82. GWT (Google Web Toolkit)

Niveau :  Supérieur

GWT (Google Web Toolkit) est un framework open source de développement d'applications web mettant en oeuvre AJAX et développé par Bruce Johnson et Google.

Mi−2006, Google a diffusé GWT qui est un outil de développement d'applications de type RIA offrant une mise en oeuvre novatrice : le but est de faciliter le développement d'applications web mettant en oeuvre Ajax en faisant abstraction des incompatibilités des principaux navigateurs.

GWT propose de nombreuses fonctionnalités pour développer une application exécutable dans un navigateur et présetnant des comportements similaires à ceux d'une application desktop :

- création d'applications graphiques s'exécutant dans un navigateur
- pas besoin d'écrire du code Javascript sauf pour des besoins très spécifiques comme l'intégration d'une bibliothèque JavaScript existante
- utilisation de CSS pour personnaliser l'apparence
- mise en oeuvre d'Ajax sans manipuler l'arbre DOM de la page mais en utilisant des objets Java
- un ensemble riche de composants (widgets et panels)
- communication avec le serveur grâce à des appels asynchrones en échangeant des objets Java et en utilisant des exceptions pour signifier des problèmes
- internationalisation
- un système de gestion de l'historique sur le navigateur
- un parser XML
- détection des erreurs à la compilation
- ...

L'utilisation de GWT présente plusieurs avantages :

- pas de code JavaScript à écrire
- utilisation de Java comme langage de développement
- une meilleure productivité liée à l'utilisation du seul langage Java (un seul langage à utiliser, mieux connu que d'autres technologies notamment JavaScript, mise en oeuvre d'un débogueur, utilisation d'un IDE Java, ...)
- hormis les styles CSS et la page HTML qui encapsule l'application, il n'y a pas d'utilisation directe de technologies web
- le code généré par GWT supporte les principaux navigateurs
- la prise en main est facile même pour des débutants ce qui lui confert une bonne courbe d'apprentissage

Le site officiel de GWT est à l'url http://code.google.com/webtoolkit/

Ce chapitre contient plusieurs sections :

- Les composants (widgets)
- Les panneaux (panels)
- La création d'éléments réutilisables
- Les événements
- JSNI
- La configuration et l'internationalisation
- L'appel de procédures distantes (Remote Procedure Call)
- La manipulation des documents XML
- La gestion de l'historique sur le navigateur
- Les tests unitaires
- Le déploiement d'une application
- Des composants tiers
- Les ressources relatives à GWT

82.1. La présentation de GWT

Le code de l'application est entièrement écrit en Java notamment la partie cliente qui devra s'exécuter dans un navigateur. Ce code Java n'est pas compilé en bytecode mais en JavaScript ce qui permet son exécution dans un navigateur.

Le coeur de GWT est donc composé du compilateur de code Java en JavaScript. L'avantage du code JavaScript produit est qu'il est capable de s'exécuter sur les principaux navigateurs sans adaptation particulière du source Java puisque le compilateur crée un fichier JavaScript optimisé pour chacun de ces navigateurs.

Le code à écrire pour la partie cliente est composé de plusieurs éléments :

- la syntaxe est celle de Java 1.4 (les fonctionnalités de Java 5 sont supportées à partir de la version 1.5 de GWT)
- un sous−ensemble des API de bases du JDK notamment des packages java.lang et java.util (particulièrement les classes qui pourront être compilées en JavaScript. La liste complète de ces classes est consultable à l'url http://www.gwtproject.org/doc/latest/RefJreEmulation.html)
- un ensemble de composants graphiques nommés widgets et de panels qui sont utilisés pour réaliser l'interface graphique

La partie graphique d'une application GWT est composée d'une petite partie en HTML, de CSS et surtout de classes Java dans lesquelles des composants sont utilisés avec des gestionnaires d'événements pour définir l'interface de l'application et les réponses aux actions des utilisateurs.

GWT propose un ensemble assez complet de composants graphiques nommés widgets fournis en standard : l'ensemble des widgets inclut des composants graphiques standards (boutons, zones de saisie de texte, listes déroulantes, ...) mais contient aussi des composants plus riches tels que des panneaux déroulants, des onglets, des arbres, des boîtes de dialogues, ... Il est aussi possible de créer ses propres composants ou d'intégrer des frameworks JavaScript (ext, Dojo, Rialto, Yahoo UIL, ...)

Le code Java pour développer en GWT est très ressemblant à celui à produire pour développer une application graphique utilisant AWT :

- instancier des composants
- ajouter ces composants dans la hiérarchie des composants de la page
- utiliser des gestionnaires d'événements pour répondre aux actions des utilisateurs

Exemple :

```
public class TestBonjour implements EntryPoint {
    public void onModuleLoad() {
        Button bouton = new Button("Saluer", new ClickListener() {
            public void onClick(Widget sender) {
                Window.alert("Bonjour");
            }
        });
        RootPanel.get().add(bouton);
    }
}
```

GWT propose aussi des outils pour assurer la communication avec la partie serveur en se reposant sur AJAX et offre aussi un support de JUnit.

Ce framework propose plusieurs originalités intéressantes :

- développement majoritairement en Java et un peu d'HTML : le code JavaScript est généré par le compilateur GWT
- développée en Java, une application GWT est plus facile à déboguer en utilisant un IDE Java
- GWT fournit de nombreux composants (widgets) et un système de gestion de leur positionnement (Layout)
- l'application GWT gère le support des principaux navigateurs
- ...

Le grand avantage de GWT est que l'application web utilisant Ajax et développée avec ce framework ne nécessite essentiellement que des connaissances en Java : quelques rudiments d'HTML et CSS sont nécessaires pour développer des applications GWT mais aucun code JavaScript n'est à écrire.

Une application GWT peut être écrite avec un des IDE Java : ceci rend l'application facilement débogable avec les fonctionnalités de l'IDE.

La version 1.4 de GWT repose sur Java 1.4.

La version 1.5 de GWT repose sur Java 1.5 et offre un support des fonctionnalités de Java 5 (énumérations, generics, ...)

Une application GWT est contenue dans un module. Un module est un ensemble de classes et un fichier de configuration. Un module possède un point d'entrée (entry point) qui correspond à la classe principale qui sera utilisée au lancement de l'application.

