

Communiquer avec un serveur avec le Google Web Toolkit

Le Google Web Toolkit (GWT) est surtout connu pour son compilateur Java vers JavaScript et son environnement intégré d'exécution en mode développement (Hosted Mode). Mais le Google Web Toolkit est une plate-forme de développement Ajax complète. La dimension de communication client/serveur n'a évidemment pas été oubliée.

GWT définit la notion de service. Cette notion est proche de celle que connaissent des frameworks comme Spring ou EJB. Il s'agit d'objets Java qui publient des services sous la forme de méthodes. Dans le cas du Google Web Toolkit, un service est implémenté sous la forme d'une Servlet d'un genre particulier à GWT. Cet article montre comment un tel service peut être construit et déployé.

Quelques petits rappels sur GWT avant d'entrer dans le vif du sujet : en mode développement (Hosted mode) tout ou presque est écrit en Java. En mode production (Web mode) une partie – la partie purement cliente – est compilée en JavaScript et est projetée sur le navigateur de l'utilisateur. La plate-forme GWT – son compilateur Java vers JavaScript en particulier – a donc besoin d'isoler ce qui est spécifique à la partie cliente. C'est pourquoi un module GWT range ses sources dans deux répertoires :

- Le premier, appelé " client ", contient toutes les classes qui devront être, à terme, transformées en JavaScript. C'est sur lui que s'appliquent les contraintes de codage imposées par GWT.
- Le second appelé " server " contient les classes qui seront exécutées sur le serveur et qui ne seront donc jamais traduites. C'est ici que nous déposerons l'implémentation de nos services.
- Il existe aussi un pseudo package appelé " public " qui contient diverses ressources (feuilles CSS, page HTML, images, etc.). Ce package ne nous intéresse pas dans le cadre de cet article. Il ne contient aucun source Java.

Je voudrais aussi rappeler qu'un module GWT est défini par un fichier de configuration placé à la racine du module (c'est-à-dire dans le répertoire qui contient les packages " client ", " server " et " public "). Il est reconnaissable à son extension : ".gwt.xml". Ce fichier nous sera utile par la suite : c'est en son sein que les services doivent être déclarés pour être reconnus et déployés par GWT.

Définir et implémenter un service

Pour définir et implémenter un service, nous devons au minimum écrire deux interfaces et une classe. La première interface décrit la liste des méthodes qui constituent notre service. La classe implémente cette interface : c'est le cœur de notre service. La seconde interface permet d'invoquer le service de manière asynchrone.

Prenons le cas d'un service mathématique trivial qui propose l'exécution d'opérations élémentaires sur des doubles :

```
public interface IMath extends RemoteService {
    double add(double a, double b);
    double subtract(double a, double b);
}
```

```
double multiply(double a, double b);
double divide(double a, double b);
}
```

Cette interface – un POJI très classique, à l'exception de son héritage – doit être placée dans le package " client ". Grâce à l'héritage de " RemoteService " (package com.google.gwt.user.server.rpc), une interface définie par GWT, notre interface sera reconnue par lui.

La seconde étape consiste à implémenter ce service. Nous allons pour cela définir une classe dans le package " server " cette fois. GWT demande deux choses :

- Qu'elle hérite de la classe " RemoteServiceServlet " (package com.google.gwt.user.server.rpc)
 - Qu'elle implémente l'interface IMath que nous avons définie plus haut.
- Une classe du package " server " va donc dépendre de l'interface présente dans le " package " client. C'est ainsi et c'est légitime : le code " client " reste toujours disponible sous sa forme bytecode. C'est du code partagé. En mode production, notre interface existera donc en deux versions : en Java et en JavaScript !

L'implémentation du service est un exercice trivial :

```
public class SMath extends RemoteServiceServlet implements IMath {
    public double add(double a, double b) {
        return a+b;
    }
    public double subtract(double a, double b) {
        return a-b;
    }
    public double multiply(double a, double b) {
        return a*b;
    }
    public double divide(double a, double b) {
        return a/b;
    }
}
```

Que le service soit adhérent au Google Web Toolkit est à peine visible ! Tout cela pourrait sembler suffisant, mais ce n'est pas le cas. Car dans Ajax il y a " A " comme " asynchrone ". Or rien de ce que nous avons écrit ne permet de faire des appels asynchrones à notre service. C'est la raison pour laquelle il faut définir une interface supplémentaire, directement dérivée de l'interface du service :



```
public interface IMathAsync {
    void add(double a, double b, AsyncCallback cb);
    void subtract(double a, double b, AsyncCallback cb);
    void multiply(double a, double b, AsyncCallback cb);
    void divide(double a, double b, AsyncCallback cb);
}
```

Cette interface est définie par copie de l'interface de service à laquelle :

- On enlève l'héritage d'interface " RemoteService ",
- On ajoute au nom de l'interface le suffixe " Async ",
- Les méthodes gardent le même nom mais perdent leur valeur de retour (remplacées par " void "),
- Un paramètre supplémentaire de type " AsyncCallback " (package com.google.gwt.user.server.rpc) est ajouté en fin de liste.

