**Cours du 11 janvier**

**J2EE – Java 2 Platform Enterprise Edition .**

Professeur : B. Chauvet

Java : 1995  
J2E : 1999

JSR : Java Specification Request

Apache est un serveur web, qui transmet à Tomcat (conteneur web) pour taches dynamiques.

Serveurs d’application :  
weblogic, JBoss Redhat, Websphere, Glassfish – Apache

**Servlet** [**clic**](https://fr.wikipedia.org/wiki/Servlet) **:** quand on reçoit une requête, elle est transmise à Tomcat qui la transmet au serveur web, qui sait à quel servlet l’envoyer.  
Le servlet traite la requête ; c’est la logique de controler.

**JSF** [**clic**](https://fr.wikipedia.org/wiki/JavaServer_Faces)**:** Java Server Faces (RichFaces et PrimeFaces)

**JSTL :** Java Standard Tag Library

**JDBC :** Java DataBase Connectivity

**JPA :** Java Persistence API

**Web services : JAX – ws - 🡪 SOAP  
 JAX – RS 🡪 Rest**

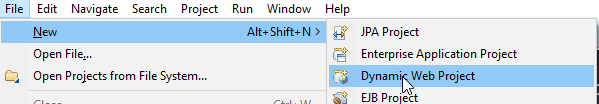
**RMI** [**clic JMDoudoux**](https://www.jmdoudoux.fr/java/dej/chap-rmi.htm) **:** Remote Method Invocation

**JNDI :** Java Naming & Directory Interface

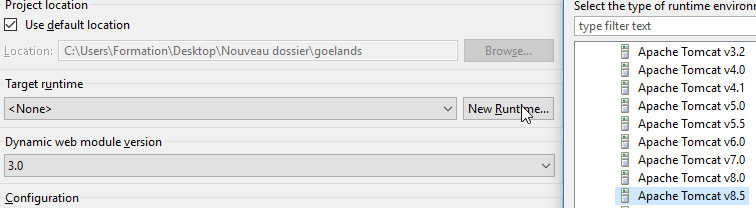
**JMS :** Java Messaging Service

JINI : JINI Is Not Initials  
LINUX : LINUX Is Not Unix

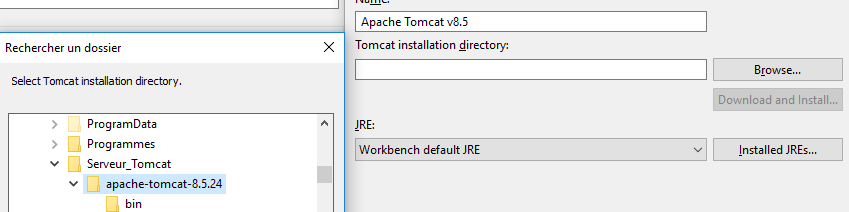
Création projet :



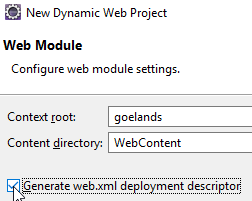
New runtime pour le serveur TomCat



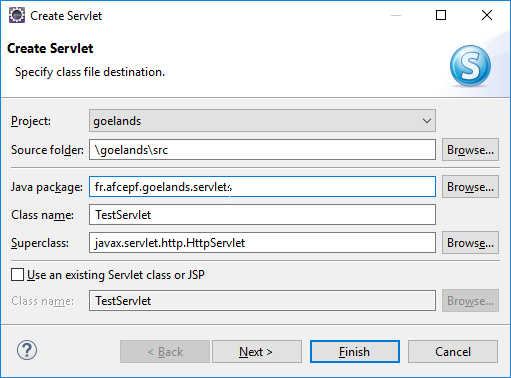
Ensuite on choisit le dossier dans Browse ou l’on a mis TomCat sur notre machine :



Ensuite, Next (pas Finish !), deux fois.  
On coche :

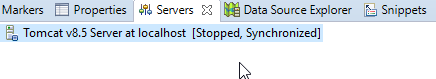


On a désormais notre projet, on va créer notre premier servlet.  
Clic droit sur le projet, add new servlet :