Côté serveur, il est possible d'utiliser toutes les technologies capables de traiter des requêtes http (Java, .Net, PHP, ...). Java est particulièrement bien adapté à cette tâche grâce à ses nombreuses API et frameworks disponibles.

Une application de gestion développée avec GWT est donc composée de classes Java :

- pour la partie IHM : les classes Java sont compilées en JavaScript pour permettre l'exécution de l'application dans un navigateur
- pour la partie serveur : les classes assurent les traitements métiers et la persistance de données.

Remarque : la partie serveur n'a pas d'obligation à être développée en Java. Elle peut être développée avec d'autres plate−formes mais GWT offre des facilités pour l'utilisation de Java

Lors du déploiement, l'application web sera générée à partir du code Java pour produire les codes HTML et JavaScript requis pour la partie cliente.

Une application GWT peut être exécutée dans deux modes :

- mode hôte (hosted mode) : il est utilisé lors de la phase de développement. Dans ce mode, l'application est exécutée sous la forme de bytecode dans une JVM. Ce mode permet donc la mise en oeuvre d'un débogueur pour faciliter la mise au point de l'application. Il utilise une version personnalisée d'un navigateur fourni par GWT en fonction de l'OS (Internet Exploreur sous Windows et Firefox sous Linux) et une machine virtuelle Java qui permet de transformer le code Java et de l'afficher dans le navigateur.
- mode web (web mode) : dans ce mode, le code Java est compilé pour générer les codes HTML et JavaScript de la partie client. L'application peut ainsi être exécutée dans les navigateurs supportés par GWT.

82.1.1. L'installation de GWT

Il faut télécharger GWT à l'url : http://code.google.com/webtoolkit/download.html

La version utilisée dans ce chapitre est la 1.3.3 sous Windows. Le fichier téléchargé se nomme donc gwt−windows−1.3.3.zip. Il faut décompresser le contenu de ce fichier dans un répertoire du système en utilisant un outil

gérant le format zip comme l'utilitaire jar fourni avec le JDK.

82.1.2. GWT version 1.6

La version 1.6 de GWT fut publiée en mai 2006. Cette version apporte de nombreuses évolutions notamment :

- une nouvelle structure pour les projets qui facilite le packaging de la partie serveur sous la forme d'une archive war
- un nouvel environnement d'exécution local qui utilise Jetty
- de nouveaux mécanismes de gestion des événements et le support d'évènements natifs
- de nouveaux composants (DatePicket, DateBox, LazyPanel)
- corrections de bugs
- ...

82.1.2.1. La nouvelle structure pour les projets

La structure des répertoires des projets GWT a été adaptée notamment pour respecter le standard des applications web et faciliter la création du packaging de déploiement sous la forme d'une archive war.

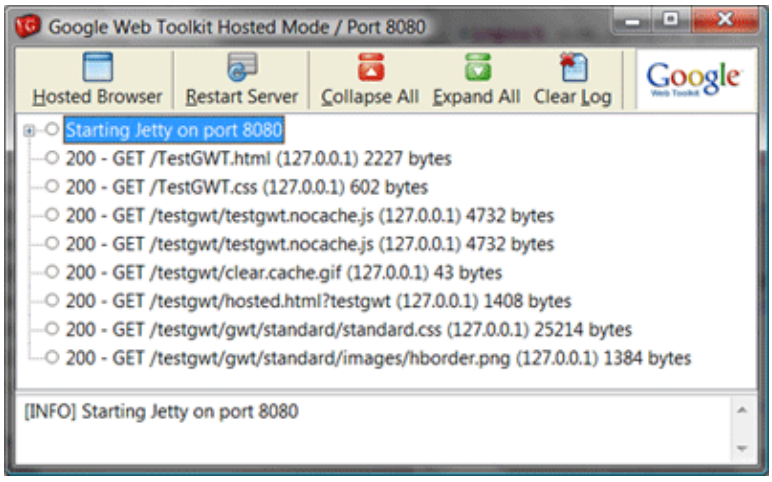
Le répertoire cible de génération des livrables se nomme d'ailleurs /war : ce répertoire contiendra le résultat des compilations mais aussi les ressources statiques nécessaires aux modules de l'application. Ainsi certaines adaptations sont nécessaires par rapport à l'organisation des projets des versions antérieures notamment :

- Le fichier de configuration web.xml de la partie serveur se trouve directement dans le sous−répertoire /war/WEB−INF.
- Toutes les bibliothèques requises par la partie serveur pour traiter les appels GWT−RPC doivent être ajoutées dans le sous−répertoire /war/WEB−INF/lib
- La page HTML qui contient l'application ainsi que les ressources requises doivent être mises dans le sous−répertoire war et non plus dans le sous−répertoire public comme c'était le cas dans les versions précédentes. Il est toujours possible d'inclure des ressources dans les sous répertoires des packages du module mais il faut que celles−ci soient spécifiques au module et manipulées dans le code. Pour accéder à une telle ressource, il est maintenant obligatoire d'utiliser la méthode GWT.getModuleBaseURL() pour obtenir le préfixe de l'url de la ressource.

Deux nouveaux outils sont proposés pour exploiter cette nouvelle structure de projet :

- HostedMode : permet l'exécution en mode hosted en remplacement de GWTShell
- Compiler : permet la compilation du code Java en JavaScript en remplacement de GWTCompiler

L'application GWTShell utilisait un serveur Tomcat embarqué. L'application HostedMode utilise un serveur Jetty embarqué.



La partie graphique cliente possède un nouveau bouton « Restart Server » qui permet de redémarrer le serveur Jetty : cela permet de prendre en compte des modifications dans la partie serveur sans avoir à arrêter et relancer l'application graphique cliente comme c'était le cas avec GWTShell.

82.1.2.2. Un nouveau système de gestion des événements

Le système de gestion des événements par listener est remplacé par un système de gestion par handler.

Les principales différences entre le deux systèmes sont :

- Les méthodes de type EventHandler ne possèdent qu'un seul paramètre de type GwtEvent. Par exemple, la classe ClickHandler possède la méthode onClick(ClickEvent)
- Chaque EventHandler ne possède qu'une seule méthode : ceci évite d'avoir à écrire des méthodes vides mais peut nécessiter de remplacer un listener par plusieurs handlers selon les besoins

La création de ses propres composants est facilitée car il n'est plus nécessaire de gérer manuellement les listeners. Tous les composants possèdent un objet de type HandlerManager dont le but est de gérer les handlers enregistrés auprès du composant.

