

# Google Web Toolkit, AJAX facile ?

Google Web Toolkit (GWT) est un *framework*, et un ensemble d'outils, permettant de développer des interfaces Web riches, de type RIA (*Rich Internet Application*), fondées sur AJAX.

GWT propose une approche atypique par rapport à d'autres frameworks AJAX. En effet, le code de l'IHM n'est pas réalisé en JavaScript, mais en Java. Un outil de GWT permet de générer l'IHM à partir du code Java. Ainsi, le code Java n'est pas compilé en *bytecode*, mais transformé en code JavaScript et en HTML.

Le code JavaScript généré est capable de prendre en compte les subtilités d'interprétation de JavaScript par les différents navigateurs du marché : Internet Explorer, Mozilla Firefox, Safari, Opera ... De cette manière, un développeur Java, peu au fait des technologies Web, est capable de développer de manière productive des applications Internet riches.

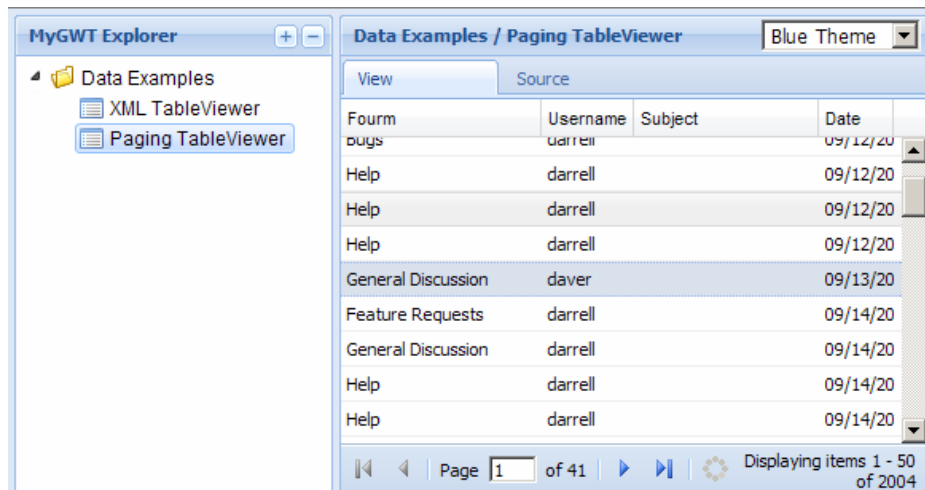


Figure 1: Exemple de rendu avec MyGWT

## La boîte à outils

GWT met à disposition deux outils :

- Java to JavaScript Compiler (JJSC),
- Host Web Browser (HWB).

**JJSC** permet de parser le code Java produit pour générer une interface Web sous forme de multiples fichiers HTML et JavaScript.

Quant à lui, **HWB** est un serveur qui fournit un environnement d'exécution pour le développement. HWB permet de développer des applications GWT de manière productive.

Le cycle de développement se déroule alors de la manière suivante :

- développement en Java avec l'API GWT,
- génération du HTML et du JavaScript,
- déploiement dans le serveur HWB (ou dans un serveur Web).

L'outil HWB permet de simplifier notablement les étapes de génération et de déploiement, jusqu'à les rendre pratiquement transparentes lors du développement.

GWT met également une API à disposition du développeur. Cette API se compose de différents modules fournis en standard. Ces modules fournissent des services tels que :

- l'affichage d'éléments graphiques (*Widget* et *Panel*),
- l'émulation du JRE (*Java Runtime Environment*),
- l'intégration native de JavaScript,
- l'internationalisation,
- les tests unitaires,
- la communication avec le serveur.

Un *Widget* est un composant graphique tel qu'un bouton, ou bien encore une liste déroulante. Un *Panel* est un composant graphique qui peut contenir divers autres composants graphiques tels que des *Widgets* ou encore des *Panels*. Ainsi, le *Panel* permet de structurer et d'organiser l'IHM.

