, il suffit de créer l'arborescence de l'application, de se placer dans le répertoire racine de cette arborescence et d'exécuter la commande :

jar cvf nom_web_app.war .

Toute l'arborescence avec les fichiers qu'elle contient sera incluse dans le fichier **nom_web_app.war**.

7.2. Le déploiement d'une application web

Le déploiement d'une archive web dans un serveur d'application est très facile car il suffit simplement de copier le fichier .war dans le répertoire par défaut dédié aux applications web. Par exemple dans Tomcat, **c'est le répertoire webapps**. Attention cependant, si chaque

conteneur qui respecte les spécifications 1.1 des JSP sait utiliser un fichier .war, leur exploitation par chaque conteneur est légèrement différente.

Par exemple avec Tomcat, il est possible de travailler directement dans le répertoire webapps avec le contenu de l'archive web décompressé. Cette fonctionnalité est particulièrement intéressante lors de la phase de

développement de l'application car il n'est alors pas obligatoire de générer l'archive web à chaque modification pour réaliser des tests. Attention, si l'application est redéployée sous la forme d'une archive .war, il faut obligatoirement supprimer le répertoire qui contient l'ancienne version de l'application.

8. Intégration du serveur Tomcat dans Eclipse

Ce tutorial a pour objet de présenter la configuration d'Eclipse pour développer des applications web sous Tomcat. Il commence par l'installation du *plugin* Tomcat dans Eclipse, puis la prise en charge par Eclipse d'un serveur Tomcat. Enfin, on montre comment créer un premier projet Web sous Eclipse, et sa mise au point sous Tomcat.

1. Création d'un serveur Tomcat sous Eclipse

Avant de créer ce serveur, on doit s'assurer que le *plugin* Tomcat est bien installé dans Eclipse. Cela se passe dans la fenêtre des *Preferences* d'Eclipse, à laquelle on peut accéder dans le menu *Windows*.

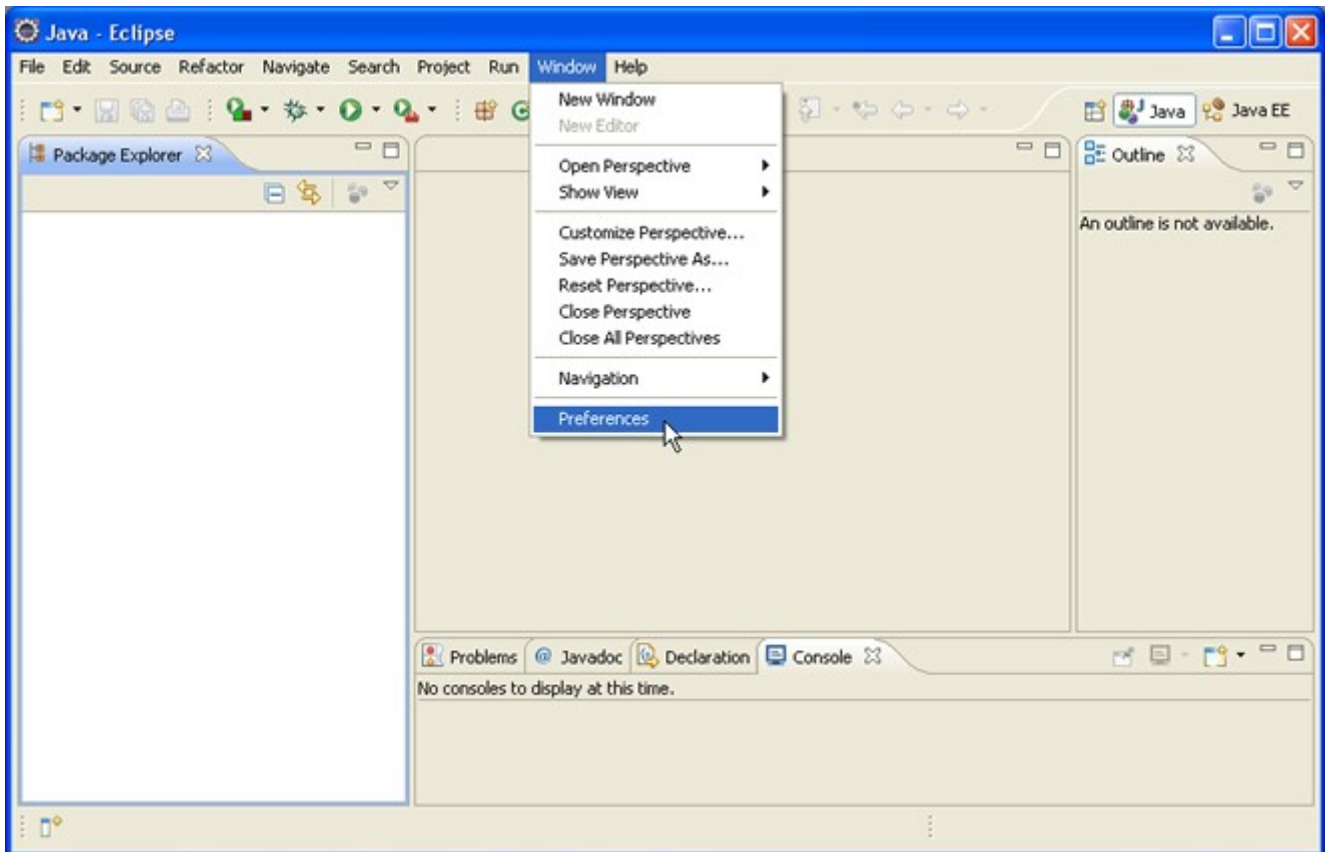


Figure 1. Ouvrir les préférences d'Eclipse

Cette fenêtre comporte énormément d'options, que l'on peut filtrer. La capture d'écran suivante montre celle qui nous intéresse : les *Server Runtime Environments*.

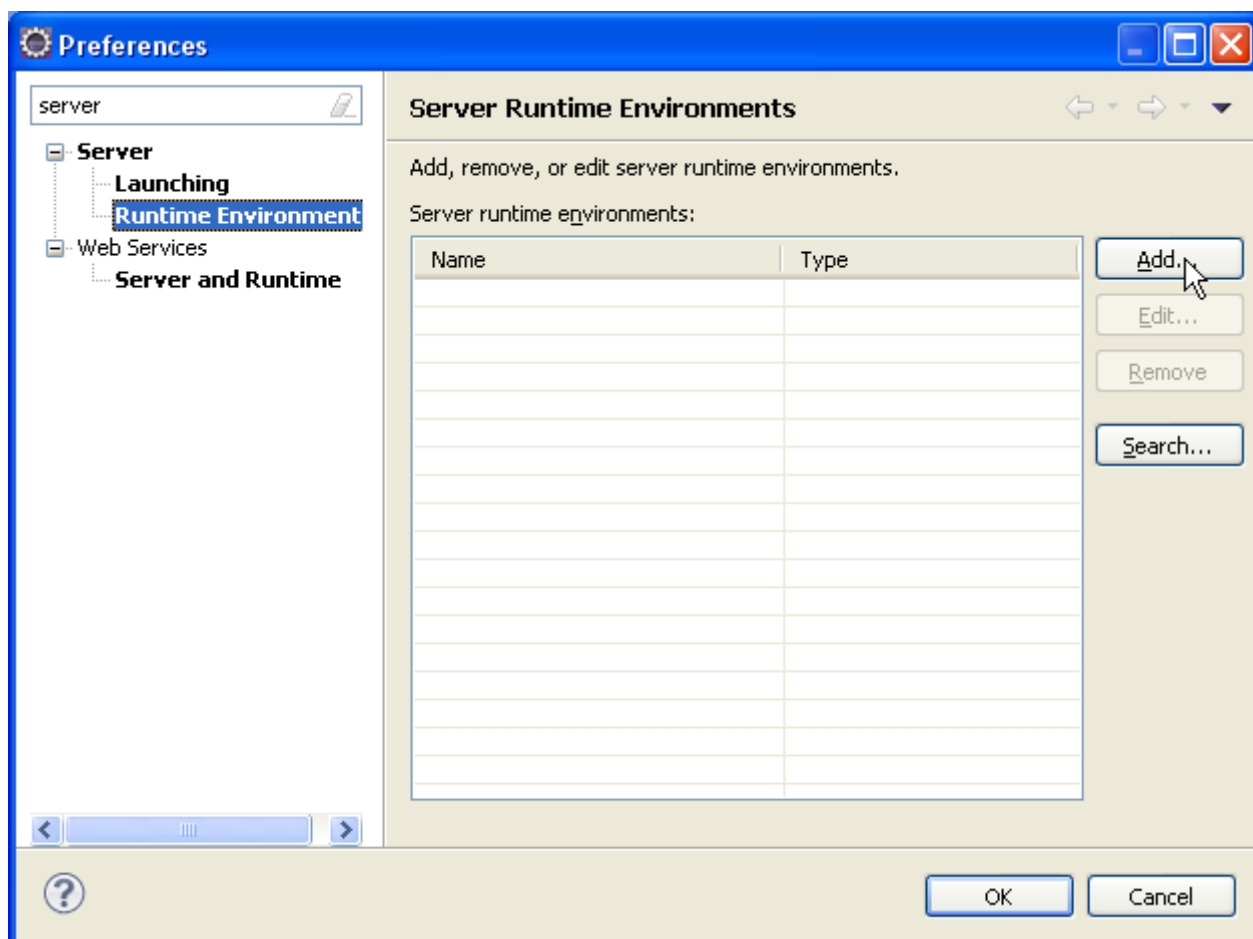


Figure 2. Ouvrir le panneau des serveurs disponibles

Pour le moment aucun serveur n'est disponible, ce qui est normal, puisque l'on en n'a déclaré aucun.

Cliquer sur le bouton *Add...* ouvre la liste des *plugins* qu'Eclipse possède déjà. Si Tomcat n'apparaît pas dans cette liste, on peut la compléter par téléchargement en clique sur le lien *Download additional server adapters*, en haut droite de ce panneau.

Dans notre cas, les *plugins* pour les différentes versions de Tomcat sont déjà présents, on sélectionne celui qui correspond à la version 6 de Tomcat.

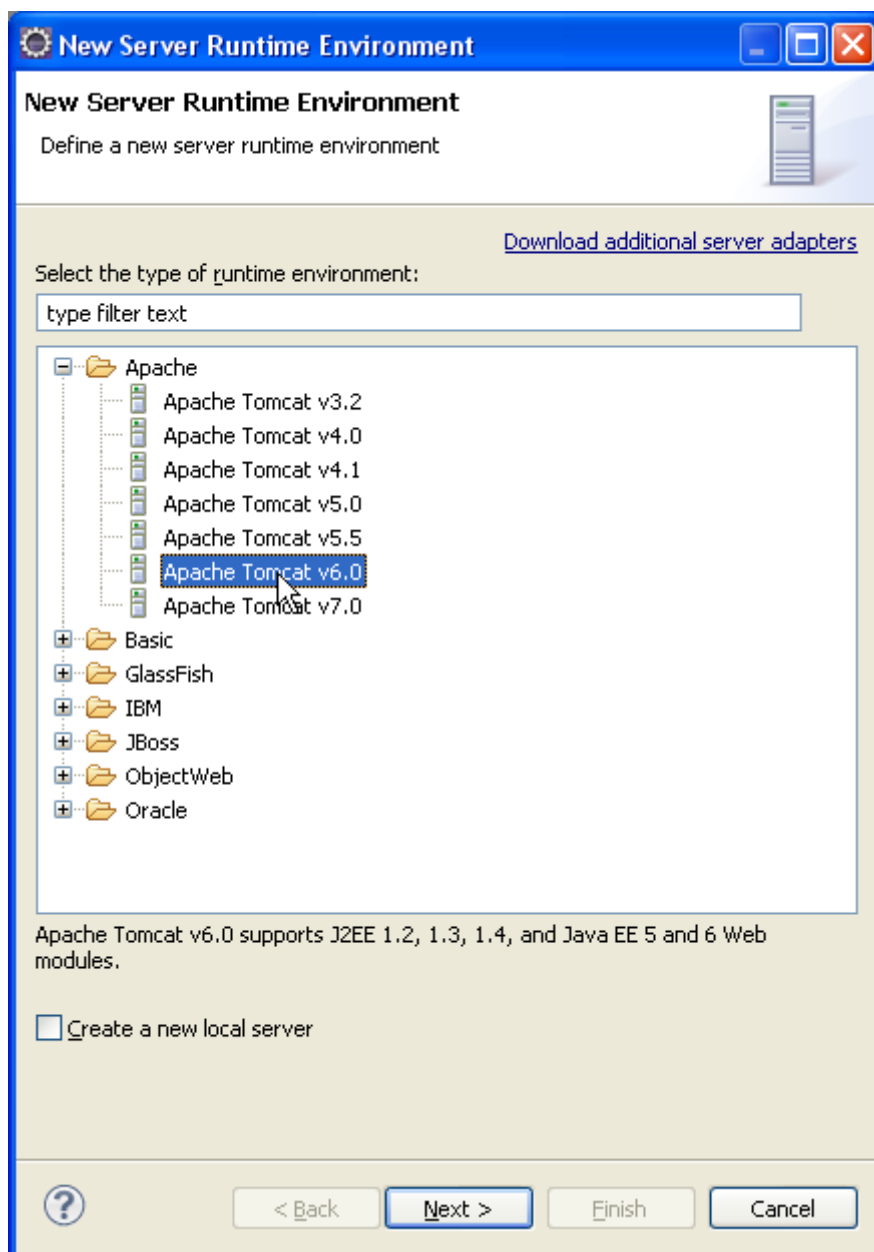


Figure 3. Sélection de Tomcat version 6

L'étape suivante consiste à indiquer à Eclipse où se trouve le répertoire d'installation de Tomcat. On peut le préciser en cliquant sur le bouton *Browse...* et en allant sélectionner le bon répertoire.