La méthode addDomHandler() permet de gérer des événements natifs tels que ClickEvent par exemple : le handler est alors invoqué à l'émission de l'événement.

Exemple :

```
Button bouton = new Button("fermer");

bouton.addClickListener(new ClickListener() {
  @Override
  public void onClick(Widget sender) {
    // traitement du clic sur le bouton

  }
});
```

Exemple :

```
Button bouton = new Button("fermer");

bouton.addClickHandler(new ClickHandler() {
  @Override
  public void onClick(ClickEvent event) {
    // traitement du clic sur le bouton

  }
});
```

Certains handlers existants ont dû être adaptés :

Exemple :

```
final DisclosurePanel panel = new DisclosurePanel("Cliquez pour ouvrir");

panel.addEventHandler(new DisclosureHandler() {
  public void onClose(DisclosureEvent event) {
    panel.getHeaderTextAccessor().setText("Cliquez pour ouvrir");
  }

  public void onOpen(DisclosureEvent event) {
    panel.getHeaderTextAccessor().setText("Cliquez pour fermer");
  }
});

panel.add(new Image("images/logo_java.jpg"));
panel.setWidth("300px");
```

```
    RootPanel.get("app").add(panel);
```

Exemple :

```
    final DisclosurePanel panel = new DisclosurePanel("Cliquez pour ouvrir");

    panel.addOpenHandler(new OpenHandler<DisclosurePanel>() {
      @Override
      public void onOpen(OpenEvent<DisclosurePanel> event) {
        panel.getHeaderTextAccessor().setText("Cliquez pour fermer");
      }
    });

    panel.addCloseHandler(new CloseHandler<DisclosurePanel>() {
      @Override
      public void onClose(CloseEvent<DisclosurePanel> event) {
        panel.getHeaderTextAccessor().setText("Cliquez pour ouvrir");
      }
    });

    panel.add(new Image("images/logo_java.jpg"));
    panel.setWidth("300px");

    RootPanel.get("app").add(panel);
```

82.1.2.3. De nouveaux composants

Les composants DatePicker et DateBox permettent à l'utilisateur de sélectionner une date dans un calendrier.

Le panneau de type LazyPanel permet de retarder la création des objets utiles pour son rendu lorsqu'il sera affiché pour la première fois : ceci peut permettre d'améliorer le temps de démarrage de l'application qui n'est plus obligée d'instancier les composants de tous les éléments de l'application.

82.1.3. GWT version 1.7

GWT 1.7 est une mise à jour mineure qui apporte un meilleur support pour les dernières versions des navigateurs (Internet Explorer 8, Firefox 3.5 et Safari 4) et corrige quelques bugs majeurs. Elle a été publiée en juillet 2009.

Cette version est téléchargeable à l'url http://code.google.com/p/google−web−toolkit/downloads/list.

Pour un projet en GWT 1.6, il suffit simplement de le recompiler avec la version 1.7, sans modifier le code, pour que l'application soit compatible avec les nouvelles versions des navigateurs supportés.

82.2. La création d'une application

GWT propose plusieurs scripts pour générer des projets GWT composés d'une structure de répertoires et de fichiers fournissant le minimum pour développer un projet.

Pour créer un nouveau projet, il faut créer un nouveau répertoire et utiliser l'application applicationCreator fournie avec GWT. Cet outil permet de créer une petite application d'exemple qui peut facilement servir de base pour le développement d'une application utilisant GWT.

La version Windows de GWT contient un script pour lancer l'application ApplicationCreator. Il suffit d'exécuter ce script avec en paramètre le nom pleinement qualifié de la classe principale de l'application. Le dernier package de cette classe doit se nommer obligatoirement client pour éviter une erreur lors de l'exécution de ApplicationCreator

Résultat :

```
D:\gwt-windows-1.3.3>ApplicationCreator com.jmdoudoux.testgwt.monapp
'com.jmdoudoux.testgwt.monapp': Please use 'client' as the final package, as in
'com.example.foo.client.MyApp'.
It isn't technically necessary, but this tool enforces the best practice.
Google Web Toolkit 1.3.3
ApplicationCreator [-eclipse projectName] [-out dir] [-overwrite] [-ignore] className

where
  -eclipse    Creates a debug launch config for the named eclipse project
  -out        The directory to write output files into (defaults to current)
  -overwrite  Overwrite any existing files
  -ignore     Ignore any existing files; do not overwrite
and
  className   The fully-qualified name of the application class to create
```

Par défaut, les fichiers générés le sont dans le répertoire principal. Pour préciser le répertoire à utiliser (celui-ci doit exister), il faut utiliser le paramètre -out.

Résultat :

```
D:\gwt-windows-1.3.3>mkdir MonApp

D:\gwt-windows-1.3.3>ApplicationCreator -out MonApp com.jmdoudoux.testgwt.client
.MonApp
Created directory MonApp\src
Created directory MonApp\src\com\jmdoudoux\testgwt
Created directory MonApp\src\com\jmdoudoux\testgwt\client
Created directory MonApp\src\com\jmdoudoux\testgwt\public
Created file MonApp\src\com\jmdoudoux\testgwt\MonApp.gwt.xml
Created file MonApp\src\com\jmdoudoux\testgwt\public\MonApp.html
Created file MonApp\src\com\jmdoudoux\testgwt\client\MonApp.java
Created file MonApp\MonApp-shell.cmd
Created file MonApp\MonApp-compile.cmd
```

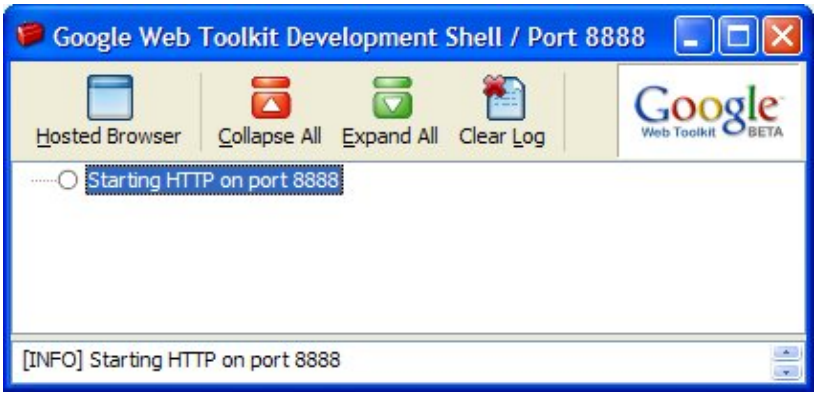
Plusieurs répertoires et fichiers sont créés :