## Le modèle de programmation

Le modèle de développement de GWT est fondé sur une approche programmatique, par opposition à JSP, par exemple, qui est d'avantage focalisé sur un modèle déclaratif à base de balises.

Une application GWT se construit par assemblage de modules GWT. Chaque module a un point d'entrée qui est une simple classe Java.

Grâce à un exemple, nous allons présenter le modèle de programmation de GWT, qui s'appuie sur les modules et les événements. L'exemple consiste en une application simple constituée d'une unique page. Cette page comporte un simple bouton ; un message s'affiche lorsque l'on presse ce bouton.

```
<module>
  <inherits name='com.google.gwt.user.User' />

  <entry-point
    class='com.neoxia.gwtrecipe.init.ui.client.GwtRecipeInit' />

  <stylesheet src='css/default.css' />
</module>
```

### Listing 1 : Déclaration du module GWT

Le Listing 1 présente le fichier XML qui permet de déclarer le module GWT. La balise `entry-point` permet de définir la classe correspondant au point d'entrée du module.

```

public class GwtRecipeInit implements EntryPoint {
    final Label label = new Label();

    public void onModuleLoad() {
        final Button button = new Button("Click me");
        button.addClickListener(new ClickListener() {
            public void onClick(Widget sender) {
                label.setText("Welcome !!!");
            }
        });

        RootPanel.get("slot1").add(button);
        RootPanel.get("slot2").add(label);
    }
}

```

### Listing 2 : Point d'entrée du module GWT

Le Listing 2 présente le code du point d'entrée du module. Une classe associée à un point d'entrée doit implémenter l'interface `EntryPoint` fournie par GWT. La méthode `onModuleLoad()` de cette interface permet de décrire les actions à effectuer lors du chargement du module. Cette méthode construit l'IHM composée de deux *Widgets* : un bouton (`Button`) et un libellé (`Label`).

Un *listener* d'événement est attaché au bouton grâce à la méthode `addClickListener` de la classe `Button`. Il s'agit d'une classe anonyme qui implémente l'interface `ClickListener` et l'unique méthode de cette interface, `onClick`. Ainsi, lorsque ce bouton est pressé, le texte du `Label` est modifié.

Le bouton et le label sont insérés respectivement dans des *panels* racines baptisés `slot1` et `slot2`. Les *panels* racines sont obtenus à partir de la méthode statique `get()` de la classe `RootPanel` qui retourne elle-même un objet `RootPanel`.

```

<html>
  ...
  <table align=center>
    <tr>
      <td id="slot1"></td> <td id="slot2"></td>
    </tr>
  </table>
  ...
</html>

```

### Listing 3 : Page HTML réceptacle du module GWT

Le Listing 3 présente le code HTML de la page réceptacle du module. Deux balises HTML sont munies d'un identifiant, respectivement `slot1` et `slot2`. Ces identifiants sont référencés dans le code du point d'entrée (Listing 2). Ainsi, le bouton et le label sont respectivement insérés dans les balises identifiées par `slot1` et `slot2`.

## La communication entre client et serveur

Dans les applications d'Entreprise, les traitements et les données sont, la plupart du temps, centralisés sur un serveur. Afin d'accéder à ces traitements et ces données, le client, développé en GWT, doit pouvoir communiquer avec le serveur.

GWT s'appuie sur un *pattern* défini pour implémenter une communication asynchrone avec le serveur. Ce pattern suit une approche de programmation par convention. Il s'agit en fait de développer deux interfaces,

l'une utilisée côté client et l'autre côté serveur. L'une et l'autre interface doivent respecter des conventions et des règles de nommage définies.

