**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**

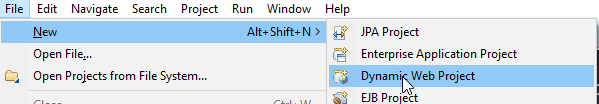
**RMI** [**clic JMDoudoux**](https://www.jmdoudoux.fr/java/dej/chap-rmi.htm) **:** Remote Method Invocation

**JNDI :** Java Naming & Directory Interface

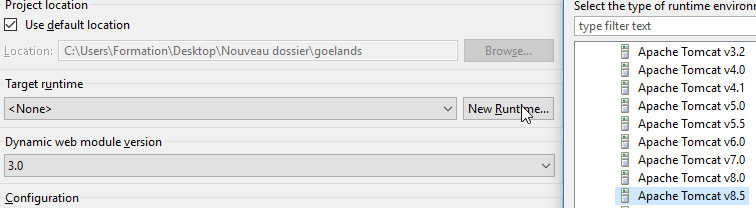
**JMS :** Java Messaging Service

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

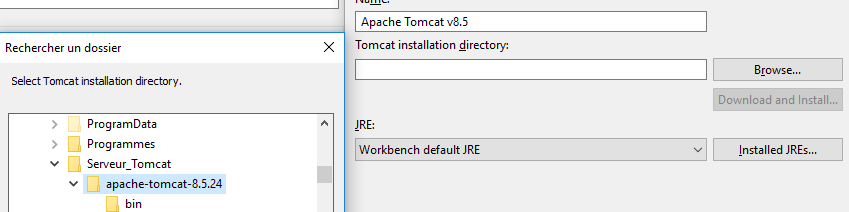
Création projet :



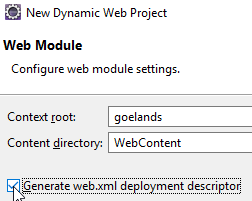
New runtime pour le serveur TomCat



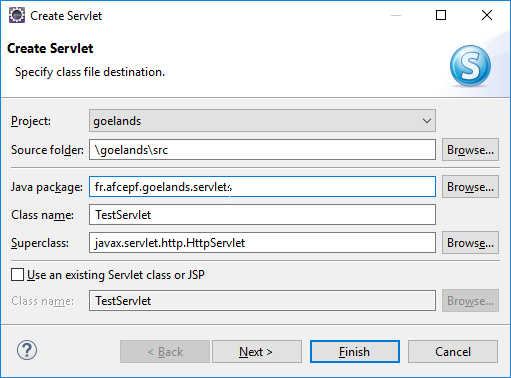
Ensuite on choisit le dossier dans Browse ou l’on a mis TomCat sur notre machine :



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

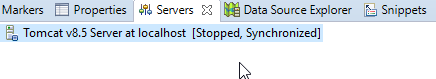


On a désormais notre projet, on va créer notre premier servlet.  
Clic droit sur le projet, add new servlet :



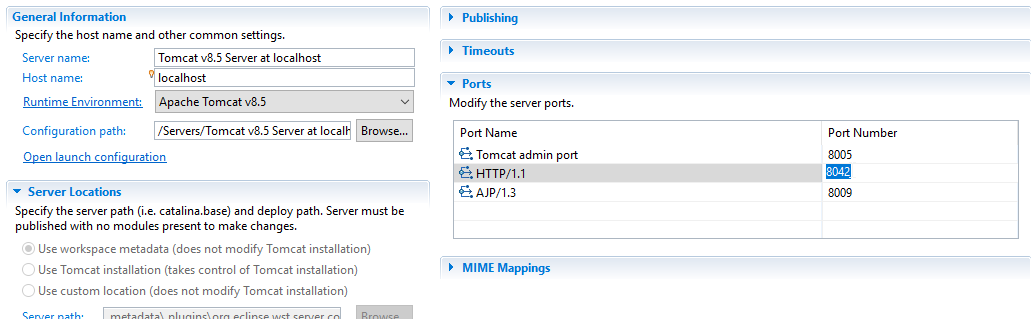
On peut ensuite run as server.

Dans la barre en bas, on a un onglet Server

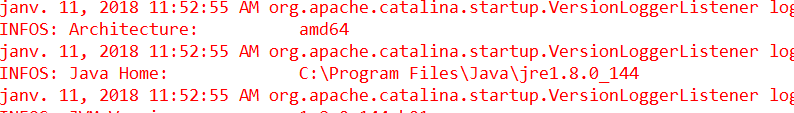


Double clic dessus :

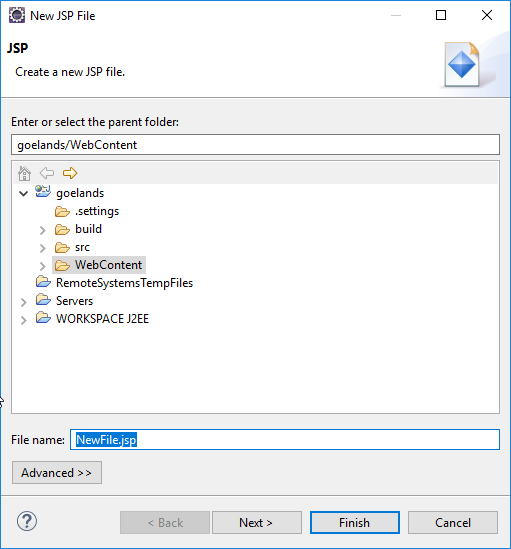
On change le port, s’il ne fonctionnait pas



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 :

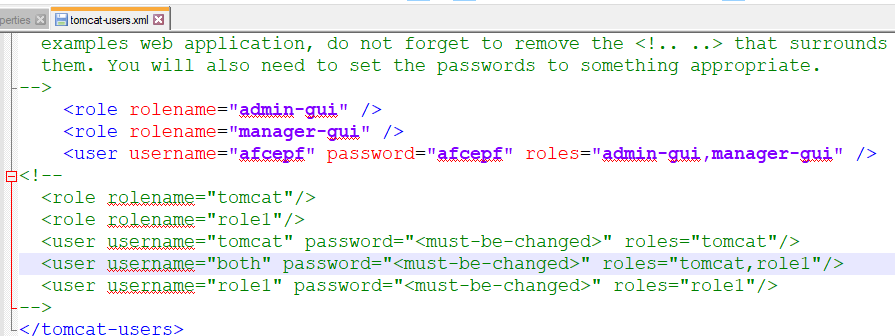
Ports : http : 8080  
 ajp : 8009  
 https : 8443