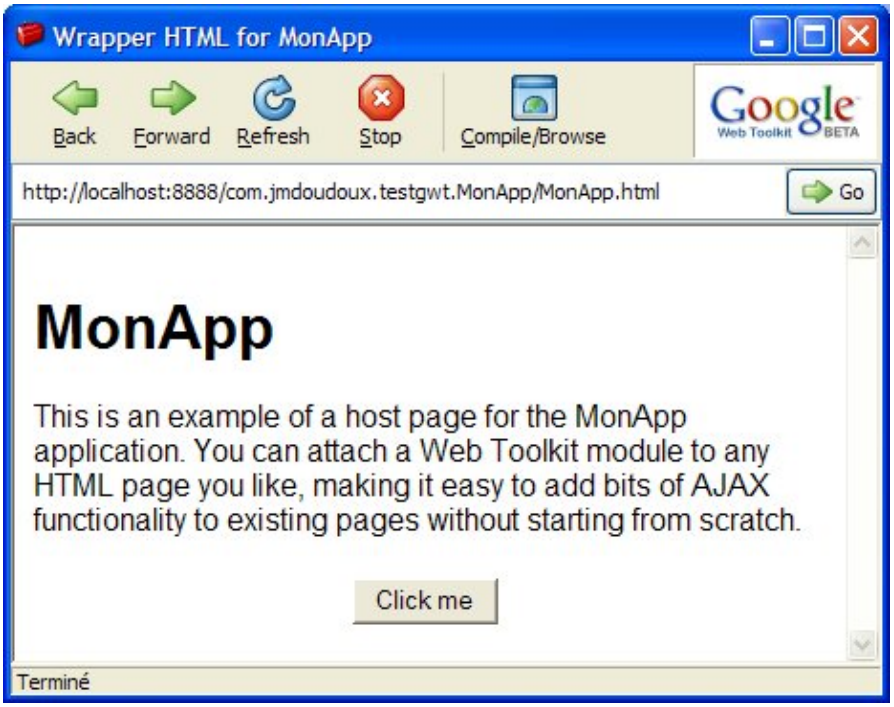
- le répertoire src composé de plusieurs sous-répertoires contient les sources de l'application générée : le sous-répertoire public contient les pages html et le sous-répertoire client contient les sources Java
- le fichier MonApp.gwt.xml contient la configuration de l'application
- le fichier MonApp-shell.cmd permet d'exécuter l'application en mode hôte
- le fichier MonApp-compile.cmd permet de compiler et d'exécuter l'application en mode web

Pour exécuter l'application en mode hôte, il suffit donc de lancer le script MonApp-shell.cmd

Résultat :

```
D:\gwt-windows-1.3.3>cd MonApp

D:\gwt-windows-1.3.3\MonApp>dir
 Le volume dans le lecteur D s'appelle Java
 Le numéro de série du volume est D8C0-0514

 Répertoire de D:\gwt-windows-1.3.3\MonApp

30/07/2007  22:10    <REP>          .
30/07/2007  22:10    <REP>          ..
30/07/2007  22:10               186 MonApp-compile.cmd
30/07/2007  22:10               195 MonApp-shell.cmd
30/07/2007  22:10    <REP>          src
               2 fichier(s)              381 octets
               3 Rép(s)  16 037 978 112 octets libres

D:\gwt-windows-1.3.3\MonApp>MonApp-shell.cmd
```





Pour vérifier la bonne exécution de l'application, il suffit de cliquer sur le bouton "Click me" pour voir apparaître un message.

82.2.1. L'application générée

L'application générée se compose de plusieurs fichiers qui constituent sa base.

Les fichiers sources de l'application sont stockés dans un package qui contient toujours trois sous−répertoires :

Package	Rôle
client	Contient les classes qui composent la partie interface graphique de l'application. Seules les classes de ce package et de ses sous−packages seront compilées en Java
public	Contient les ressources statiques web : pages HTML, images, JavaScript, feuilles de style CSS, ...
server	Contient les classes qui seront exécutées côté serveur

L'application repose sur une page html nommée nom_du_projet.html dans le répertoire public.

Le code de l'application est contenu dans la classe nom_du_projet.java du répertoire client.

Une application GWT est contenue dans un module. La configuration d'un module est le fichier nom_du_projet.gwt.xml.

82.2.1.1. Le fichier MonApp.html

Le fichier MonApp.html du sous−répertoire public contient la structure de la page de l'application.

Le fichier html d'une application GWT est généralement très simple : la page html sert d'enveloppe pour recevoir les différents composants graphiques qui seront ajoutés grâce à du code Java.

Exemple :

```
<html>
<head>
<title>Wrapper HTML for MonApp</title>
<style>
  body,td,a,div,.p{font−family:arial,sans−serif}
  div,td{color:#000000}
  a:link,.w,.w a:link{color:#0000cc}
  a:visited{color:#551a8b}
  a:active{color:#ff0000}
</style>
<meta name='gwt:module' content='com.jmdoudoux.testgwt.MonApp'>

</head>
<body>
<script language="javascript" src="gwt.js"></script>
<iframe id="__gwt_historyFrame" style="width:0;height:0;border:0"></iframe>
<h1>MonApp</h1>
<p>This is an example of a host page for the MonApp application. You
can attach a Web Toolkit module to any HTML page you like, making it
easy to add bits of AJAX functionality to existing pages without
starting from scratch.</p>
<table align=center>
  <tr>
    <td id="slot1"></td>
    <td id="slot2"></td>
  </tr>
</table>
</body>
</html>
```

Les deux balises <td> possèdent des identifiants distincts : ils définissent des conteneurs dans lesquels les composants vont être ajoutés. Chaque identifiant sera utilisé dans le code Java pour obtenir une référence sur le conteneur.

Le script JavaScript gwt.js est utilisé pour lancer l'application notamment en exécutant la version de l'application dédiée au navigateur utilisé.

L'iframe est utilisé dans le mécanisme de gestion de l'historique de navigation.