Une fois le répertoire validé, l'installation de Tomcat que l'on a spécifiée apparaît dans la liste des *Server Runtime Environments*.

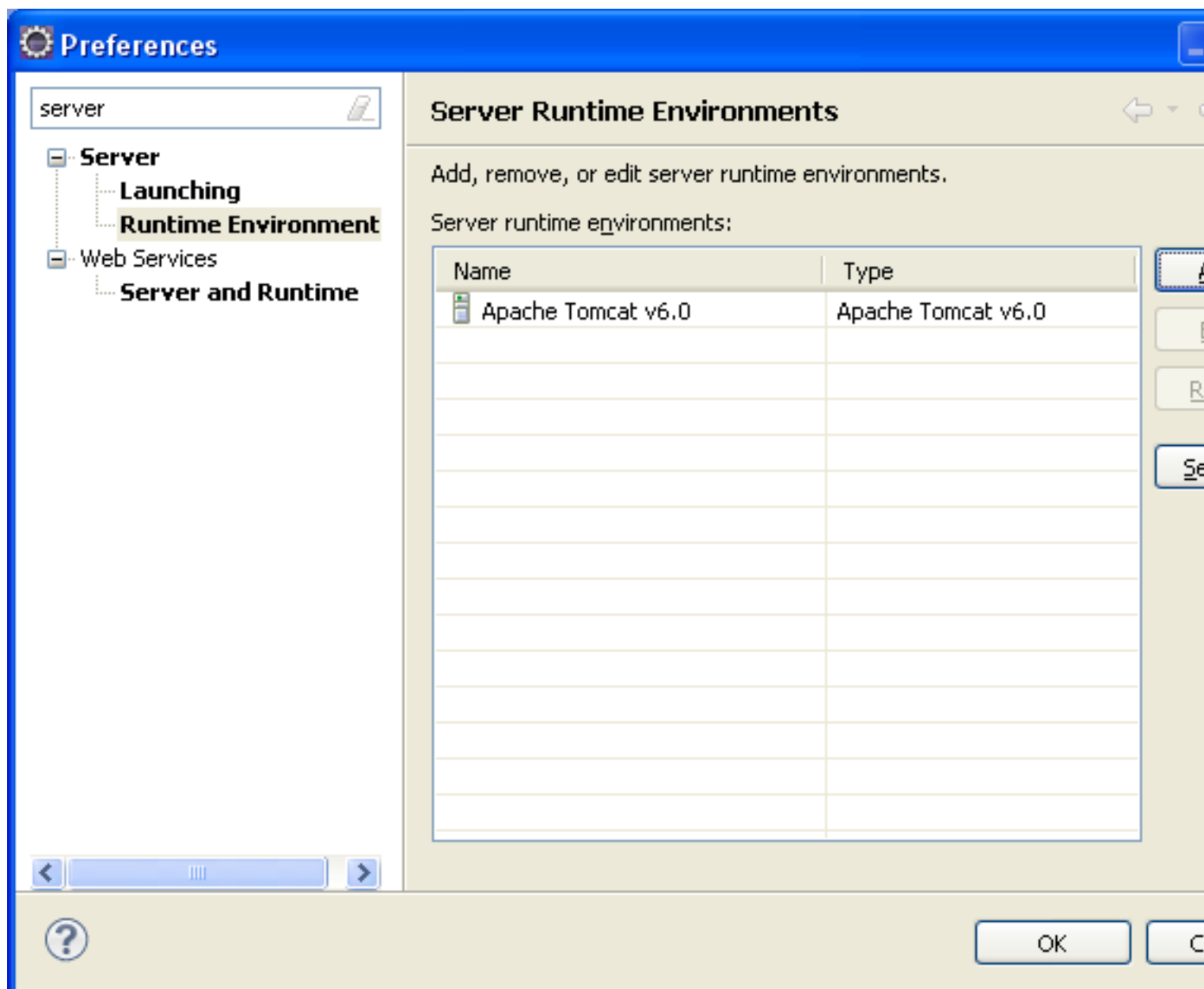


Figure 4. L'installation de Tomcat dans Eclipse est validée

Notre installation est validée, nous allons pouvoir passer à l'étape suivante.

2. Création d'un premier projet Web

Un projet Web est un type de projet spécial sous Eclipse. Spécial, en ce sens qu'il est attaché à un serveur d'applications (ici Tomcat, ce pourrait être un autre serveur), et qu'il expose des fonctionnalités particulières pour créer des servlets, des pages, ou des services web (que nous ne présenterons pas ici).

La création d'un projet Web débute comme un projet Eclipse normal. Comme il ne s'agit pas d'un projet Java classique, on sélectionne la rubrique *Project...* du sous-menu *New*.

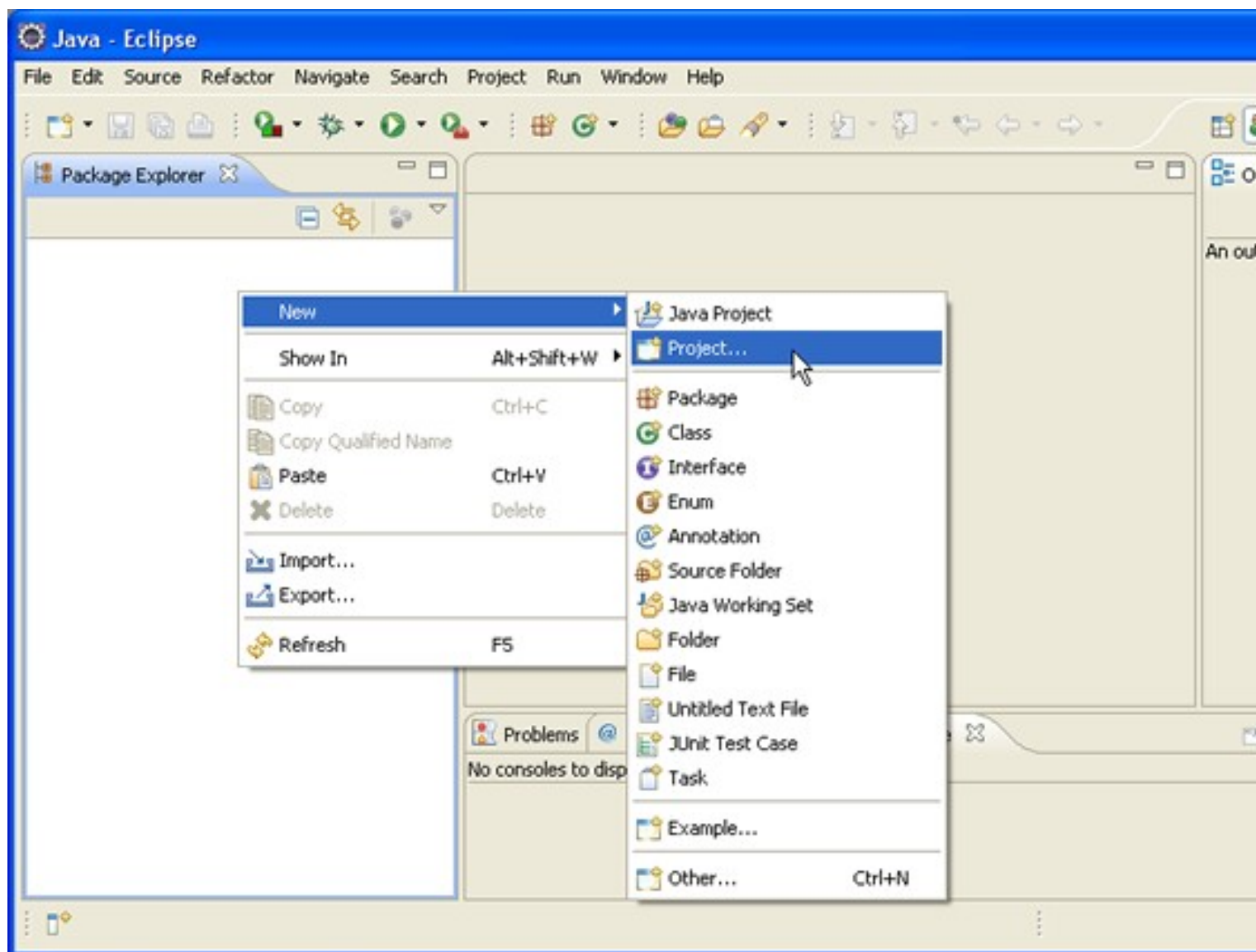


Figure 5. Création d'un projet web - 1

Dans le panneau qui s'ouvre, on sélectionne alors *Dynamic Web Project*, sous le nœud *Web*.

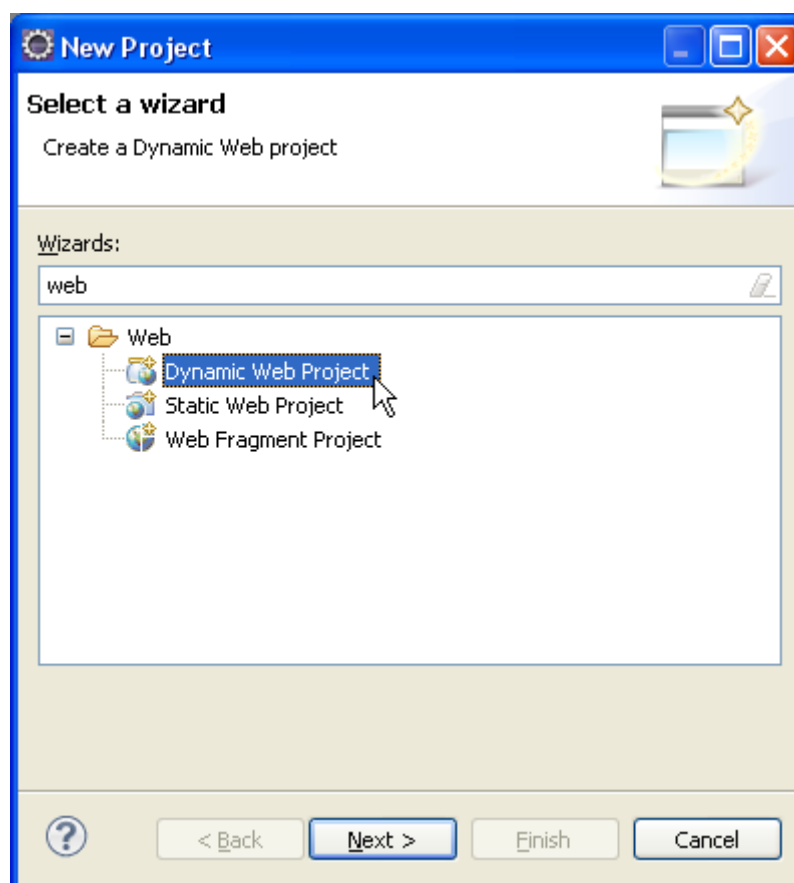


Figure 6. Création d'un projet web - 2

Le panneau qui s'ouvre alors est un peu plus compliqué que celui d'un projet Java classique. En particulier, il nous demande de choisir le serveur que ce projet va utiliser pour fonctionner. Dans notre exemple, on sélectionne le serveur Tomcat que l'on vient de créer. La version de l'API Servlet est sélectionnée automatiquement : 2.5, version la plus récente supportée par Tomcat v6.

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location
☒ Use default location
Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v6.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership
☐ Add project to an EAR
EAR project name:

Working sets
☐ Add project to working sets
Working sets:

Figure 7. Création d'un projet web - 3

Le panneau suivant nous demande dans quel répertoire on souhaite ranger nos fichier source. On conserve le choix par défaut : le répertoire src.

