

Tutoriel pour développeur WebObjects: Hello World

Ce nouveau volet de la série de tutoriaux sur WebObjects fait suite au billet sur [l'installation](#) de l'environnement de travail. Nous allons ici créer notre premier projet et en expliquer les différentes parties et le fonctionnement général.

Tout au long de ce tutoriel je vais considérer que vous avez déjà une connaissance normale du langage Java et d'eclipse. Si tel n'est pas le cas je vous conseille de vous former dans un premier temps sur ces outils car sinon vous risquez d'éprouver quelques difficultés à suivre ce qu'il se passe.

Pour ce tutoriel et les suivants, je m'inspire fortement des screencasts de David LeBer WebObjects, WOLips and Wonder tutorial [part 1](#) et [part 2](#) qui sont disponible sur le site de WOCommunity avec tous les autres [screencasts](#) que je vous recommande vraiment.

Créer un projet Wonder

Première étape, nous allons créer dans notre joli eclipse disposant du plug-in WOLips un projet Wonder.

Vous êtes habitué à eclipse donc aucune difficulté jusqu'ici: File > New > Wonder Application.

Première étape, le nom du projet. Pour illustrer notre tutoriel, nous allons créer une petite application permettant à un utilisateur de créer sa collection de séries et de se souvenir du dernier épisode regardé. Notre projet s'appellera donc « Series ». Vous pouvez également choisir votre workspace eclipse pour ce projet.

Deuxième étape vous pouvez choisir le package de base de vos classes et de vos composants. Le standart pour le package des composants est d'utiliser le package de

base auquel est ajouté `.components`, c'est à dire que pour un package de base `xxx.yyy` vous utiliserez le package `xxx.yyy.components`. Dans notre exemple je vais utiliser comme racine `com.pierreschambacher.series`.

Etape intermédiaire, si vous avez d'autres projets vous pourrez indiquer que vous les utilisez dans votre nouvelle application. Si vous n'avez pas d'autre projet eclipse cet écran n'apparaît normalement pas, s'il apparaît choisissez `next`.

Dernière étape, l'import d'EOModels existants. Nous n'avons pas non plus d'EOModel existant, cliquez donc sur `Finish`. Pour information cette étape vous permettra d'utiliser un modèle déjà existant.

Vous pouvez voir qu'eclipse vient de créer votre projet avec tout un tas de dossiers, packages et fichiers déjà écrits.

Classes, packages, ressources, ...

Faisons un tour des dossiers et fichiers générés pour maîtriser la structure de notre projet.

Sources

Tout d'abord vous remarquerez trois classes dans le package de base que vous avez fourni au wizard: `Application`, `DirectAction` et `Session`.

Application

Il s'agit de votre classe main. Peu de choses sont à faire avec ce fichier si ce n'est de la configuration au démarrage comme l'indique le commentaire inséré par le template.

DirectAction

Les direct actions sont des méthodes dites "stateless". Elles n'ont pas besoin d'une session, bien qu'il soit possible de la transmettre par cookie par exemple, et sont accessible directement par leur URL à tout moment et par n'importe qui. La méthode la plus classique que l'on trouvera dans un fichier de direct action est la méthode d'authentification sur un site web, car on ne souhaite par forcément qu'elle soit couplée à une page en particulier et n'est associée à aucune session.

Session

Comme son nom l'indique il s'agit de la classe qui sera instanciée pour créer des sessions pour les utilisateurs. Les attributs d'instance que vous y ajoutés dureront aussi longtemps que la session de l'utilisateur durera. C'est ici généralement que l'on retrouve stocké l'utilisateur connecté et quelques autres informations. Lorsque vous vous trouvez dans un composant WebObjects, la session est simplement accessible à l'aide de la méthode `session()` qui par défaut demandera un cast vers le type `Session` de votre projet, le type renvoyé étant une classe de plus haut niveau.

Dans le package `components` se trouve un unique fichier.