82.2.1.2. Le fichier MonApp.gwt.xml

Ce fichier contient la définition et la configuration du module notamment :

- la classe qui fait office de point d'entrée dans l'application
- les dépendances
- les directives de compilation

C'est un fichier XML dont l'extension est .gwt.xml. Le tag racine est le tag <module>

Le tag <inherits> permet de préciser les fonctionnalités de base qui composeront le module.

Le tag <entry−point> permet de préciser la classe pleinement qualifiée qui est le point d'entrée de l'application : l'attribut class permet de préciser le nom de la classe principale de l'application.

Exemple :

```
<module>

  <!-- Inherit the core Web Toolkit stuff.                  -->
  <inherits name='com.google.gwt.user.User' />

  <!-- Specify the app entry point class.                   -->
  <entry-point class='com.jmdoudoux.testgwt.client.MonApp' />

</module>
```

82.2.1.3. Le fichier MonApp.java

La classe MonApp contient le code de l'application avec notamment :

- la définition des composants de l'interface graphique
- la définition des gestionnaires d'événements (listeners ou handlers) pour répondre aux actions de l'utilisateur
- les traitements en réponse aux événements

La méthode onModuleLoad() est le point d'entrée de l'application. Cette méthode contient la définition de l'IHM de l'application.

La mise en oeuvre des gestionnaires d'événements est similaire à celle d'autres framework permettant le développement d'interfaces graphiques tels que AWT, Swing ou SWT. Elle repose sur l'enregistrement de listeners généralement définis sous la forme de classes anonymes.

Exemple :

```
package com.jmdoudoux.testgwt.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;

/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */
public class MonApp implements EntryPoint {

  /**
   * This is the entry point method.
   */
  public void onModuleLoad() {
    final Button button = new Button("Click me");
    final Label label = new Label();

    button.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
        if (label.getText().equals(""))
          label.setText("Hello World!");
        else
          label.setText("");
      }
    });

    // Assume that the host HTML has elements defined whose
    // IDs are "slot1", "slot2".  In a real app, you probably would not want
    // to hard-code IDs.  Instead, you could, for example, search for all
    // elements with a particular CSS class and replace them with widgets.
    //
    RootPanel.get("slot1").add(button);
    RootPanel.get("slot2").add(label);
  }
}
```

Important : les classes de la partie client ne peuvent pas faire référence aux classes de la partie serveur.

82.3. Les modes d'exécution

Comme évoqué en introduction, une application GWT peut être exécutée dans deux modes :

- Le mode hôte (hosted mode) : ce mode est utilisé pour le développement et la mise au point de l'application car il permet la mise en oeuvre d'un débogueur
- Le mode web (web mode) : ce mode est utilisé pour le déploiement et l'exploitation de l'application par les utilisateurs

82.3.1. Le mode hôte (hosted mode)

Dans ce mode, l'application est exécutée de façon hybride sous la forme de code Java exécuté dans un navigateur spécial. Ceci permet notamment l'utilisation du débogueur d'un IDE.

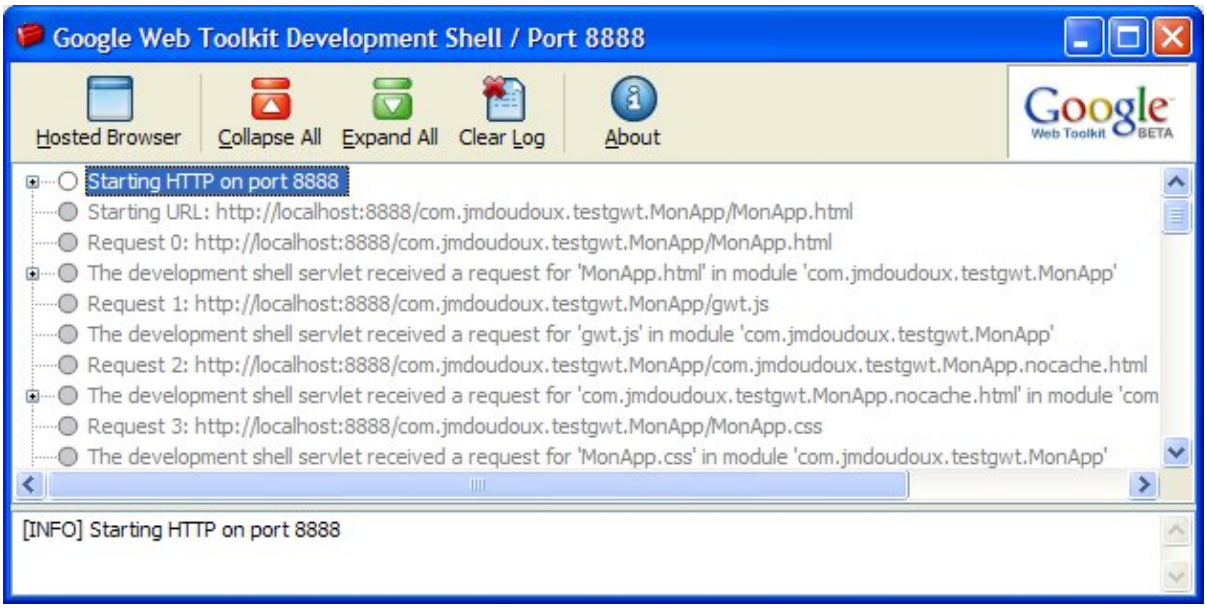
L'environnement d'exécution est composé d'une console, d'un conteneur web (Tomcat ou Jetty selon la version de GWT) et d'un navigateur dédié (Internet Exploreur sous Windows et Firefox sous Linux).

Remarque : le mode hôte n'est disponible que sous Windows et Linux.

