

Premiers pas avec le Google Web Toolkit

Google nous a gratifiés au printemps dernier d'un kit de développement qui tranche avec tout ce que nous connaissions jusque là en matière de réalisation d'applications Web en Java. Il s'agit, avec cet outil, d'injecter de l'AJAX (Asynchronous JavaScript And XML) dans nos réalisations.

Le propos de cet article est de vous guider dans l'installation de ce toolkit et dans l'écriture d'une toute première application.

AJAX, le nouveau héros du Web ?

AJAX n'est pas une technologie ni un produit, c'est une idée. L'idée est d'utiliser conjointement JavaScript et XMLHttpRequest. JavaScript est un langage de scripting présent depuis longtemps sur tous les navigateurs Web du marché. XMLHttpRequest est un mécanisme d'appel de procédure distante, éventuellement asynchrone, introduit dès 1999 dans Internet Explorer (et disponible sur les produits concurrents). XMLHttpRequest permet d'interroger le serveur sans provoquer de rechargement complet de la page, à l'inverse des échanges HTML/http. Le code d'IHM peut alors être écrit entièrement en JavaScript, être exécuté sur le navigateur, et faire appel au besoin au serveur, grâce à XMLHttpRequest.

L'enfant terrible de la famille AJAX, c'est JavaScript. Développer du code robuste et portable avec JavaScript n'est pas devenu subitement plus facile en 2006, juste parce que l'on s'est mis à beaucoup parler d'AJAX ! Les problèmes que posent Javascript existent toujours. Il est donc toujours difficile de développer et tester du Javascript. Pour être productif avec ce langage, deux stratégies ont été envisagées :

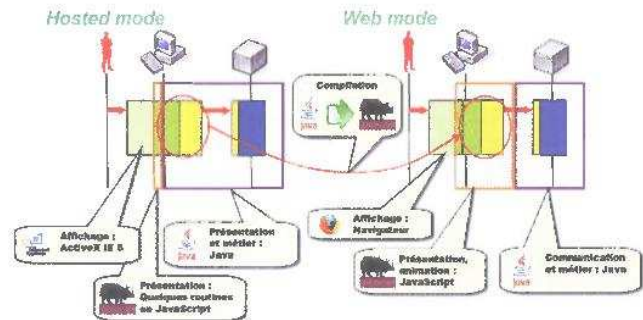
- Proposer un environnement de développement qui "comprend" le Javascript et soit capable d'intégrer ou de simuler correctement un navigateur Web comme IE ou Firefox,
- Proposer des bibliothèques de composants Javascript qui soulagent le développeur en lui proposant des composants prêts à l'emploi.

La solution proposée par Google Web Toolkit (GWT) est beaucoup plus originale et radicale : puisqu'il est difficile de programmer en Javascript, passons-nous de Javascript ! Programmons notre interface homme machine en Java – nous disposons déjà, pour cela, d'excellents outils : compilateur, débogueur, framework de tests. Puis, quand notre application fonctionne, transformons une partie du code Java en code Javascript. Le pari étonnant de Google est que cette transformation peut être automatisée à l'aide d'un compilateur, en fait : un traducteur Java vers Javascript. Et effectivement, ça marche.

Les grands principes du Google Web Toolkit

Une application AJAX écrite avec le GWT est un ensemble de classes Java dont certaines construisent et animent l'interface homme machine proprement dite et d'autres fournissent informations et services métiers à cette interface.

Les classes de la première catégorie, devant, in fine, s'exécuter sur un navigateur, sont destinées à être transformées en Javascript. Les classes de la seconde catégorie doivent s'exécuter sur le serveur. Elles ne seront pas transformées. Toutes les classes écrites dans un projet GWT ne sont donc pas traduites en Javascript.



Il existe deux modes d'exécution : le mode "hosted" et le mode "Web". En développement, on utilise le mode "hosted". Dans ce mode, on utilise un navigateur spécial, fourni par GWT et qui s'appuie, pour l'affichage proprement dit, sur IE (Windows) ou Gecko (Linux). Ce navigateur de développement abrite une machine virtuelle Java capable de manipuler les objets graphiques qui composent l'interface. Cette manipulation est réalisée à l'aide d'une API Java, elle aussi fournie par GWT.

En production, on utilise le mode "Web". Il faut au préalable "compiler" l'application afin que GWT transforme la partie cliente en Javascript. L'application peut ensuite être exécutée à partir de la plupart des navigateurs du marché.

