Google WebToolKit

Mathieu PASSENAUD
6 septembre 2009

Introduction

Ce document explique

Google WebToolKit est un ensemble d'outils mis à disposition des développeurs pour créer des applications web grâce au langage Java.

La première partie explique brièvement les principes de base pour créer une application et vous familliariser avec l'outil.

La deuxième partie explique le mode de communication entre le serveur et le navigateur du client grâce à la technologie AJAX.

La troisième partie expose les diverses améliorations et optimisations qu'apportent le web toolkit aux applications web.

L'auteur

Etudiant en informatique, passionné de nouvelles technologies web.

Pré-requis

Une connaissance assez poussée du langage Java et de tous les concepts de la programmation orientée objet.

Quelques notions de HTML.
Un éditeur de texte, ou un environnement de développement (Eclipse).

Première partie

Présentation de WebToolKit

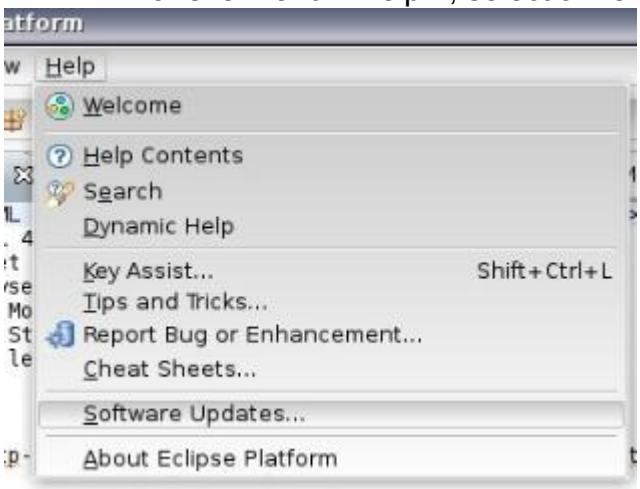
Si vous programmez déjà, surtout en Java, vous avez constaté qu'il est possible et aisé de faire des interfaces graphiques évoluées, créées dynamiquement avec une gestion des événements sur chacun des éléments.

Avec WebToolKit, Google a mit à disposition des utilisateurs les mêmes concepts mais avec la technologie web. C'est à dire que vous programmez entièrement en Java, en utilisant les principes de la programmation graphique sauf qu'elle sera appliquée à une page web et que le tout pourra s'exécuter dans n'importe quel navigateur web. Le programme que vous avez écrit en Java est interprété et transformé en une série de codes html, javascript et autres. Je ne m'attarderai pas sur cette partie, il y a déjà énormément à voir sur WebToolKit pour en tirer le meilleur.

