

TD – Java EE 1

Omar ABOU KHALED, Stefano CARRINO, Joël DUMOULIN

Buts du travail

- Familiarisation avec Java EE
- Création de Servlets
- Mise en place de vues JSP
- Introduction aux Java Beans
- Utilisation d'EL et de JSTL

Composants nécessaires

Les composants suivants sont nécessaires à ce travail pratique :

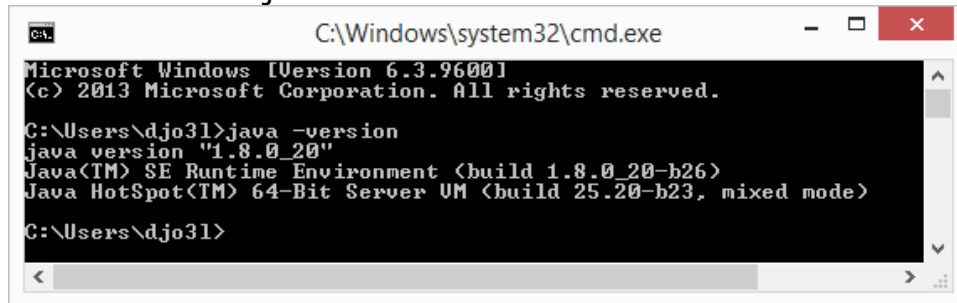
Composants	Remarques
Java JDK 7	http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Eclipse Luna (4.4) for Java EE Developers	https://www.eclipse.org/downloads/
Apache Tomcat 8	http://tomcat.apache.org/download-80.cgi
JSTL 1.2	http://download.java.net/maven/1/jstl/jars/jstl-1.2.jar

Travail à réaliser

1. Installation
2. Création d'un projet web
3. Servlet
4. JSP
5. JavaBean
6. EL
7. JSTL
8. A vous de jouer !

1 Installation

- Le Java JDK 8 doit être installé
 - o Vérification : > java -version



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

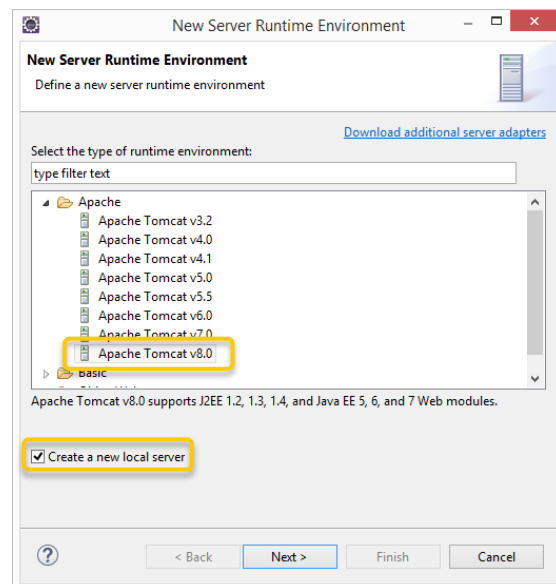
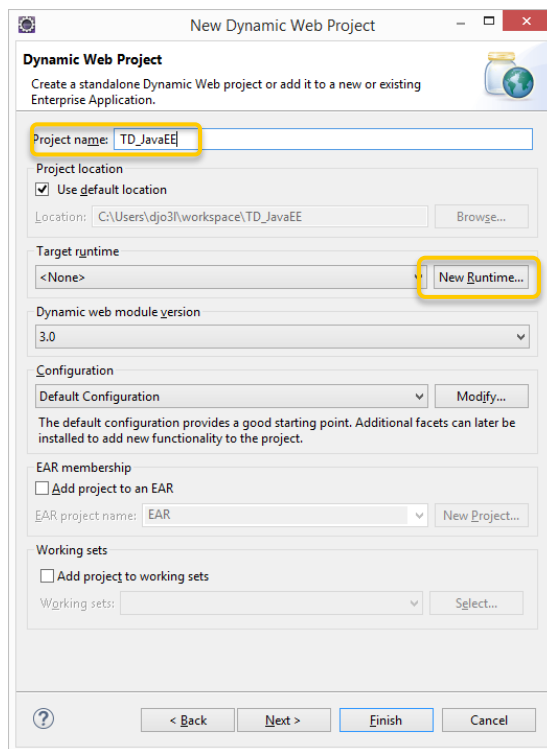
C:\Users\djo31>java -version
java version "1.8.0_20"
Java(TM) SE Runtime Environment (build 1.8.0_20-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.20-b23, mixed mode)