Pour exécuter une application dans le mode hôte, il faut exécuter le script dont le nom se compose du nom de l'application et se termine par −shell. Plusieurs options peuvent être fournies à l'environnement d'exécution :

- −noserver
- −out
- −gen
- −logLevel level : permet de préciser niveau de trace. Level peut prendre les valeurs : ERROR, WARN, INFO, TRACE, DEBUG, SPAM, ALL

Exemple avec l'option −logLevel ALL



Le mode hôte, facilite grandement l'écriture et la mise au point de l'application en permettant :

- l'exécution de l'application
- la modification du code source, sa recompilation

- de relancer l'application simplement en cliquant sur le bouton Refresh

L'environnement d'exécution affiche deux fenêtres :

- la fenêtre "Google Web Toolkit Development Shell / Port 8888" : affiche les messages du serveur et permet d'interagir avec lui
- le navigateur qui affiche l'application

Lorsque l'application est exécutée en mode hôte :

- il n'y a pas besoin de compiler et déployer l'application à chaque modification
- pour tester une modification faite dans le code et compilée en bytecode, il suffit simplement de cliquer sur le bouton de rafraichissement du navigateur : ceci permet de tester rapidement des modifications
- il est possible d'utiliser le débogueur d'un IDE en positionnant des points d'arrêts

82.3.2. Le mode web (web mode)

Dans le mode web, la partie cliente de l'application doit être compilée en JavaScript. Un script de compilation est généré lors de la création de l'application. Le nom de ce script est composé du nom de l'application suivi de « −compile.cmd ».

Le compilateur possède plusieurs options :

- −logLevel
- −treeLogger
- −gen
- −out
- −style : style de lisibilité du code généré : OBFUSCATED (style par défaut), PRETTY ou DETAILED

Le compilateur génère plusieurs fichiers correspondant à chaque navigateur supporté par le compilateur et éventuellement un pour chaque langue mise en oeuvre pour internationaliser l'application. Ceci permet de réduire la taille du fichier JavaScript de l'application car elle ne contient que du code pour le navigateur et la Locale utilisés.

Résultat :

```
D:\gwt−windows−1.3.3\MonAppProjet>MonApp−compile.cmd
Output will be written into D:\gwt−windows−1.3.3\MonAppProjet\www\com.jmdoudoux.
testgwt.MonApp
Copying all files found on public path
Compilation succeeded
```

Le répertoire www est créé : il contient un sous−répertoire qui porte le nom du package principal de l'application. Ce répertoire contient les fichiers générés.

Résultat : le contenu du répertoire de D:\gwt−windows−1.3.3\MonAppProjet\www\com.jmdoudoux.testgwt.MonApp

```
D:\gwt−windows−1.3.3\MonAppProjet\www\com.jmdoudoux.testgwt.MonApp>dir
 Le volume dans le lecteur D s'appelle Java
 Le numéro de série du volume est D8C0−0514
13/08/2007  23:03    <REP>          .
13/08/2007  23:03    <REP>          ..
13/08/2007  23:03            38 805 0A3A82524C61CDBB6FEA7286E3F85391.cache.html
13/08/2007  23:03               801 0A3A82524C61CDBB6FEA7286E3F85391.cache.xml
13/08/2007  23:03            38 865 4D7E1609F2BC0A4229FFC55F98976B68.cache.html
13/08/2007  23:03               798 4D7E1609F2BC0A4229FFC55F98976B68.cache.xml
13/08/2007  23:03            38 628 921FF51D706A4D67E10268B5DD22D7D7.cache.html
13/08/2007  23:03               801 921FF51D706A4D67E10268B5DD22D7D7.cache.xml
13/08/2007  23:03            38 422 942F11931D582C6DF0166C3EA388BE17.cache.html
13/08/2007  23:03               798 942F11931D582C6DF0166C3EA388BE17.cache.xml
13/08/2007  23:03             2 900 com.jmdoudoux.testgwt.MonApp.nocache.html
13/08/2007  23:03            17 336 gwt.js
13/08/2007  23:03               444 history.html
13/08/2007  23:03               501 MonApp.css
```

```
13/08/2007  23:03               512 MonApp.html
13/08/2007  23:03                82 tree_closed.gif
13/08/2007  23:03                78 tree_open.gif
13/08/2007  23:03                61 tree_white.gif
              16 fichier(s)          179 832 octets
```

Les fichiers .cache.html contiennent le code JavaScript pour chaque navigateur. Le nom du fichier est encrypté. Chacun de ces fichiers possède un fichier avec le même nom et l'extension .cache.xml.

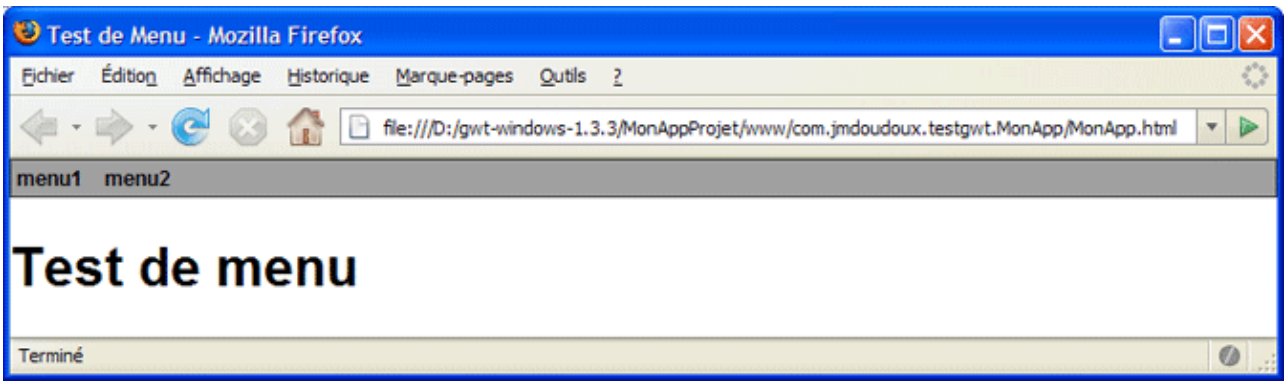
Ces fichiers donnent des informations sur le contenu des fichiers