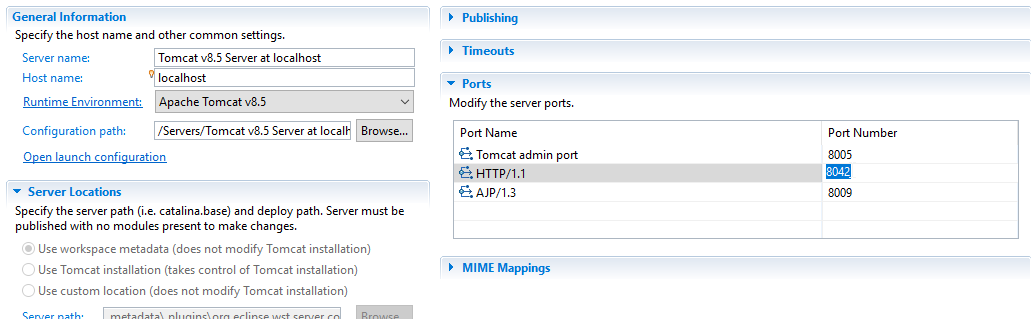
On peut ensuite run as server.

Dans la barre en bas, on a un onglet Server

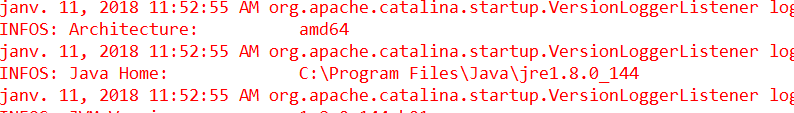


Double clic dessus :

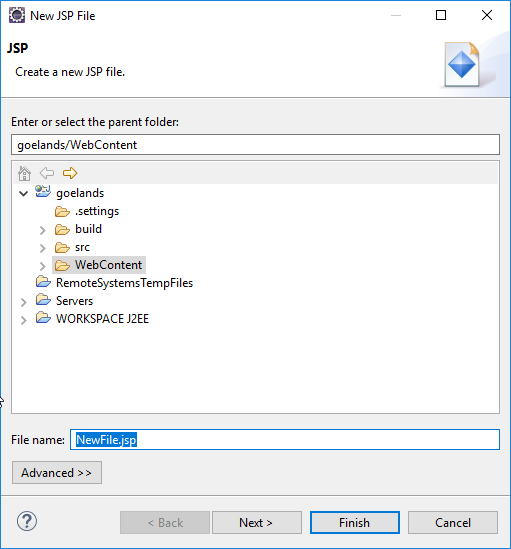
On change le port, s’il ne fonctionnait pas



On peut à nouveau run, on a en console les données du server au lancement !



On se fait un JSP :



Modèe vue-contrôleur.  
Afficher les données, c’est le boulot de la vue.

Le contrôleur récupère le modèle, définit quelle est la vue à utiliser, puis détermine la vue et y passe le modèle.

Ensuite, la vue a terminé, le contrôleur peut renvoyer le résultat.

Le servlet est le contrôleur : il reçoit une request, la servlet s’adresse alors au business, qui récupère l’article, jsp

Le modèle c’est les données que le jsp récupère.

[Clic cours internet](http://www.lipn.fr/~rozenknop/Cours/IIG/Cours6/cours/modele_vue_contr%C3%B4leur.html)

**Cours du 14 mars**

**J2EE – Java 2 Platform Enterprise Edition .**

Professeur : S. Péricard

Servlet : classe 🡪 HttpServlet 🡪 container de Servlet (Serv d’application : Tomcat, JBoss (Standard JEE), …)

Quant au HttpServlet : doGet, doPost, init

Init : équivalent de blocs statics en JSE, appel au démarrage de l’application si load-on-startup

Au déploiement de l’application sur le serveur (archive.war), décompression de l’application, lecture du web.xml

Le web.xml porte une information, la seule que l’on ne peut avoir avec Maven :   
c’est un fichier xml, donc dans la déclaration de son en-tête, son schéma qui permet d’avoir la **version des servlet**!  
On peut aussi y définir les pages d’accueil, les timeout de session, les servlet, des filtres, des listener, etc…

**Si dans le xml, on est dans une version de servlet avant 3.x pas d’annotations !  
En version 2.5 par exemple, les @nnotations éventuelles ne seront pas lues.**

Thread.currentThread().getContextClassLoader().getResource(« . »)

La servlet permet la création de la réponse  
Elle fait office de **controller** (+vue).