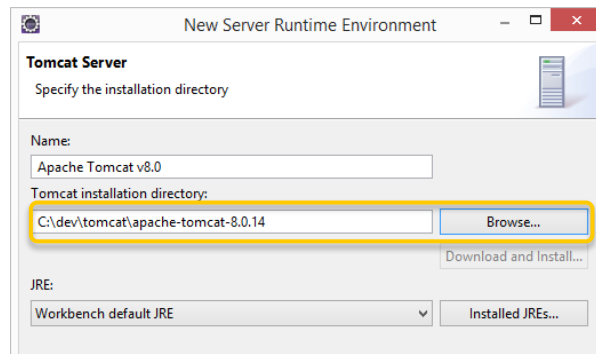
C:\Users\djo31>
```

- Décompressez Eclipse
 - o Ex : sous C:\dev\eclipse
- Décompressez Tomcat
 - o Ex : sous C:\dev\tomcat

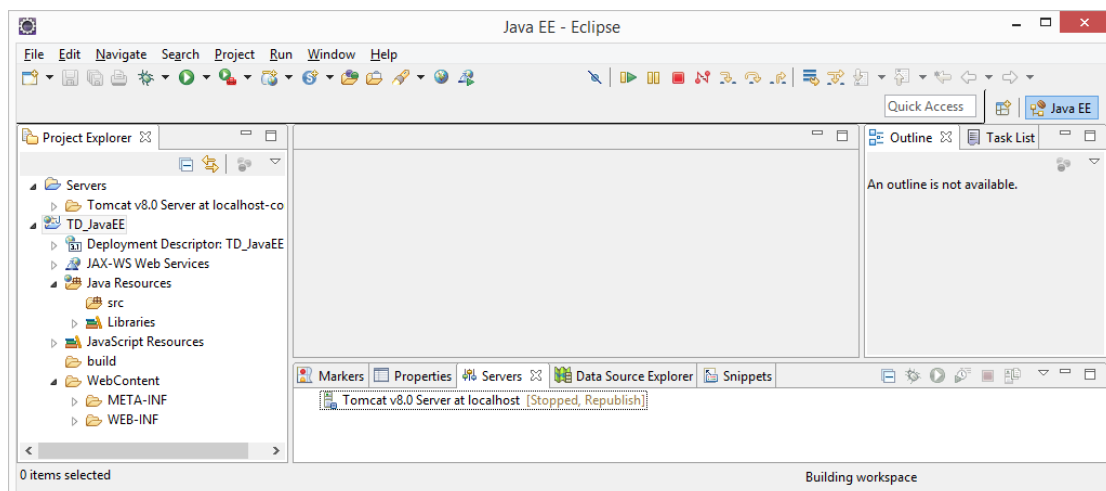
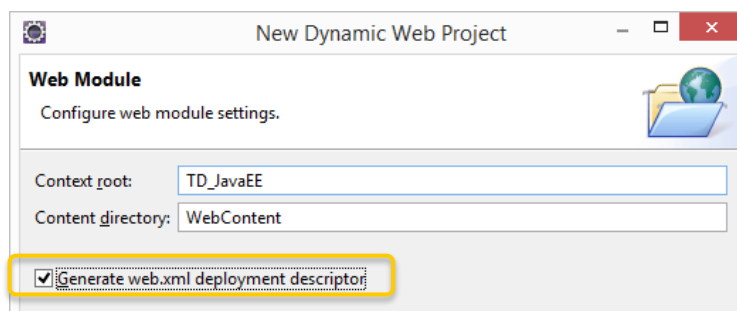
2 Création d'un projet web

- New project : CTRL+N → Web → Dynamic Web Project

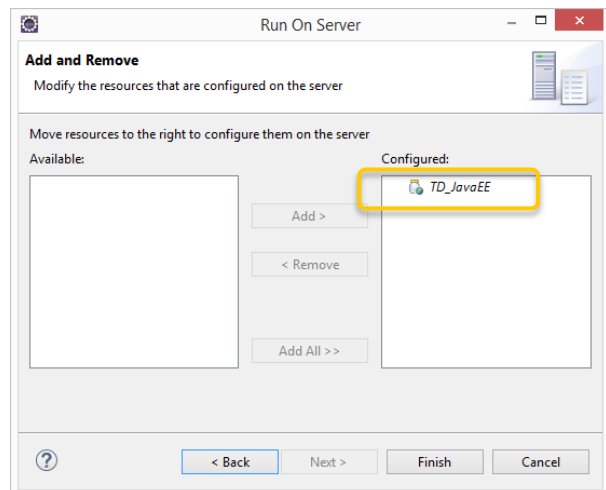
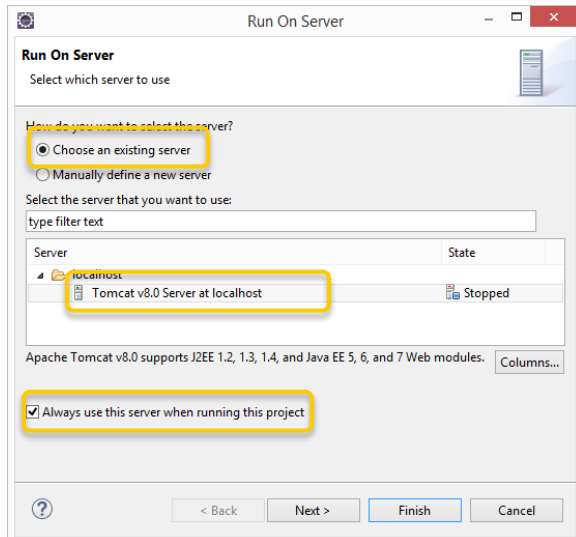




- Finish → Next → Next



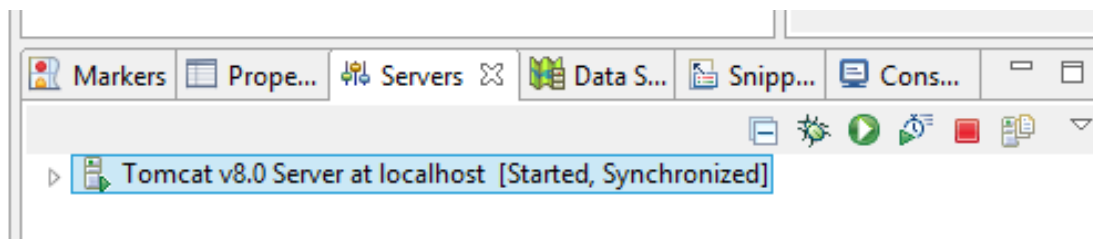
- Clic droit sur le projet → Run As → Run on Server



- Finish
 - o Si message du Firewall, accepter !

Remarque :

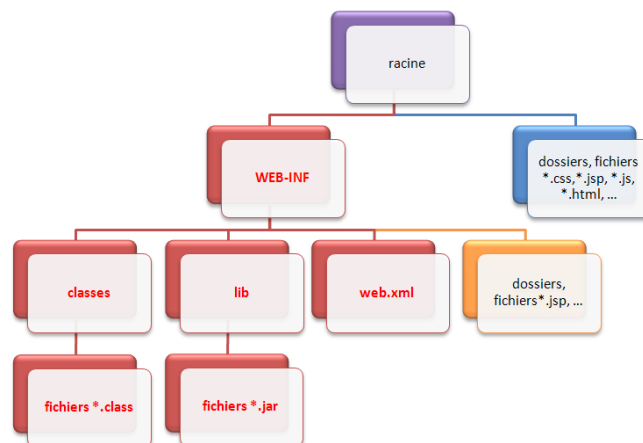
- A la fin de Servers/Tomcat v8.0.../server.xml
 - o `<Context docBase="TD_JavaEE" path="/TD_JavaEE" reloadable="true" source="org.eclipse.jst.jee.server:TD_JavaEE"/></Host>`



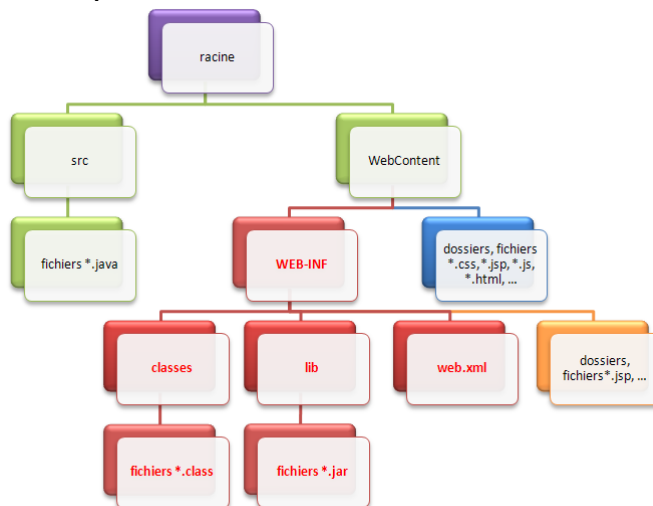
- Création d'une page HTML
 - o CTRL+N → Web → HTML File → Next
 - Nommer le nouveau fichier « HelloWorld.html » et le placer sous le dossier « WebContent » du projet
 - Finish
 - Ajouter un message « Hello world ! »
 - Run

2.1 Notes

- Tomcat n'est pas un serveur d'applications Java EE au sens complet du terme.
- La configuration du serveur passe principalement par deux fichiers : server.xml et web.xml.