Déploiement : webapps

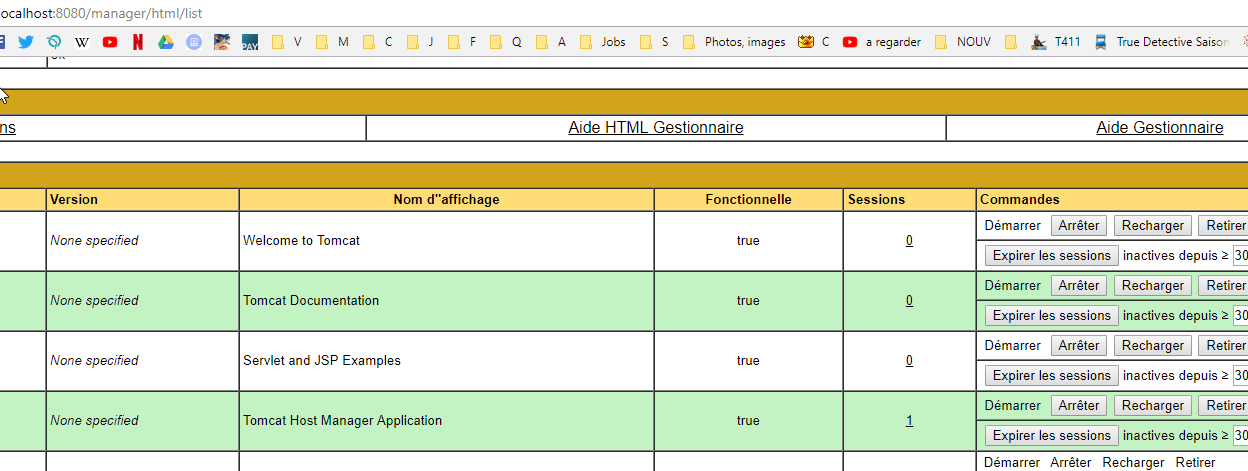
Une instance de Tomcat ne gère pas au dessus d’environ 20 utilisateurs, cluster ensuite (sorte d’autre instance ?)

On installe Tomcat 8.5.29

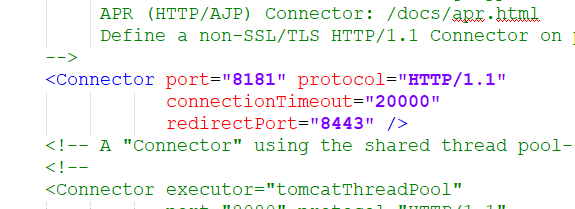
Dans le dossier config on change le tomcat-users.xml, on ajoute l’user :



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

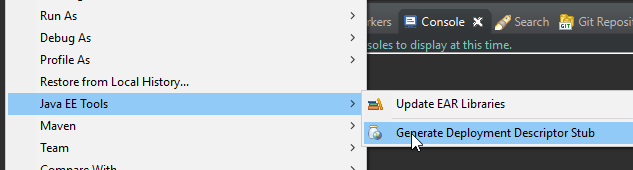
On va créer un premierWeb

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)

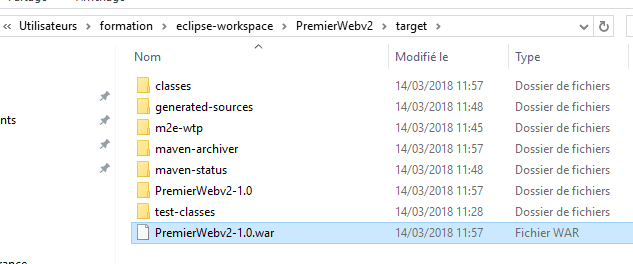
<servlet>  
 <servlet-name>CommeOnVeut</servlet-name>  
 <servlet-class>fr. … PremierServlet</servlet-class>  
</servlet>

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

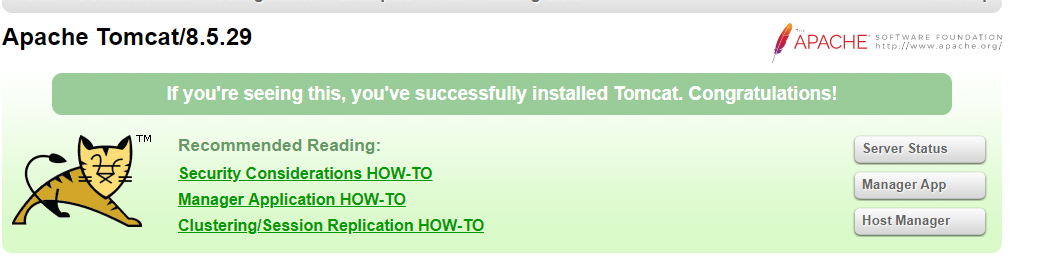
Générer le web.xml, clic droit sur le projet :  


Important de changer ceci, car hébergement a changé :

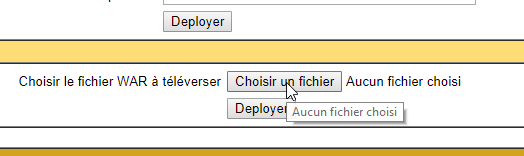


Le maven install permet de créer une archive en dossier target :  
  


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



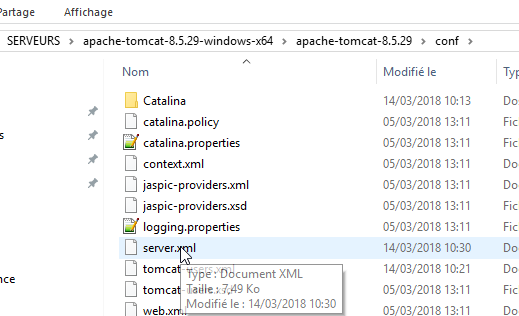
Manager App puis :

