GWT
Google Web Toolkit

Yves Bekkers

Présentation de GWT

- Outil développé par Google depuis 2006
- Principe
  - Utilisation massive d'appels à distance asynchrones
  - Développement WEB orienté composants
  - Développer en un seul langage : Java
  - Lors du déploiement un compilateur traduit le code client écrit en Java vers une application WEB classique (HTML+Javascript)
  - Abstraction complète de la complexité liée à HTML, Javascript, CSS

Application WEB dynamique RIA

- Comme une application AJAX : pas de rechargement de page html entre les requêtes
- Développer une application AJAX sans écrire de code Javascript
  - Développement
    - Écrire l'interface WEB en Java
    - Débugger l'interface en Java dans un « mode hôte »
  - Production
    - Déployer l'application en utilisant le compilateur croisé GWT pour générer les pages HTML et le code Javascript à partir du code Java

Portabilité

- Les applications GWT sont supportées automatiquement par tous les principaux navigateurs
  - IE
  - Mozilla Firefox
  - Safari
  - Google Chrome

Communication avec le serveur

- Deux types de communication
  - GWT RPC (Remote Procedure Call)
    - À base d'une servlet coté serveur
    - asynchrone
  - HTTP XML, JSON (JavaScript Object Notation)

Communications RPC

- Appel de méthodes à distance basé sur les servlet
- Esprit RMI
  - Créer une interface qui spécifie les méthodes distantes
  - Implémenter l'interface coté serveur
- Déroulement d'un appel de méthode à distance
  - Sérialisation des arguments
  - Appel de la méthode distante
  - Sérialisation du résultat
  - Le client désérialise le résultat
- Échange d'objets java sérialisés au dessus du protocole HTTP, asynchrone

Format XML

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New"
  onclick="CreateNewDoc()" />
    <menuitem value="Open"
  onclick="OpenDoc()" />
    <menuitem value="Close"
  onclick="CloseDoc()" />
  </popup>
</menu>
```


Format JSON

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value":"New","onclick":"CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```


Pourquoi GWT

- Pas besoin de connaitre Javascript
- Javacript est un faux langage à objet
- Typage statique de votre application
  - Détection de bug à la compilation
- On reste dans un même univers de développement : Java
  - Mise au point du code client en Java
- Support des outils de développement javascript/Ajax laisse à désirer

Gwt et les applets

- Même idée de départ
  - Déporter l'activité sur le client
  - Décharger le serveur
  - Limiter les échanges d'information
- Code plus simple avec GWT
- GWT offre en plus un mécanisme de RPC
  - Remote Procedure Call : appel de méthode à distance
  - Sérialisation automatique des objets échangés

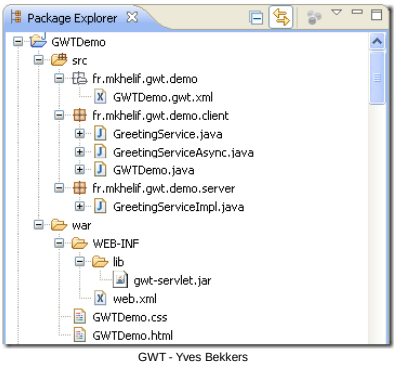
Composants de GWT

- le **compilateur Java vers JavaScript**
- un **navigateur spécialement modifié** pour permettre l'exécution (et le débogage) de code Java natif sans nécessiter la compilation JavaScript
- une **bibliothèque d'émulation JRE**: il s'agit d'une implémentation en JavaScript d'un sous-ensemble de la bibliothèque de classes Java standard (en particulier quasiment tout le package java.lang et une partie de java.util)
- une **bibliothèque de composants graphiques** contenant des Widgets de base permettant la construction d'une interface graphique

Un outil de développement sous Eclipse

- Un plugin pour Eclipse 3.4
- site de mise à jour
  - http://dl.google.com/eclipse/plugin/3.4

Structure d'un nouveau projet sous Eclipse



Conventions

- Module sous Eclipse
  - Deux fichiers avec une nommage cohérant
    - `monMudule.gwt.xml` le descripteur de module
    - `monModule.html` la page html hôte du module
  - Plusieurs modules sont autorisés dans un même projet, ils ont chacun leur descripteur, point d'entrée et leur page HTML Hôte
  - Un module n'a qu'une page HTML hôte
  - Si un module a plusieurs points d'entrée ils apparaissent tous dans la même page

Le dossier war

- Le dossier war, représente l'archive WAR qui sera exportée/déployée. Lors de la compilation du projet, les fichiers seront créés dans un sous-dossier de celui-là.
  - Le répertoire lib du WEB-INF contient les librairies GWT.
- Toutes les ressources (images, css, …) doivent être placées dans le répertoire war à l'instar d'une application Web classique.

Le dossier src

- On trouve deux packages
  - `Client`
  - `Server`
- On trouve aussi un fichier `GWTDemo.gwt.xml`

GWT en « mode hôte » – mise au point

- Application peut s'exécuter en mode hôte « hosted mode »
  - La JVM exécute le code client à l'intérieur d'une fenêtre de navigateur hôte
  - Mise au point facilitée
    - Éditer le source
    - Rafraichir
    - Examiner les résultats
    - Recommencer

GWT en « mode WEB » - déploiement

- Une fois testé le code en mode hôte on le compile et on le déploie sur un serveur (Tomcat d'Apache par exemple)
- Une application déployée est dite en « mode WEB »
- Une fois déployé le code client est du pur HTML+Javacript

Support des CSS

- Séparation du code et de la présentation
- Code Java