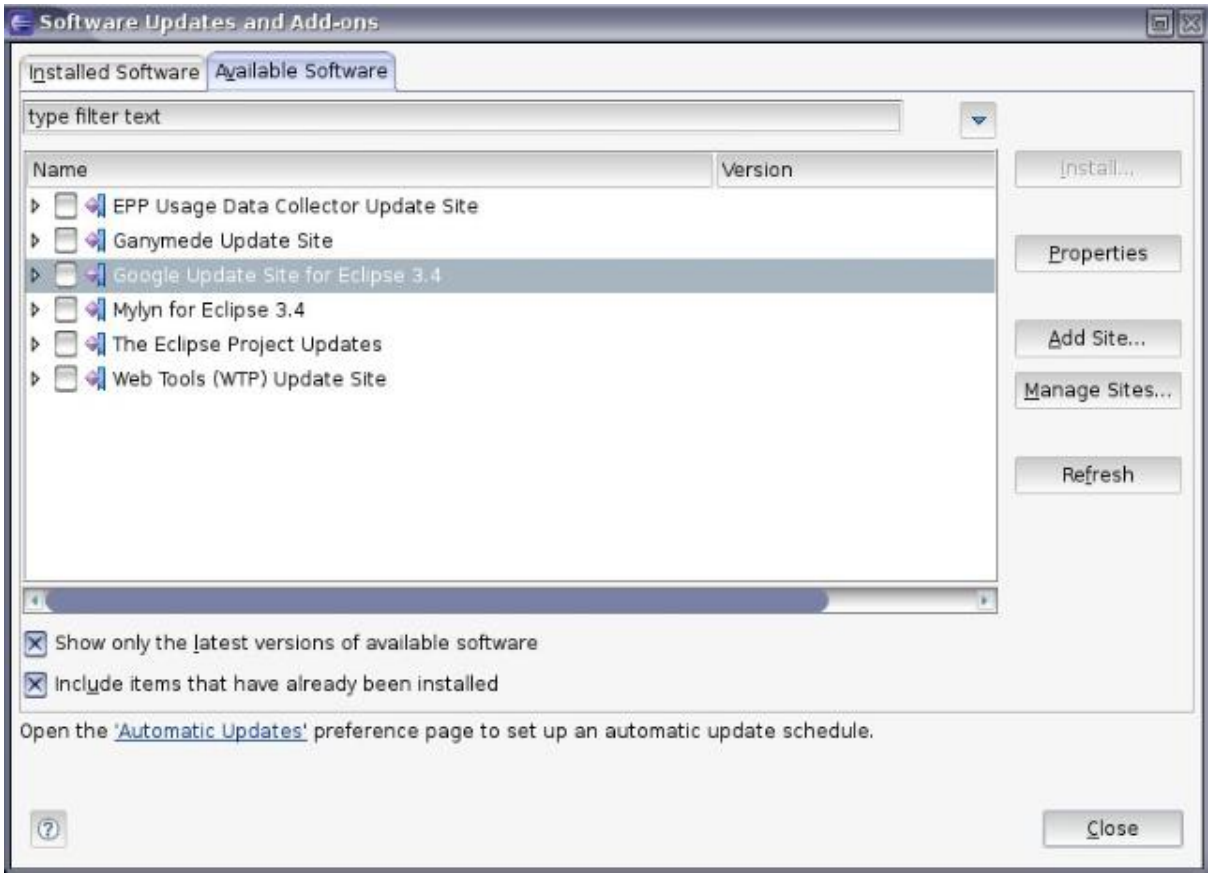
Installation

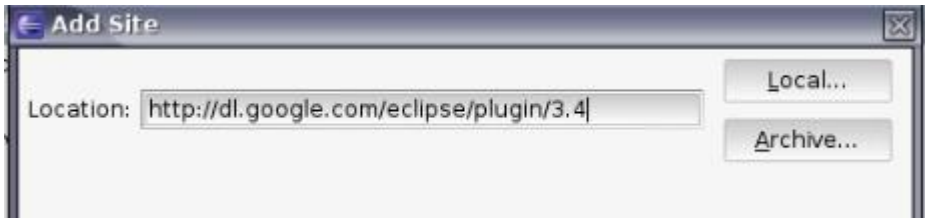
Pour ma part, j'utilise directement le plugin pour Eclipse. Pour l'installer, reportez-vous à la documentation fournie par Google. Il s'agit juste de l'ajout d'un plugin pour Eclipse.

Dans le menu « help », sélectionnez « Software Updates... ».



Cliquez ensuite sur «Add site » dans la fenêtre.