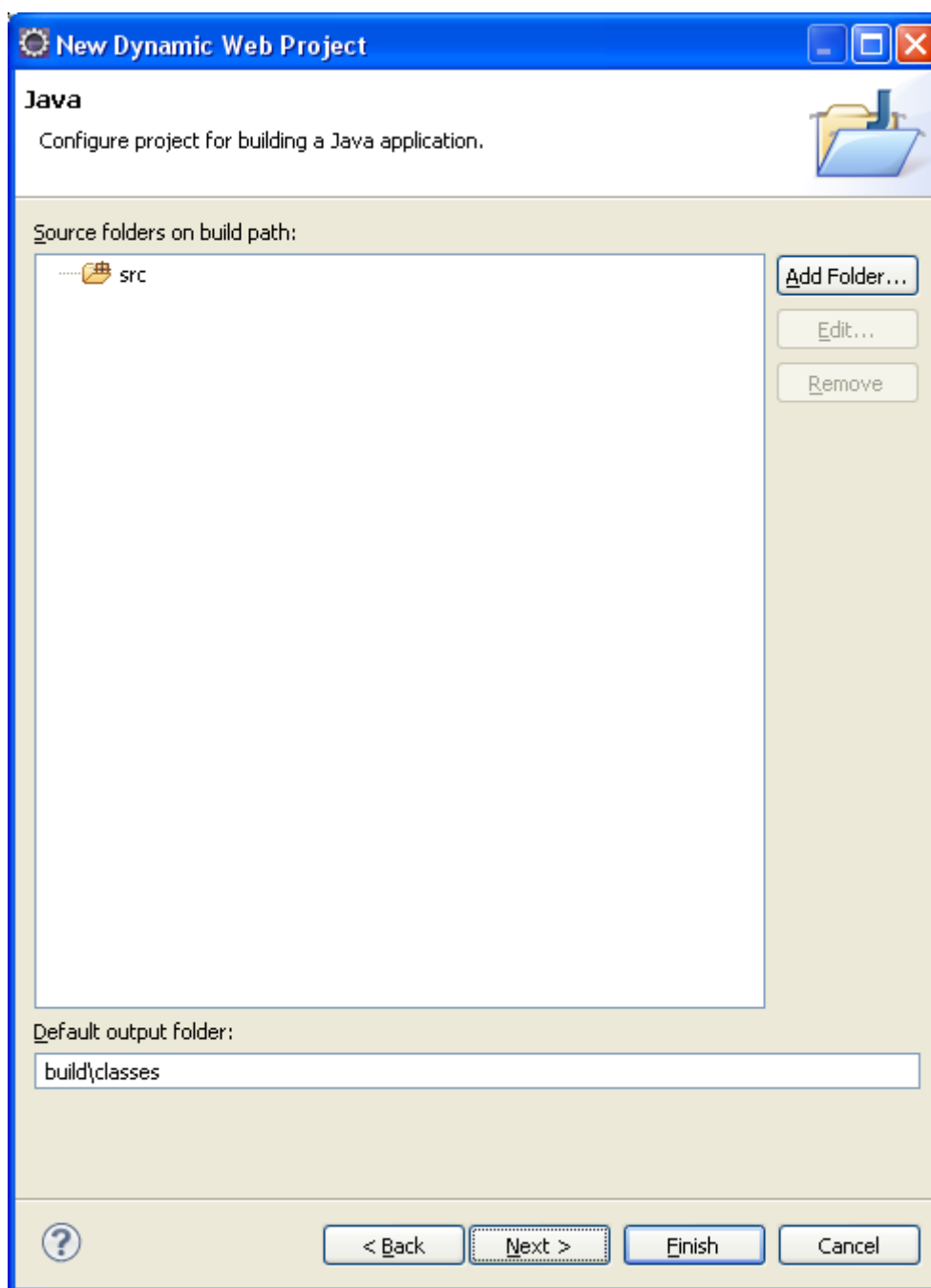


Figure 8. Création d'un projet web - 4

Le dernier panneau nous demande dans quel répertoire on souhaite ranger les ressources de ce projet web : le répertoire WebContent. C'est dans ce répertoire que l'on pourra ranger les différentes ressources de notre projet : pages HTML, tout fichier multimedia, fichier Javascript, etc...

On peut également générer automatiquement le fichier web.xml de notre application web, ce qui est une commodité offerte par Eclipse. Ce fichier sera rangé automatiquement dans WebContent/WEB-INF.

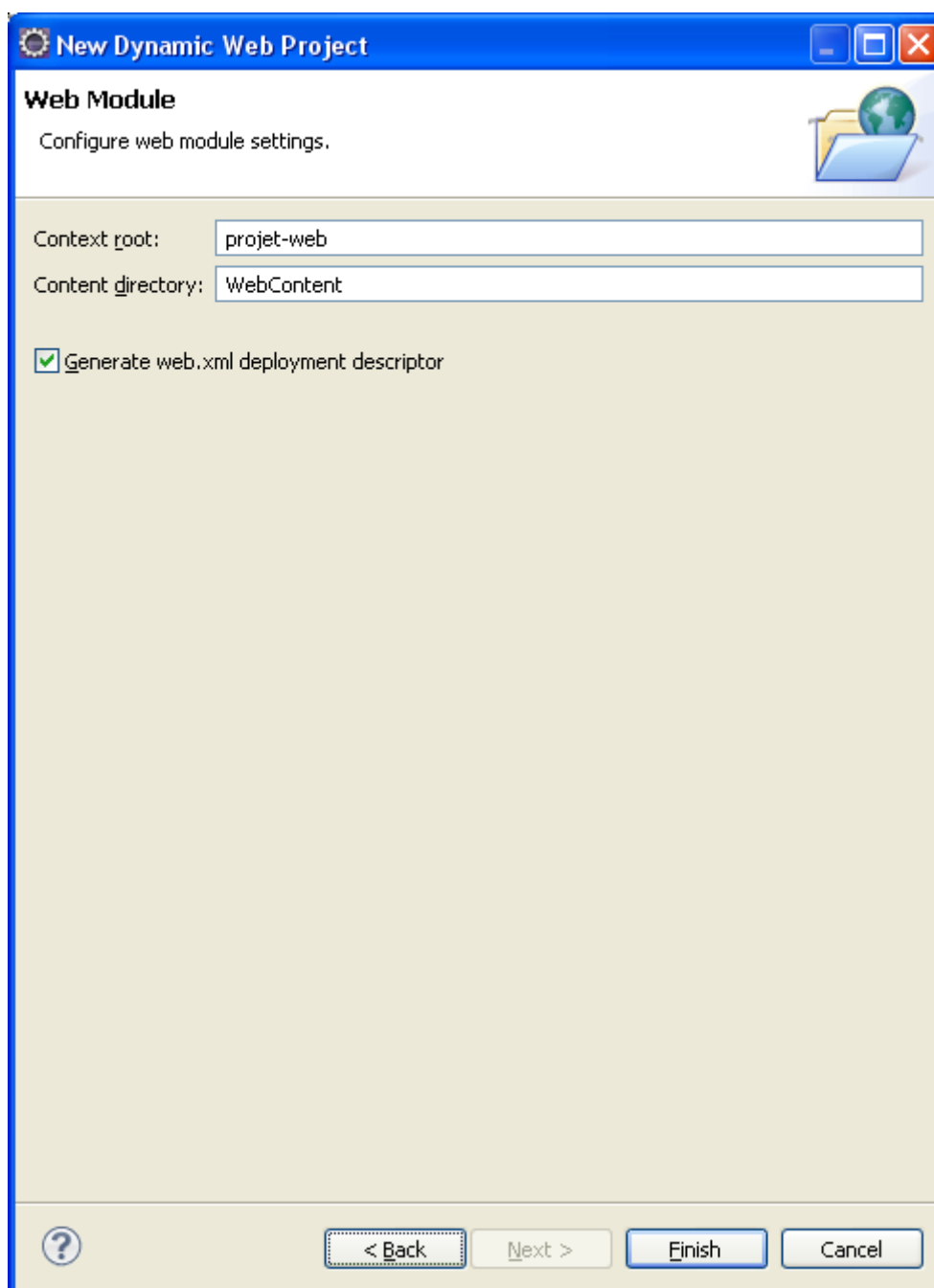


Figure 9. Création d'un projet web - 5

Une fois notre projet créé, Eclipse nous demande si l'on souhaite utiliser la *perspective Java EE*. Cette perspective est bien adaptée au développement des projets Java EE (elle est même faite pour ça !), on accepte donc cette proposition.

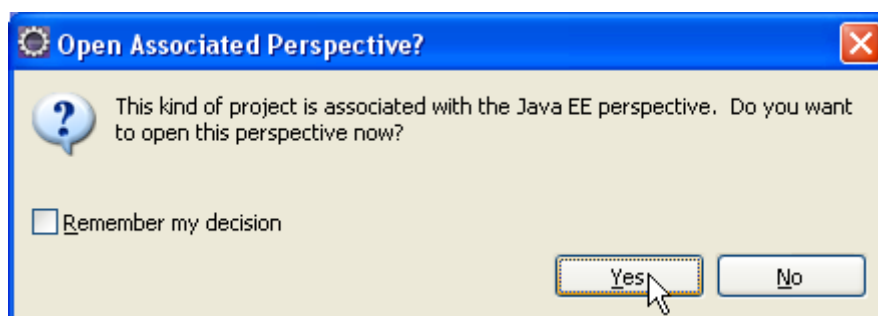


Figure 10. Utilisation de la perspective Java EE

Sur la partie gauche de cette perspective, se trouve la vue *Project Explorer* (qui ressemble beaucoup à *Package Explorer*), dans laquelle notre nouveau projet apparaît.

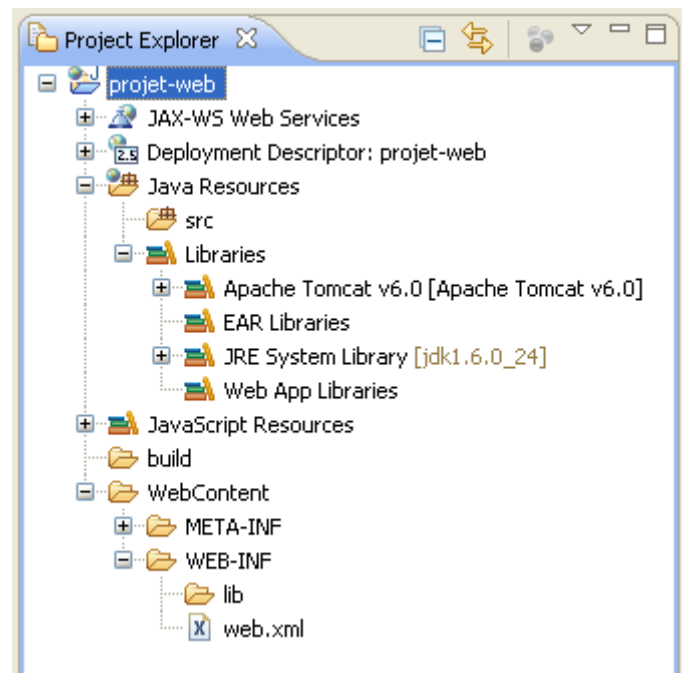


Figure 11. Structure de notre projet dans la vue *Project Explorer*

Notre projet est maintenant créé, et l'on va pouvoir passer à l'étape suivante : créer une première servlet.

3. Création d'une première servlet

La perspective Java EE nous montre une frise d'icône qui nous permet de créer directement des classes Java de servlet. C'est que nous allons utiliser pour notre première servlet.

La première rubrique du menu qui s'ouvre est celle qui nous permet de créer une servlet.

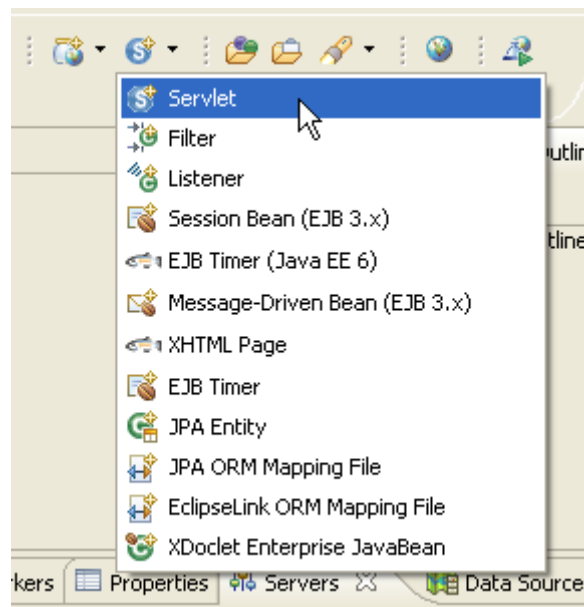


Figure 12. Création d'une première servlet

Un panneau s'ouvre alors, qui nous permet de spécifier la classe de cette servlet : son nom, et le nom du *package* dans lequel cette classe sera placée. Une servlet étant nécessairement `HttpServlet`, on ne doit donc prendre garde à ce point si l'on modifie la classe qu'étend notre servlet.