```
Public ListWidget(String Item) {
   …
   setStyleName("gwt-ListWidget");
}
```

- Fichier CSS

```
.gwt-ListWidget {
   background-color:black;
   color:red;
}
```

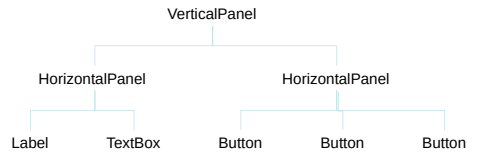

Interface graphique

Composants graphiques (Widgets) disponibles

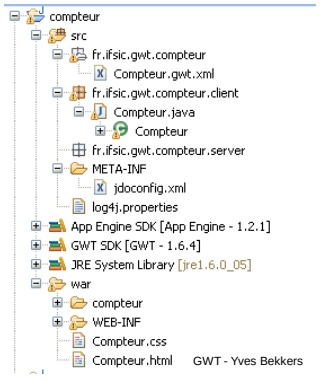
- Des composants graphiques classiques
  - Panneaux
  - Boutons
  - Cases à cocher
  - Tables / Grilles
  - Boîtes de dialogues
  - Primitive HTML (dont les images et les hyperliens)
  - Menus et barres de menus
  - Fenêtres défilantes
  - Onglets
  - Arbres
- Plusieurs composants non disponibles dans GWT ont été implémentées dans des librairies tierces, comme Ext GWT, GWT Component Library, GWT-Ext,GWT Widget Library,GWTiger,Rocket GWT,dojo etc .

Un exemple : le compteur

Structure de l'interface



Structure de l'application sous Eclipse



Notion de module

- Un module est un ensemble de services cohérents réutilisables
- Toute application est incluse dans un module
- Un module comporte
  - Un nom de module
  - Un descripteur de module (document `Monmodule.gwt.xml`)
  - Une page HTML hôte
  - Un éventuel point d'entrée

Nom du document descripteur

- Sous Eclipse par défaut si le projet s'appelle `monProjet` le descripteur s'appelle `MonProjet.gwt.xml`
- Le nom du document descripteur peut être changé

Le document `Compteur.gwt.xml`

- À placer dans le répertoire `src/fr/ifsic/gwt/compteur`

```
<module rename-to='compteur'>
  <inherits name='com.google.gwt.user.User'/>
  <inherits
   name='com.google.gwt.user.theme.standard.Standard'/>