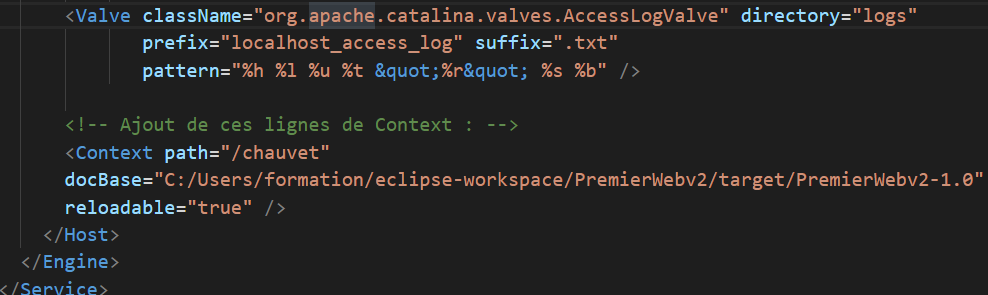


On chope le chemin de l’app déployée en target (.war)

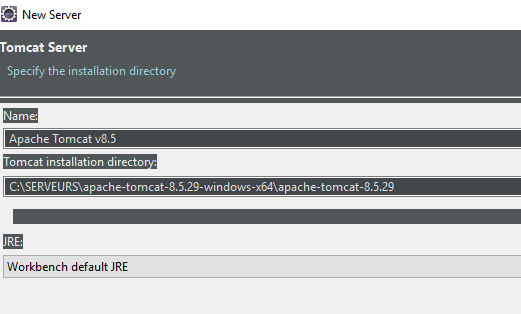
On déploit !

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

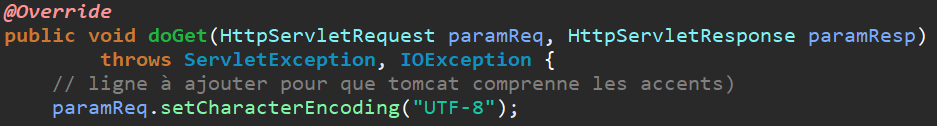
request.getParameter(«nom ») : String

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

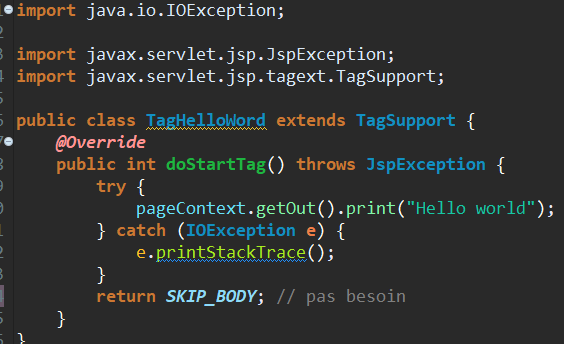
Le **% est un caractère interdit en XHTML**, donc on ne peut pas y inclure les balises <% %>

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)



**. MVC2 .**

Un unique controller, par lequel toutes les actions passent.

Sous controllers qui s’occupent du traitement demandé.

Dans une application, on pourrait faire du MVC2 (controller unique) avec une servlet controller, qui va répondre à url-pattern : \*.action

Elle sera ainsi en charge de vérifier ce qui a été envoyé, afin de déclencher la bonne action et de pouvoir ensuite forward.  
C’est aussi la solution d’un filtre (qui intercepte tous avec url-pattern : /\* )  
Exemple : si .html, .jsp etc 🡪 laisse passer sans rien faire.  
Sinon, si c’est une action il vérifie avant d’exec.

Pour JSF servlet : modèle MVC2 événementiel  
urlt-pattern : \*.jsf  
 /faces/\*

Struts : servletController  
🡪 url-pattern :\*.do 🡪 action, actionForm, Vue

Struts différencie événements et attributs.

Comment Struts va-t-il pouvoir en chargeant une simple vue, se déclencher pour faire du traitement (en MVC2) ?  
Le but étant de toujours appeler le controller, même en appelant simplement une vue.

🡪 Struts2 utilise un filtre (qui filtre tout : /\*) : un peu comme dans JSF, une couche d’abstraction permettant d’éliminer la lead des servlets (pas de javax.Servlet)  
Une classe Action qui gère les attributs de la vue, et qui gère les événements via des méthodes (tout est regroupé dans une classe, contrairement à Struts 1).  
Système de root controller (qui vient de Symphony), permettant de faire des sous controllers par package, par bloc de type d’actions.

Du coup, qu’est-ce qui permet de faire du traitement dans la vue ?  
Deux choses : les taglib et .

Struts a ses taglib, qui déclenchent le controller de Struts.  
En fait on a des sous-root, Struts controllait l’ensemble des actions de la même façon ; ce n’est plus le cas avec Struts 2 qui permet de différencier l’action selon la catégorie.

Spring : autre MVC 🡪 Struts2 + @nnotations

C’est simplement l’ajout d’@nnotations.

Nous allons faire du Struts 1 (v1.3) et du Struts 2 (v2.5).

Une action 🡪 Action : execute (form, mapping, req, resp)

UnForm 🡪 ActionForm : validate (): actionErrors, reset()

Avec le Form, mapping avec une vue (.jsp), taglib:html, logic (if, …)

Si Struts reçoit un Get (/page.jsp, form action= « toto.do »)  
Struts quand il va lire la page, va voir le formulaire. De là, il peut voir l’action et vérifie l’action de la vue (définie par le formulaire). Déclanheemnt du traitement.  
Une action est liée à un form ; Struts voit l’action, et le form à utiliser : soit il l’instancie, soit il le récupère de la session.  
S’il l’instancie, il utilise la méthode reset() pour remplir la vue.  
Il retourne ensuite la réponse.  
Il peut y avoir des erreurs si l’action n’existe pas, si le form n’existe pas, et si dans la méthode reset() erreurs dans les traitements.