- Une application web Java EE doit respecter une architecture bien définie :



- Eclipse modifie l'architecture des applications pour les intégrer correctement à son système.

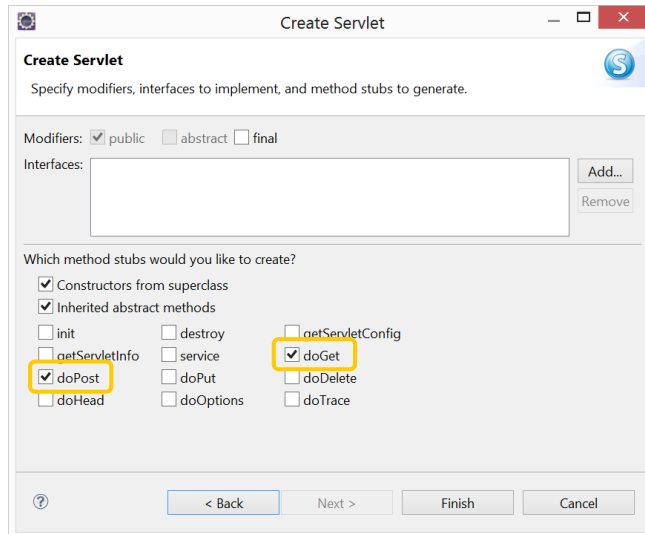


3 Servlet

3.1 Création

- Dans l'explorateur du projet, sur *ProjectName* → *Deployment Descriptor* → *Servlet*, cliquez-droit puis *New* → *Servlet*
- Spécifiez un nom de classe (ex : HelloWorld) et un package (ex : ch.si.servlets), puis *Next*
- L'écran suivant permet de configurer les informations de la servlet, qui seront reportées dans le fichier web.xml de l'application. Vous pouvez laisser la configuration par défaut.
- Le dernier écran vous permet de paramétrer la servlet, en indiquant par exemple les différentes méthodes à auto-générer (ex : doGet(),

doPost(), etc.). Assurez-vous que les méthodes doGet() et doPost() soient cochées. → *Finish*



- Le code auto généré :
 - L'annotation indiquant au serveur que cette classe est une servlet, et qu'elle est accessible via l'URL */HelloWorld*. Auparavant, ce type de configuration devait se faire dans le descripteur de déploiement. L'utilisation d'annotation est elle extrêmement concise et pratique.