Les servlet sont des pages html avec quelques balises :  
<%@Page ... %>

<%Code Java%>  
  
Taglib : gestion des espaces de nom  
Un taglib a toujours un préfix et un URI  
<%@taglib prefix=’’ ’’ uri=’’ ’’ %>

Objets servlet :  
request  
out : PrintWriter (qui vient de response.getWriter) 🡪 permet d’écrire dans la réponse

Out.write(« <html> ») ; 🡪 va être interprêté

L’interpréteur transforme les ${EL} en sa valeur, les <% codeJava %> en code, et <balises :specifique …/> en classe adéquate.

On a quatre conteneurs d’attributs pour les expression language :  
1- scope request  
2- session  
3- page  
4- application

Serveur.xml 🡪 config serveur 🡪 SHUTDOWN : 8005  
 🡪 Connecteurs :

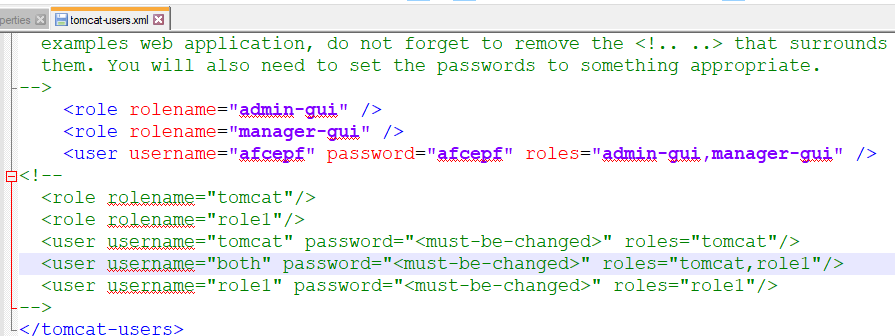
Ports : http : 8080  
 ajp : 8009  
 https : 8443

Déploiement : webapps

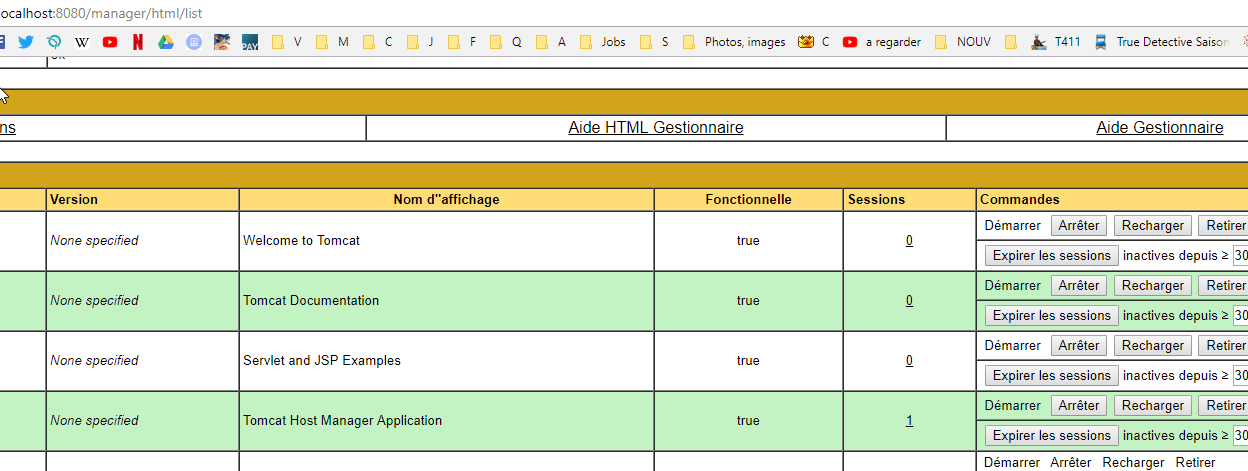
Une instance de Tomcat ne gère pas au dessus d’environ 20 utilisateurs, cluster ensuite (sorte d’autre instance ?)