Ensuite, l’utilisateur fait un submit (post, sur action .do) : on considère que Struts est paramétré avec url-pattern : \*.do  
Post /action.do  
1. Struts voit l’action  
2. Il instancie le form ou le récupère.  
3. Il peuple les données du form avec les paramètres de la request  
4. Ensuite, il va passer par sa méthode validate() du form.  
Si erreurs, cela revient en réponse (avec erreurs).  
Dans le cas contraire, il va pouvoir :  
5. instancier l’action  
6. Execute() de l’action pour ensuite  
7. Forward le traitement à l’endroit demandé.

La Servlet Struts va donc permettre la cohérence des informations (existence etc), l’instanciation du form, et permet la validation des propriété.  
Il vérifie qu’on a bien dans la vue des propriétés existantes.

On appelle une vue ou une action 🡪 on passe par le servlet controller.

Pour tout cela, Struts possède un fichier de configurations nommé **struts-config.xml** ; dans Struts 2, c’est **struts.xml**  
Ce fichier définit des éléments de type form-beans afin de déclarer les beans de l’application.

Un form-bean va déclarer un nom et un type  
<form-bean name=’’UnForm’’  
 type=’’fr.afcepf. … .UnForm’’>

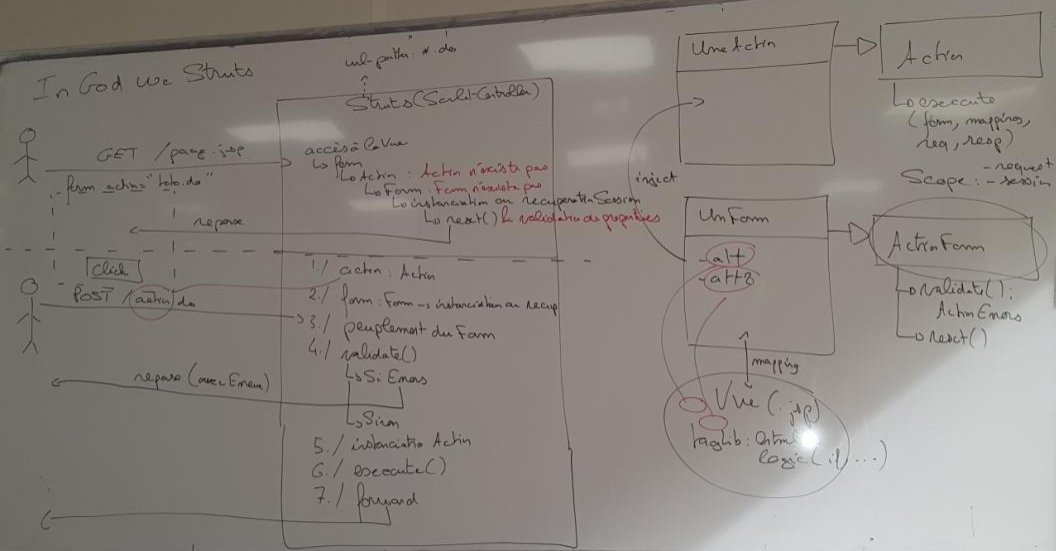
Pour récupérer la valeur du label après traitement (TP) on utilise une EL :  
${unForm.label}

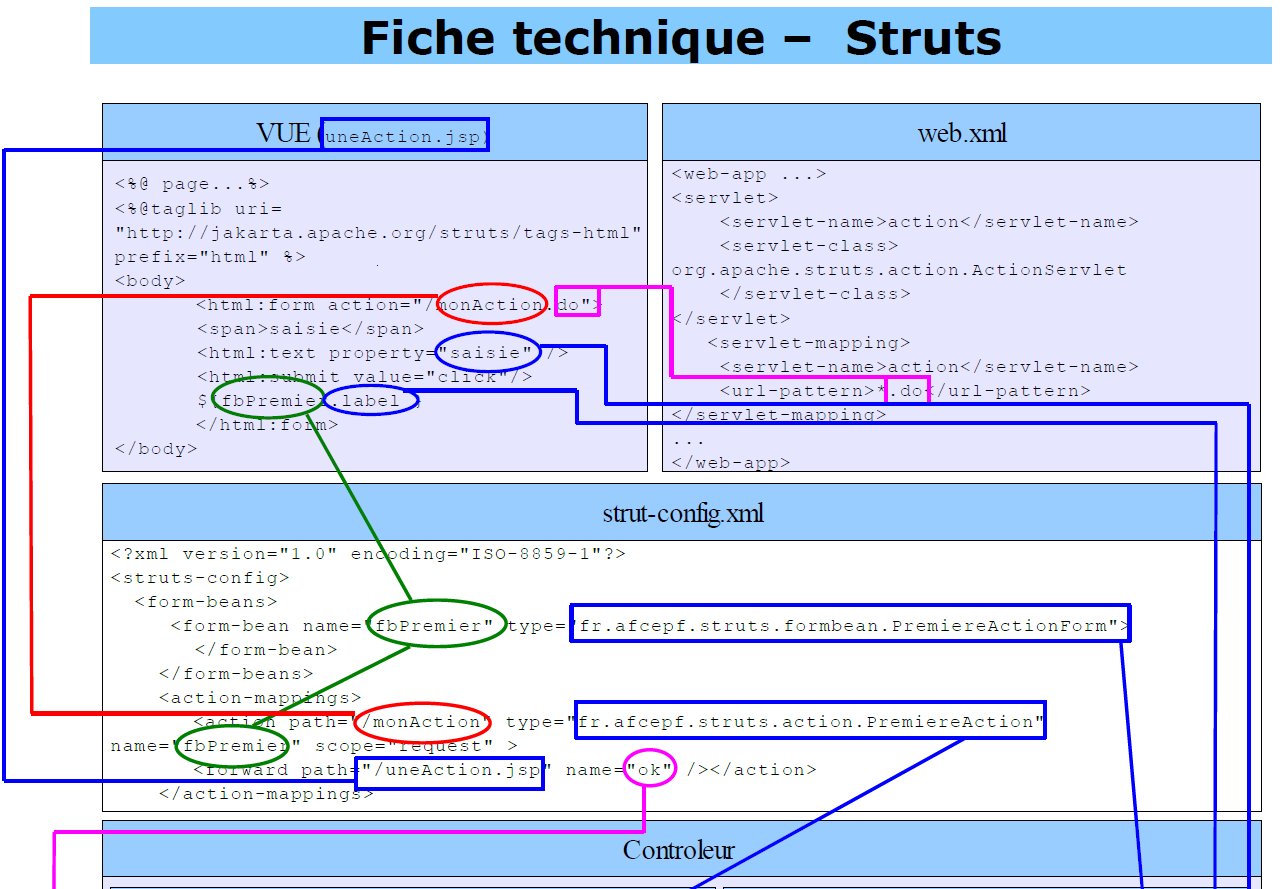
On va devoir déclarer des action-mapping qui permettent de déclarer nos actions : on va pouvoir déclarer autant d’actions que l’on souhaite.  
L’action doit avoir un path, qui définit à quoi elle doit répondre.  
Toto.do