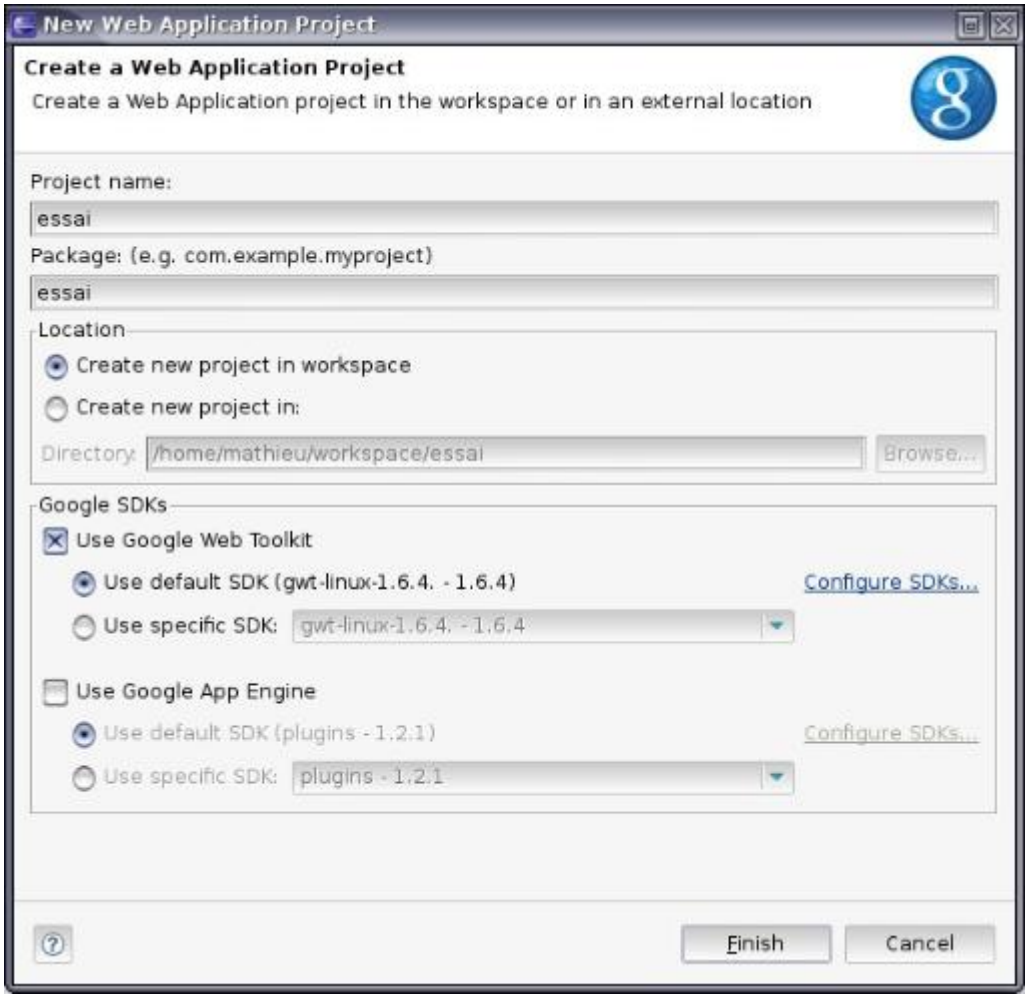




Dans la boîte de dialogue, entrez l'adresse de téléchargement du plugin, cette adresse est proposée par Google : http://dl.google.com/eclipse/plugin/3.4
Normalement, une fois l'opération terminée, vous devez avoir trois icônes supplémentaires en haut à gauche de la fenêtre principale d'Eclipse.

Architecture du projet

Quand vous créez un nouveau projet (en cliquant sur le bouton bleu), sélectionnez de créer une application uniquement avec le WebToolkit (l'appEngine a été vu précédement dans un autre document).



Le premier projet sera nommé « essai ». Une fois le projet créé, vous remarquerez que vous avez déjà une application fonctionnelle mais quelque peu complexe.

Pour une première fois avec WebToolKit, je vous ferai tout reprendre de 0.

Le répertoire « src » contient les sources en Java qui constituent le programme général.
Le répertoire « war » contient les pages web fixes qui servent de base à la création de l'application.
Le répertoire « WEB-INF » contient les fichiers de configuration de l'application. Nous les verrons un peu plus loin dans ce document.

La page web de base

Dans un projet de ce type, il y a au moins une page web de base, en html dans laquelle il y a des éléments qui contiendront les divers objets qui seront ajoutés avec le programme Java.

La page « essai.html » contient déjà des éléments. Allez à la fin et enlevez tout entre « <h1> » et « </table> ». Je vais vous faire ajouter des éléments au fur et à mesure pour bien comprendre comment cela fonctionne.
Insérez le code suivant dans la page web, à la place de ce que vous venez de supprimer :

```
<table border=0>
	<tr>
		<td id= "case1">
		</td>
		<td id="case2">
		</td>
	</tr>
</table>
```

Les deux cases (« case1 » et « case2 » serviront de « conteneur » pour des éléments actifs que nous ajouterons dans le programme Java.

Le programme Java

Dans le paquetage « essai.client », éditez le fichier « essai.java ». Supprimez l'intégralité des choses après la ligne 40 (déclaration de la méthode « onLoad »), en laissant bien deux accolades à la fin du fichier, sans trop regarder ce qu'il y a dedans afin de bien comprendre la suite :-).

Tout ce que vous écrirez en Java sera retransformé en JavaScript par le moteur applicatif fourni par Google. Tout comme en programmation graphique avec Swing, vous utiliserez des éléments pré-définis avec une gestion des événements.

Pour lancer l'application, faites un clic droit sur le projet puis « Run As » et « Web application ». Le serveur est lancé avec un navigateur dans lequel s'affiche l'application. Inutile de fermer et d'arrêter le serveur dès que vous modifiez l'application, cliquez simplement sur le bouton « Restart Server » pour relancer le serveur.

Création d'un programme de base

Dans la méthode « onLoad » de la classe « Essai », ajoutez ces deux lignes de code en essayant de les comprendre :

```
final Button monBouton = new Button("Mon bouton");
RootPanel.get("case1").add(monBouton);
```

Cela va créer un bouton et l'ajouter à la page web à l'emplacement récupéré grâce au RootPanel. Simple non ?

Vous pouvez continuer avec d'autres cases et divers éléments mis à disposition tels que :

- ListBox
- CheckBox
- FileUpload
- Hyperlink
- Image
- TextBox
- RichTextArea
- ToggleButton
- ...

Ils sont tous disponibles dans le paquetage com.google.gwt.user.client.ui.

La programmation événementielle

Sur chaque élément que vous ajoutez, vous pouvez y appliquer un écouteur qui réagira en fonction du type d'évènement.

Revenez à la base avec l'unique bouton dans la première case. La deuxième case servira pour afficher un message au moment où on cliquera sur le bouton.

Un écouteur n'est pas un « Listenner » comme en programmation avec Swing, mais un « Handler ». A part cela, l'utilisation est exactement identique.

Sur le bouton « monBouton », ajoutez un « ClickHandler » que je nomme « MyClickHandler ». Pour cela, il suffit d'écrire :

```
monBouton.addClickHandler(new MyClickHandler());
```

A la fin de votre classe (ou dans un autre fichier), créez la classe « MyClickHandler » qui implémente l'interface ClickHandler avec toutes ses méthodes (comme avec Swing).

```
class MyClickHandler implements ClickHandler {
	@Override
	public void onClick(ClickEvent event) {
		//la méthode se déclenche uniquement sur clic
		RootPanel.get("case2").add(new HTML("Clic sur le bouton");
	}
}
```

Relancez le serveur et admirez... La page n'est pas rechargée et le texte HTML est bien ajouté à chaque clic sur le bouton.