On installe Tomcat 8.5.29

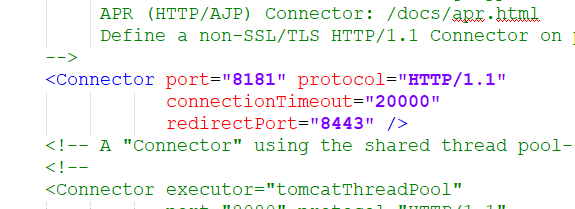
Dans le dossier config on change le tomcat-users.xml, on ajoute l’user :



Go sur le local host  

Nb : on tue une commande Tomcat (lancée avec un .bat) avec CTRL + C  


On change aussi dans le server.xml certaines données sur les ports :  


On est donc en localhost :8181

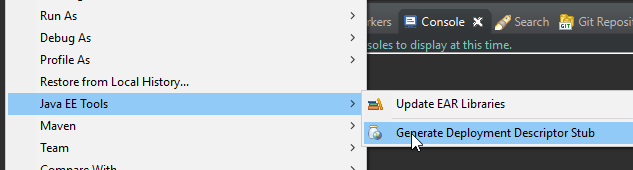
On va créer un premierWeb

On a donc besoin de pom.xml contenant :  
source  
target 1.8  
encoding UTF8  
FailOnMissingWebXml :false

PremierServlet  
C’est une classe qui hérite de HttpServlet qui est une classe abstraite (méthodes doGet, doPost, init)

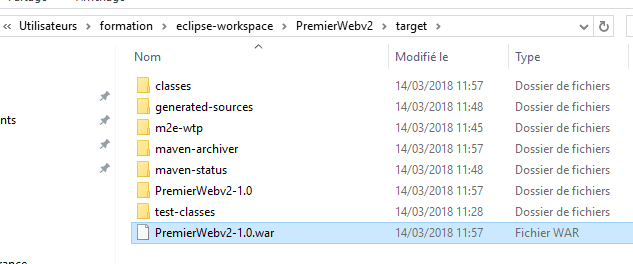
<servlet>  
 <servlet-name>CommeOnVeut</servlet-name>  
 <servlet-class>fr. … PremierServlet</servlet-class>  
</servlet>

<servlet-mapping>  
 <servlet-name>CommeOnVeut</servlet-name>  
 <url-pattern >/ ou \*</ url-pattern >  
</servlet-mapping>

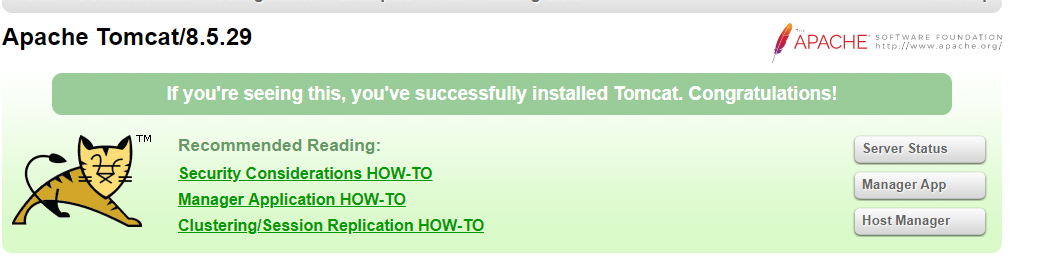
Générer le web.xml, clic droit sur le projet :  


Important de changer ceci, car hébergement a changé :

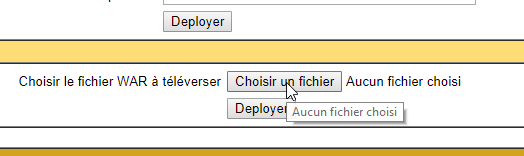


Le maven install permet de créer une archive en dossier target :  
  


On va sur le localhost :port (ici port = 8181)



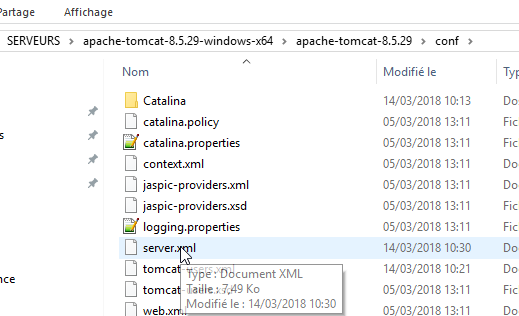
Manager App puis :