  <entry-point
   class='fr.ifsic.gwt.compteur.client.Compteur'/>
</module>
```

- Déclaration du point d'entrée du module

Nom d'un module

- Sous Eclipse si un projet s'appelle `monProjet` que le module se trouve dans la paquetage `fr.ifsic.test` le module s'appelle par défaut `fr.ifsic.test.MonProjet`
- On peut renommer un module dans le fichier de configuration `MonProjet.gwt.xml` grâce à l'attribut `rename-to` de l'élément `<module>`

La page html hôte

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
   charset=UTF-8"/>
    <link type="text/css" rel="stylesheet" href="Compteur.css"/>
    <title>Compteur</title>
    <script type="text/javascript" language="javascript"
            src="compteur/compteur.nocache.js"></script>
  </head>

  <body>
    <h1>Compteur</h1>
    <div id="mainCompteur" align="center"/>
  </body>
</html>
```

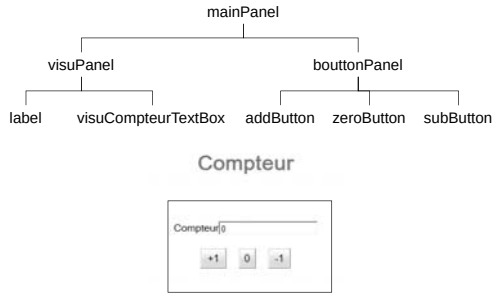
- **L'élément <body> peut être totalement vide**

La classe `Compteur`

```
public class Compteur implements EntryPoint {
   public void onModuleLoad() {
      …
   }
   …
}
```

- Point d'entrée du module : la méthode `onModuleLoad`
  - Remplace la méthode `main` d'une application java

Structure de l'interface



Déclaration des composants de l'interface

```
VerticalPanel mainPanel = new VerticalPanel();
HorizontalPanel visuPanel = new HorizontalPanel();
HorizontalPanel buttonPanel = new HorizontalPanel();
TextBox visuCompteurTextBox = new TextBox();
Button addButton = new Button("+1");
Button zeroButton = new Button("0");
Button subButton = new Button("-1");
Label label = new Label("Compteur");
```

- Composants de la librairie `com.google.gwt.user.client.ui`

Placement des composants de l'interface

```
mainPanel.setHorizontalAlignment(
     HasHorizontalAlignment.ALIGN_CENTER);
visuPanel.add(label);
visuPanel.add(visuCompteurTextBox);

buttonPanel.setSpacing(20);
buttonPanel.add(addButton);
buttonPanel.add(zeroButton);
buttonPanel.add(subButton);

mainPanel.add(visuPanel);
mainPanel.add(buttonPanel);

RootPanel.get("mainCompteur").add(mainPanel);
```


Placement dans la page HTML

- Dans le code Java

```
RootPanel.
  get("mainCompteur").add(mainPanel);
```

- Dans la page html
- `<div id="mainCompteur"/>`

Placement par défaut

```
RootPanel.get().add(mainPanel);
```

- `mainPanel` est ajouté au composant `<body>` de la page html

Mémorisation de la valeur du compteur

- Attribut `cpt` de la classe `Compteur`

```
public class Compteur implements EntryPoint {
   private int cpt = 0;
   public void onModuleLoad() {
      …
   }
   …
}
```

Gestion des évènements sur les boutons

```
addButton.addClickHandler(new ClickHandler() {
   public void onClick(ClickEvent event) {
      cpt++;
      refresh();
   }
});
```

- Le méthode `refresh()` met à jour l'interface

Mise à jour de l'interface

```
private void refresh() {
   visuCompteurTextBox.setText(""+cpt);
}
```