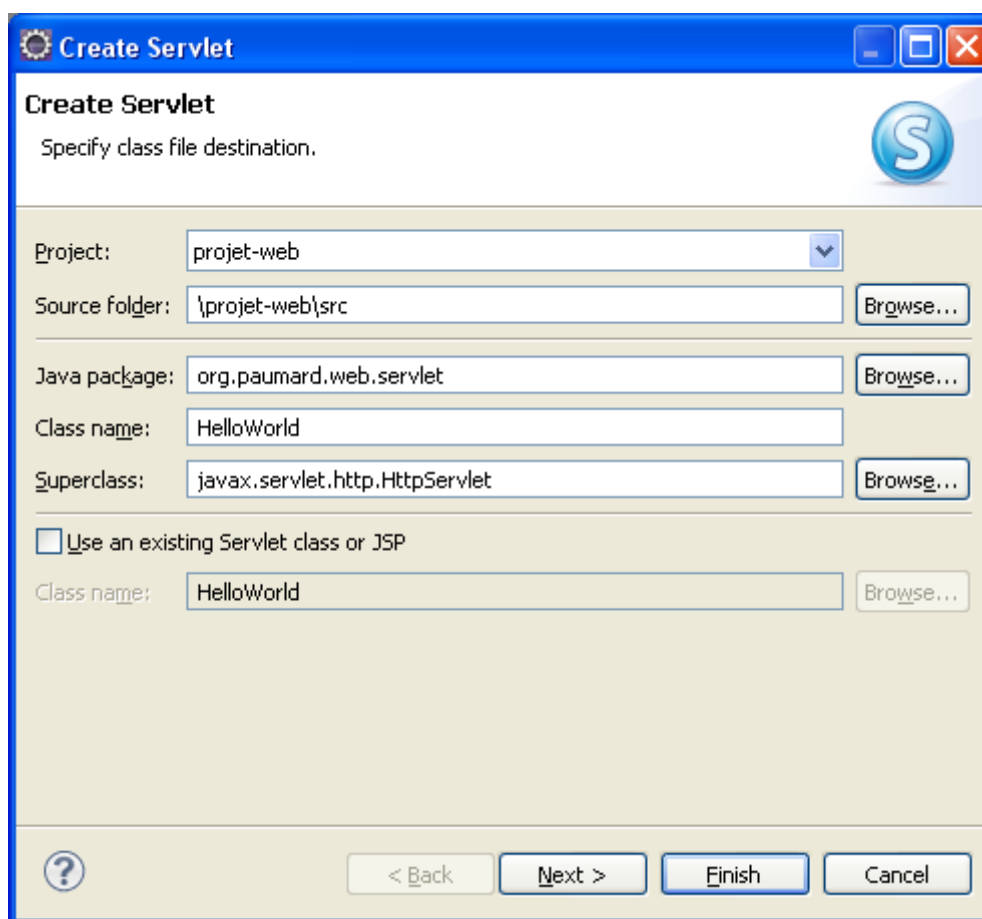


Figure 13. Création d'une première servlet : fixer la classe

Le panneau suivant nous permet de fixer les informations sur cette servlet, qui seront écrites dans le fichier web.xml de notre application.

- Le nom de notre servlet, utilisé pour s'y référer dans le fichier web.xml.
- Ses paramètres d'initialisation.
- Le (ou les) paramètres *URL mappings*, qui fixeront l'URL à laquelle notre servlet sera disponible.

Create Servlet

Enter servlet deployment descriptor specific information.

Name:

Description:

Initialization parameters:

Name	Value	Description
------	-------	-------------

URL mappings:

/HelloWorld

< Back Next > Finish Cancel

Figure 14. Création d'une première servlet : fixer ses paramètres

Le dernier panneau de création précise le contenu de la classe qu'Eclipse va nous générer. Par commodité, Eclipse peut créer pour nous différentes méthodes standard de notre servlet, telles que `init()`, `doGet()` ou `doPost()`.

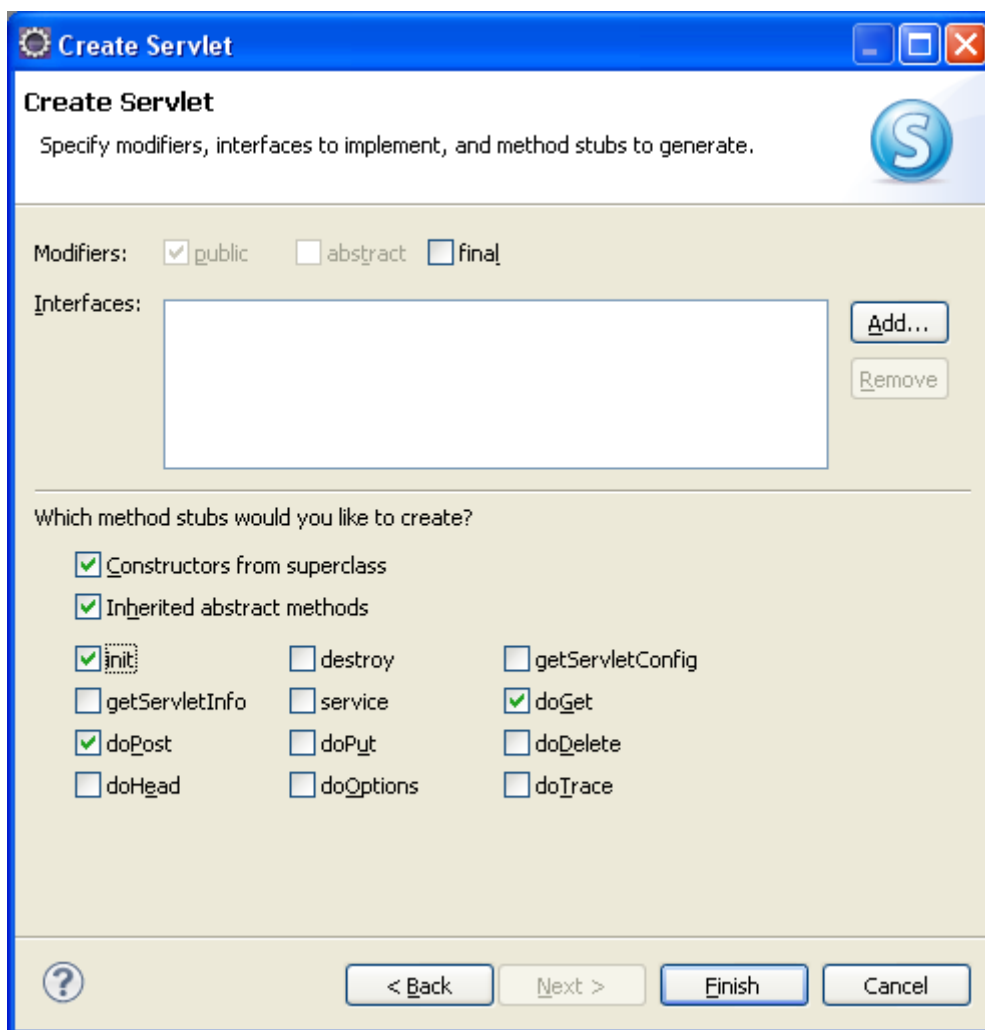


Figure 15. Création d'une première servlet : méthodes à générer

Une fois ces étapes déroulées, notre classe de servlet est créée, et apparaît dans la structure de notre projet.

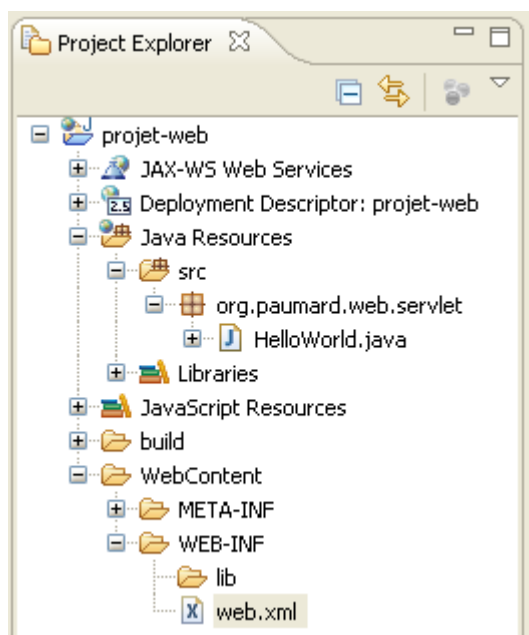


Figure 16. Servlet créée dans la structure du projet

Le fichier web.xml a bien été modifié, et prend bien en compte la servlet qui vient d'être créée.



Figure 17. Servlet déclarée dans le fichier web.xml

Notre servlet est très simple, son code est le suivant.

Exemple 1. Code d'une première servlet

```
public class HelloWorld extends HttpServlet {

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

        PrintWriter pw = response.getWriter();
        pw.write("Hello world !");
    }

    protected void doPost(
        HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        doGet(request, response);
    }
}
```

Une fois cette classe créée, il nous reste à la faire fonctionner dans Tomcat, ce qui est l'objet de la prochaine étape.

4. Première installation de Tomcat

Une servlet doit être exécutée dans un serveur d'applications, dans notre exemple il s'agit de Tomcat. La première étape consiste à créer une instance de serveur Tomcat, associée à notre espace de travail.

La création d'une instance de Tomcat (comme de tout autre serveur d'application) peut se faire directement dans la vue *Servers*, présente dans le bas de la perspective *Java EE*. Le menu contextuel de cette vue permet de créer cette nouvelle instance.

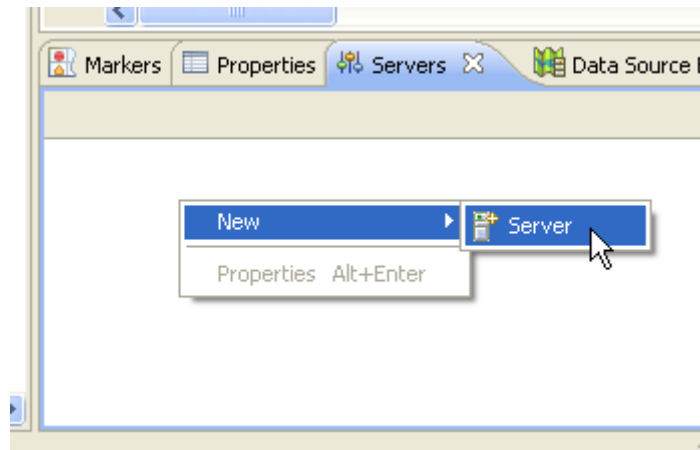


Figure 18. Création d'un serveur Tomcat dans la vue Servers

Le panneau qui s'ouvre alors permet de sélectionner le serveur qui va être créé. Pour cela, il faut renseigner les différents éléments du panneau.

- Le type de serveur : ici Tomcat v6.
- Le *Server host name* : ici localhost.
- Le *Server name* : ici Tomcat v6.0 Server at localhost. Ce nom est un nom logique utilisé par Eclipse. Il peut être utile de modifier sa valeur par défaut lorsque l'on fait fonctionner plusieurs serveurs de même type sur une même machine.
- Le *Server runtime environment* : c'est ici que l'on sélectionne le Tomcat que l'on a installé précédemment. On peut aussi installer de nouveaux serveur en cliquant sur le lien *Add...* qui se trouve à côté de ce sélecteur.