Grâce à un exemple simple, nous allons montrer l'implémentation d'un service coté serveur et de l'invocation asynchrone de ce service coté client. Le service comporte une méthode qui retourne un message de bienvenue adressé à une personne dont le nom est passé en paramètre.

```
public interface WelcomeService extends RemoteService {
    public WelcomeMessage getMyWelcomeMessage(String name);
}
```

#### Listing 4 : Interface 'coté serveur' du service

Le Listing 4 montre l'interface du service dans sa version 'coté serveur'. Cette interface doit hériter de l'interface GWT `RemoteService` pour indiquer qu'il s'agit de l'interface coté serveur.

```
public class WelcomeMessage implements IsSerializable {
    private String message;
    private Date date;
    ...
}
```

#### Listing 5 : Objet de transfert retourné par le service

Le Listing 5 montre la classe de l'objet de transfert retourné par le service. Cette classe doit implémenter l'interface `IsSerializable` de GWT.

```
public interface WelcomeServiceAsync {
    public void getMyWelcomeMessage(String name, AsyncCallback callback);
}
```

#### Listing 6 : Interface 'coté client' du service

Le Listing 6 montre l'interface du service dans sa version 'coté client', utilisée par le client GWT. Cette interface doit respecter un certain nombre de conventions :

- Le nom de l'interface doit correspondre à celui de l'interface 'coté serveur', suffixé par `Async`. Dans notre cas, l'interface est alors baptisée `WelcomeServiceAsync`.
- L'interface doit avoir les mêmes méthodes que l'interface `WelcomeService`, à quelques différences près, au niveau de la signature.
- Chaque méthode de l'interface doit avoir `void` comme type de retour.
- Chaque méthode de l'interface doit comporter les mêmes paramètres. Cependant, chaque méthode doit comporter un paramètre supplémentaire de type `AsyncCallback`, qui doit figurer en dernière position.

```

public class WelcomeServiceServlet
    extends RemoteServiceServlet
    implements WelcomeService {

    public WelcomeMessage getMyWelcomeMessage(String name) {
        WelcomeMessage result = new WelcomeMessage();
        result.setDate(new Date(System.currentTimeMillis()));
        result.setMessage("welcome " + name);
        return result;
    }
}

```

#### Listing 7 : Implémentation 'côté serveur' du service

Le Listing 7 présente l'implémentation 'coté serveur' du service. D'une part, le service doit implémenter l'interface `WelcomeService` (interface 'coté serveur' du service). D'autre part, l'implémentation du service est en fait une *Servlet* et doit hériter de la classe `RemoteServiceServlet` de GWT. Cette *Servlet* assure l'exposition du service à destination du client.

```

<module>
...
    <servlet path="/welcomeServiceServlet"
        class="com.neoxia.gwtrecipe.rpc.ui.servlet.
        WelcomeServiceServlet"/>
</module>

```

#### Listing 8 : Déclaration de la *Servlet*

Le Listing 8 présente le code ajouté à la déclaration du module pour définir la *Servlet* auprès de GWT. La *Servlet* doit également être déclarée dans le descripteur de déploiement de l'application Web.

```

WelcomeServiceAsync service =
    (WelcomeServiceAsync) GWT.create(WelcomeService.class);

ServiceDefTarget endpoint = (ServiceDefTarget) service;

String moduleRelativeURL =
    GWT.getModuleBaseURL() + "welcomeServiceServlet";

endpoint.setServiceEntryPoint(moduleRelativeURL);

AsyncCallback callback = new AsyncCallback() {
    public void onSuccess(Object objectResult) {
        WelcomeMessage result = (WelcomeMessage) objectResult;
        label.setText(result.getMessage() + ", " + result.getDate());
    }

    public void onFailure(Throwable caught) {
        label.setText("Callback : failure, cause : " + caught);
    }
};

service.getMyWelcomeMessage(textBox.getText(), callback);

```

#### Listing 9 : Invocation asynchrone du service

Le Listing 9 présente le code d'invocation asynchrone du service par le client. Le service est invoqué via l'interface `WelcomeServiceAsync` (l'interface 'coté client' du service). Cette interface est récupérée via la

méthode statique `create()` de la classe `GWT` en passant en argument le type de l'interface `WelcomeService` (l'interface 'coté serveur').