```
@WebServlet("/HelloWorld")
```

- La classe étend la classe abstraite *HttpServlet*.

```
public class HelloWorld extends HttpServlet {
    ...
}
```

- Le squelette pour la gestion des requêtes get/post prend en paramètres la requête ainsi qu'un objet de type *HttpServletResponse* pour gérer la réponse.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}
```

3.2 Get

- Dans la servlet, ajoutez le code suivant à la méthode doGet() afin de mettre en place un HelloWorld :

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head>");
out.println("<title>Hello world servlet</title>");
out.println("</head>");
out.println("<body>");

out.println("<h1>Hello World Servlet</h1>");
out.println("Hello World!");

out.println("</body>");
out.println("</html>");
out.close();
```

- Testez le bon fonctionnement de votre servlet

3.3 Post

- Créez un petit formulaire dans une page HTML (*HelloWorld.html* à placer dans le répertoire *WebContent*), afin d'utiliser notre servlet et afficher un message Hello personnalisé à l'aide d'un paramètre *name*

```
/* A completer */
<form action="" method="">
  <label>Name</label>
  <input type="text" name="name"/>
  <input type="submit"/>
</form>
```

- Complétez la méthode `doPost()` afin de récupérer la valeur du paramètre *name* (voir `request.getParameter()`) et afficher le message.

3.4 Notes

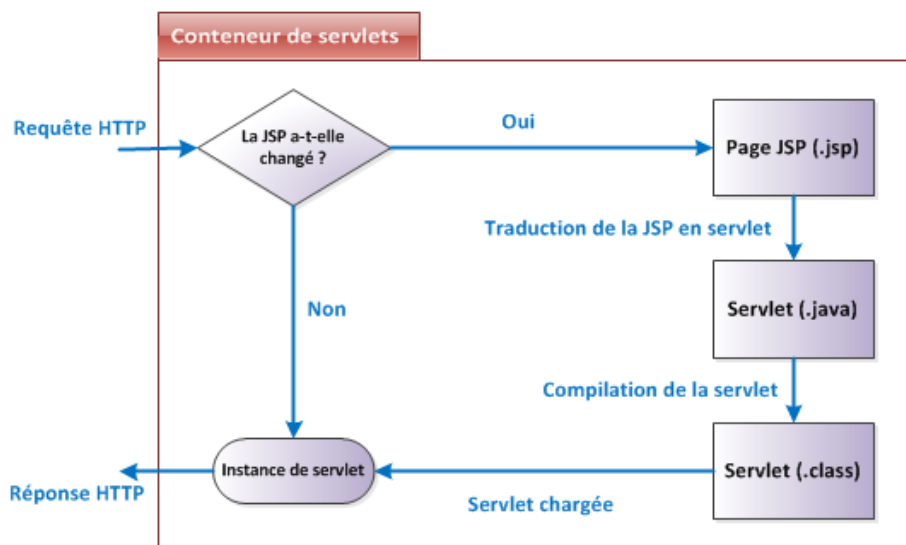
- Le client envoie des requêtes au serveur grâce aux méthodes du protocole HTTP, notamment GET, POST et HEAD.
- Le conteneur web place chaque requête reçue dans un objet `HttpServletRequest`, et place chaque réponse qu'il initialise dans l'objet `HttpServletResponse`.
- Le conteneur transmet chaque couple requête/réponse à une servlet : c'est un objet Java assigné à une requête et capable de générer une réponse en conséquence.
- La servlet est donc le point d'entrée d'une application web.
- Pour pouvoir traiter une requête HTTP de type GET, une servlet doit implémenter la méthode `doGet()` ; pour répondre à une requête de type POST, la méthode `doPost()` ; etc.
- Une servlet n'est pas chargée de l'affichage des données, elle ne doit donc pas s'occuper de la présentation (HTML, CSS, etc.).

4 JSP

- La technologie servlet est trop difficile d'accès et ne convient pas à la génération du code de présentation
 - o Les pages JSP sont en quelque sorte une abstraction "haut niveau" de la technologie servlet.
- On peut résumer la technologie JSP en une technologie offrant les capacités dynamiques des servlets tout en permettant une approche naturelle pour la création de contenus statiques. Ceci est rendu possible par :
 - o un langage dédié : les pages JSP sont des documents au format texte, à l'opposé des classes Java que sont les servlets, qui décrivent indirectement comment traiter une requête et construire une réponse. Elles contiennent des balises qui combinent à la fois simplicité et puissance, via une syntaxe simple, semblable au HTML et donc aisément compréhensible par un humain ;
 - o la simplicité d'accès aux objets Java : des balises du langage rendent l'utilisation directe d'objets au sein d'une page très aisée ;
 - o des mécanismes permettant l'extension du langage utilisé au sein des pages JSP : il est possible de mettre en place des balises qui n'existent pas dans le langage JSP, afin d'augmenter les fonctionnalités accessibles.

4.1 Création

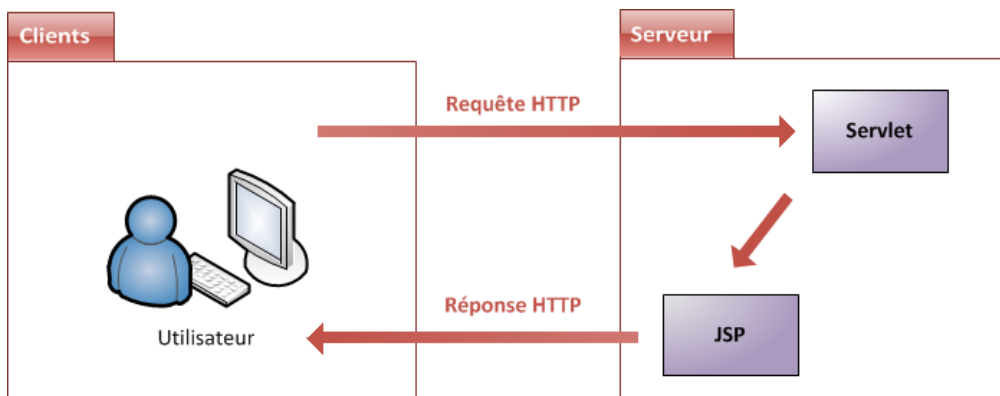
- Vous pourriez créer une nouvelle page JSP, mais pour cet exercice, modifiez simplement l'extension de votre *HelloWorld.html*
- Si vous rechargez cette page, vous devez obtenir exactement la même chose. Seulement, du côté serveur, toute votre page JSP va être convertie en code Java (servlet), compilé, puis chargé. Cette compilation ne se fait que la première fois, et ensuite uniquement si il y a des modifications.



- Déplacez la page JSP dans le répertoire WEB-INF
 - o La page n'est plus accessible, car les ressources sous ce dossier sont cachées automatiquement
 - o Nous devons la rendre disponible via une Servlet...
 - o Remplacez le corps de la méthode doGet() :