Vous diposez d'une multitude d'écouteurs, en fonction du type d'objet dont vous pourrez partie.

Le gestionnaire d'affichage

Tout comme avec Swing, vous pouvez utiliser des « layout » pour la mise en forme des différents éléments que vous ajoutez dans une page web.
Voiçi ceux que j'ai listés :

- DockPanel
- AbsolutePanel
- CaptionPanel
- CellPanel
- ComplexPanel
- DeckPanel
- DecoratorPanel
- DecoratedStackPanel
- DecoratedPopupPanel
- DecoratedTabPanel
- DisclosurePanel
- FlowPanel
- FocusPanel
- FormPanel
- HorizontalPanel
- HorizontalSplitPanel
- HTMLPanel
- LazyPanel
- Panel
- PopupPanel
- ScrollPanel
- SimplePanel
- StackPanel
- TabPanel

- VerticalPanel
- VerticalSplitPanel

Dans un emplacement de la page, ajoutez le « panel » qui vous convient. Les éléments sont ajoutés directement sur le « panel » et non sur la page (comme avec Swing).

Un gestionnaire d'affichage peut en contenir d'autres.

Les boîtes de dialogue

Il est possible grâce à WebToolKit de créer des boîtes de dialogue directement dans la page du navigateur. Ces boîtes sont entièrement codées en Javascript. Elles peuvent être animées.

L'objet utilisé est une « DialogBox » sur laquelle on définit :

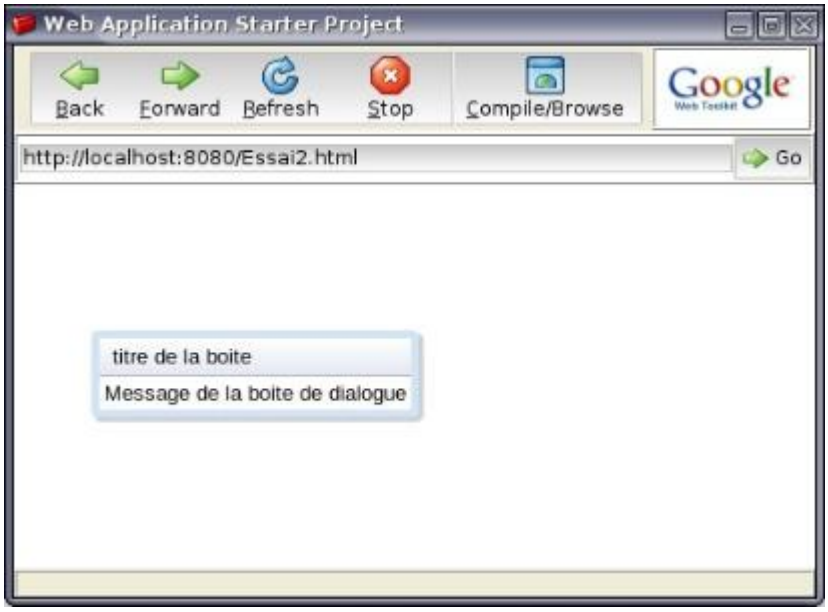
- un titre
- un panel contenant des éléments

Il existe bien sûr tout un ensemble d'options applicables sur une boîte de dialogue. La méthode « hide() » permet de masquer une boîte de dialogue.

Voiçi un exemple qui créé une boîte de dialogue simple avec un message.

```
final DialogBox boiteDeDialogue = new DialogBox();
boiteDeDialogue.setText("titre de la boite");
boiteDeDialogue.setAnimationEnabled(true);
VerticalPanel panel = new VerticalPanel();
panel.add(new HTML("Message de la boite de dialogue"));
panel.setHorizontalAlignment(VerticalPanel.ALIGN_RIGHT);
boiteDeDialogue.setWidget(panel);
boiteDeDialogue.center();
```

Le résultat obtenu est le suivant :



Création d'une application complète

Dans cet exemple, nous allons créer une application qui vous permettera de réviser vos facultés de calcul mental...

Le but est de tirer deux nombres au hasard, de les multiplier et de demander à l'utilisateur de trouver le résultat. La page HTML ne devra jamais être rechargée.

Pour commencer, créez un nouveau projet dans Eclipse en utilisant uniquement le WebToolKit (et non AppEngine).

La page web de base

Modifiez la structure de la page pour qu'elle ressemble à ceci :

Le tableau est composé de 4 cellules, une qui comporte l'id « multi1 », une autre pour le signe de l'opération, la troisième qui s'appelle « multi2 » et la dernière qui servira pour le résultat. Elle s'appelle donc « resultat ». Une zone de texte y sera insérée, quand l'utilisateur appuiera sur la touche entrée cela validera le résultat.

Au bout de 10 opérations, les résultats seront affichés dans une boîte de dialogue.