Main

Ce fichier au nom un peu trompeur ne contient pas le main de votre application (il est dans la classe `Application`, regardez juste au dessus :-p). Il s'agit de la page principale de votre application qui a été gentiment déjà créée pour vous. Ou plutôt une partie du composant `Main`, voir juste ci-dessous.

Components

À côté du dossier Sources, vous remarquez un autre dossier appelé Components qui contient un élément Main WO, et conjointement dans le package `com.pierreschambacher.series.components` vous remarquez un `Main.java`. Sachez donc qu'un composant graphique WebObjects est constitué de plusieurs éléments portant tous le même nom mais des extensions différentes:

- `Composant.html`: fichier de template contenant du texte, des balises html et des balises WebObjects. C'est le principal fichier sur lequel vous allez travailler.
- `Composant.java`: fichier de code qui permet de définir le comportement de votre page. N'oubliez pas que vous êtes cependant dans une vue et le code contenu ici devrait être minimaliste.
- `Composant.wod`: vous allez définir ici la nature des webobjects défini dans votre fichier HTML et mettre en place le binding des variables à l'aide du fichier java, mais j'y reviendrais plus tard.
- `Composant.api`: définition des bindings visibles depuis l'extérieur de votre composant. Vous pouvez définir qu'un binding est requis pour obliger le programmeur à vous le fournir par la suite.
- `Composant.woo`: contient des informations comme la version de WebObjects, l'encodage des caractères et la configuration des DisplayGroups. Nous n'aborderons pas ce fichier, inutile d'y toucher.

Libraries

Rien de particulier ici, il s'agit du dossier où déposer les fichiers jar nécessaires à votre projet avant de les ajouter dans votre Build Path.

Resources

C'est dans ce dossier que se trouve le fichier de configuration Properties. C'est également ici que plus tard nous ajouterons notre modèle mais il est encore trop tôt pour cela.

WebServerRessources

Images, vidéos, fichiers de style CSS ... les ressources statiques de votre site vont ici.
Vous êtes libre de les organiser dans une hiérarchie de dossier.

woproject

NE TOUCHEZ PAS !! :-p

Run as ...

Rien de compliqué pour cette opération. Faites un clic droit sur votre projet, choisissez « Run as... » ou plutôt « Debug as... », puis WOApplication.

eclipse recherche le main et vous demande de lui indiquer lequel choisir, prenez bien évidemment celui qui se trouve dans votre classe Application (dans mon cas dans le package com.pierreschambacher.series).

À cette étape, votre navigateur préféré se lance et affiche le message « Hello World ». Bravo vous venez de lancer votre première application WebObjects.

Modifier un composant

Maintenant que vous avez pris connaissance de l'arborescence de votre nouveau projet, nous allons essayer de le modifier un petit peu !

REMARQUE IMPORTANTE: dans la mesure du possible lorsque vous avez une instance de l'application en train de fonctionner et que vous modifiez du code, WOLips va essayer de modifier votre instance « à chaud ». Des fois votre modification permettra un tel remplacement mais des fois le changement sera trop important pour le permettre et vous aurez une fenêtre d'alerte « Hot Code Replace Failed » vous demandant quoi faire. Ne vous inquiétez pas, contentez vous de cliquer sur « Terminate » afin de tuer l'instance courante. Vous aurez simplement à relancer l'application une fois vos modifications terminées.

Ouvrez le dossier Components et double-cliquez sur Main et sa petite icône de pièce de puzzle. Eclipse ouvre une fenêtre partagée horizontalement en deux. La partie haute contient du HTML tout ce qu'il y a de plus classique et la partie basse est vide complètement.

Mon premier WebObject

Vous remarquez dans le code HTML un message « Hello World ». Supprimez ce texte et rentrez à la place la balise suivante:

```
<webobject name = « helloString » />
```

Si vous sauvegardez maintenant, vous remarquez qu'Eclipse n'est pas très heureux parce que vous avez utilisé un WebObject helloString qu'il ne connaît pas. Ajoutez donc les lignes suivantes dans la partie basse de la fenêtre jusqu'à présent encore vite:

```
helloString: WOString {  
    }
```

Sauvegardez et vous remarquez qu'Eclipse n'est toujours pas heureux parce que vous utilisez un WOString sans définir le binding « value » qui est requis ! Insérez donc entre les accolades:

```
value = « Hello World ! »;
```

Et cette fois si après sauvegarde, Eclipse est heureux. Relancez l'application et vous devriez avoir votre Hello World ! affiché dans votre navigateur préféré. Vous pouvez changer la chaîne de caractères pour ce que vous voulez, sauvegarder et actualiser la page, vous remarquez que le message affiché dans le navigateur change !