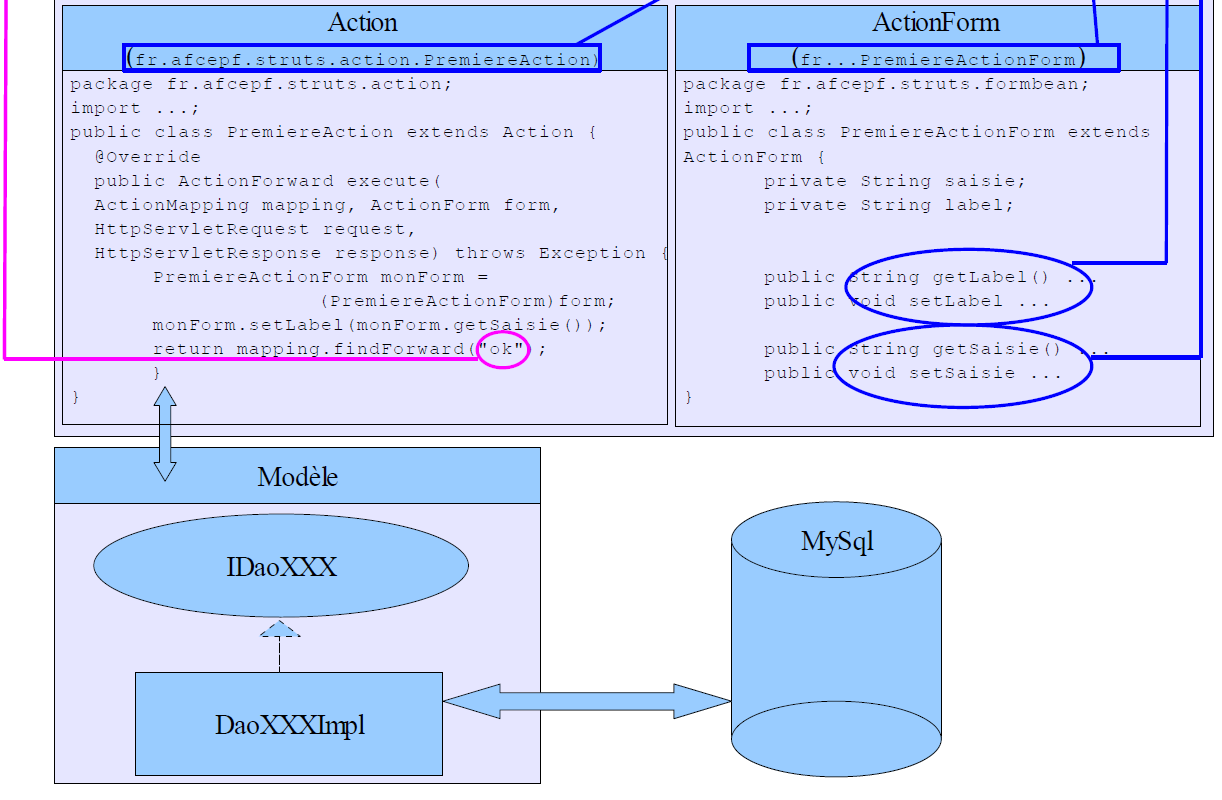
<action-mapping>  
 \*<action path) »/toto » type = « fr.afcepf. ….uneAction”  
 name=”unForm” scope=”request ou session”  
 input=”/premiere.jsp”>  
 <forward …>  
</action-mapping>

Le .do est définit par l’url-pattern de Struts, ce dernier vérifie l’action qui est appelée : il la lit, puis vérifie qu’elle existe bien.  
Une fois la vérification faite, le type de l’action étant type=’’fr.afcepf. … .uneAction’’, va chercher l’action.

findForward : ou rediriger le traitement.  
SI on met findForward(« ok ») Struts va chercher dans sa config ou rediriger le traitement « ok ».





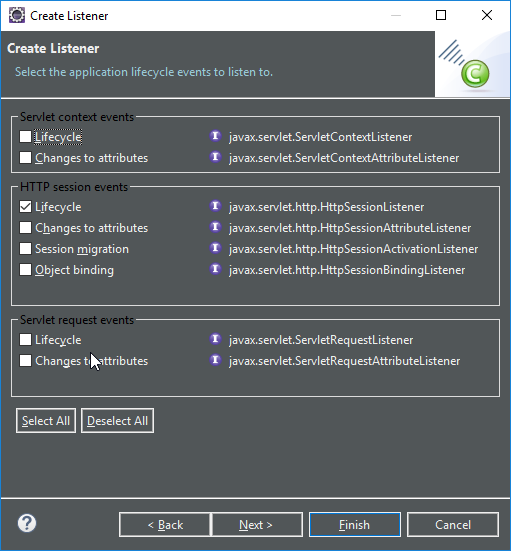


Dans le TP, on crée les classes pour le Form, pour l’Action,  
On a préalablement édité le pom.xml évidemment (packagé en war pour pouvoir avoir JBoss Tools et envoyer le déployment),  
On s’occupe alors de la création du struts-configs.xml  
Ajout eds form-bean, et de l’action-mapping.