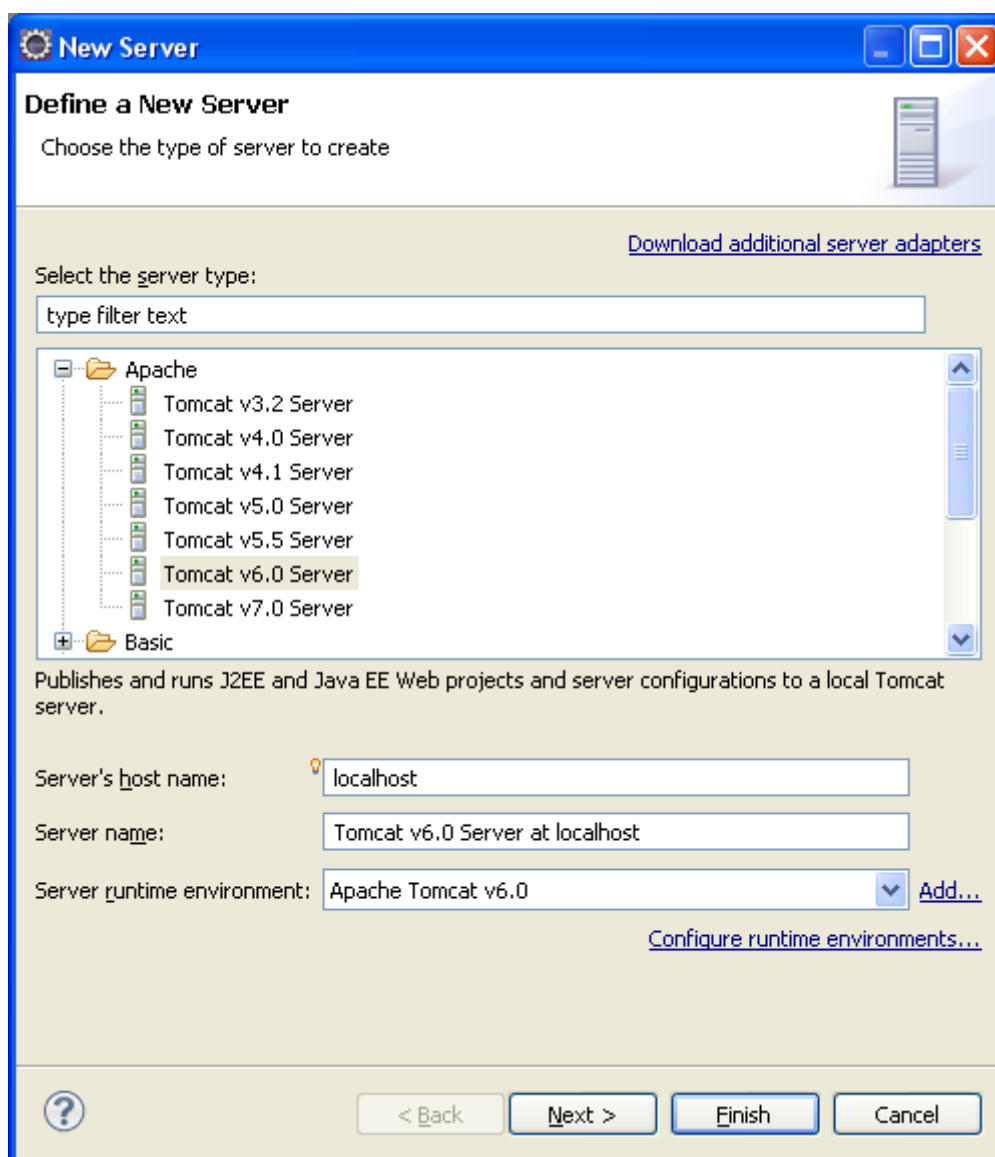


Figure 19. Sélection du serveur

Une fois l'instance de serveur créé, il faut lui ajouter des projets. Ici nous avons créé un serveur Tomcat, capable d'exécuter des servlets, on peut donc lui ajouter des projets qui contiennent des servlets. Eclipse ne nous propose ici que les projets web. Si nous avons d'autres projets Java dans notre espace de travail, ils n'apparaîtraient pas dans cette liste.

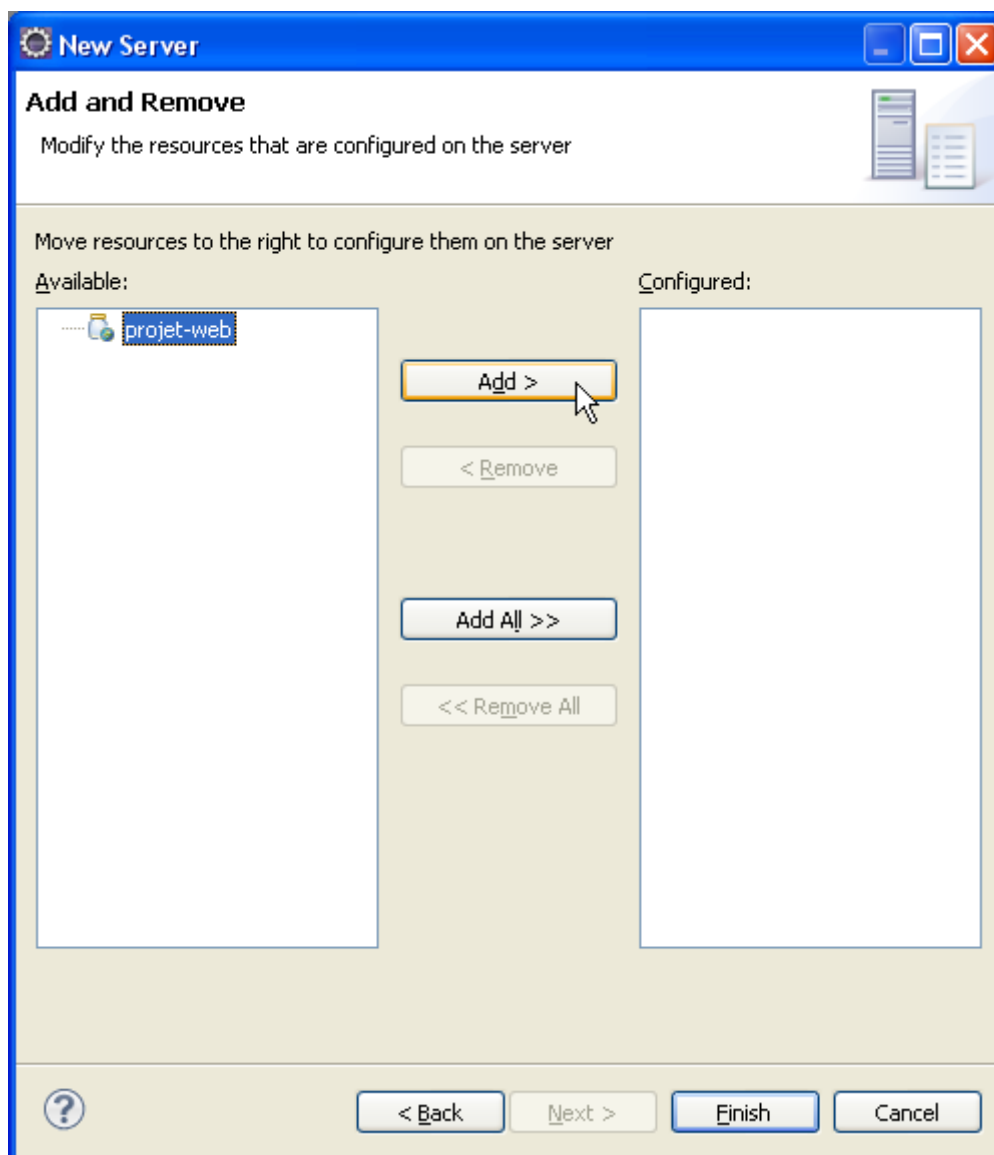


Figure 20. Ajout d'un projet web au serveur

Cette étape était la dernière : notre serveur est maintenant créé, et notre projet web lui a été ajouté. Notons que l'on peut aussi ajouter des projets à un serveur en les glissant / déposant à partir de la vue *Project explorer*.

Notre vue *Servers* a maintenant l'allure suivante.

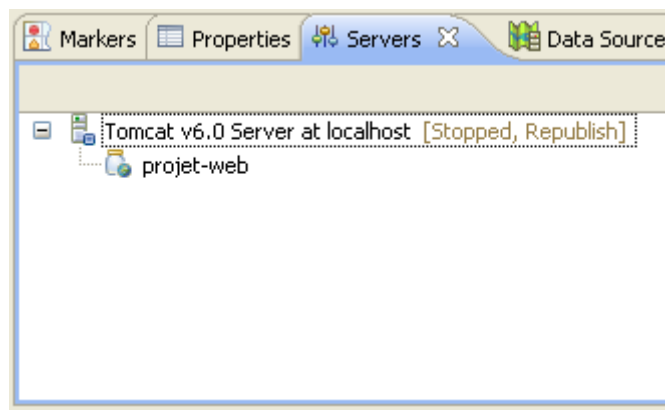


Figure 21. Vue *Server* une fois Tomcat installé

5. Exécution d'une première servlet

Lorsque toute la configuration précédente a été effectuée, l'exécution d'une servlet devient assez simple. Il suffit de sélectionner la traditionnelle option *Run as...* du menu contextuel de la classe que l'on veut exécuter. Comme cette classe ne comporte pas de méthode *main()*, la rubrique *Java Application* n'est pas présente. En revanche, nous avons la rubrique *Run on Server*, que nous allons sélectionner.

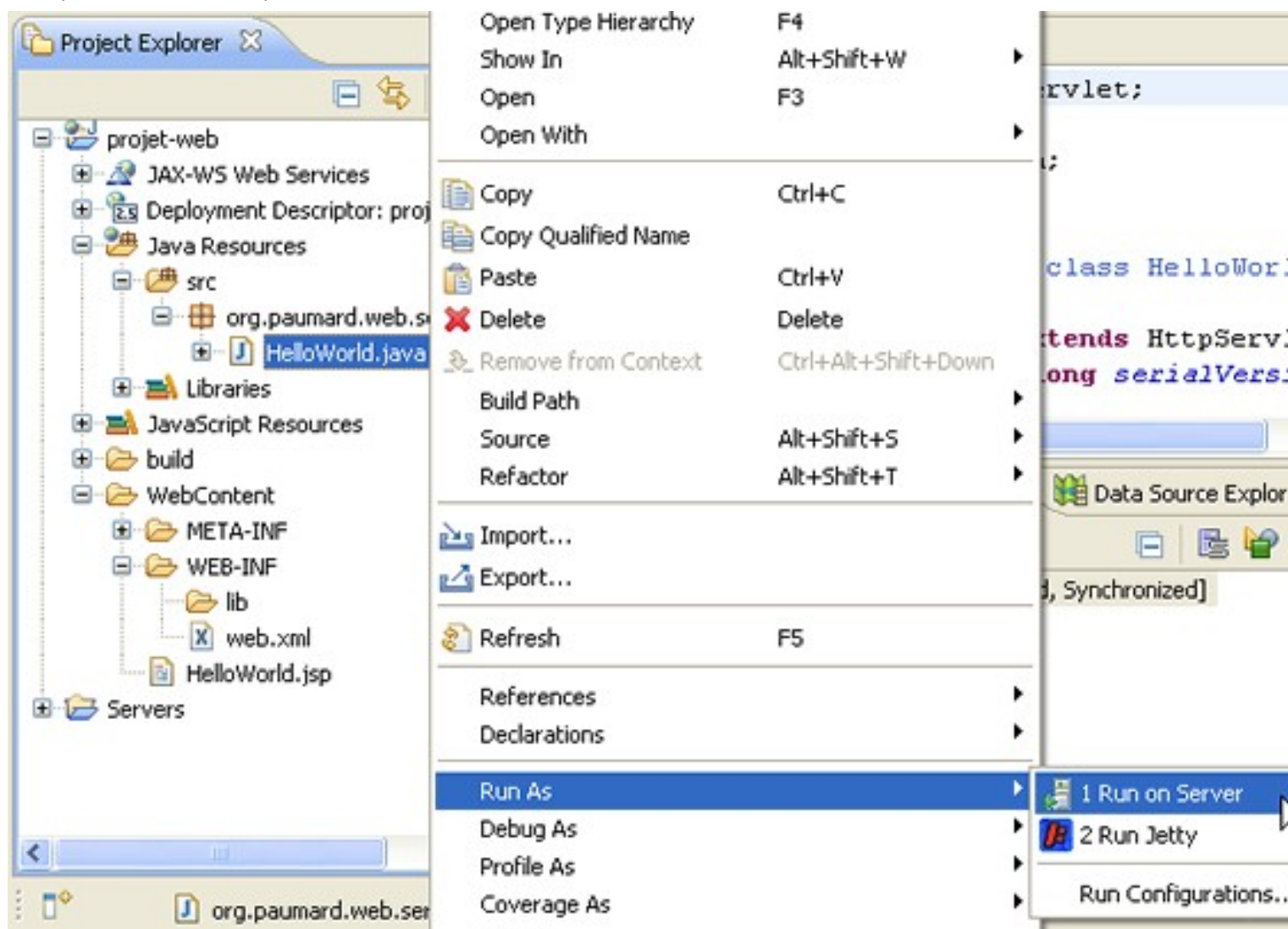


Figure 22. Lancement d'une servlet

Eclipse nous demande alors quel serveur va exécuter cette servlet. Ici nous n'en avons qu'un, dans une configuration complexe, il pourrait y en avoir plusieurs.