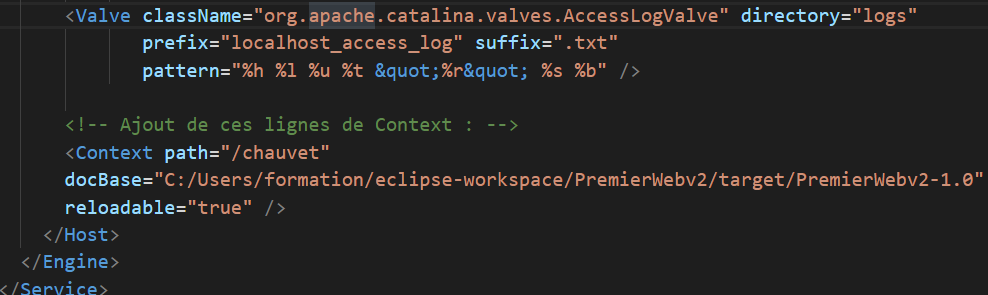


On chope le chemin de l’app déployée en target (.war)

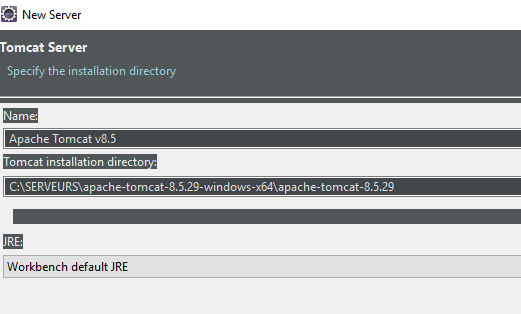
On déploit !

Autre façon de faire : retour au server.xml et ajout de lignes :





Ajout du serveur dans Eclipse :



doGet et doPost contiennent un objet Request et un objet Response

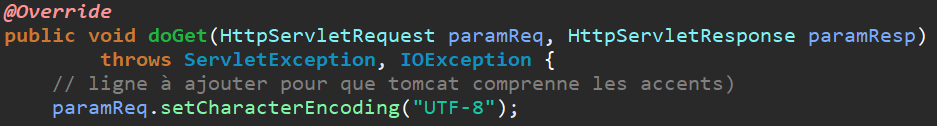
request.getParameter(«nom ») : String

Les requêtes peuvent contenir des params dans l’entête ou dans le corps.  
Ensemble de valeurs en tableau de String []

Parameters par map (clef valeur)

Request.getParametersValues(« nom ») : String []

Définir la réponse :  
1/ On peut utiliser response.sendRedirect(« /resource ») ;  
2/ request.getRequestDispatcher(« /resource …»).forward(req, resq) ; 🡪 deléguer à un autre composant  
3/ response.getWriter().print(« .. ») ; 🡪 écrire nous-même la réponse



Entre l’arrivée dans le container et l’arrivée à la Servlet adéquate, la request peut passer par des filtres !  
Permet de gérer la sécurité, entre autre.

On peut ainsi définir un comportement avant & après avoir appelé une servlet.

On va créer un filtre :  
saisie : (champs de text) + bouton clic 🡪 /\* un filtre

15 mars :

JSP de deux formats :  
XHTML (qui désormais intègre le HTML5)  
.jsp

Le **% est un caractère interdit en XHTML**, donc on ne peut pas y inclure les balises <% %>

Pages .jsp :  
<%@Page ... %>  
<%@taglib prefix= ’’ ’’ uri=’’ ’’ %>

Après lancement sur le serveur de la jsp, la classe compilée apparait dans un dossier (à retrouver : tmp0 > work > apache un truc comme ça)

JSTL : Java Standard TagLib

Composé de cinq grands éléments :  
- le core (formes conditionnelles, foreach etc)  
- les functions  
- format : formatDate, decimal, …  
- pour le XML  
- historique mais encore inclus, sql (qui permet de faire du jstl dans la jdbc)