Cette interface d'appel est placée dans le même package que l'interface de service. Sur " client " donc.

Le paramètre supplémentaire est nécessaire pour signifier à GWT quels sont les traitements qui doivent être effectués en cas de succès ou d'échec de l'invocation du service. Une invocation échoue si elle se termine par la levée d'une exception. Nous retrouverons cette interface "AsyncCallback" lorsque nous montrerons comment un service peut être appelé dans le code client. Il ne nous reste plus qu'à déployer ce service. Il suffit pour cela de le déclarer dans le fichier de configuration du module à l'aide d'une balise " servlet " :

```
<module>
...
<servlet path="/mathService"
class="com.objetdirect.myapp.server.SMATH"/>
</module>
```

Appeler un service

Regardons maintenant comment notre code client peut invoquer ce service. La première chose est de fabriquer un proxy qui désigne le service. Ce proxy sera créé et initialisé à l'aide des deux lignes suivantes :

```
ServiceDefTarget target = (ServiceDefTarget) GWT.create(IMath.class);
target.setServiceEntryPoint(GWT.getModuleBaseURL() + "mathService");
```

Notez que Service est identifié par une URL dont GWT nous fournit la base et qu'il suffit de compléter par l'étiquette (path) que nous avons déclarée dans le fichier de configuration " .gwt.xml " .

Il faut ensuite créer un objet " réflexe " qui va indiquer ce qu'il convient de faire en cas de réussite ou d'échec. Généralement cet objet est créé sous la forme d'une classe anonyme – ce que GWT accepte sans problème :

```
AsyncCallback cb = new AsyncCallback() {
    public void onFailure(Throwable caught) {
        Window.alert("Oops try again");
    }

    public void onSuccess(Object result) {
        Window.alert("Yes ! The result is : "+result);
    }
};
```

La méthode activée en cas de réussite – onSuccess – reçoit en paramètre la valeur retournée par notre service, un " double " dans notre cas. Pour les types Java simples – int, float, double, etc. – la valeur de retour est passée en version " objet " - Integer, Float, Double.

La méthode activée en cas d'échec – onFailure – reçoit en paramètre l'exception qui a provoqué l'arrêt du service.

L'appel du service en lui-même est effectué par la ligne suivante :

```
((IMathAsync)target).add(12.0, 14.0, cb);
```

L'exécution du code qui précède provoque l'apparition d'une fenêtre d'alerte sur le navigateur affichant la valeur 26.

Grandeur et servitude de l'asynchronisme

Noter que l'appel est asynchrone, c'est-à-dire que la main est rendue au code client avant que le service soit exécuté sur le serveur et que son résultat ne soit communiqué au client. Comment faire alors pour enchaîner des appels de services de manière synchrone ? Par exemple, comment écrire la procédure qui effectue les instructions suivantes :

- Multiplication de " a " par " a " donnant " a2 "
- Multiplication de " b " par " b " donnant " b2 "
- Multiplication de " b " par " a " donnant " ab "
- Multiplication de " ab " par 2 donnant " a2b "
- Ajout de " a2 " et " b2 " donnant " a2pb2 "
- Ajout de " a2pb2 " et " a2b " donnant " total "
- Affichage de total dans une fenêtre d'alerte

C'est possible, mais ce n'est pas si simple, car GWT ne semble pas offrir de solution pour appeler un service de manière synchrone.

Il faut, non pas faire suivre les appels, mais les imbriquer ! Le premier appel invoque la méthode " multiply ". Dans la méthode " onSuccess " de son objet réflexe, on insère un second appel à la méthode " multiply ". Dans l'objet réflexe de ce second appel on insère un troisième appel à la méthode " multiply " et ainsi de suite. Comme notre " procédure " contient six appels, il ne faudra pas moins de six objets réflexes imbriqués pour l'implémenter !

```
final a;
final b;
final double a2;
final double b2;
final double ab;
...
AsyncCallback cb1 = new AsyncCallback() {
    public void onFailure(Throwable caught) {}

    public void onSuccess(Object result) {
        a2 = ((Double)result).doubleValue();
        AsyncCallback cb2 = new AsyncCallback() {
            public void onFailure(Throwable caught) {}

            public void onSuccess(Object result) {
                b2 = ((Double)result).doubleValue();
                AsyncCallback cb3 = new AsyncCallback() {
                    ...
```

```
}  
((MathAsync)target).multiply(a, b, cb3);  
}  
};  
((MathAsync)target).multiply(b, b, cb1);  
}  
};  
((MathAsync)target).multiply(a, a, cb1);
```

Il est aisé de constater que ce code est beaucoup plus complexe et obscur qu'une suite d'instructions qui seraient exécutées de manière synchrone comme cela est traditionnellement fait dans une méthode Java. Conséquence : comme pour toute bonne chose, il faut savoir ne pas abuser de l'asynchronisme. Dans notre cas, il est temps de rajouter une méthode de service supplémentaire sur notre serveur qui calcule $(a+b)^2$ en un seul appel.