Positionnement des éléments

Il faut insérer seulement une zone de texte dans la partie « resultat » ainsi que les deux nombres à multiplier qui seront tirés au hasard. On utilise l'objet « RootPanel » sur lequel on applique la méthode « get() » et on y positionnement l'élément voulu (« HTML » pour les deux nombres, « TextBox » pour le résultat).

Dans la classe principale (multipli.java dans mon cas), supprimez intégralement le contenu de la méthode « onModuleLoad() » ainsi que la classe d'écouteur après celle-ci.

Créez la zone de texte avec le constructeur « new TextBox() ». L'objet est appelé « resultat » dans mon exemple. La zone est redimensionnée à 30 pixels de large avec la méthode « setWidth ». Attention, mettez entre guillemets la valeur, seule une chaine de caractères est acceptée par cette méthode. Ajoutez la zone de texte à la page de base avec la ligne suivante :

```
RootPanel.get("resultat").add(resultat);
```

Cela doit vous donner quelque chose de ce genre :

Nous allons créer une méthode « setNumbers » qui va prendre en argument deux entiers qui seront positionnés respectivement dans les cases « multi1 » et « multi2 ». Cette méthode sera appelée une fois que les deux nombres

seront tirés au hasard. Attention à bien retirer les éléments déjà présents avant d'en ajouter de nouveaux !
Voiçi ce à quoi elle doit ressembler :

```
public static void setNumbers(int multi1, int multi2){
		RootPanel.get("multi1").clear();
		RootPanel.get("multi2").clear();
		RootPanel.get("multi1").add(new HTML("<b>"+multi1+"</b>"));
		RootPanel.get("multi2").add(new HTML("<b>"+multi2+"</b>"));
	}
```

Création de l'écouteur

Il faut positionner un écouteur sur la zone de texte pour savoir quand l'utilisateur aura appuyé sur la touche entrée. Cet écouteur est du type « KeyPressHandler ».

Ajoutez la ligne suivante pour appliquer l'écouteur à l'objet :

```
resultat.addKeyPressHandler(new Ecouteur());
```

La classe « Ecouteur » est créée dans le même fichier pour plus de commodité :

```
class Ecouteur implements KeyPressHandler{
	@Override
	public void onKeyPress(KeyPressEvent event) {
		if(event.getCharCode()==KeyCodes.KEY_ENTER){
			Noyau.compare(new Integer(Multipli.resultat.getValue()));
			Multipli.resultat.setValue("");
		}
	}
}
```

Il suffira de remplir dans le « if » plus tard pour déclencher les actions.

Une classe supplémentaire

Il nous faut absolument une autre classe dans laquelle sera gérée l'application. J'appelle cette classe « Noyau ». Elle est mise dans le même paquetage que la classe principale.

Les opérations sont toutes mémorisées dans une « ArrayList » sous forme d'une seule chaîne de caractères. Ces chaînes pourront être réutilisées plus tard pour avertir l'utilisateur là où il s'est trompé.

Il faut aussi deux autres variables entières qui ne sont autres que les deux nombres à multiplier. Une dernière variable entière nous permettera de compter le nombre d'opérations effectuées pour s'arrêter au bout de la dixième.

La première étape consiste à vérifier si on ne dépasse pas les 10 opérations. Ensuite, on tire au sort deux nombres au hasard. La classe « Math » contient une méthode « Random » qui retourne un nombre au hasard entre 0 et 1. Ce nombre multiplié par 10 et transtypé en entier fera l'affaire.

```
multi1=(int) (Math.random()*10)+1;
```

Une fois les deux nombres tirés, on incrémente la variable qui contient le nombre d'opérations et on envoie les deux nombres à l'interface pour les afficher.

Une dernière méthode est appelée une fois que l'utilisateur a validé. Le nombre entré par l'utilisateur est passé en argument. Une comparaison est effectuée, si le résultat est bon on passe à l'opération suivante, sinon on mémorise le calcul dans la liste et on passe à l'opération suivante.

Si on dépasse les 10 opérations, on appelle une méthode de la classe « Multipli » qui affichera la boite de dialogue avec les calculs dont le résultat a été erroné et les variables seront ré-initialisées.

La boite de dialogue

Grâce à une méthode à laquelle on passera la liste des opérations erronées, on prendra chaque entrée de la liste que l'on affichera dans une boite de dialogue personnalisée. Une boucle « for » va parcourir la liste et afficher chaque calcul qui a été mémorisé.

Il faudra aussi penser à ajouter un bouton à la boite de dialogue qui nous permettra de la fermer. Le bouton aura son propre écouteur qui fermera la boite de dialogue avec la méthode « hide() ».