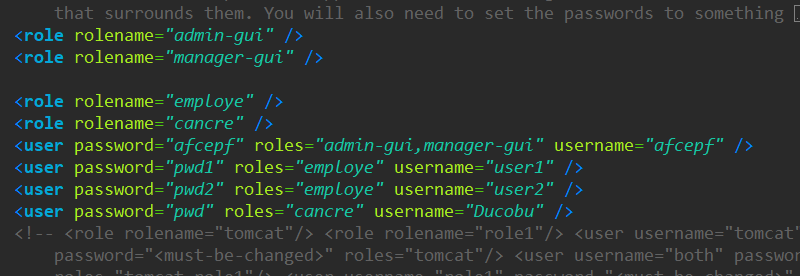
Balise <html :errors />  
Toutes les erreurs qui se sont passées dans la méthode validate().

La classe ActionSupport hérite de Action.  
Avec Struts 2 :  
on définit des configurations pour un package donné.

Création de listeners



Au niveau sécurité, on change dans les configs de Server (créé par Tomcat dans les projets lors de son premier lancement) :



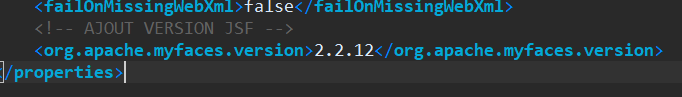
En fait presque tout se fait dans web.xml pour la sécurité.  
On y indique les rôles, les pages de redirection, la description des rôles et comment on demande les mots de passe.

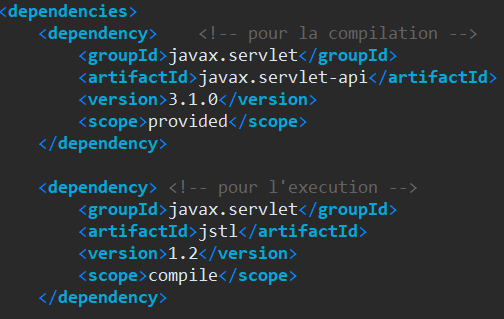
Tomcat-users.xml permet de donner les rolename, et décrire les users autorisés.

**. JSF2 .**

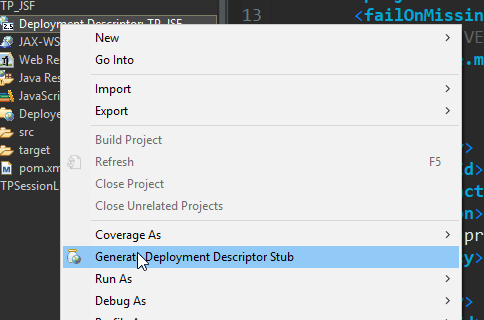
Rajout dans le pom X :

Dans les propriétés : version JSF

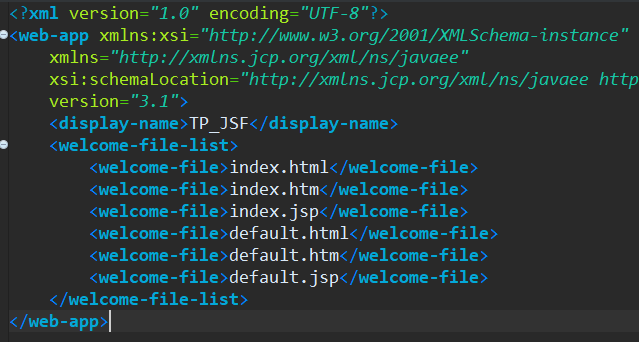


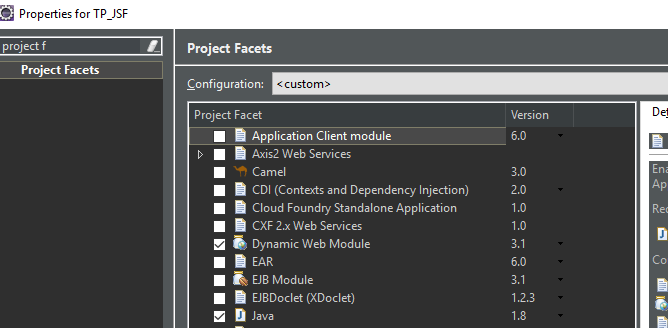
Dans les dépendances :  


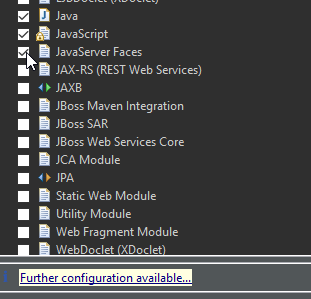
Ceci est visible page 11 du poly de D. Defrance (Jsf2-juin2014, 74 pages)

On peut faire générer :  


Petti bug Eclipse : il faut supprimer le fichier web.xml généré, puis refaire la manip.  
Le web.xml généré est alors de la bonne version.. :



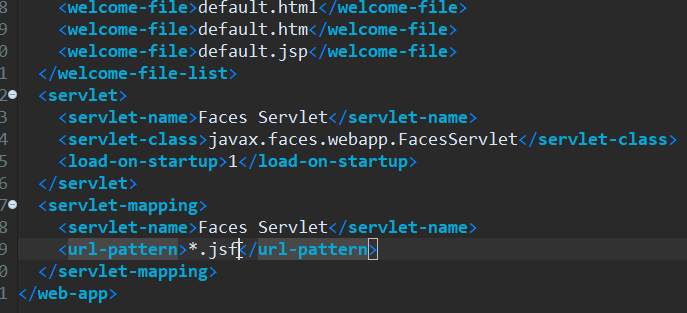
On gère ensuite les Project Facet (en propriétés du projet) :  


Il faut s’il est déjà coché, décocher JavaServerFaces, puis recocher (après avoir close and apply) :  


