*FLORENTIN Cynthia*

*Stage du 1 juillet au 31 juillet 2009*

# Création d'un serveur web à la compilation et exécution de TOM


Le but était de mettre en place un service web qui avait la faculté de permettre à quiconque d'essayer d'écrire du Tom et de voir le résultat de l'exécution.

## 1 Choix de la technologie

Il a été décidé d'utiliser pour la construction du site des java server page (JSP) et des servlets. Pour cela, il a fallu disposer d'un serveur TOMCAT. Nous avons choisi d'utiliser la dernière version du serveur à savoir 6.0, le JRE 6.0, le web module dynamic 2.5 et Javascript Toolkit 1.0

L'utilisation de PHP avait été pensé au départ, mais pour respecter la contrainte : « l'exécution du code dois se faire côté client », cela fut vite abandonné.
Pour se conformer à la demande, nous avons créer les fichiers de compilation sur le serveur par l'intermédiaire d'un script bash et avons ensuite, créer un jar qui doit s'exécuter côté client par l'intermédiaire d'une applet.

Pour plus de faciliter, nous avons décider de développer sous l'IDE Eclipse. La plus grande difficulté fût de configurer l'IDE pour qu'il y est les éléments nécessaire au développement web : jsp, XHTML, XML, servlet et bien évidemment, la configuration du serveur Tomcat pour qu'il fonctionne en parallèle de l'IDE.

Il a été décidé de suivre la norme W3C lors de l'écriture en XHTML. Par contre, pour l'écriture dans les jsp, nous avions essayé de suivre la norme de JSP2.0, mais cela a posé plus de soucis. Nous avons donc décidé de finaliser le projet sans la syntaxe xml de JSP2.0, mais cela pourra être modifier dans des versions futures du site web.

## 2 Développement de l'application web dynamique

Le choix des technologie défini précédemment, nous obligé à créer plusieurs type de fichiers distinct et séparé dans différent dossiers que nous allons vous présenter. Nous avons décidé d'appeler ce projet web dynamique : TOMCO

## 2.1 L'arborescence

L'architectue des JSP veut que pour le modèle MVC (Modéle-Vue-Controleur) nous ayons des beans pour les Modèles, des Java Server page pour les Vues et des servlets pour les Contrôleurs. De plus, comme il y avait aussi des fichiers utilisables par le client et le serveur, il fallut ajouter ces contraintes dans l'arborescence.
TOMCO est séparé par 3 types de répertoires :

- **JavaRessources** qui contient toutes les sources liées à java. Nous pouvons y retrouver 3 packages : Applet, Conf et Servlet.
  Applet n'est pas réellement un package au sens de java, il a été construit en tant que source folder. Les fichiers class créer de ce répertoire sont ensuite envoyé vers l'output folder : TOMCO/WebContent/applet . Ce dossier contient toutes les classes nécessaires à la visualisation de l'applet qui permet d'afficher le résultat de l'exécution de la classe créer à base de tom. Nous y retrouvons par exemple un jar d'exécution de tom : tom-runtime.jar qui contient seulement quelques dossiers de tom-runtime-full.jar.En effet, il n'y a que les dossiers share, aterm, jjtraveler et tom. En cas de modification de fonctionnalité importante du langage, il faudra bien évidemment régénérer ce jar.
  Conf et Servlet sont dans source folder dont la sortie est renvoyé vers TOMCO/build/classes
  Conf est un package qui permet de conserver dans un seul et même endroit la configuration des fichiers. On y retrouve le fichier *ConfString* qui contient tous les champs qui sont appelée lors de la création de la vue ou des contrôleurs. Comme par exemple, l'expression régulière pour trouver le nom de la classe qui a été donné dans le champs code source de la page web, ou les noms des boutons ou les liens vers les fichiers qui sont nécessaire au bon fonctionnement de la page web. Certains de ces champs était limité à l'action sur ma propre machine. Il ne faut pas oublié de les modifier pour que cela soit possible de faire la même chose sur un serveur d'application. (`String root="workspace2/TOMCO/"`)
  Servlet contient 3 fichiers qui extends tous de httpServlet. *FormulaireServlet* est le contrôleur pour la partie gauche du formulaire qui contient un champs éditable où on peut mettre son code source et 2 boutons. Le boutons « RUN » permet de lancer le script qui lance la compilation de tom et javac. Si tout s'est bien passé, on récupère un jars qui contient le fichier class nécessaire à l'exécution de l'applet. Le bouton « RESET » permet de récupérer le formulaire comme si nous n'avions jamais lancé de requête. Le fichier *FormRServlet* quant à lui contient permet l'exécution du bouton stop aprés l'applet. Ce bouton permet d'arréter l'applet en cas de boucle sans changer ce qui a été rentré dans le champ du code source. Et pour finir *JarAppletServlet* permet de charger en lien le jar pour l'exécution de l'applet.
- **WebContent** qui contient tous les fichiers auquel le client aura accès. C'est pour cela qu'on y retrouve la page de style css , les pages jsp et xhtml , les images, les classes de l'applet ainsi que le javascript. On y retrouve aussi le dossier WEB-INF qui contient la configuration de l'exécution du site web (time-out des sessions, liens vers les classes de httpSessionListener, etc … )
- **ScriptBash** est un répertoire qui contient le script permettant de compiler le fichier tom et java. Ce fichier est un script bash qui configure les variables d'environnement pour TOM et il permet de créer le jar qui sera stockée coté serveur. Si pour une raison ou une autre le fichier java ou class n'est pas créer , ce script permet d'afficher à quel moment il s'est arrêté ainsi que les erreurs renvoyées sur le terminal. Tout ceux ci sera montrer dans le champs à droite dans le site web.