```
this.getServletContext().getRequestDispatcher("/WEB-INF/HelloWorld.jsp").forward(request, response);
```

- depuis notre instance de servlet, nous appelons la méthode `getServletContext()`. Celle-ci nous retourne alors un objet `ServletContext`, qui fait référence au contexte commun à toute l'application : celui-ci contient un ensemble de méthodes qui permettent à une servlet de communiquer avec le conteneur de servlet ;
- celle qui nous intéresse ici est la méthode permettant de manipuler une ressource : `getRequestDispatcher()`. Elle retourne un objet `RequestDispatcher`, qui agit ici comme une enveloppe autour de notre page JSP. Il est impératif d'y préciser le chemin complet vers la JSP, en commençant obligatoirement par un / ;
- nous utilisons enfin ce dispatcher pour réexpédier la paire requête/réponse HTTP vers notre page JSP via sa méthode `forward()`.
- Remarque : le dossier `WebContent` existe uniquement pour Eclipse !



4.2 Echange de données

- Partage de données depuis la Servlet

```
String message = "Message from the servlet!";  
request.setAttribute("msg", message);
```

- Récupération de données dans la JSP

```
<p>Get attribute:
<%
String attribute = (String) request.getAttribute("msg");
out.println(attribute);
%>
</p>

<p>Get parameter:
<%
String parameter = (String) request.getParameter("name");
out.println(parameter);
%>
</p>
```

4.3 Balises

- Les commentaires JSP (<%-- ... --%>) ne sont pas, au contraire des commentaires HTML (<!-- ... -->), présent dans le code HTML généré.
- Les balises de déclaration (<% ! ... %>) permettent de déclarer des variables et fonctions.

```
<%! String hello = "World"; %>
```

- Les balises de scriptlet - « script » + « servlet » !- (<% ... %>) permettent d'utiliser du code java.

```
<%
for(int i=0; i < 5; i++){
    out.println(name + " " + i);
}
%>
```

- La balise d'expression est un raccourci de la scriptlet de print :

```
<% out.println("toto"); %>
Peut être remplacé par:
<%= "toto" %>
```

- Les directives JSP permettent

- o D'importer un package

```
<%@ page import="java.util.List, java.util.Date" %>
```

- o D'inclure d'autres pages JSP

```
<%@ include file="uneAutreJSP.jsp" %>
```

- o D'inclure des bibliothèques de balises (Taglib)

```
<%@ taglib uri="maTagLib.tld" prefix="tagExemple" %>
```

- o De définir des propriétés et informations relatives à une page JSP

4.4 Notes

- Une page JSP ressemble en apparence à une page HTML, mais en réalité elle est bien plus proche d'une servlet : elle contient des balises derrière lesquelles se cache du code Java.
- Une page JSP est exécutée sur le serveur, et la page finale générée et envoyée au client est une simple page HTML : le client ne voit pas le code de la JSP.
- Idéalement dans le modèle MVC, une page JSP est accessible à l'utilisateur à travers une servlet, et non pas directement.
- Le répertoire /WEB-INF cache les fichiers qu'il contient à l'extérieur de l'application.
- La méthode `forward()` de l'objet `RequestDispatcher` permet depuis une servlet de rediriger la paire requête/réponse HTTP vers une autre servlet ou vers une page JSP.
- Un attribut de requête est en réalité un objet stocké dans l'objet `HttpServletRequest`, et peut contenir n'importe quel type de données.
- Les attributs de requête sont utilisés pour permettre à une servlet de transmettre des données à d'autres servlets ou à des pages JSP.
- Un paramètre de requête est une chaîne de caractères placée par le client à la fin de l'URL de la requête HTTP.
- Les paramètres de requête sont utilisés pour permettre à un client de transmettre des données au serveur.

4.5 Taglib

Vous pouvez écrire vos propres librairies de tag. Les étapes de la création d'une nouvelle taglib sont les suivantes :

1. Création du fichier de description taglib (TLD)
2. Création de la classe Java
3. Définition du taglib dans le *web.xml*
4. Déclaration du taglib dans le fichier JSP, et utilisation du taglib

Exemple : <http://adiguba.developpez.com/tutoriels/j2ee/jsp/taglib/>

5 JavaBean

Le JavaBean désigne un composant réutilisable. C'est en fait une simple classe Java, qui respecte certains standards :

- doit être une classe publique ;
- doit avoir au moins un constructeur par défaut, public et sans paramètres. Java l'ajoutera de lui-même si aucun constructeur n'est explicité ;
- peut implémenter l'interface `Serializable`, il devient ainsi persistant et son état peut être sauvegardé ;
- ne doit pas avoir de champs publics ;

- peut définir des propriétés (des champs non publics), qui doivent être accessibles via des méthodes publiques `getter` et `setter`, suivant des règles de nommage.

Concepts :

- **Les propriétés** : un bean est conçu pour être **paramétrable**. On appelle "propriétés" les champs non publics présents dans un bean. Qu'elles soient de type primitif ou objets, les propriétés permettent de paramétrer le bean, en y stockant des données.
- **La sérialisation** : un bean est conçu pour pouvoir être **persistant**. La sérialisation est un processus qui permet de sauvegarder l'état d'un bean, et donne ainsi la possibilité de le restaurer par la suite. Ce mécanisme permet une persistance des données, voire de l'application elle-même.
- **La réutilisation** : un bean est un composant conçu pour être **réutilisable**. Ne contenant que des données ou du code métier, un tel composant n'a en effet pas de lien direct avec la couche de présentation, et peut également être distant de la couche d'accès aux données (nous verrons cela avec le modèle de conception DAO). C'est cette indépendance qui lui donne ce caractère réutilisable.
- **L'introspection** : un bean est conçu pour être **paramétrable de manière dynamique**. L'introspection est un processus qui permet de connaître le contenu d'un composant (attributs, méthodes et événements) de manière dynamique, sans disposer de son code source. C'est ce processus, couplé à certaines règles de normalisation, qui rend possible une découverte et un paramétrage dynamique du bean !

5.1 Bean Student

- Créez un bean `Student.java` qui contient les propriétés suivantes:
 - o Une chaîne de caractères *name*
 - o Un booléen *working*
 - *Attention : vous devez respecter les conventions de nommages.*
- Exemple :*