Comme défini par les conventions, la méthode `getMyWelcomeMessage()` de l'interface `WelcomeServiceAsync` prend un argument supplémentaire de type `AsyncCallback`. L'interface `AsyncCallback` permet de notifier, de manière asynchrone, le client, du dénouement de l'invocation. Cette interface comporte deux méthodes :

- La méthode `onSuccess()` est appelée si l'invocation asynchrone réussit, le résultat de l'invocation est alors passé en argument.
- La méthode `onFailure()` est appelée si l'invocation échoue, l'exception à l'origine de l'échec est alors passé en argument.

L'interface `AsyncCallback` est implémentée grâce à une classe anonyme. L'instance de cette classe est passée en argument lors de l'appel de la méthode `getMyWelcomeMessage()` de l'interface `WelcomeServiceAsync`. En cas de réussite, le texte du label est mis à jour pour afficher le résultat ; en cas d'échec, le texte du label est modifié pour refléter l'erreur.

## Pour conclure

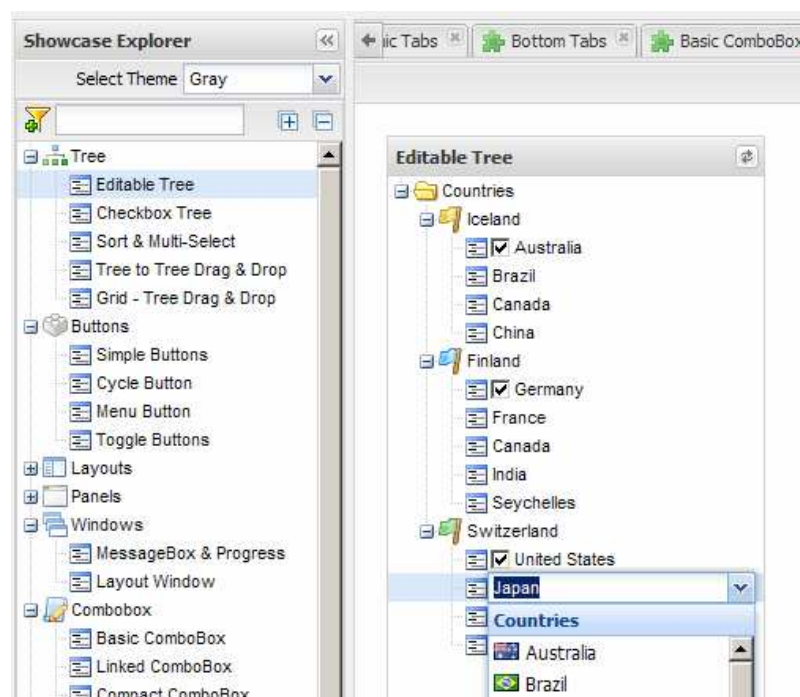


Figure 2 : Exemple de rendu avec GWT-Ext

Contrairement à beaucoup d'autres *frameworks*, GWT offre la possibilité de développer des applications Web riches (RIA), fondées sur AJAX, sans être un expert de JavaScript. Une autre grande force de GWT est la composition de l'IHM à partir de *Widgets*, qui fournit un modèle de réutilisation à base de composants graphiques. À ce jour, il existe d'ailleurs deux importantes bibliothèques de *Widgets*, qui permettent de construire des IHM extrêmement riches et élégantes : MyGWT (voir Figure 1) et GWT-Ext (voir Figure 2). De manière générale, une communauté très active produit de nombreuses extensions et *Widgets*.

Globalement, le bilan de GWT est extrêmement positif. Il n'est cependant pas dénué de certains défauts, tels que l'absence de support de Java 5 (types paramétrés, énumération, boucle `for` améliorée ...), un modèle de programmation pas toujours très POJO, et parfois quelques difficultés d'intégration.

Cependant, la version 1.4 actuelle de GWT est tout à fait opérationnelle, et l'on attend avec impatience les améliorations de la version 1.5.

## Références bibliographiques et Web

- MyGWT
  - ▶ <http://mygwt.net/deploy/dev/explorer/>
- GWT-Ext
  - ▶ <http://www.gwt-ext.com/demo/>

**N. le C.**