Exemple : le fichier4D7E1609F2BC0A4229FFC55F98976B68.cache.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cache-entry>
        <rebind-decision in="com.google.gwt.user.client.ui.impl.TextBoxImpl"
          out="com.google.gwt.user.client.ui.impl.TextBoxImpl"/>
        <rebind-decision in="com.google.gwt.user.client.impl.DOMImpl"
          out="com.google.gwt.user.client.impl.DOMImplMozilla"/>
        <rebind-decision in="com.google.gwt.user.client.impl.HistoryImpl"
          out="com.google.gwt.user.client.impl.HistoryImplStandard"/>
        <rebind-decision in="com.google.gwt.user.client.ui.impl.FormPanelImpl"
          out="com.google.gwt.user.client.ui.impl.FormPanelImpl"/>
        <rebind-decision in="com.jmdoudoux.testgwt.client.MonApp"
          out="com.jmdoudoux.testgwt.client.MonApp"/>
        <rebind-decision in="com.google.gwt.user.client.ui.impl.PopupImpl"
          out="com.google.gwt.user.client.ui.impl.PopupImpl"/>
</cache-entry>
```

Ainsi, le fichier 4D7E1609F2BC0A4229FFC55F98976B68 correspond au code pour le navigateur Mozilla.

L'ouverture du fichier MonApp.html dans un navigateur lance l'application



Pour diffuser l'application qui ne contient pas de partie serveur, il suffit de copier les fichiers générés, hormis les fichiers .xml, dans un serveur web.

Pour une application qui contient une partie serveur, il faut packager l'application dans un war. Cette archive doit contenir :

- la partie cliente : les fichiers HTML et JavaScript générés ainsi que les ressources statiques (CSS, images, ...)
- la partie serveur : les fichiers .class, le fichier web.xml, les bibliothèques requises (gwt-servlet.jar, ...)

82.4. Les éléments de GWT

GWT se compose de plusieurs éléments :

- le compilateur qui compile du code Java en code JavaScript

- JSNI qui permet l'utilisation de code JavaScript dans le code Java
- JRE Emulation Library qui est un sous−ensemble des classes de base de Java
- une API qui fournit de nombreuses fonctionnalités : composants graphiques pour IHM, appels RPC vers un serveur, gestion de l'historique de navigation, parseur de documents XML, tests unitaires avec JUnit, ...

82.4.1. Le compilateur

Le compilateur GWT de code Java en JavaScript est encapsulé dans la classe com.google.gwt.dev.GWTCompiler.

Le compilateur traite l'entry point pour chacune de ses dépendances. Le compilateur utilise les fichiers sources mais n'utilise pas les fichiers .class.

Le compilateur des versions antérieures à la version 1.5 de GWT ne supporte que du code source respectant la syntaxe de Java 1.4 : les fonctionnalités de Java 5 ne sont donc pas supportées avant la version 1.5 de GWT. Cette restriction n'est valable que pour le code de la partie cliente qui sera transformé en JavaScript. La partie serveur n'est pas concernée par cette restriction puisqu'elle est compilée en bytecode pour être exécutée dans la JVM du serveur.

Le compilateur génère un fichier JavaScript par navigateur et par langage si l'internationalisation est utilisée dans l'application. Le fichier pour le navigateur concerné sera chargé au lancement de l'application. L'intérêt majeur est de limiter le code JavaScript au navigateur utilisé : ceci évite d'avoir à gérer de nombreuses opérations de tests sur le navigateur comme cela est fréquent dans le code JavaScript.

Cette fonctionnalité est aussi mise en oeuvre pour chaque langue utilisée pour internationaliser l'application. Le code JavaScript ne contient que les libellés pour la langue du fichier.

Le compilateur peut mettre en oeuvre des techniques d'obfuscation du code JavaScript généré afin de le protéger et surtout de réduire sa taille.

Au final, le code contenu dans chaque fichier généré est le plus réduit possible.

82.4.2. JRE Emulation Library

Pour utiliser certaines classes de la bibliothèque de base de Java, GWT propose le JRE Emulation Library qui contient les classes fréquemment utilisées dans les applications. Ces classes sont un sous−ensemble de la bibliothèque correspondant à celles qui peuvent être transformées en JavaScript par le compilateur.

Les classes du package java.lang incluses dans le JRE Emulation Library sont :

Boolean	Byte	Character
Class	Double	Float
Integer	Long	Math
Number	Object	Short
String	StringBuffer	System
Throwable	Error	Exception

Il existe aussi des différences entre leur utilisation dans une JVM et dans le JRE Emulation Library. Ces différences sont essentiellement imposées par la conversion du code en JavaScript.

Certaines fonctionnalités de ces classes sont ainsi différentes de leurs homologues de la bibliothèque Java, par exemple :

- System.out et System.err sont utilisables dans le mode hosted mais n'ont aucun effet dans le mode web

- les expressions régulières utilisées dans certaines méthodes la classe String (replaceAll(), replaceFirst()) sont différentes
- ...

Attention : JavaScript ne propose pas de support pour les entiers sur 64 bits représentés par une variable de type long en Java. Le compilateur transforme les types long en double. Le fonctionnement de l'application peut donc être différent dans le mode host et web.

La méthode getStackTrace() de la classe Throwable n'est pas utilisable dans le mode web.

Les assertions (mot clé assert) sont ignorées par le compilateur.

JavaScript n'est pas multithread : tout ce qui concerne le multithreading dans le langage Java est donc inutilisable et ignoré par le compilateur.

L'API reflexion permettant une utilisation dynamique des objets n'est pas utilisable. L'API GWT propose uniquement la méthode GWT.getTypeName() : elle renvoie une chaîne de caractères qui correspond au type de l'objet fourni en paramètre.

JavaScript ne propose pas le support pour la finalisation des objets lors de leur traitement par le garbage collector.

JavaScript ne propose pas le support d'une précision constante dans les calculs en virgule flottante. Il n'est donc pas recommandé d'effectuer de tels calculs dans la partie cliente. Des calculs en virgule flottante peuvent être réalisés mais leur précision n'est pas garantie.

La sérialisation proposée par Java n'est pas supportée par GWT qui a son propre mécanisme pour les appels de type RPC vers le serveur.

Pour des raisons de performance, il n'est pas recommandé d'utiliser des objets de type Long, Float ou Double comme clé pour des objets de type Map.

Les classes du package java.lang incluses dans le JRE Emulation Library sont :

AbstractCollection	AbstractList	AbstractMap
AbstractSet	ArrayList	Arrays
Collections	Date	HashMap
Stack	Vector	Collection
Comparator	EventListener	Iterator
List	Map	RandomAccess
Set		

82.4.3. Les modules

La classe qui est le point d'entrée d'un module doit implémenter l'interface EntryPoint. Cette interface définit la méthode void onLoadModule().

Un module contient un fichier descripteur au format XML. Son nom est composé du nom du module suivi de .gwt.xml

Il permet de préciser :

- Les autres modules utilisés
- Le nom de la classe qui sert de point d'entrée
- Les chemins des fichiers sources qui doivent être compilés en JavaScript
- Les chemins pour trouver les ressources publiques (CSS, images, javasript, ...)

Ce fichier est stocké dans le répertoire contenant la classe qui sert de point d'entrée au module.

Le tag <module> est le tag racine.

Le tag <inherits> permet de préciser un autre module qui sera utilisé. L'attribut name permet de préciser le nom du module.

Le tag <source> permet de préciser le répertoire qui contient des sources à compiler grâce à l'attribut path.

Le tag <stylesheet> permet de préciser une feuille de style CSS. L'attribut src permet de préciser le nom du fichier CSS.

Le tag <servlet> permet de définir une servlet qui sera utilisée pour les communications de type RPC avec le serveur en mode hosted. L'attribut path permet de préciser l'uri de la servlet. L'attribut class permet de préciser le nom pleinement qualifié de la classe qui encapsule la servlet.

82.4.4. Les limitations

Le code de l'application est compilée en JavaScript : seules les fonctionnalités compilables en JavaScript peuvent être utilisées. Ainsi par exemple, il n'est pas possible d'utiliser le type primitif long puisque JavaScript ne supporte pas le 64 bits. Cependant le code se compile parfaitement puisque chaque variable de type long est convertie en type double ce qui peut provoquer des effets de bord.

JavaScript n'est pas multithread : il faut en tenir compte lors du développement de l'application.

82.5. L'interface graphique des applications GWT

Pour la partie graphique de l'application, GWT propose un ensemble de composants de deux types :

- widgets : ce sont des contrôles utilisateurs soit de base (bouton, zone de texte, case à cocher, bouton radio, ...) soit plus riches en fonctionnalités (barre de menu, onglet, treeview, ...)
- panels : ces composants se chargent d'assurer la disposition des composants qui leurs sont rattachés à l'image des layouts manager de Swing. Certains panels proposent aussi de l'interactivité avec l'utilisateur.

En plus de ces composants proposés en standard par GWT, il est possible de développer ses propres composants.

Il existe plusieurs projets open source qui développent d'autres composants (calendrier, grille, ...) ou des composants qui encapsulent des bibliothèques JavaScript existantes (Scriptaculous, Google Search et Map, ...).

Les composants possèdent une double représentation :

- en Java, lors de l'écriture du code et de l'exécution de l'application dans le mode hôte
- dans l'arbre DOM de la page une fois le code compilé en JavaScript et exécuté dans le mode web

L'organisation des composants n'est pas assurée par des layouts mais par des panneaux qui rendent mieux en HTML. Par exemple :

- HorizontalPanel : les composants sont mis les uns à coté des autres de gauche à droite
- FlowPanel : arrange les composants qu'il contient les uns à côté des autres en allant du haut à gauche vers le bas à droite
- AbsolutePanel : permet de préciser les coordonnées des composants
- ...

GWT propose un ensemble complet de composants graphiques (widgets) et panneaux (panels).

L'état de l'interface graphique est maintenu sur le client dans une application GWT.

Exemple :