Cependant sachez qu'il existe deux façons d'écrire les templates WebObjects. Vous venez d'utiliser l'ancienne façon, voyons à présent la nouvelle écriture apportée par le projet Wonder et que personnellement je préfère.

Supprimez complètement le contenu de la fenêtre du bas, le fichier WOD. Dans le fichier HTML, supprimez votre balise <webobject> et remplacez la par celle-ci:

```
<wo:str value = « Hello World ! » />
```

Vous remarquez que l'on a plus besoin de nommer notre variable ce qui je vous l'assure est un gros point positif. De plus à l'usage cette notation est plus pratique pour relire un fichier et savoir rapidement ce qu'il fait car on sait immédiatement ce que fait une balise sans avoir à consulter le WOD. Relancez l'application pour vérifier que tout va bien.

Un peu de dynamisme

Afficher une chaîne de caractère, c'est bien mais on voudrait pouvoir afficher quelqu'un d'un peu plus dynamique. Nous sommes au XXIème siècle quand même !

Chaîne de caractère

Dirigez vous vers votre fichier Main.java. Ajoutez une méthode currentTime renvoyant un objet String comme suit:

```
public String currentTime() {  
    return new java.util.Date().toString();  
}
```

À présent dans votre fichier Main.html, remplacez dans la balise wo:str le value= »Hello World ! » par value= »\$currentTime ». La présence du caractère \$ indique qu'il s'agit d'un binding. WebObjects va alors rechercher dans le fichier Java:

1. une méthode getCurrentTime renvoyant un objet et ne prenant aucun paramètre
2. une méthode publique nommée currentTime, renvoyant un objet et ne prenant aucun paramètre
3. un attribut public nommé currentTime

Vous aurez compris dans notre cas que WebObjects sera satisfait de trouver notre méthode fraîchement créée. Dans le cas où plusieurs de ces éléments existent, ils sont choisis dans l'ordre indiqué, c'est donc une méthode get qui sera choisie en premier.

Relancez l'application, vous voyez l'heure actuelle affichée dans votre navigateur. Si vous actualisez la page, l'heure est mise à jour.

Une date

Nous avons renvoyé une chaîne de caractère afin d'afficher la date dans notre fenêtre. Cependant il existe une façon un peu plus propre de faire la même chose.

Changez la signature de la méthode `currentTime`, qui ne renvoie plus un objet `String` mais un objet `NSTimestamp`. Il suffit de créer une nouvelle instance de cet objet:

```
public NSTimestamp currentTime() {  
    return new NSTimestamp();  
}
```

Vous constatez alors que votre page affiche l'heure actuelle, mais il faut bien le dire, dans un format absolument dégueulasse ! Il faut à tout prix remédier à cela.

Non non, modifier la méthode dans le fichier java n'est pas une bonne idée, ajoutez plutôt l'attribut `dateFormat` à votre balise `wo:str` comme suit:

```
<wo:str  
    value = « $currentTime »  
    dateFormat = « HH:mm:ss dd/MM/yyyy » />
```

Actualisez la page ... c'est mieux non ?

Un peu d'Histoire

WebObjects a été développé par la société NeXT en Objective C. Les classes utilisées en Objective C sont précédées par NS (NeXT Step). La société est rachetée plus tard par Apple mais les classes garderont historiquement ce préfixe. À partir de la version 4 de WebObjects, Java remplace petit à petit l'Objective C jusqu'à l'avoir complètement remplacé aujourd'hui mais afin de ne pas perdre les personnes ayant développé pendant longtemps en Objective C, les méthodes du framework ont été portées vers Java avec leur nom et leurs méthodes. Ce choix est fait au détriment en revanche des programmeurs Java qui devront apprendre à remplacer leurs classes habituelles par celles de WebObjects.