Le code complet

Classe Noyau.java

```
package multipli.client;

import java.util.ArrayList;

public class Noyau {
	private static int multi1;
	private static int multi2;
	private static ArrayList<String> operations = new ArrayList<String>();
	private static int enCours=0;

	public static void tirerNombres(){
		if (enCours>10){
			Multipli.resultats(operations);
			enCours=0;
			operations = new ArrayList<String>();
			tirerNombres();
		}else{
			multi1=(int) (Math.random()*10)+1;
			multi2=(int) (Math.random()*10)+1;
			Multipli.setNumbers(multi1, multi2);
```

```
                        enCours++;
                }
        }
        public static void compare(int resultat){
                if(resultat==(multi1*multi2)){
                        tirerNombres();
                }else{
                        operations.add(multi1+" x "+multi2+" = <b>"+multi1*multi2+"</b>");
                        tirerNombres();
                }
        }
}
```

Classe Multipli.java

```
package multipli.client;
import java.util.ArrayList;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.event.dom.client.KeyCodes;
import com.google.gwt.event.dom.client.KeyPressEvent;
import com.google.gwt.event.dom.client.KeyPressHandler;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
public class Multipli implements EntryPoint {

        public static final TextBox resultat = new TextBox();
        public static DialogBox boiteDeDialogue;
        private static final String SERVER_ERROR = "An error occurred while "
                        + "attempting to contact the server. Please check your network "
                        + "connection and try again.";
        private final GreetingServiceAsync greetingService = GWT
                        .create(GreetingService.class);
        public void onModuleLoad() {
                resultat.setWidth("30");
                RootPanel.get("resultat").add(resultat);
                resultat.addKeyPressHandler(new Ecouteur());
                Noyau.tirerNombres();
        }
        public static void setNumbers(int multi1, int multi2){
                RootPanel.get("multi1").clear();
```

```
                RootPanel.get("multi2").clear();
                RootPanel.get("multi1").add(new HTML("<b>"+multi1+"</b>"));
                RootPanel.get("multi2").add(new HTML("<b>"+multi2+"</b>"));
        }
        public static void resultats(ArrayList<String> erreurs){
                boiteDeDialogue = new DialogBox();
                boiteDeDialogue.setText(erreurs.size() + " erreurs");
                boiteDeDialogue.setAnimationEnabled(true);
                boiteDeDialogue.setModal(true);
                VerticalPanel panel = new VerticalPanel();
                for(int i=0; i<erreurs.size(); i++){
                        panel.add(new HTML(erreurs.get(i)+"<br />"));
                }
                panel.setHorizontalAlignment(VerticalPanel.ALIGN_RIGHT);
                boiteDeDialogue.setWidget(panel);
                boiteDeDialogue.center();
                final Button boutonFermer = new Button("Fermer");
                panel.add(boutonFermer);
                boutonFermer.addClickHandler(new Click());
        }
}
class Ecouteur implements KeyPressHandler{
        @Override
        public void onKeyPress(KeyPressEvent event) {
                if(event.getCharCode()==KeyCodes.KEY_ENTER){
                        Noyau.compare(new Integer(Multipli.resultat.getValue()));
                        Multipli.resultat.setValue("");
                }
        }
}
class Click implements ClickHandler{
        @Override
        public void onClick(ClickEvent event) {
                Multipli.boiteDeDialogue.hide();
        }
}
```

Deuxième partie

L'intérêt d'une application web est aussi de pouvoir sauvegarder des données, comme si l'utilisateur était sur une machine personnelle avec son espace de stockage. Dans les applications web, le stockage se fait sur un serveur distant. Il est nécessaire de faire communiquer l'application (exécutée dans le navigateur) avec le serveur applicatif.

La communication ne doit pas se faire de manière traditionnelle avec des formulaires HTML dont on envoi les données avec la méthode GET ou POST à une page PHP ou JSP. Elle se fait continuellement sans avoir à recharger la page web (avec la technologie AJAX). Si vous utilisez Google Maps, Google Documents ou une autre application, vous remarquerez que la page n'est jamais rechargée. Les éléments évoluent dynamiquement et les appels au serveur sont effectués selon les actions de l'utilisateur.

Des langages différents

Avec le Web Toolkit, l'application est divisée en deux parties :
- la partie client, transformée en JavaScript est exécutée dans le navigateur web ;
- la partie serveur, compilée en byteCode est exécutée sur le serveur, dans un serveur applicatif Google spécifique livré avec le Web Toolkit.