```
package com.jmdoudoux.test.gwt.client;
```

```
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.FlowPanel;
import com.google.gwt.user.client.ui.Panel;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.Widget;

public class MainEntryPoint implements EntryPoint {

  private boolean etat = true;

  public MainEntryPoint() {
  }

  public void onModuleLoad() {
    final TextBox text = new TextBox();
    text.setText("AAAAA");
    final Button button = new Button();
    button.setText("Inverser");
    button.addClickListener(new ClickListener() {

      public void onClick(Widget sender) {
        etat = !etat;
        if (etat) {
          text.setText("AAAAA");
        } else {
          text.setText("ZZZZZ");
        }
      }
    });
    Panel main = new FlowPanel();
    RootPanel.get().add(main);
    main.add(text);
    main.add(button);
  }
}
```

Un clic sur le bouton "Inverser" inverse les lettres affichées

82.6. La personnalisation de l'interface

Comme une application GWT est compilée pour générer une application utilisant le DHTML et JavaScript, la personnalisation de l'interface de l'application repose sur les feuilles de style CSS.

Le rendu des composants d'une application GWT peut donc être assuré par des styles CSS.

La plupart des composants ayant un rendu graphique possèdent une classe de style CSS, par défaut, composée de gwt−suivi du nom du composant (exemple : gwt−Button, gwt−CheckBox, ...).

Il est possible d'utiliser une feuille de style CSS définie dans un fichier stocké dans le sous−répertoire public de l'application.

Exemple : le fichier monstyle.css

Résultat :

```
root {
    display: block;
}

.message
{
    color: blue;
    display: block;
    width: 450px;
    padding: 2px 4px;
    margin-top: 3px;
    text-decoration: none;
    text-align: center;
    font-family: Verdana,Arial,Helvetica,sans-serif;
    font-size: 10px;
    border: 1px solid;
    border-color: black;
    ext-decoration: none;
}

.erreur
{
    color: white;
    display: block;
    width: 450px;
    padding: 2px 4px;
    margin-top: 3px;
    text-decoration: none;
    text-align: center;
    font-family: Verdana,Arial,Helvetica,sans-serif;
    font-size: 10px;
    font-weight: bold;
    border: 1px solid;
    border-color: black;
    ext-decoration: none;
    background-color: red;
}
```

Pour que la feuille de style CSS soit prise en compte par l'application, il faut la déclarer dans le fichier de configuration du module. Cette déclaration se fait à l'aide du tag <stylesheet>. Son attribut src permet de préciser le nom du fichier CSS.

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
  <inherits name="com.google.gwt.user.User"/>
  <entry-point class="com.jmdoudoux.test.gwt.client.MainEntryPoint"/>
  <stylesheet src="monstyle.css"/>
</module>
```

Chaque composant possède plusieurs méthodes héritées de la classe UIObject et relatives aux styles CSS :

- addStyleName() : permet d'ajouter un style à la liste des styles du composant
- setStylePrimaryName() :
- setStyleName() : permet de forcer le sélecteur de classe du style utilisé en supprimant tous les styles appliqués

Exemple :

```
...
            public void onSuccess(Object result) {
                lblMessage.setStyleName("message");
                lblMessage.setText((String) result);
            }

            public void onFailure(Throwable caught) {
                lblMessage.setStyleName("erreur");
                lblMessage.setText("Echec de la communication");
```