Comme vous l'avez vu dans l'exemple précédent, il faudra utiliser NSTimestamp en lieu et place d'une Date.

Pour les tableaux, oubliez l'ArrayList, vous allez utiliser les classes NSArray et NSMutableArray. Pourquoi deux classes me direz vous ? Comme leur nom l'indique, l'un est un tableau, l'autre un tableau modifiable. Si vous avez besoin en paramètre d'un tableau que vous n'allez pas modifier, indiquez qu'il s'agit d'un NSArray, mais si vous avez l'intention de changer son contenu utilisez NSMutableArray. Si vous avez un NSArray et que vous tenez absolument à pouvoir effectuer des modifications sur ses éléments vous disposez de la méthode mutableClone qui créera un clone de votre NSArray en version Mutable.

Pour les dictionnaires, oubliez la HashMap et dites bonjour à NSDictionary et NSMutableDictionary . Je vous renvoie au paragraphe précédent pour le Mutable/non Mutable.

Si vous aimez utiliser les Sets, utilisez NSSet et NSMutableSet.

Interaction avec l'utilisateur

Suffit d'afficher la date dans son navigateur sauf si vous êtes en train de créer horloge-parlante.com ! Nous allons inviter l'utilisateur à rentrer son nom et le saluer pour le remercier.

Mise en place

Modifiez votre fichier HTML pour obtenir le résultat suivant dans la partie body:

```
<wo:form>
    Enter your name: <wo:textfield value = « $username » />
    <wo:submit value = « Say hi » action = « $sayHi » />
</wo:form>
<br />
<h1>Hello <wo:str value = « $username » /> !</h1>
```

Si vous sauvez, vous constaterez d'eclipse n'est pas vraiment heureux parce qu'il ne trouve pas username et sayHi dans votre fichier Java. Nous allons résoudre le problème. Ne quittez pas le fichier HTML, maintenez le bouton Command enfoncé et placez votre souris sur \$username, vous remarquerez que le nom se transforme en lien. Si vous cliquez, eclipse vous indique qu'il ne trouve pas username dans votre fichier Java et se propose de le créer pour vous. Voyons les différents éléments de cette boîte de dialogue

- les boutons radio vous permettent de choisir si votre binding est un objet, un NSArray d'objets ou un NSMutableArray d'objet.
- le champ texte permet de choisir le type de notre objet, ici java.lang.String nous convient parfaitement.
- les checkbox suivantes permettent de demander la génération d'un attribut, d'un get et d'un set, ce qui est coché par défaut. Vous pouvez choisir de mettre « get » ou non sur le getter avec la dernière checkbox.

Dans notre cas tout est configuré comme il faut, faite simplement OK et vous vous retrouvez dans le fichier Java où vous constatez qu'eclipse s'est chargé de générer un attribut privé username, une méthode publique username qui renvoie cet attribut et une méthode setUsername qui s'occupe de peupler l'attribut. Concernant username eclipse est heureux.

Retournez dans le fichier HTML, command-clic sur \$sayHi qui est une action et remarquez que cette fois ci eclipse propose de créer une méthode renvoyant un objet de type WOActionResult. Cliquez sur OK la configuration par défaut nous convient. Voici la méthode générée:

```
public WOActionResults sayHi() {  
    return null;  
}
```

Petite explication. Les actions sont des méthodes qui permettent de définir un comportement sur une action de l'utilisateur, par exemple ici le clic sur le bouton submit. L'objet retourné par une action est très important car il définit la prochaine page à afficher. Si la méthode retourne null, alors c'est la même page qui est regardée, sinon c'est la page retournée qui sera envoyée au client. Pour créer une page différente, utilisez la méthode pageWithName qui possède deux signatures. Nous y reviendrons un peu plus tard dans ce tutoriel dans la section « Deuxième composant ».