GWT présente une autre originalité : elle retourne aux sources de la programmation IHM graphique. Une application GWT consiste à assembler des composants graphiques (appelés widgets) et à armer des gestionnaires sur les événements reçus par ces widgets, à l'instar de ce qui se fait avec Swing, SWT ou Visual Basic. Le GWT ne vient pas s'ajouter à la pile des technologies Java/Web (servlets, JSP, JSTL, Struts) : il les remplace.

Il est possible – et même facile – de créer de nouveaux types de widgets à partir de celles (assez rudimentaires) proposées en standard par GWT. Il est aussi possible d'intégrer des frameworks purement Javascript (Dojo, Rialto, Yahoo) en insérant simplement du code Javascript au sein de classes Java (à l'aide de commentaires spéciaux). Enfin, le GWT est livré avec une solution de tests automatisés basée sur JUnit et une API d'internationalisation.

Intégrer le Google Web Toolkit et Eclipse

La première action est, bien sûr, de télécharger le GWT. Il est disponible à l'adresse : <http://code.google.com/webtoolkit/download.html>. Il se présente sous la forme d'un fichier zip que l'on décompressera. A la racine de l'arborescence du GWT, on trouve, entre autres, un utilitaire appelé projectCreator. Placez-vous sur le répertoire qui doit contenir votre nouveau projet. Invoquez projectCreator en lui indiquant un nom de projet Eclipse :

```
c:\projects>c:\gwt-windows-1.2.22\projectCreator -eclipse myapp
```



Un projet GWT est créé au sein d'un répertoire appelé "myapp". Sa structure est assez classique : on trouve le répertoire de tout développement Java (src), on découvre aussi un répertoire "tomcat" qui donne une indication sur la nature de l'outil de communication http utilisé par GWT ...

Il faut maintenant créer un module GWT. Le module est le composant d'IHM de haut niveau défini par GWT. Pour simplifier, disons qu'un module est l'équivalent d'un écran. Pour créer ce module, on utilise un autre outil appelé applicationCreator.

```
c:\projects>c:\gwt-windows-1.2.22\applicationCreator -eclipse myapp  
com.objetdirect.myapp.mymodule.client.Root
```

Notez que dans l'exemple ci-dessus, le nom de la classe est laissé à votre appréciation. C'est aussi le cas de l'arborescence des packages, à l'exception du dernier, "client" qui doit impérativement porter ce nom. Cet utilitaire fait diverses choses : il crée une classe d'entrée du module (Root) ; il construit une définition de projet Eclipse (en créant des fichiers .project et .classpath) ; il fabrique un utilitaire de compilation Java-Javascript pour ce module (Root-compile.cmd), un exécuteur autonome du module (Root-shell.cmd) et un exécuteur pour Eclipse (Root.launch).

La naissance d'une page AJAX

Nous sommes prêts maintenant à développer une page AJAX avec GWT et Eclipse. Lançons Eclipse. La première chose à faire est de demander à Eclipse de retrouver le projet que l'on vient de créer (en passant, soit par "File > New > Project ..." ou par "File > Import ... > Existing Projects into Workspace"). Le projet apparaît. On peut constater que ce projet est prêt à l'emploi (les bibliothèques du GWT ont déjà été déclarées) et qu'une implémentation exemple existe dans la classe Root.java. Notez que cette classe est dans un package appelé client et qu'au même niveau que ce package, existent deux autres packages : server et public.

- Le répertoire client contient toutes les classes qui sont utilisées dans la partie purement IHM. Ce sont les classes de ce package (et de ses sous-packages), et seulement elles, qui seront traduites en Javascript.
- Le répertoire serveur contient les classes qui sont spécifiques à la partie serveur. Lui aussi peut contenir des sous-packages.
- Le répertoire public contient les ressources autres que les classes Java : feuilles CSS, source Javascript, pages HTML, images, etc.

Les classes contenues dans la partie cliente doivent être strictement autonomes. Elles ne doivent en aucun cas référencer des classes contenues dans la partie serveur. L'inverse n'est pas vrai.

A côté de ces trois répertoires, on découvre un fichier de définition du module appelé Root.gwt.xml. Ouvrons le fichier de définition. On y trouve les déclarations suivantes :

```
<module>  
  
  <!-- Inherit the core Web Toolkit stuff.      -->  
  <inherits name='com.google.gwt.user.User' />  
  
  <!-- Specify the app entry point class.      -->  
  <entry-point class='com.objetdirect.myapp.mymodule.client.Root' />  
  
</module>
```

La clause "inherits" indique que notre module reprend les définitions standard définies pour tout module GWT.