Gestion des évènement dans la `TextBox`

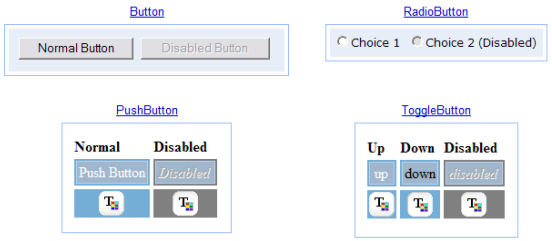
```
visuCompteurTextBox.addKeyPressHandler(
      new KeyPressHandler() {
   public void onKeyPress(KeyPressEvent event) {
      if (event.getCharCode() == KeyCodes.KEY_ENTER) {
         String newVal = visuCompteurTextBox.getText();
         cpt=Integer.parseInt(newVal);
      }
   }
});
```

- Modification du compteur lors de la réception d'un caractère *retouràlaligne*

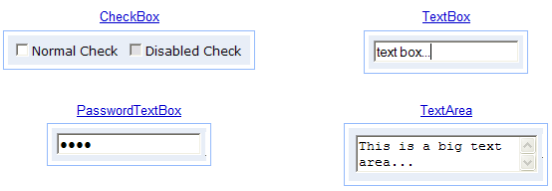
Démonstration

…

Galerie des composants - 1



Galerie des composants - 2



Galerie des composants - 3

Hyperlink

Info
Buttons
Menus
Images
Layouts

ListBox

List 0
List 0
List 1
List 2
List 3
List 4
foo0
bar0
baz0
toto0
tintin0

Galerie des composants - 4

MenuBar

Style Fruit Term
Bold
Italicized
More »
Code
~~Strikethrough~~
Underlined

Tree

foo@example.com
Inbox
Drafts
Templates

Galerie des composants - 5

Table

sender	email
markboland05	mark@example.com
Hollie Voss	hollie@example.com
boticario	boticario@example.com
Emerson Milton	emerson@example.com
Healy Colette	healy@example.com
Brigitte Cobb	brigitte@example.com
Elba Lockhart	elba@example.com