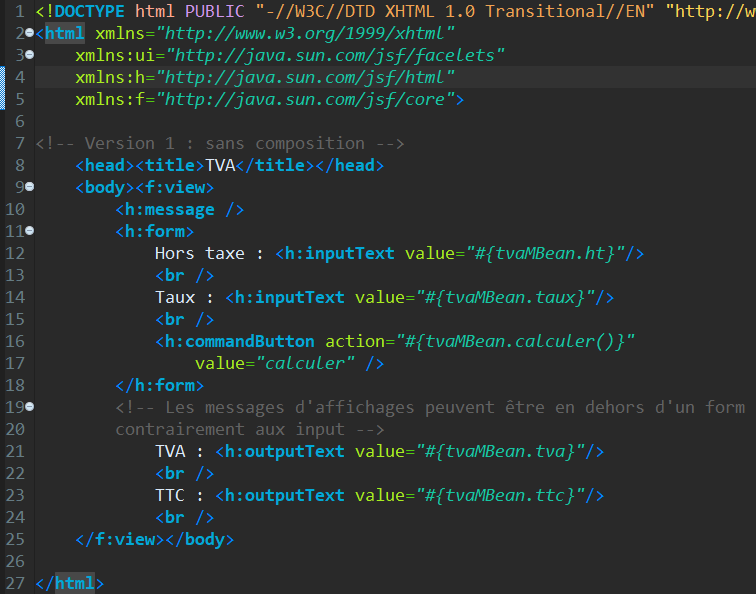
On accède donc aux configs de plus. (further configs)

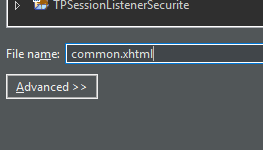
On change la valeur de l’url-pattern avec \*.jsf

Le web.xml est alors édité automatiquement:

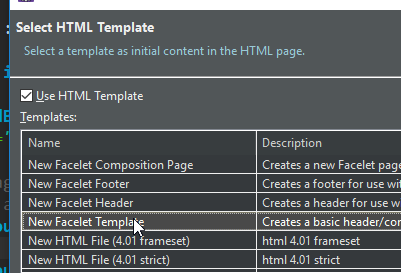


Création de la xhtml (facet blank comme template)

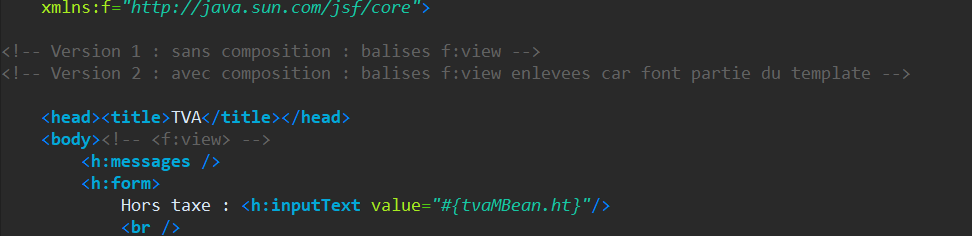


On peut créer dans le webapp un new folder pour les templates :  


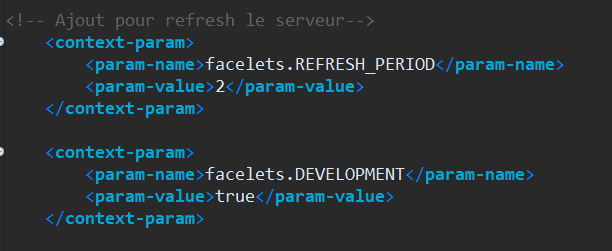
On crée dedans un xhtml avec new Facet Template :



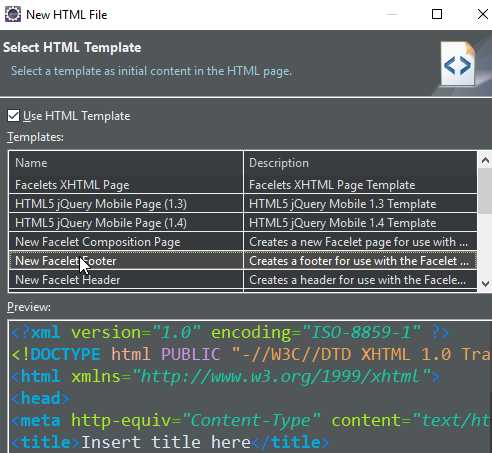
Dans la version 2, on enlève ajoute des templates donc il faut enlever le f :view (par rapport à la V1), puisque contenue dans les templates

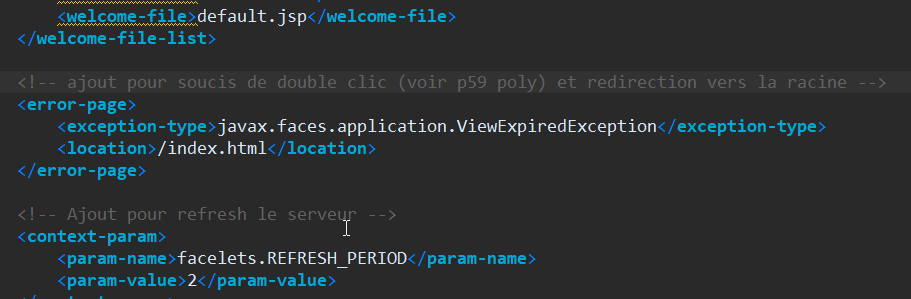


Ajout dans le web.xml pour autarefresh les changements.

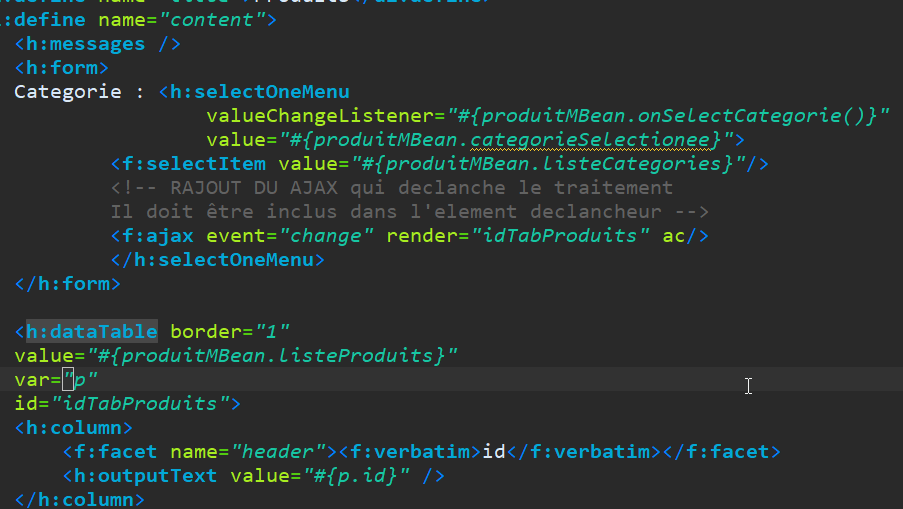


On ajoute une HTML, mais on la nomme .xhtml :  
On choisit le template désiré (ici footer) :