A la frontière des mondes

L'asynchronisme n'est pas la seule difficulté que rencontre le développeur GWT. Il faut aussi qu'il aide le compilateur à traduire paramètres et valeur de retour entre code client et code serveur. Pourquoi, me demandez-vous ? Il existe bien un mécanisme en Java qui fait cela très bien depuis fort longtemps : la sérialisation.

Mais la sérialisation Java ne s'applique pas ici pour une raison fort simple : le code client final sera traduit en JavaScript et échappe donc aux subtilités de la machine virtuelle Java. GWT possède son propre système de sérialisation, plus contraint que celui que nous connaissons de manière traditionnelle. Pour les types simples, pas de problème : GWT sait les sérialiser. Il en va de même de leur version objet (Integer, Float, etc) et pour les classes String et Date (java.util.Date).

Afin de bien marquer la différence, Google a défini une interface "IsSerializable" qui joue dans GWT le même rôle que "java.util.Serializable" pour la sérialisation Java standard.

A la demande de la communauté des développeurs, GWT accepte aussi, depuis la version 1.4, de prendre en compte dans son mécanisme spécifique, les classes marquées par "java.util.Serializable". Mais les deux mécanismes de sérialisation restent bien distincts. D'où quelques subtilités qu'il faut connaître.

Conclusion

GWT offre au développeur le complément indispensable pour que celui-ci puisse écrire des applications Ajax : un mécanisme de définition et d'appel de service. Ce mécanisme est la partie la plus subtile de GWT. Outre les contraintes apportées par une gestion obligatoirement asynchrone, il faut parfois entrer dans les méandres d'un mécanisme de sérialisation Java/JavaScript.original.

Une autre limite de ce mécanisme est qu'il refuse les apports syntaxiques du JDK 1.5. Generics et annotations ne sont donc pas acceptés. Conséquence : il n'est pas possible d'utiliser en l'état des POJO Hibernate récents. Nous verrons dans un prochain article comment résoudre ces divers problèmes. Cela nous permettra de mieux comprendre comment fonctionne la sérialisation GWT, ce que la place disponible dans ces pages ne m'a pas permis de faire pour cette fois.

■ **Henri DARMET**

Directeur Technique - Objet Direct / Homsys Group

Retour sur l'utilisation de GWT et autres API Google

En quoi GWT se distingue-t-il des autres framework AJAX ?

GWT est bien plus qu'un framework AJAX. Il faut plutôt le comparer à un L4G qui permet de coder en Java des clients DHTML / AJAX qui s'exécutent dans n'importe quel navigateur. Avec tous les avantages d'un L4G : productivité, réutilisation des composants, optimisation à la compilation, etc.



Combien de temps faut-il à un développeur Java pour créer sa première application ?

C'est toute la force de GWT. Google fait sauter la barrière de l'apprentissage du code Javascript et de l'architecture AJAX. Si bien qu'il faut très peu de temps à un développeur pour maîtriser le kit de développement. Les premiers écrans sont généralement produits en quelques heures.

Peut-on avoir confiance dans cette technologie ?

Google prend toujours son temps avant retirer le tag Bêta à une API. Pour GWT, c'est le cas depuis la version 1.4. Cette technologie est très stable, elle se comporte réellement comme on l'imagine, il n'y a pas de surprise ou de comportement étrange.

Les Widgets proposés par GWT sont-ils suffisants ?

GWT propose tous les composants de base. Cette palette est complétée par un grand nombre de frameworks open-source : GWT Ext, MyGWT, GWidget, GWT Components, ... Par ailleurs la technologie est telle qu'il est très simple de développer ses propres composants.

Parmi les autres API de Google que faut-il suivre ?

Google Gears est sans doute l'une des API les plus intéressantes pour les développeurs. Elle permet de stocker sur le poste des informations utilisateurs afin de permettre un travail hors ligne ou d'éviter de charger et recharger les mêmes données lors de chaque démarrage d'une application. Un autre projet est à surveiller : GWT Google Apis, il est encore difficile de voir ce que ce projet va donner, mais il semble bien que Google se lance dans une encapsulation systématique de ses API pour les rendre facilement utilisables par les développeurs ne maîtrisant pas ou ne voulant pas faire de javascript.

Et l'avenir ?

Google peut sans doute réapparaître là où il est le moins attendu. Une nouvelle génération de terminaux arrive sur internet avec des navigateurs très puissants. Alors, avec la puissance de GWT, il est



fort probable que l'on retrouve des applications basées sur les technologies Google dans les iPhone, les Ipad Touch ou encore les Wii.

■ **Didier Girard** - Directeur Technique de Sfeir