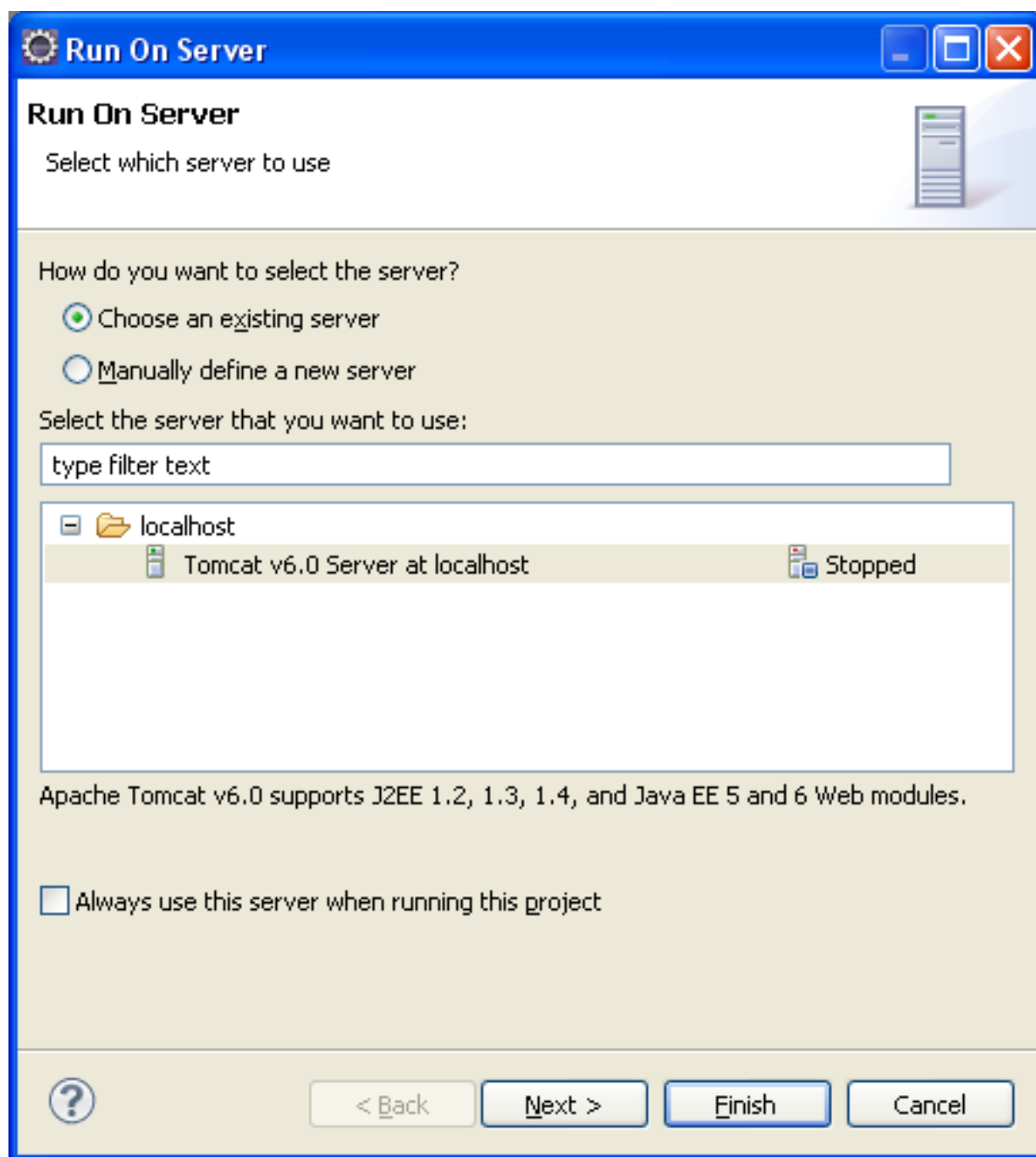


Figure 23. Choix d'un serveur d'exécution

Ici encore, il est possible d'ajouter des ressources de notre espace de travail au serveur. On peut sauter cette étape en cliquant directement sur le bouton *Finish* du panneau précédent.

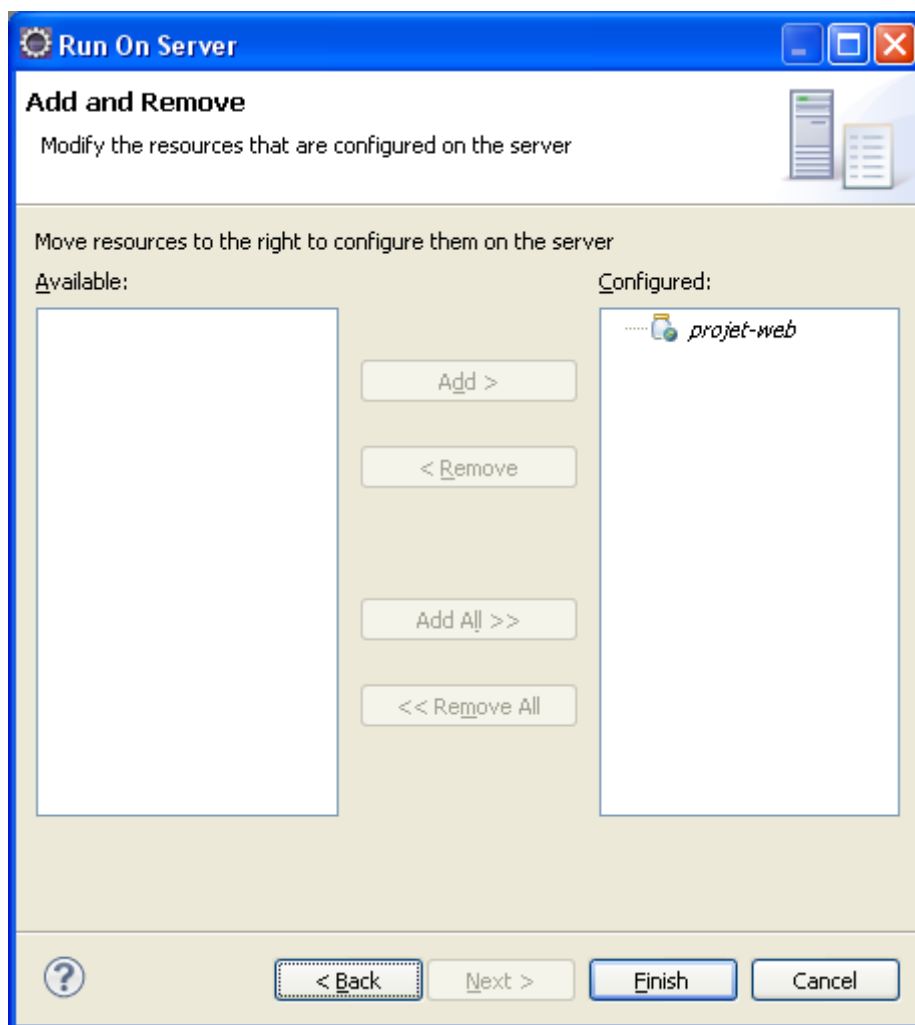


Figure 24. Sélection de ressources à ajouter au serveur

Eclipse nous ouvre enfin une vue *Navigateur*, sur l'URL associée à notre servlet. Cette URL est composée de deux éléments principaux : l'hôte sur lequel on a installé notre serveur, et l'élément d'URL auquel est associée notre servlet. Le port choisi par Eclipse est le port 8080, par défaut. Nous verrons dans la suite comment le changer.

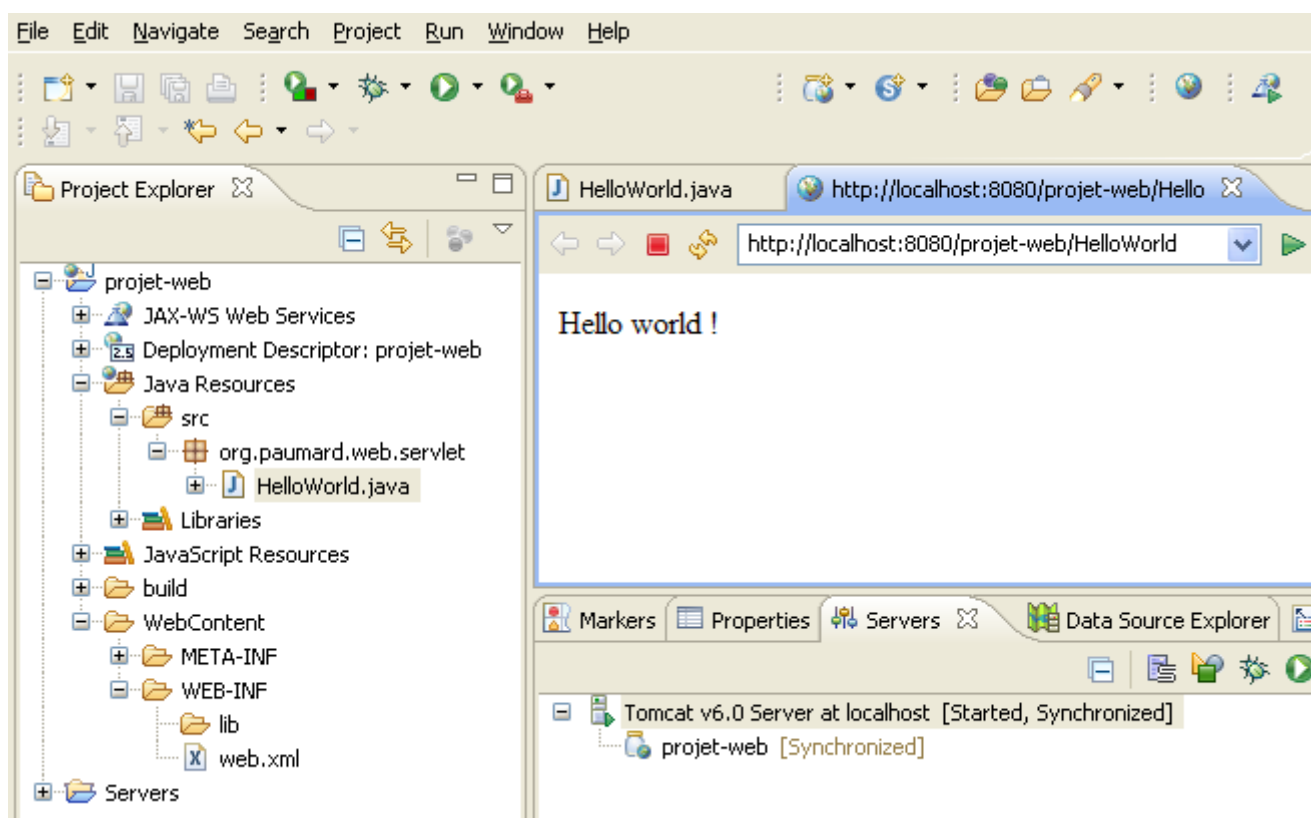


Figure 25. Une première servlet lancée

On peut également copier cette URL et la coller dans un navigateur lancé localement.

6. Création d'une première JSP

Créer et exécuter une page JSP suit le même processus qu'une servlet. Le menu contextuel du projet nous permet d'ouvrir le panneau de création d'une JSP.