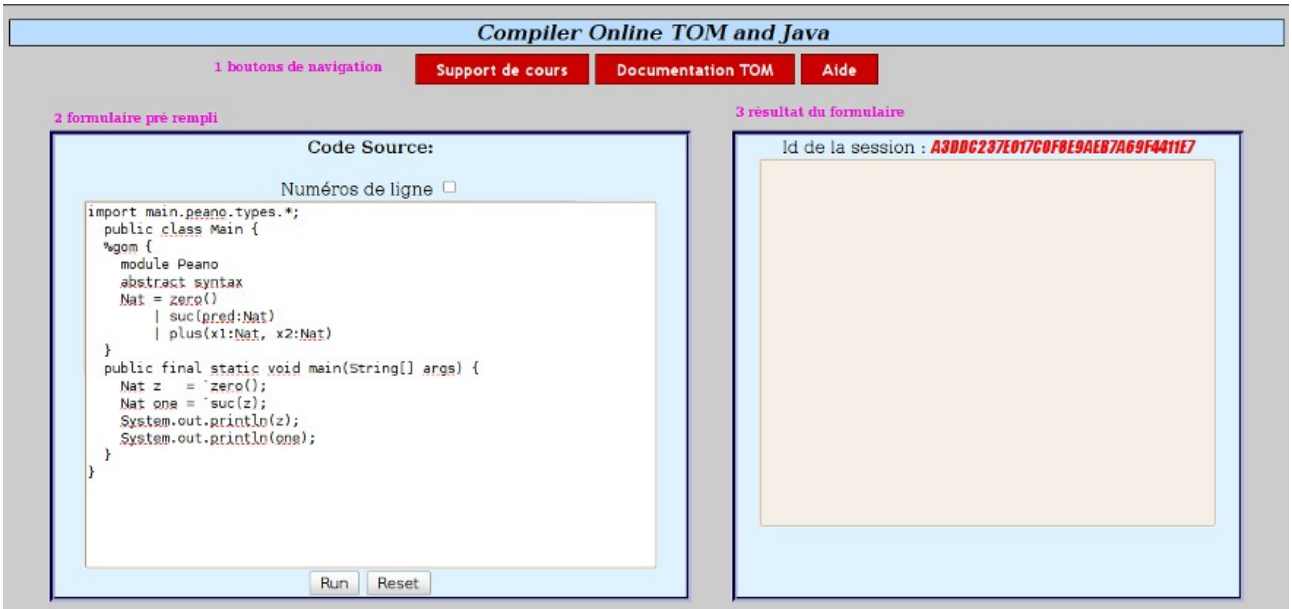
Pour de plus ample informations sur les classes cités ou sur les méthodes, nous vous invitons à consulter la javadoc et les commentaires qui ont été ajouter dans les codes sources.

## 2.2 Le fonctionnement du site web

Au départ, nous avons essayé de limiter la taille des fichiers en les découpant en plusieurs morceau. C'est pour cela que l'on a 2 fichier avec l'extension jsp. *Index.jsp* est le fichier maitre qui contient tout l'aspect visuel du site, alors que fonctionIndex.jsp est un fichier qui contient toutes les fonctions nécessaires aux fonctionnement du site.
Ainsi, si le nouveau programmeur veut changé l'aspect visuel il devra s'orienter vers Index.jsp ou miseEnPage.css. Mais s'il veut changé le retour d'une fonction ou tout autre il pourra se diriger vers les servlets qui contrôle ou fonctionsIndex.jsp
Ensuite, nous avons limité l'importance de fichier avec extension javascript qui ne sont pas automatiquement activé. Le seul élément lié au javascript et l'affichage dans le code source du numéro de ligne: numeroteLigne.js
Les images utilisées sont toute regroupé dans le dossier WebContent/image.