Dans le cas présent, il suffit de recharger la page courante, donc laissez la méthode générée telle quelle. Sauvegardez tout et lancez l'application, ça fonctionne, magique non ?

Explications

Vous n'avez pas l'impression d'avoir fait grand chose et pourtant ça marche. Faisons un petit retour sur le fonctionnement de cette page.

Concernant le `wo:str`, vous voyez qu'il y a un binding de valeur sur `username`, donc de façon très simple au moment du chargement de la page la méthode `username` est appelée et insérée dans la page. Rien n'a changé par rapport à tout à l'heure.

Pour le `wo:textfield` et le `wo:submit`, vous avez mis en place un binding à l'aide de l'élément `value` du `textfield` sur `username`. Donc au chargement de la page la valeur de `username` est utilisée pour remplir le `textfield`, mais dans le cas d'un retour d'informations, comme c'est le cas avec le clic sur le bouton `submit`, c'est la méthode `SET` qui est appelée pour modifier la valeur dans le fichier Java avec la valeur que l'utilisateur a mis dans son navigateur. Comme l'action du bouton `submit` renvoie `null`, la même page est rechargée avec cette fois ci `username` différent de `null`.

Vous l'aurez compris, certains bindings fonctionnent uniquement en `GET`, comme le `value` du `wo:str`, et d'autres fonctionnent en `GET` et `SET` comme le `value` du `textfield`. Consultez la documentation pour savoir à quel type de binding vous avez affaire, de toute façon c'est généralement assez explicite. Dernier type de binding, les actions pour réagir aux actions utilisateurs. Quelque chose que vous remarquez rapidement, votre fichier HTML a connaissance des éléments publics de votre fichier Java, mais ce dernier ignore totalement l'existence de la page HTML. Tout fonctionne par bindings, il faudra vous habituer.

Améliorations

Première chose, vous avez remarqué qu'au premier chargement il est simplement affiché « Hello ! » ce qui est un peu bête. Ajoutez sur votre `wo:str` le binding suivant:

```
valueWhenEmpty = « World »
```

Rafraichissez la page. Mieux non ?

Deuxième petite amélioration-démonstration, sur le binding value du wo:str, changez le pour mettre ceci:

```
value = « $username.toUpperCase »
```

Rafraichissez de nouveau. Vous aurez compris, rien n'empêche dans un binding d'utiliser un chaînage de méthodes.

Deuxième Composant

Dernier point que nous aborderons sur ce tutoriel, nous allons créer un deuxième composant et créer un lien entre les deux.

Sur le dossier Components, faite un clic droit New > WOComponent. Entrez comme nom « About » et assurez vous que la superclass est er.extensions.components.ERXComponent et que la case « Create HTML contents » soit cochée. Vous avez votre deuxième composant WebObjects. Mettez ce que vous souhaitez dans le HTML pour reconnaître votre page About puis retournez dans votre page Main. Ajoutez un nouveau composant dans le HTML comme ceci:

```
<wo:link string = « About » action = « $openAbout » />
```

Comme tout à l'heure, command-clic sur openAbout et laissez eclipse créer le squelette de la méthode pour vous. Dans le code remplacez le return null par ceci:

```
return pageWithName(About.class);
```

Vous disposez également de la signature suivante:

```
return pageWithName(About.class.getName());
```

Lorsque vous utilisez la méthode `pageWithName`, utilisez toujours le style `MonComposant.class` et jamais « `MonComposant` ». Cela au cas où vous souhaiteriez renommer votre composant un jour. La méthode `pageWithName` va instancier votre composant et l'initialiser, et en étant retourné par l'action vous indiquez à `WebObjects` qu'il s'agit de la prochaine page à afficher. Compilez et lancez l'application et observez le résultat.

Conclusion

Suite à ce tutoriel vous devriez avoir compris comment créer une page, y insérer du HTML et des composants mélangés. Vous savez mettre en place des bindings avec le code Java, changer la page courante, créer des données dynamiques. Vous êtes cependant un peu léger concernant les composants que vous connaissez, le prochain tutoriel sera consacré à la liste des principaux composants que l'on utilise 80% du temps.