```
String toto;  
  
public void setToto(String value){  
    toto = value;  
}  
  
public String getToto(){  
    return toto;  
}
```

5.2 Configuration du projet sous Eclipse

Afin de rendre vos objets accessibles à votre application, il faut que les classes compilées à partir de vos fichiers sources soient placées dans un dossier "classes", lui-même placé sous le répertoire `WEB-INF`.

- Par défaut Eclipse ne procède pas ainsi et envoie automatiquement vos classes compilées dans un dossier nommé "build". Afin de changer ce comportement, il va falloir modifier le *Build Path* de notre application.
- Clic droit sur le dossier du projet → Build Path → Configure Build Path...
 - o Sélectionnez alors l'onglet *source*, puis regardez en bas le champ *Default output folder*
 - o C'est ici qu'il faut préciser le chemin vers **WEB-INF/classes** afin que nos classes, lors de leur compilation, soient automatiquement déposées dans le dossier pris en compte par notre serveur d'applications.
- Nous allons ainsi pouvoir manipuler nos beans directement depuis nos servlets et nos JSP !

5.3 Use bean

- Dans la Servlet, instanciez un bean et transmettez-le à la JSP comme vu précédemment
- Dans la JSP, on peut :
 - o Utiliser un bean à l'aide de l'action `<jsp:useBean id="beanId" class="path.to.class" />`
 - o Modifier un bean à l'aide de l'action `<jsp:setProperty name="beanId" property="propertyName" value="propertyValue" />`
 - Remarque : si vous utilisez le caractère * pour le paramètre *property*, chaque champs va être bindé automatiquement. Exemple : binder les paramètres d'une requête POST automatiquement sur les champs d'un JavaBean.
 - o Afficher la valeur d'une propriété d'un bean à l'aide de l'action `<jsp:getProperty name="beanId" property="propertyName" />`. La même opération peut être faite plus simplement avec `<%= beanId.getpropertyName() %>`
- Récupérez le bean et affichez ses propriétés

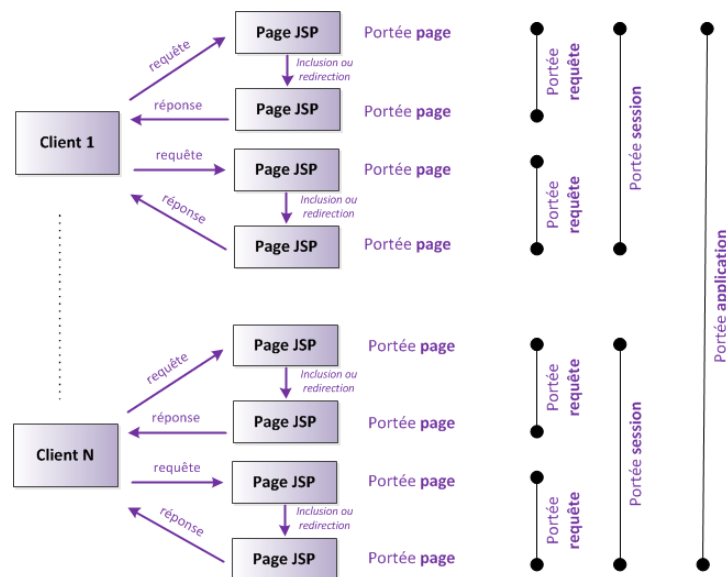
5.4 Portée

Un concept important intervient dans la gestion des objets par la technologie JSP : la **portée des objets**. Souvent appelée visibilité, ou *scope* en anglais, elle définit tout simplement leur **durée de vie**.

Il existe au total quatre portées différentes dans une application :

- **Page** (JSP seulement) : les objets dans cette portée sont uniquement accessibles dans la page JSP en question ;
- **Requête** : les objets dans cette portée sont uniquement accessibles durant l'existence de la requête en cours ;
- **Session** : les objets dans cette portée sont accessibles durant l'existence de la session en cours ;

- **Application** : les objets dans cette portée sont accessibles durant toute l'existence de l'application.



- On peut spécifier le scope d'un bean de la façon suivante :

```
<jsp:useBean id="student" class="ch.si.beans.Student" scope="request" />
```

6 EL

Les *expression language* permettent de s'affranchir des scriptlets dans les pages JSP, afin de ne pas trop mélanger du code Java et du code de présentation (HTML, XHTML, etc.).

Leur syntaxe : `${ expression }`.

Elles permettent notamment :

- Faire des tests en utilisant des opérateurs
 - o Arithmétiques (+ - * / %)
 - o Logiques (&& || !)
 - o Relationnels (== != < > <= >=)
 - o Deux autres types de test sont fréquemment utilisés au sein des expressions EL :
 - les conditions ternaires, de la forme : *test ? si oui : sinon*
 - les vérifications si vide ou null, grâce à l'opérateur *empty*

Exemples :

```

<!-- Logiques sur des booléens -->
${ true && true } <br /> <!-- Affiche true -->
${ true && false } <br /> <!-- Affiche false -->
${ !true || false } <br /> <!-- Affiche false -->