TabBar

1634 1640 1642 1662

Galerie des composants - 6

DialogBox

About the Mail Sample

Google

This sample app
construction of a
GWT's built-in w
see how easy it

Close

PopupPanel

Bob Saget
Ludwig von Beethoven
Richard Feynman
Alan
John

Richard Feynman
richard@example.com

Galerie des composants - 7

StackPanel

Mail
Tasks
Get groceries
Walk the dog
Contacts

HorizontalPanel

0 1 2

Galerie des composants - 8

VerticalPanel

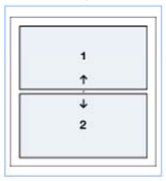
0
1
2

FlowPanel

0 1 2
3 4

Galerie des composants - 9

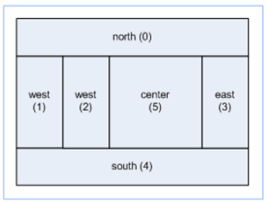
VerticalSplitPanel



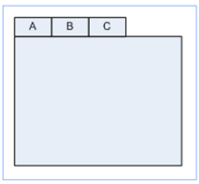
HorizontalSplitPanel

Galerie des composants - 10

DockPanel

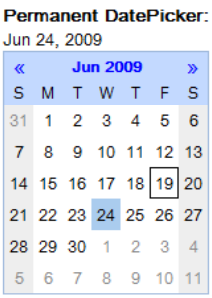


TabPanel



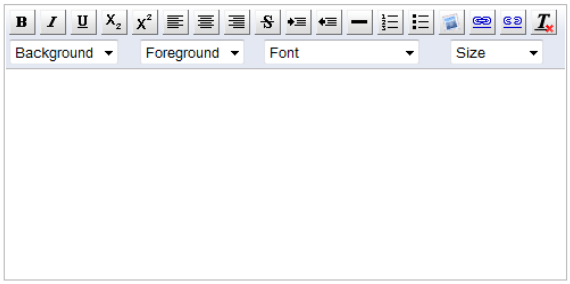
Choisir une date

- DatePicker



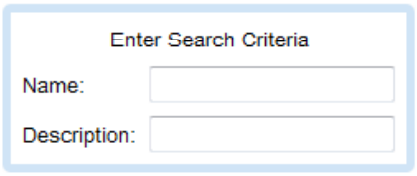
Texte riche

- RichTextArea



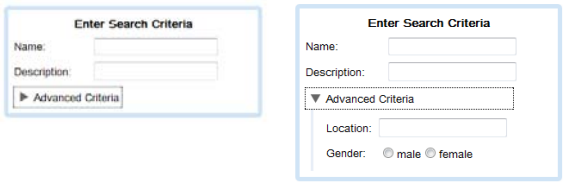
Panneau décoré

- DecoratorPanel

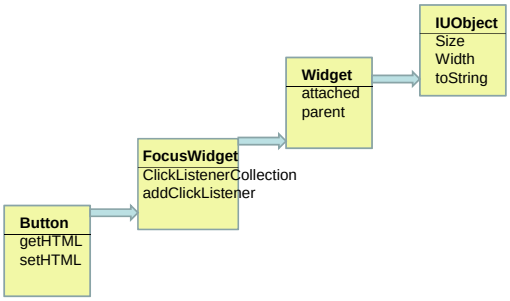


Panneau extensible

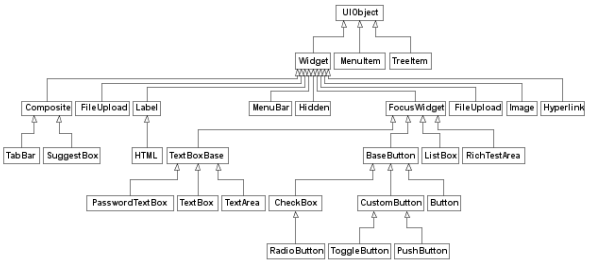
- DisclosurePanel



Héritage - exemple du bouton



Hiérarchie des composants



Fabrication d'une table

```
FlexTable table = new FlexTable();
table.setText(0, 0, "Nom");
table.setText(1, 0, "Prénom");
table.setText(2, 0, "Email");
table.setWidget(0,1, new TextBox());
table.setWidget(1,1, new TextBox());
table.setWidget(2,1, new TextBox());
```

Nom
Prénom
Email

Fabrication d'un arbre 1

```
class Demo extends Composite {
   public Demo() {
      Tree tree = new Tree();
      initWidget(tree);
      TreeItem outerRoot = new TreeItem("aaaa");
      outerRoot.addItem("bbbb");
      outerRoot.addItem("cccc");
      outerRoot.addItem("dddd");
      outerRoot.addItem(new CheckBox("eeee"));
      tree.addItem(outerRoot);
...
```

- aaaa
  - bbbb
  - cccc
  - dddd
  - eeee
- ffff
  - gggg
  - hhhh
  - iiii

Fabrication d'un arbre 2

```
...
      TreeItem innerRoot = new TreeItem("ffff");
      innerRoot.addItem("gggg");
      innerRoot.addItem("hhhh");
      innerRoot.addItem(new CheckBox("iiii"));

      outerRoot.addItem(innerRoot);
   }
}
```

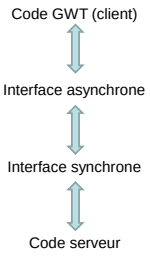
- aaaa
  - bbbb
  - cccc
  - dddd
  - eeee
- ffff
  - gggg
  - hhhh
  - iiii

Appel de méthodes à distance

RPC

Principe de fonctionnement

On demande un service mais on n'attend pas la réponse. Celle-ci viendra en son temps grâce à un « *objet de rappel* ».



Créer un service

1. Coté client, définir une interface dite synchrone qui étend `RemoteService` et qui liste toutes les méthodes RPC
2. Coté serveur, implémenter l'interface et étendre `RemoteServiceServlet`
3. Coté client, définir une interface asynchrone basée sur la première interface avec le même nombre de méthodes que la première interface

Interface synchrone coté client

- Spécification de vos services

```
import
  com.google.gwt.user.client.rpc.RemoteService;
public interface MyService extends
  RemoteService {
  public T myMethod(T1 p1, T2 p2, T3 p3);
}
```