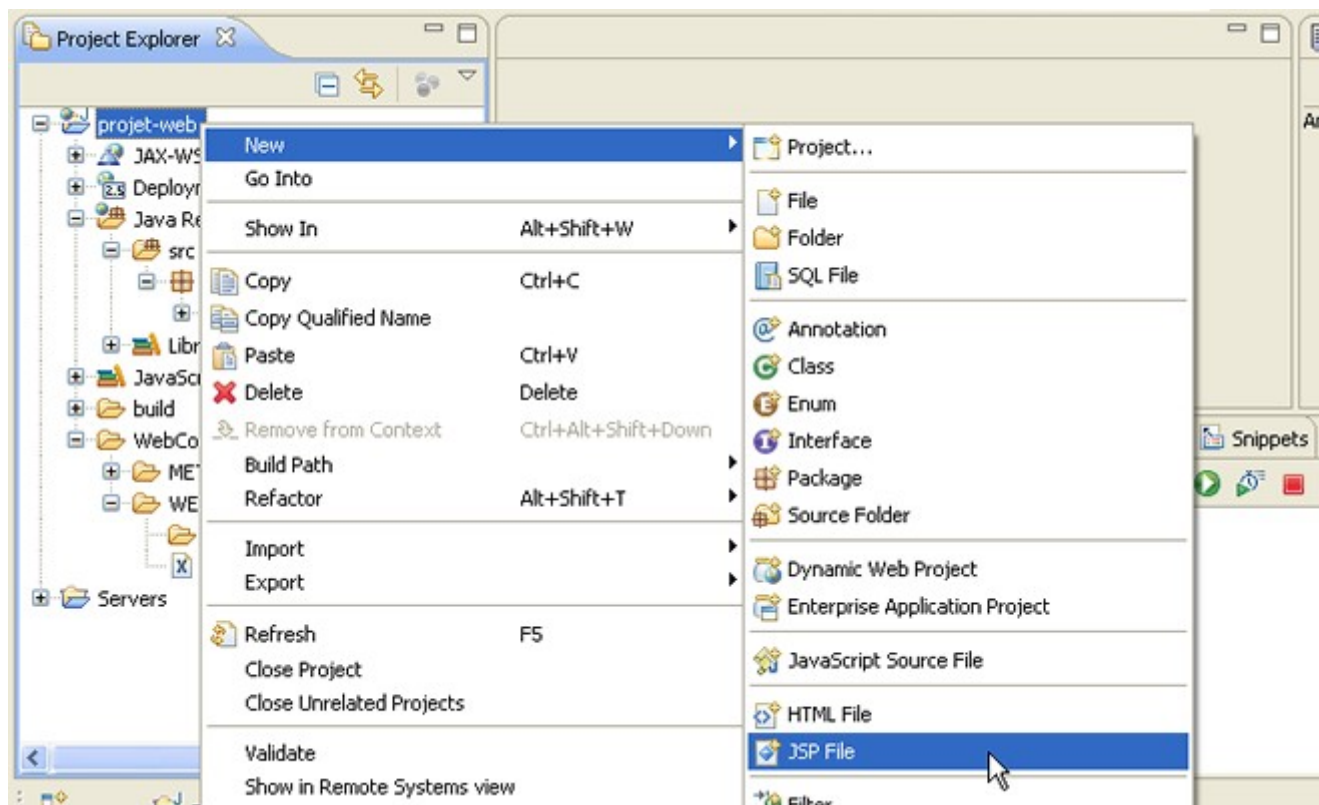


Figure 26. Création d'une JSP

Le premier panneau qui s'ouvre alors permet de fixer dans quel répertoire notre JSP va être créée. Ce répertoire doit se trouver sous celui que nous avons déclaré comme étant celui qui contient nos ressources web, au niveau de notre projet. Par défaut, il s'agit du répertoire WebContent.

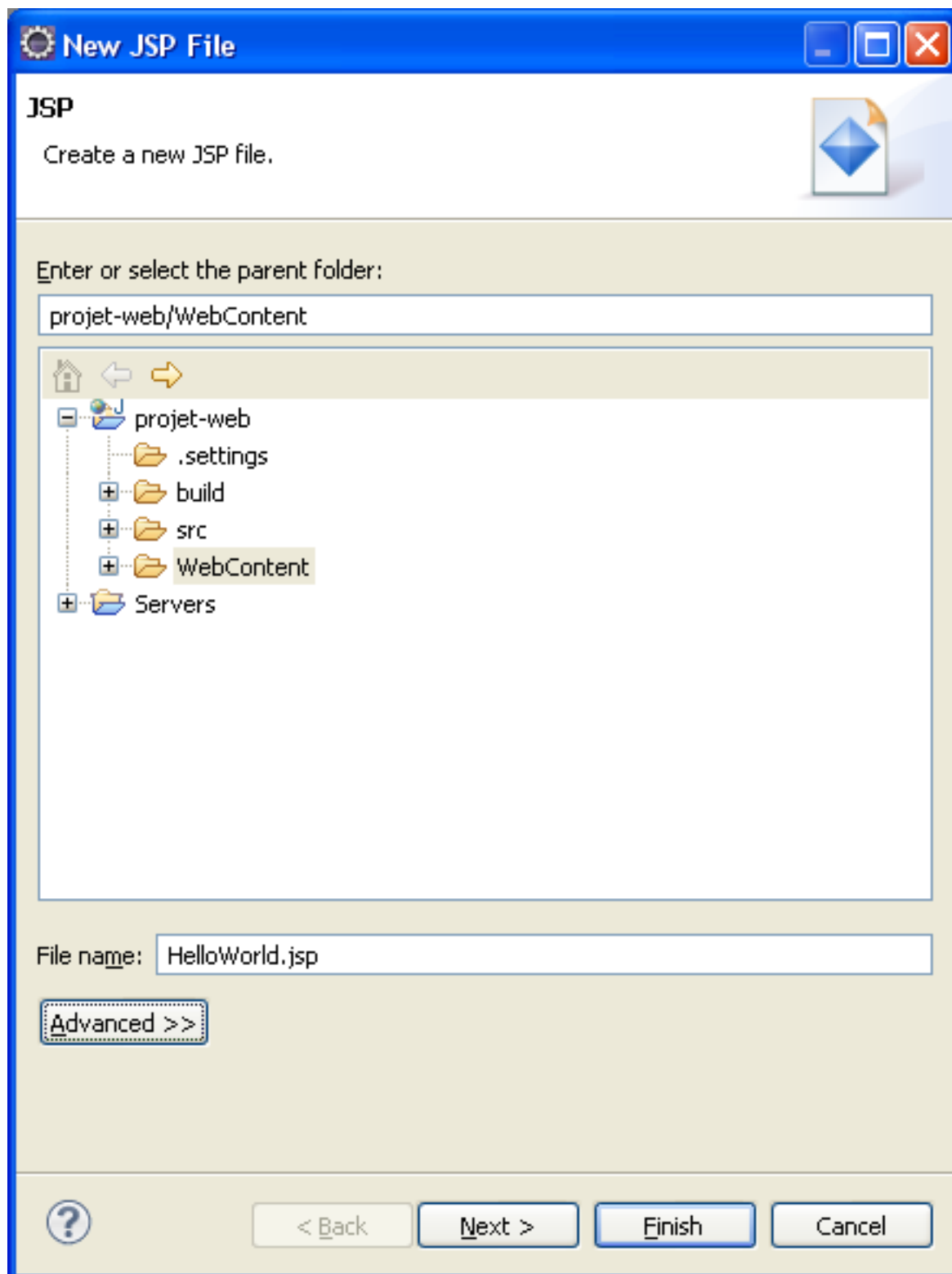


Figure 27. Choix du répertoire de création de notre JSP

Le second panneau est plus technique. Il nous demande le type de la JSP que nous voulons créer. Il y a sept possibilités, et l'objet de ce tutorial n'est pas de présenter les différences qui existent entre chacune. On sélectionne donc la quatrième : *JSP with html markup*.



Figure 28. Choix du type de JSP

Il ne nous reste plus qu'à valider la création de notre JSP. Elle apparaît sous le nœud WebContent de notre projet dans la vue *Project Explorer* sous la forme suivante.

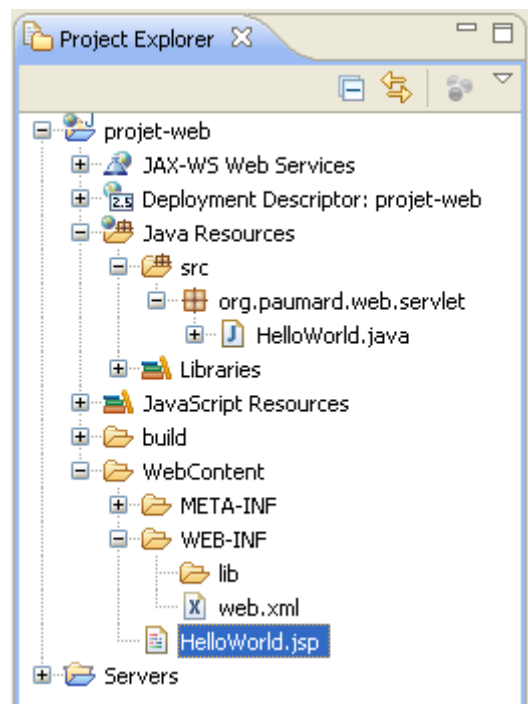


Figure 29. Structure de notre projet avec une JSP

7. Exécution d'une première JSP

Mettons un peu de code dans notre JSP avant de l'exécuter.