<!-- Calculs arithmétiques -->
${ 10 / 4 } <br /> <!-- Affiche 2.5 -->

```

```

${ 10 mod 4 } <br /> <!-- Affiche le reste de la division entière, soit 2 -->
${ 10 % 4 } <br /> <!-- Affiche le reste de la division entière, soit 2 -->
${ 6 * 7 } <br /> <!-- Affiche 42 -->
${ 63 - 8 } <br /> <!-- Affiche 55 -->
${ 12 / -8 } <br /> <!-- Affiche -1.5 -->
${ 7 / 0 } <br /> <!-- Affiche Infinity -->

<!-- Compare les caractères 'a' et 'b'. Le caractère 'a' étant bien situé avant le caractère
'b' dans l'alphabet ASCII, cette EL affiche true. -->
${ 'a' < 'b' } <br />

<!-- Compare les chaînes 'hip' et 'hit'. Puisque 'p' < 't', cette EL affiche false. -->
${ 'hip' gt 'hit' } <br />

<!-- Compare les caractères 'a' et 'b', puis les chaînes 'hip' et 'hit'. Puisque le premier
test renvoie true et le second false, le résultat est false. -->
${ 'a' < 'b' && 'hip' gt 'hit' } <br />

<!-- Compare le résultat d'un calcul à une valeur fixe. Ici, 6 x 7 vaut 42 et non pas 48, le
résultat est false. -->
${ 6 * 7 == 48 } <br />

<!-- Vérifications si vide ou null -->
${ empty 'test' } <!-- La chaîne testée n'est pas vide, le résultat est false -->
${ empty '' } <!-- La chaîne testée est vide, le résultat est true -->
${ !empty '' } <!-- La chaîne testée est vide, le résultat est false -->

<!-- Conditions ternaires -->
${ true ? 'vrai' : 'faux' } <!--Le booléen testé vaut true, vrai est affiché -->
${ 'a' > 'b' ? 'oui' : 'non' } <!--Le résultat de la comparaison vaut false, non est affiché -->
${ empty 'test' ? 'vide' : 'non vide' } <!--La chaîne testée n'est pas vide, non vide est
affiché -->

```

- Manipuler des objets (JavaBeans / collections / objets implicites (ex : paramètres))
 - o Pour les beans :
 - `${ beanId.propertyName }`
équivalent à
`<jsp:getProperty name="beanId" property="propertyName"/>`

6.1 Exercice

- Ré-écrivez la récupération des données avec des EL (paramètres, attributs, beans)
- Utilisez une EL pour afficher un message personnalisé si l'étudiant est en train de travailler.

6.2 Notes

- La technologie EL est fondée sur les JavaBeans et sur les collections Java.
- Les expressions EL remplacent les actions standard de manipulation des objets.
- Une expression EL permet d'effectuer des tests, interprétés à la volée lors de l'exécution de la page.

- L'interprétation des expressions EL peut être désactivée via une section dans le fichier web.xml.

7 JSTL

JSTL est un ensemble de balises qui implémentent des fonctionnalités diverses, permettant d'alléger le code du programmeur. Elle permet de mettre en place des boucles, des tests conditionnels, du formatage des données, la manipulation de données XML, etc.

Par exemple, une boucle affichant les éléments d'une liste passée en paramètres à la requête :

- En scriptlet :

```
<%@ page import="java.util.List, java.util.ArrayList" %>
<%
    List<Integer> list = (ArrayList<Integer>)request.getAttribute("tirage");
    for(int i = 0; i < list.size();i++){
        out.println(list.get(i));
    }
%>
```

- Avec la bibliothèque *core* de JSTL :