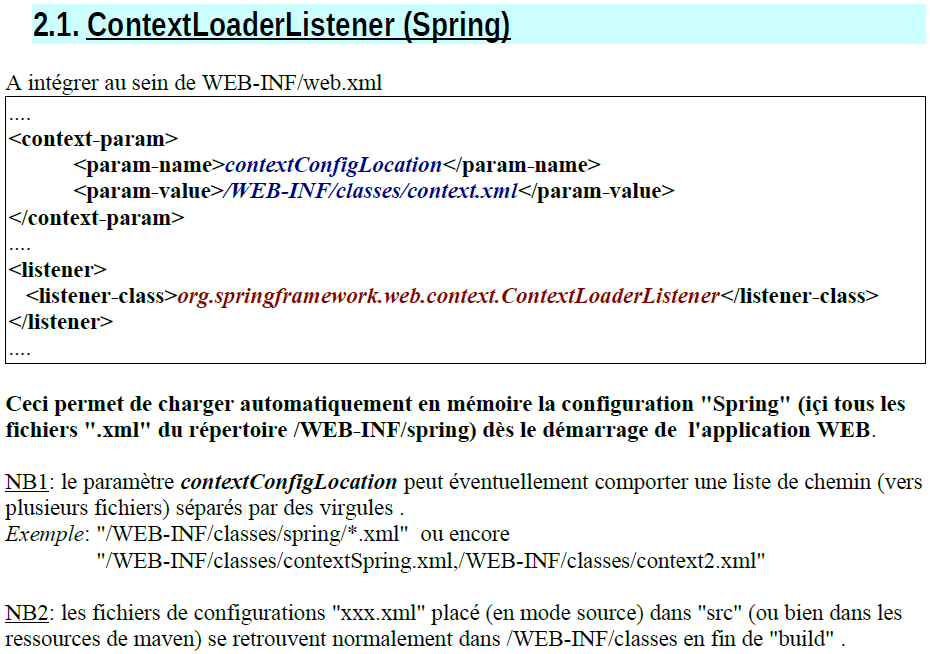
On peut ajouter pour éviter l’erreur décrite page 59 du poly de D. Defrance :  


En cas de double submit, aucun ne passe donc (utile par exemple pour un virement bancaire ou un achat de billet d’avion).

TP Produit : ajout du listener et appel avec <f :ajax> :  


**.** **SPRING .**

Installation poly page 65 (JSF2 2014 mes couilles)

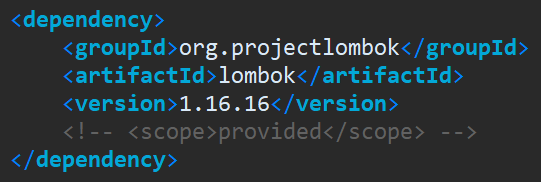


Ceci aussi :

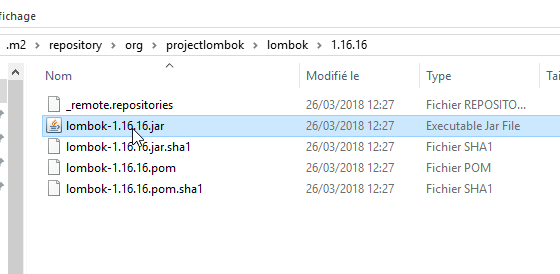


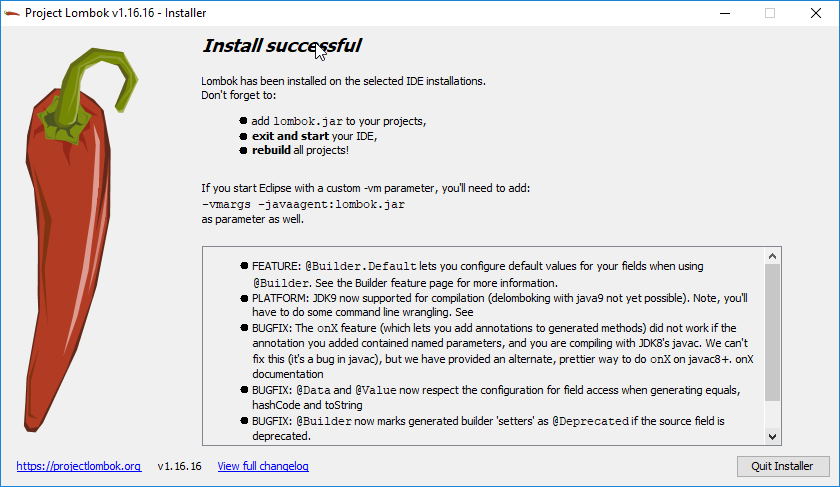
En gros, ça permet d’aller rechercher automatiquement dans la bibliothèque de Spring un managedBean

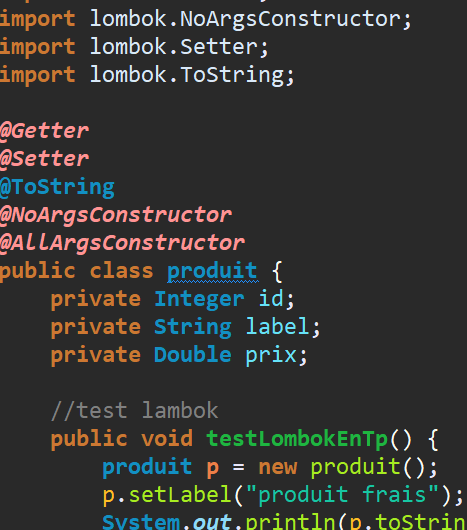
Dépendance Lambok pour les get set etc, utile, simplifie le code.



Lancer le .jar au cas ou Eclipse n’assimile pas direct le bail





Utilisation pour les get set du coup :  


**@JSonIgnore** a le même effet que **@XmlTransient**: ne prend pas en compte l’élément annoté, lors du scan / production du JSon (ou Xml)