Exemple 2. Code d'une première JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    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>Insert title here</title>
</head>
<body>
    Bonjour le monde, il est <%=new java.util.Date() %>
</body>
</html>
```

Bien que n'étant pas (encore !) une classe Java, une JSP s'exécute de la même manière qu'une servlet, en allant chercher la rubrique *Run on Server* qui se trouve dans le sous-menu *Run as...* du menu contextuel de cette JSP.

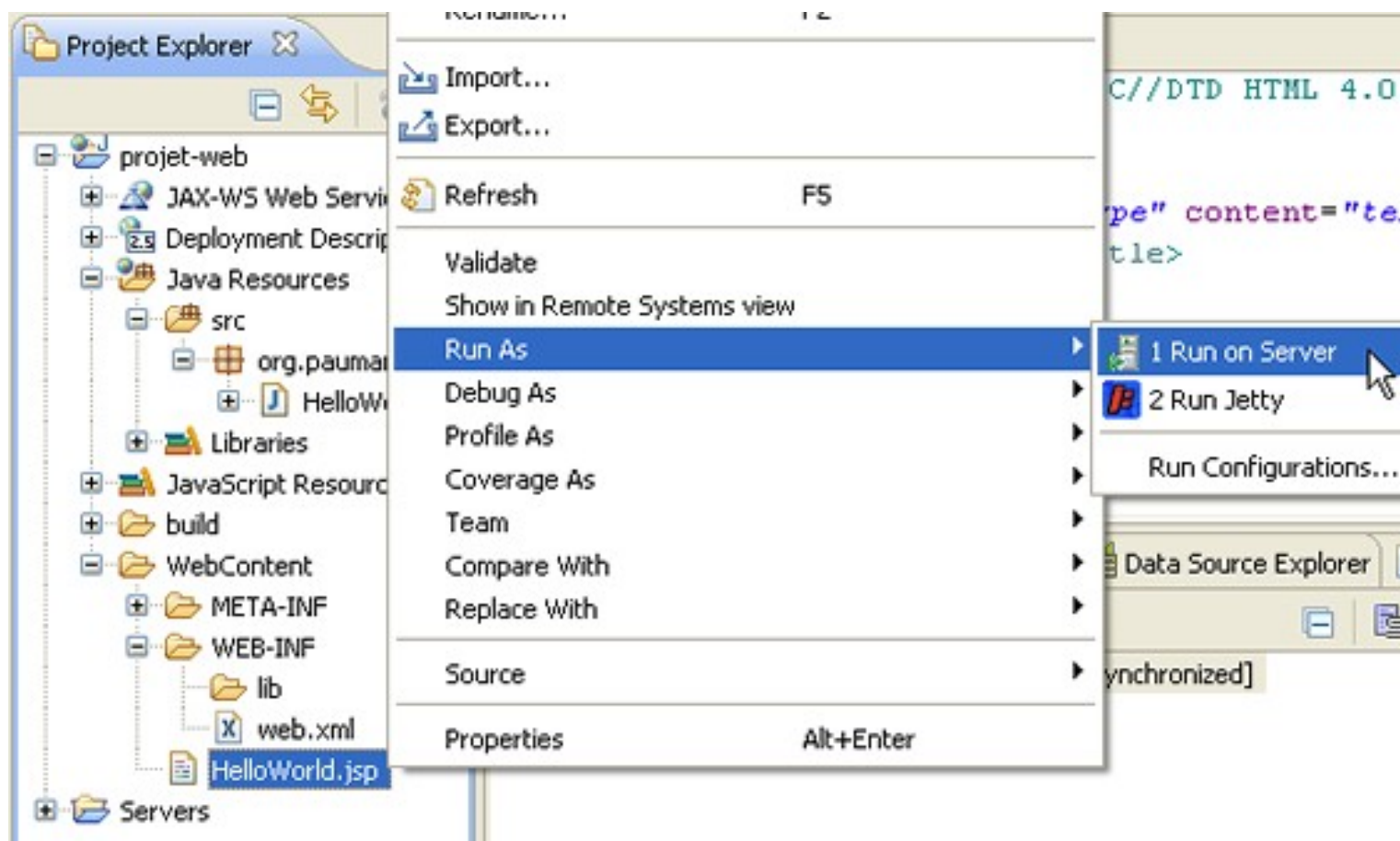


Figure 30. Exécution de notre première JSP

Eclipse nous demande alors quel serveur il doit utiliser pour cette exécution. Dans notre exemple nous n'en avons créé qu'un seul, le choix est donc facile.

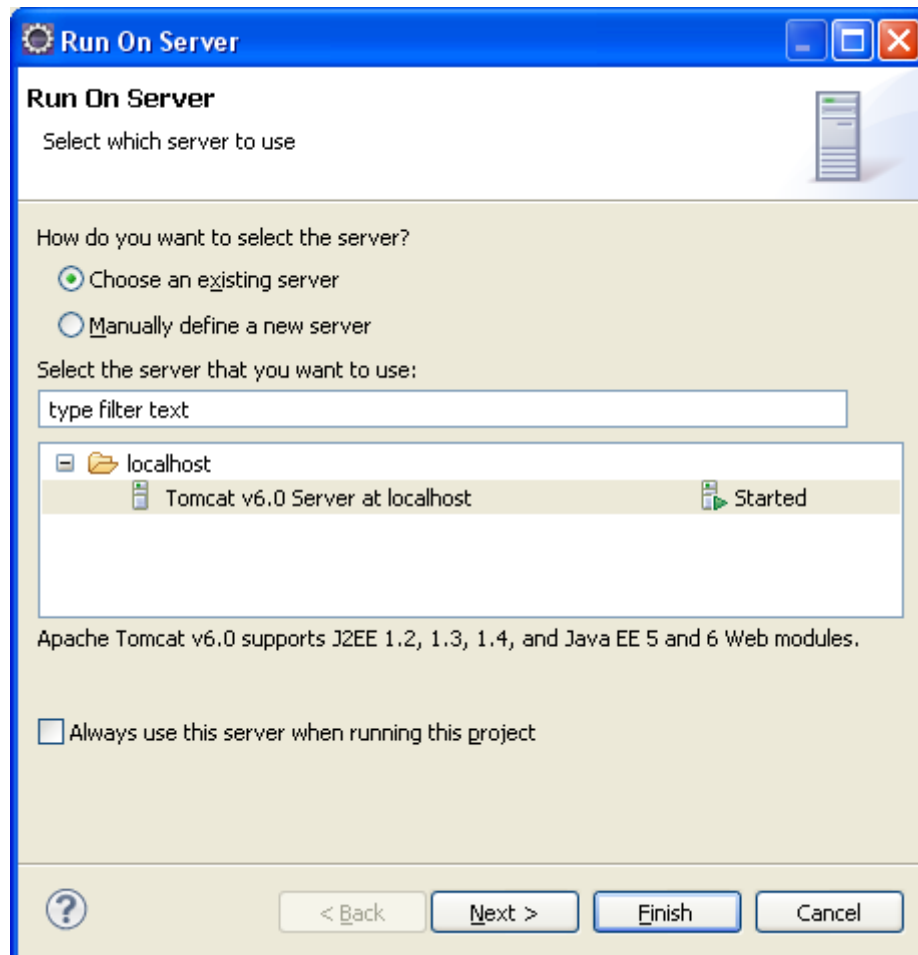


Figure 31. Choix du serveur d'exécution

De même que pour l'exécution d'une servlet, on peut alors changer les ressources connues de ce serveur.

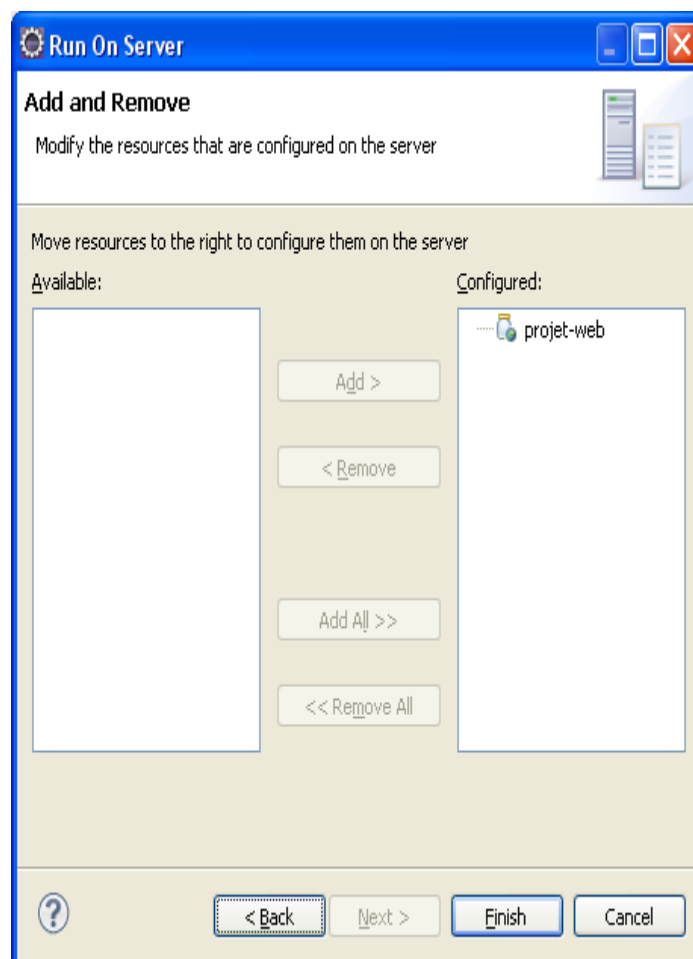


Figure 32. Choix des ressources pour l'exécution

Tout comme pour une servlet, notre exécution se fait dans le navigateur intégré à Eclipse. Eclipse nous l'affiche dans sa vue, comme sur la capture suivante.

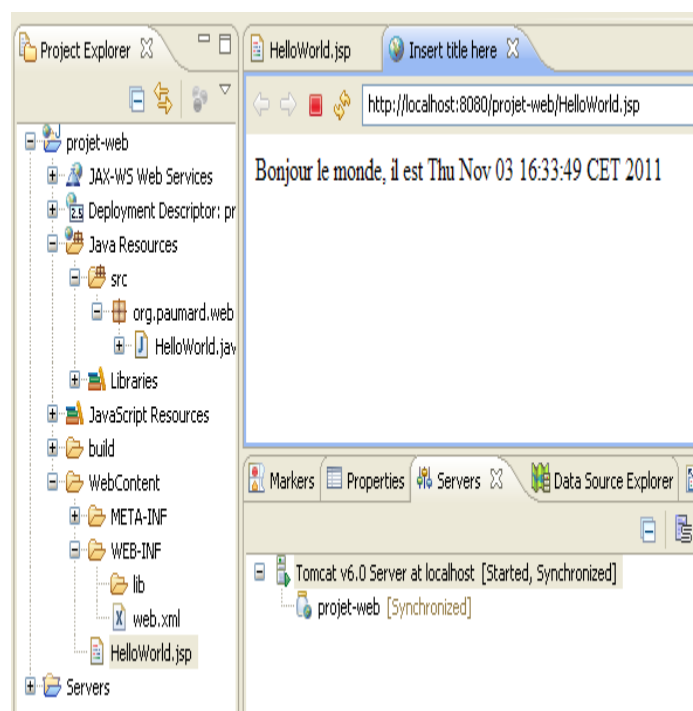


Figure 33. Exécution d'une JSP dans un navigateur

On peut bien sûr copier l'URL affichée par ce navigateur, et la coller dans un autre navigateur.

Lorsque l'on dit "exécuter une JSP" (ou une servlet) dans un navigateur, il ne faut pas se méprendre sur le sens des choses. La servlet, ou la JSP est exécutée dans une JVM, dans Tomcat, et ce que l'on observe dans la navigateur, ce n'est que l'affichage de la page contenant le code HTML généré par cette servlet ou cette JSP.

8. Arrêt et redémarrage de Tomcat

Notons que la vue *Servers* présente des boutons qui nous permettent d'arrêter, et de relancer tomcat. Le lancement de Tomcat peut se faire en mode normal ou en mode *debug*, ce qui permet d'exécuter pas à pas nos servlets, nos JSPs, et nos services web si l'on en avait.

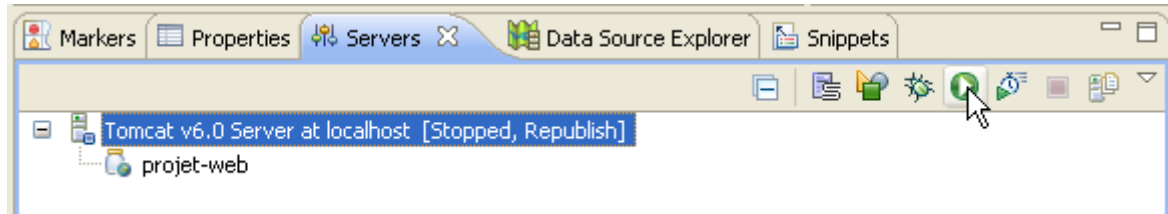


Figure 34. Lancement de Tomcat en mode normal

On peut aussi "rafraîchir" Tomcat. Cela signifie que l'on force le rechargement de toute ressource (servlet ou JSP notamment) qui aurait été modifiée dans Eclipse, alors que Tomcat était en cours d'exécution.

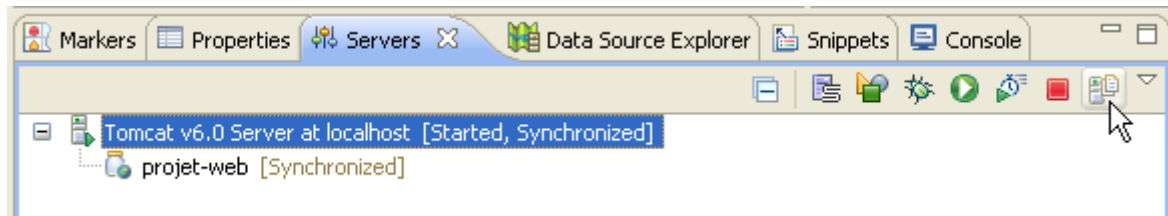


Figure 35. Rafraîchissement de Tomcat

Enfin on peut accéder à un panneau de configuration de Tomcat, simplement en double-cliquant dessus dans la vue *Servers*. Ce panneau s'affiche dans une vue spéciale, qui porte le nom que l'on a donné à l'instance. Elle permet de configurer de nombreux paramètres, dont le port d'écoute HTTP de Tomcat, ce qui est utile en cas d'instances multiples par exemple.

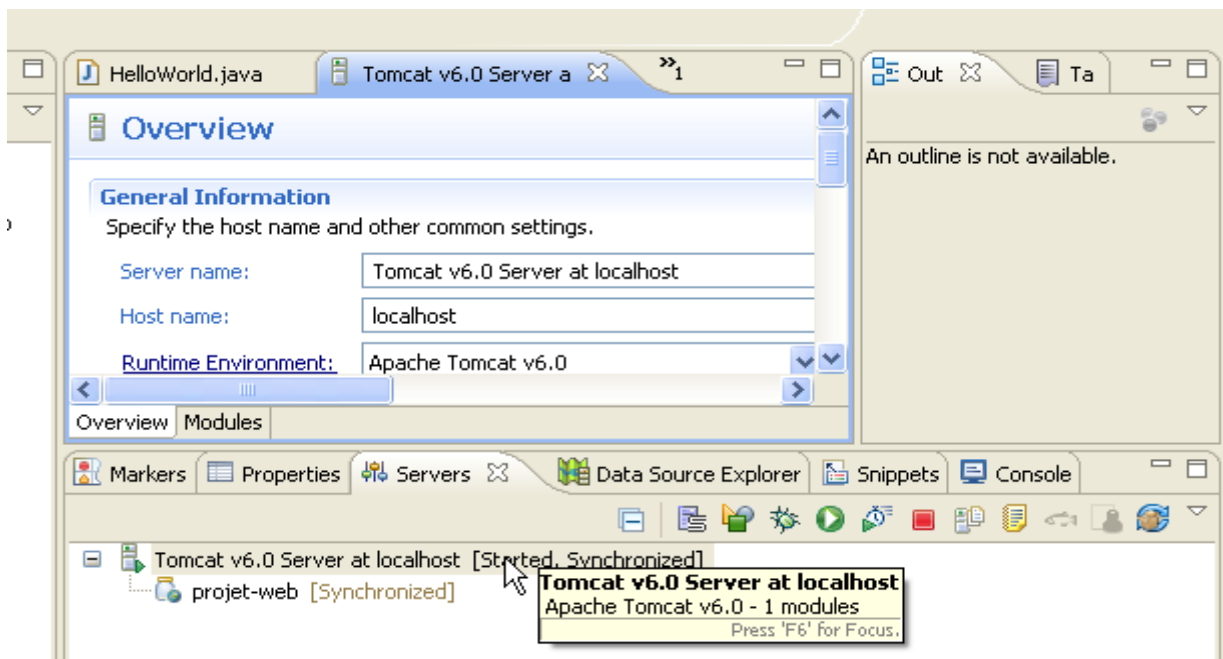


Figure 36. Panneau de configuration de Tomcat