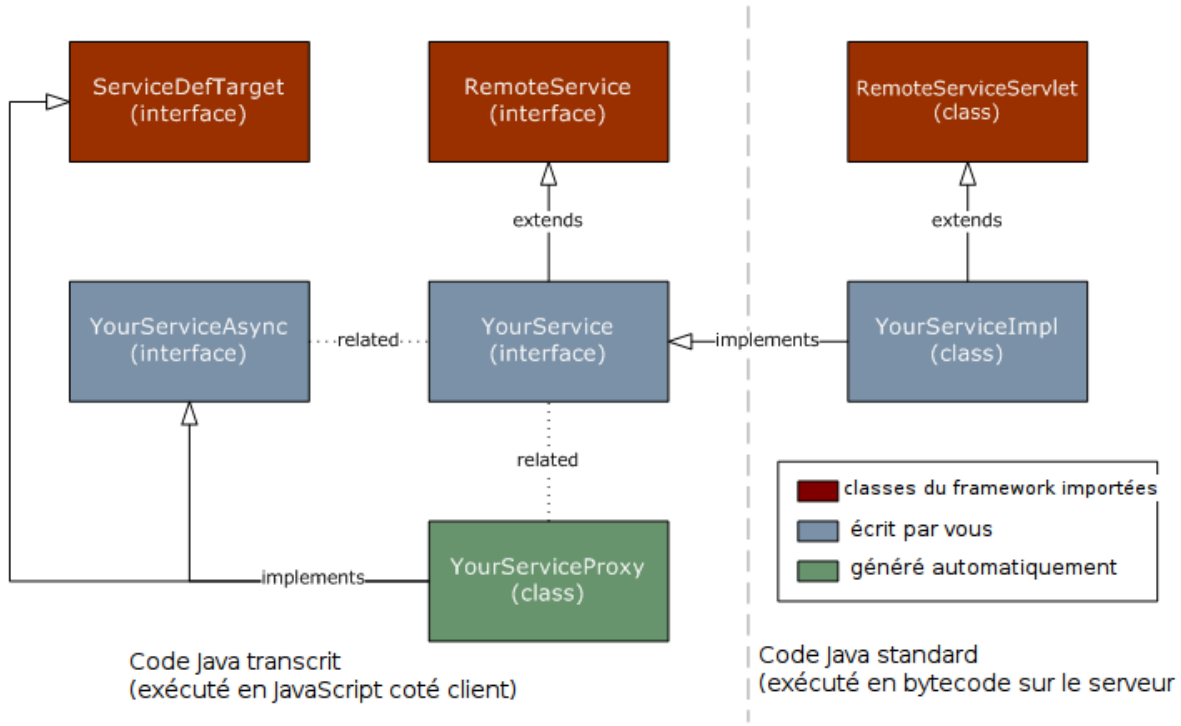
Avec le Web Toolkit, les procédures d'appel (RPC, Remote Procedure Call) sont basées sur les servlets Java, le serveur applicatif exécutant un programme Java comme serveur.

Le même projet, mais deux utilisations différentes

La partie client se trouve dans les paquetages nommés **client**. Toute cette partie est entièrement transformée en JavaScript pour être exécutée par le navigateur web. Il est donc facile de comprendre que les possibilités sont limitées et qu'il n'est pas question d'utiliser des librairies Java dont on sait que l'utilisation sera impossible une fois transformé en JavaScript (par exemple la sérialisation n'est pas disponible).

La partie serveur (dans les paquetages **server**) est quand à elle exécutée sur le serveur directement dans la machine virtuelle Java. Donc l'intégralité des possibilités du langage Java sont conservées ainsi que quelques ajouts concoctés par Google pour permettre la communication avec l'application client en JavaScript.

Principe



(schéma traduit de http://code.google.com/webtoolkit/doc/1.6/DevGuideServerCommunication.html)

Une petite explication et un exercice pratique s'imposent.

La communication se fait grâce à des services. Google a créé une petite astuce en réutilisant un concept présent dans le langage Java : les interfaces.
Une interface est présente coté client, l'implémentation de cette interface est faite dans une classe exécutée cotée serveur. Pour la communication ? Il se débrouille, vous n'avez rien à faire.

Deux interfaces doivent être créées, l'une pour la communication synchrone et l'autre pour la communication asynchrone. Au delà, je ne sais absolument pas comment le système gère cet aspect en interne.

Une interface (coté client) hérite de l'interface RemoteService. Quant à la classe, elle hérite de RemoteServiceServlet et elle implémente l'interface client.

Cette vision des choses permet de comprendre le fonctionnement très rapidement sans devoir apprendre de nouveaux concepts ni de nouveaux langages. L'astuce de programmation utilisée par Google est très simple et tout bon programmeur en Java n'aura aucun mal à s'adapter à cet aspect.

Deux interfaces, une classe

Attention, suivez bien car cet aspect est à la fois simple mais aussi très surprenant. Une interface coté client décrit la classe coté serveur (en héritant de l'interface RemoteService). Les méthodes et leurs retours sont spécifiés.

Il faut aussi créer une deuxième interface, très ressemblante à la précédente. En effet, les retours ne peuvent pas se faire comme si le programme était exécuté en local, il faut toute une procédure en interne pour transférer les objets sur le réseau. Cette interface est dite "Asynchrone" et elle comporte les mêmes méthodes avec le retour void et un objet supplémentaire du type AsyncCallback dans lequel le retour sera inséré par le système. Aucune classe dans le système implémente cette interface.

Dans une classe client, il faut créer l'objet de communication asynchrone en invoquant la méthode create() sur l'objet GWT (exemple : GWT.create(monConnecteur.class) me retourne un objet du type monConnecteurAsync).
Ensuite, pour appeler une méthode et récupérer le résultat :

```
monObjetAsynchrone.methode(argument, new AsyncCallback<int>(){
public void onSuccess(int result) {
    // si la connection réussit
   }

   public void onFailure(Throwable caught) {
    // si la connection échoue
   }
});
```

La classe coté serveur reste classique, elle peut utiliser toutes les possibilités offertes par la machine virtuelle Java.