La clause "entry-point" indique un point d'entrée du module. Un point d'entrée est une classe Java qui est chargée et exécutée au démarrage du module. Cette classe est la fameuse Root que nous avons créée à l'aide de l'outil applicationCreator. Elle contient une méthode :

```
public void onModuleLoad() { ... }
```

C'est cette méthode qui sera invoquée lorsque le module sera chargé. C'est au sein de cette méthode que l'on construira le contenu de notre page. Contentons-nous ici d'une IHM triviale, constituée d'un unique bouton :

```
public void onModuleLoad() {  
    Button button = new Button("Cliquez");  
    RootPanel.get("slot0").add(button);  
}
```

La seconde ligne de la méthode accroche notre bouton à un élément de l'arbre des éléments graphiques qui constituent la page affichée par le navigateur. Mais où est donc définie cette page ? Retournons dans le répertoire public. Nous trouvons une page HTML, appelée Root.html. Cette page est presque vide :

```
<html>  
  <head>  
    ...  
    <meta name='gwt:module' content='com.objetdirect.myapp.mymodule.Root'>  
  </head>  
  
  <body>  
    ...  
    <script language="javascript" src="gwt.js"></script>  
    <div id="slot0"></div>  
  </body>  
</html>
```

C'est tout à fait normal. En GWT, la page HTML n'est qu'un socle sur lequel sera bâti toute l'IHM. On aurait pu aussi se dispenser de la balise div et construire à partir de l'élément de base, à savoir celui représenté par la balise body. Cette page contient aussi les références nécessaires pour que le code Javascript de notre module soit chargé.



Dans notre exemple, notre bouton sera intégré au sein de la balise div d'identifiant " slot0 ". Ajoutons le traitement d'un événement de sélection pour notre bouton :

```
public void onModuleLoad() {  
    final Button button = new Button("Cliquez");  
    button.addClickListener(new ClickListener() {  
        public void onClick(Widget sender) {  
            button.setText("Ok");  
        }  
    });  
    RootPanel.get("slot0").add(button);  
}
```

La manière de faire ressemble fort aux pratiques rencontrées dans la programmation Swing, AWT ou SWT : on crée un gestionnaire d'événement sous la forme d'une classe anonyme en implémentant une interface de traitement d'événement (ClickListener). Cette classe ne contient qu'une méthode : onClick qui est activée lorsque le bouton est sélectionné. Notez que la signature de cette méthode est simplifiée : elle ne reçoit pas d'objet événement qu'il faut consulter pour connaître les conditions de déclenchement de l'événement. Le contexte est passé directement en paramètre.

Le premier cri

Il ne nous reste plus qu'à exécuter notre page. Pour cela, il suffit d'activer le menu d'exécution d'Eclipse : le lanceur Root.launch y est déjà intégré. Deux fenêtres apparaissent :

- La première appelée : " Google Web Toolkit Development Shell / Port 8888 " matérialise notre serveur.

- La seconde appelée Wrapper HTML for Root " est notre navigateur dans le mode " hosted ". Cette seconde page affiche le bouton que nous avons programmé. Cliquons dessus : son label est modifié. Succès ! Fermons les deux fenêtres. Compilons notre application pour qu'elle puisse fonctionner en mode " Web ". Il suffit pour cela d'activer (à partir de la vue " package explorer " d'Eclipse) l'utilitaire Root-compile.cmd. Relançons l'application GWT. Ouvrons un véritable IE ou un navigateur Firefox. En entrant l'URL de la page HTML qui sert de socle à notre module GWT (<http://localhost:8888/com.objetdirect.myapp.mymodule.Root/Root.html>), le même bouton apparaît. Il a exactement le même comportement. Mais cette fois, c'est du Javascript qui est exécuté ... Cette initiation à la programmation GWT n'est que le début de l'exploration des nombreuses possibilités offertes par cet étonnant outil. Il montre néanmoins quelles sont ses grandes caractéristiques : programmation Java plutôt que Javascript, programmation événementielle " traditionnelle ", affranchissement complet par rapport aux technologies Java/Web, prise en main très rapide. Il nous reste encore beaucoup de sujets à voir : comment communiquer avec le serveur, comment intégrer des modules Javascript " prêts à l'emploi ", comment définir ses propres composants graphiques et comment tester ou internationaliser une page GWT. Tous ces thèmes feront l'objet de prochains articles.



■ Henri DARMET

Directeur Technique - Objet Direct / Homsys Group

A propos de Homsys Group :

Créé en 1991, Homsys Group est spécialisé autour de la Business Intelligence (Homsys) et les technologies Objet et Internet (Objet Direct). Homsys Group est implanté à Paris, Marseille, Lyon, Toulouse, Bordeaux, Grenoble et Rennes. www.homsysgroup.com - www.objetdirect.com