Le client n'aura accès qu'à ce que contient le dossier WebContent. C'est pourquoi quand on compile le dossier applet , il doit y avoir le dossier de sortie des .class dans ce dossier. Pour le déploiement, on construira un .war. Pour la construction du war, on se mettra au niveau du WebContent.

Le site web en lui même ressemble à ceci :




Pour faciliter l'apprentissage du nouvel utilisateur, nous avons mis un code source initial qui est l'utilisation des nombre de Peano.

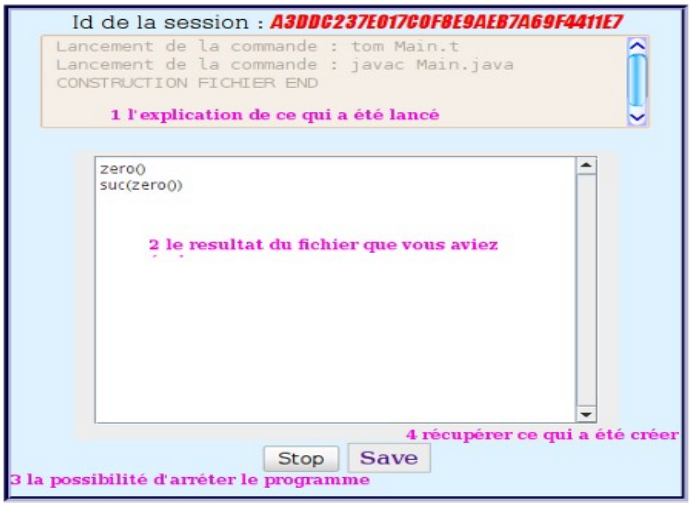
A chaque nouvel connexion, on utilise l'identifiant de la session, pour créer un répertoire temporaire sur le serveur. Pour cela, on utilise le fait qu'à chaque nouvelle session (qui a une durée de 60 minute, ceci est réglé dans web.xml), on crée un répertoire. A la fin de la session, ce répertoire est supprimé, ainsi que tout ce qu'il contient. Le fait qu'il y est un identifiant de session, permet en cas de fermeture du navigateur de revenir sur la page et de ne rien perdre des informations précédemment entré. A chaque appel de *FormulaireServlet*, on supprime les fichiers dans ce dossier.

Appuyer sur Reset, permet de revenir à l'état initial. Cela ne change pas l'identifiant de la session car il y a une durée de session qui doit être respecté quoiqu'il arrive.

C'est le fait d'appuyer sur Run qui permet de créer le fichier tom et de le stocker dans ce répertoire temporaire. Si on avait afficher les numéro de ligne, le fait de cliquer sur Run enlèverai ces numéros pour être sûr de ne pas les avoir dans le fichier tom créer. Ensuite, cela exécute plusieurs

commandes: tom nomFichier.t, javac nomFichier.java, java nomFichier.class et jar cf etud.jar *

On obtient alors cette fenêtre :




En cas d'erreur, on montre tous ce que les compilateurs ont pu écrire, pour que l'utilisateur puisse corriger sans problème. Dans le cas d'une erreur, on ne va pas jusqu'à la création du jar et donc la fenêtre 2 ainsi que les boutons seront inexistant.
Si tout c'est bien passé, on propose soit d'arrêter l'exécution de l'applet soit d'enregistrer les fichiers créer en enregistrant le jar qui contient tous les fichiers.
Dans le cas de l'utilisation de la fonctionnalité « stop », on efface la console 1 et 2 de l'image ci dessus, mais pas ce qu'il y avais dans le code source.
Pour le bouton « save », on ne fais que enregistrer le fichier.

En ce qui concerne la configuration du site web, on le retrouve dans le fichier web.xml.

# 3 Élément à ajouter

Pour le moment, il n'y a que très peu de fonctionnalité, mais on pourrait pensé qu'il serait utile de rentre le site capable de remplacer entièrement un éditeur de texte en ajoutant du javascript pour faire les tabulations dans le textArea afin d'avoir une bonne indentation, ou le chargement direct de fichier d'extension t, java ou txt dans le textArea.

On pourrait en plus, essayer de modifier le site pour pouvoir y faire des projets. Dans ce cas, la fonction qui permet d'exécuter les commandes devrait être différentes donc dans formulaireServlet, il faudrait une nouvelle fonction.

Certaines personnes ont même pensé que la coloration syntaxique serait intéressante dans le cadre du code source. Ainsi qu'une coloration dans le cadre du résultat de l'exécution, pour différencier les cas d'erreur, des cas normaux.