Des noms formattés

Le GWT est exigent concernant les noms attribués à chaque classe et interface. L'interface porte le nom que l'on veut, l'interface asynchrone porte le suffixe "Async" et la classe qui implémente porte le suffixe "Impl".
Astuce : servez-vous de la force d'eclipse ! Quand vous créez votre interface, eclipse vous proposera et vous générera la classe asynchrone associée en adaptant le nom et les arguments des méthodes

Une déclaration nécessaire

Il faudra adapter le web.xml pour lui déclarer les ressources (ce fichier est analysé et interprété par Jetty).
Il faut déclarer la servlet ainsi que la ressource web qui appelera la servlet (l'url).

```
<servlet>
  <servlet-name>monServiceImpl</servlet-name>
  <servlet-class>
    appli.server.monServiceImpl
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>monServiceImpl</servlet-name>
  <url-pattern>/appliweb/monService</url-pattern>
</servlet-mapping>
```

Prenez toujours garde aux noms que vous donnez, le GWT est exigent sur les noms des classes et interfaces.

Un exemple de sauvegarde des instances

Avant de se lancer, il faut réfléchir à la solution qui est la moins coûteuse en bande passante et en ressources. L'idéal reste de gérer les objets en local (dans le navigateur chez le client) et de transmettre juste les objets à sauvegarder au serveur.

Toutes les classes sont définies dans le paquetage client, et un objet qui va gérer les sauvegardes sur disque sera instancié coté serveur. Le client transmettera les objets à sauvegarder directement en appelant une méthode (par exemple save(monObjet)) et le serveur fera une simple sérialisation de son coté. Simple non ?

Pour stocker des données coté client, deux solutions sont possibles :
- proposer à l'utilisateur de télécharger un fichier généré
- utiliser le plugin Google gears (article à venir)

Troisième partie

Une application web doit être simple à utiliser, épurée mais aussi très rapide. Les terminaux sont de plus en plus puissants certes, mais les 3ghz que l'on a dans nos pc ne sont pas encore dans nos téléphones.

De plus les connections internet doivent être utilisées à juste dose, ce n'est pas parce qu'elle est rapide qu'il faut en abuser. Elle reste infiniment plus lente qu'un traitement en local.

La gestion des navigateurs

Vous n'avez rien à faire, le web toolkit adapte son application selon le navigateur qui l'exécute. En effet, Google chrome est le plus rapide à exécuter le javascript, Firefox le navigateur le plus utilisé du web, internet explorer ne remplit pas (encore) les standards de la W3C... il faut donc réussir à contenter tout le monde.

La certitude que vous pouvez avoir, c'est que quelque soit le navigateur web (récent) utilisé, votre application fonctionnera sans aucun problème.

Les images

Avant de vous expliquer, livrez-vous à une petite expérience :
- Allez sur http://www.google.fr
- faites une recherche quelconque
- enregistrez la page de résultat avec votre navigateur sur votre disque dur
- allez dans le dossier qui contient les diverses ressources de la page enregistrée sur votre disque dur (en effet, quand vous enregistrez une page sur votre disque dur, vous avez la page html et un dossier qui contient les divers éléments tels que fichiers javascript ou images).

Normalement, vous devriez trouver plein de petites images :
les petits éléments de search wiki (si disponible), le logo google, les boutons pages suivantes et précédentes etc...

- surprise, vous n'avez pas une multitude d'images, mais une seule qui les regroupe toutes :

Le Web toolkit permet de regrouper en une seule image toutes les images de la page web. Quel en est l'intérêt ? Vous ne faites qu'une seule demande au serveur, le transfert est réalisé d'un seul bloc et les images sont ensuite traîtées en local. Un temps précieux peut être gagné.

Le type ImageBundle permet cette performance et ce processus de regroupement des images en une seule.

Les extensions disponibles sont .gif, .jpg ou .png

Pour plus d'infos : http://code.google.com/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideImageBundles

L'optimisation du code

Même en ayant des méthodes de programmation mauvaises, en utilisant les mauvaises boucles, le compilateur est capable d'optimiser au maximum le code javascript pour en tirer les meilleures performances. Si le plugin Google Gears est installé, l'améliorateur d'exécution fonctionnera aussi.

La communication client/serveur

Tout comme le reste, les appels sont optimisés. Comme ils sont asynchrones, il est assez aisé pour le logiciel de regrouper plusieurs appels en un seul pour optimiser au mieux les transferts.

Il faudra tout de même prendre toutes les précautions nécessaires lors de la programmation pour éviter des transferts incessants entre le serveur et le client. Le client dispose de ressources matérielles, peut être plus que les serveurs des fois. Cet aspect est aussi à prendre en compte.