```
<c:forEach var="item" items="${tirage}" >
    <c:out value="${item}" />
</c:forEach>
```

JSTL est constitué des 5 librairies suivantes :

Area	URI	Subfunction	Prefix
Core	http://java.sun.com/jsp/jstl/core	Variable support	c
		Flow control	
		URL management	
		Miscellaneous	
XML	http://java.sun.com/jsp/jstl/xml	Core	x
		Flow control	
		Transformation	
I18N	http://java.sun.com/jsp/jstl/fmt	Locale	fmt
		Message formatting	
		Number and date formatting	
Database	http://java.sun.com/jsp/jstl/sql	SQL	sql
Functions	http://java.sun.com/jsp/jstl/functions	Collection length	fn
		String manipulation	

Pour pouvoir utiliser JSTL, il est nécessaire de la déclarer comme suit (à la manière d'intégrer une taglib), par exemple pour la bibliothèque *core* :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- Comme Tomcat n'est pas un serveur d'application, il n'implémente pas totalement la spécification Java EE. Nous devons donc ajouter manuellement la librairie JSTL (par exemple Glassfish, qui respecte bien la spécification Java EE, inclus JSTL).
 - o Ajoutez la librairie *jstl-1.2.jar* dans le répertoire *WEB-INF/lib*

7.1 Exercice

- Ajoutez une liste d'éléments au bean Student (ex : liste de langage de programmation), et affichez la liste des éléments en utilisant JSTL
- Essayez d'afficher une liste d'éléments stockés dans un fichier XML
 - o Créez un document XML et placez-le dans le répertoire WebContent
 - o Dans une page JSP, récupérez le document XML (*c:import*)
 - o Parsez le document (*x:parse*)
 - o Parcourez le document parsé (*x:forEach*)
 - o Affichez un ou plusieurs champs de l'élément récupéré (*x:out*)

7.2 Notes

- La JSTL est composée de cinq bibliothèques de balises standard ;
- Elle permet d'éviter l'utilisation de code Java dans les pages JSP ;
- Elle permet de réduire la quantité de code à écrire ;
- Elle rend le code des pages JSP plus lisible ;
- Sous Tomcat, il faut placer son fichier .jar sous /WEB-INF/lib pour qu'elle soit correctement intégrée.

8 A vous de jouer!

En utilisant les notions que vous venez d'apprendre et en respectant le modèle MVC, réalisez une petite application utilisant les technologies suivantes :

- Servlet
- JSP
- JavaBean
- EL
- JSTL

Synthèse

Faites une synthèse de ce travail, et détaillez notamment la façon dont est mis en place le modèle MVC dans une application Java EE.

Références

- [1] <http://docs.oracle.com/javaee/7/tutorial/doc/>
- [2] <http://fr.openclassrooms.com/informatique/cours/creez-votre-application-web-avec-java-ee>
- [3] <https://jstl.java.net>
- [4] <http://www.jsptut.com>
- [5] <http://adiguba.developpez.com/tutoriels/j2ee/jsp/taglib/>

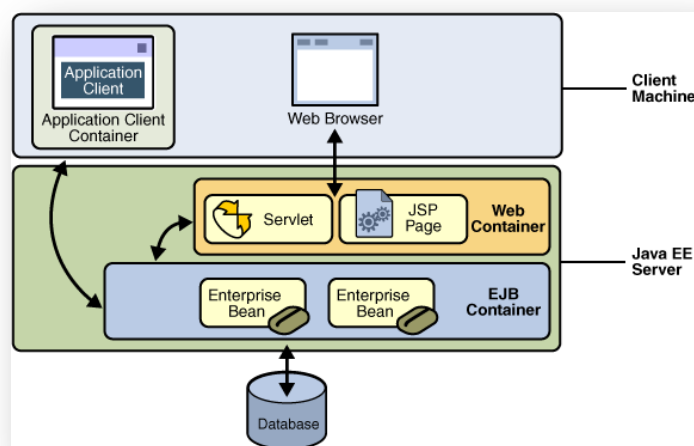
Complément théorique

Déploiement d'une *Enterprise Application aRchive* – *EAR*

Un livrable Java EE est une archive possédant une des extensions suivantes : **.ear**, **.war**, **.jar**, **.rar**. Les livrables sont déployés dans un serveur d'application Java EE (conteneur Web + conteneur EJB) ou dans un simple conteneur web (moteur de servlet/JSP comme Tomcat).

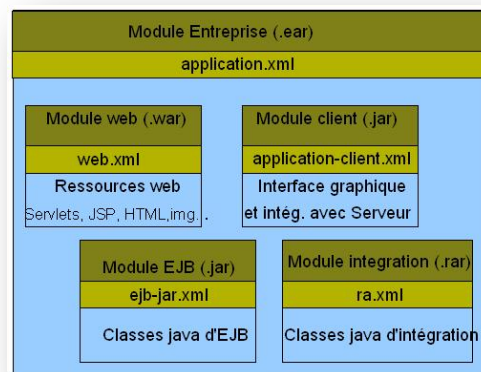
Développer une application Java EE revient à créer les différentes livrables, suivant la complexité des besoins de l'application. Le développement de chaque livrable peut être confié à une équipe ou plusieurs équipes de développement réparties par exemple dans plusieurs pays. Les développeurs livrent alors leur travail dans un référentiel commun.

Nous présentons ci-dessous le contenu de chaque livrable ainsi que leur relations et contribution dans une application Java EE.



EAR

La livraison englobant toutes les autres est celle d'extension ear (Enterprise ARchive). Utile dès que l'application est complexe, impliquant des services d'entreprise (Transactionnel, Sécurité, communication multiserveurs, Base de données, Mainframe, Reporting, fournisseur de Message...).



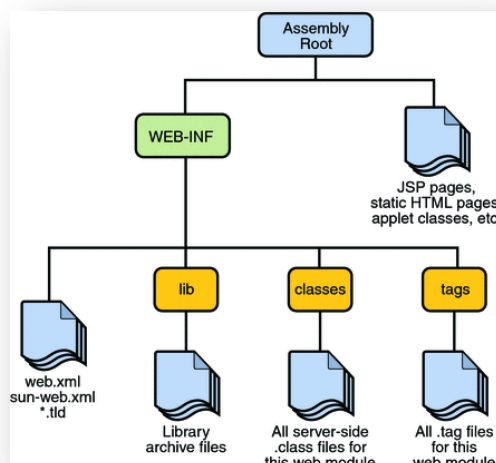
Le fichier EAR doit être déployé dans un serveur d'application labellisé Java EE (ex : Glassfish, Jboss, etc.).

Livraison WAR

La plupart des applications Java EE sont livrées dans un module WAR, contenant les écrans de l'application, les composants d'accès aux données ainsi que des composants métiers.

Vous trouverez dans ce livrable :

- Ecrans de l'application (Pages (X)HTML, JSP)
- Images de l'application
- Eléments de présentation (Feuilles de style CSS, XSL, Tld)
- Classes Java métier (JavaBean) ou d'accès aux données
- Fichier de configuration web.xml



Le fichier WAR peut être déployé dans un serveur d'application labellisé Java EE (ex: Glassfish, JBoss, etc.) ou dans un simple moteur de servlet comme Tomcat.

JAR / EJB

Les EJB (Enterprise JavaBean) sont des composants java métiers évoluant dans le conteneur EJB d'un serveur d'applications Java EE. Ils bénéficient de plusieurs services offerts par le conteneur web, dont la sécurité et la transaction déclarative.

Un EJB est livré sous forme de .jar indépendant ou à l'intérieur d'une livraison globale .ear.

JAR / Clients

La livraison JAR (Java ARchive), correspond typiquement à un client JAVA qui interrogerait un EJB. Un Jar client est livré sous forme de .jar indépendant ou à l'intérieur d'une livraison globale .ear

RAR

Un fichier RAR Java EE (Ressource Archive) est une livraison déployée dans un serveur labellisé Java EE et nécessaire pour permettre aux applications Java EE de communiquer avec des systèmes tiers (SAP, Mainframe, Reporting, fournisseur de Messages, etc.).