Implémentation coté serveur

- Implémentation de vos services

```
public class MyServiceImpl extends
  RemoteServiceServlet implements
    MyService {

  public T myMethod(T1 p1, T2 p2, T2 p3) {
    // …
    return r;
  }
}
```


Interface asynchrone coté client

- Implémentation de vos services

```
interface MyServiceAsync {
  public void myMethod(T1 p1, T2 p2,
  T2 p3, AsyncCallback<T> callback);
}
```


Appel de méthodes asynchrones

- Un appel de méthode asynchrone nécessite de passer un objet de rappel (callback object)
- L'objet de rappel est notifié lorsque l'appel asynchrone est terminé
- En attendant l'appelant continu son activité
- Remarquez que les méthodes asynchrones ont un type résultat void

Conventions de nommage

- Nom des classes et interfaces
  - `MyService` interface synchrone
  - `MyServiceImpl` implementation
  - `MyServiceAsync` interface asynchrone
- Chaque méthode dans l'interface synchrone doit avoir une méthode correspondante dans l'interface asynchrone avec un paramêtre extra en dernier argument de type AsyncCallback

Conventions de nommage - bis

- Méthode de l'interface synchrone
- `public Integer[] myMethod(String s, int i)`
- Méthode de l'interface asynchrone
- `public void myMethod(String s, int i, AsyncCallback<Integer[]> callback)`

Utilisation de services

- Créer une instance de service

```
MonServiceAsync monService = GWT
          .create(MonService.class);
```

- Appel de méthode à distance

```
monService.maMethode(p1,p2, new
  AsyncCallBack<T>() {
  public onFailure(Trowable t) {...}
  public onSuccess(T result) {...}
})
```


Chemin d'accès à un service

- Le chemin d'accès à un service est composé de deux parties
  - `/nomDuModule/nomDuService`
- Le nom du service peut être défini de deux manières
  - Statiquement par annotation dans l'interface synchrone

```
@RemoteServiceRelativePath("service")
```

  - Dynamiquement

```
ServiceAsync service = GWT.create(Service.class);
ServiceDefTarget test = (ServiceDefTarget) service
test.setServiceEntryPoint("/module/service");
```


Chemin d'accès à un service - bis

- Les déclarations de path pour la servlet doivent être cohérentes
  - Dans le descripteur de module

```
<servlet class= "MaServlet"
     path="/module/service"/>
```

  - Dans le descripteur web.xml

```
<url-pattern>/module/service</url-
 pattern>
```

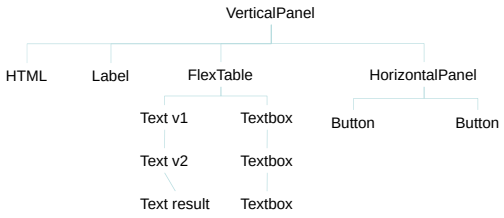

Un second exemple

La calculette déportée

Interface utilisateur

Deux opérations `addition` et `multiplication` déportées sur le serveur

Structure de l'interface



Déclaration des composants

```
final VerticalPanel mainPanel = new VerticalPanel();
final TextBox v1Field = new TextBox();
final TextBox v2Field = new TextBox();
final TextBox resultField = new TextBox();
final HorizontalPanel lesBoutons = new
  HorizontalPanel();
final FlexTable lesFields = new FlexTable();
final HTML titre = new HTML("<h1>Calculette
  déportée</h1>");
final Label invit = new Label("Entrez deux nombres
  puis cliquez sur un des boutons");
```


Remplissage du conteneur vertical

```
mainPanel.setSpacing(5);
mainPanel.setHorizontalAlignment(
   HasHorizontalAlignment.ALIGN_CENTER);
mainPanel.add(titre);
mainPanel.add(invit);
mainPanel.add(lesFields);
mainPanel.add(lesBoutons);
```


Remplissage de la table

```
lesFields.setText(0,0,"V1");
lesFields.setText(1,0,"V2");
lesFields.setText(2,0,"Résultat");
lesFields.setWidget(0,1,v1Field);
lesFields.setWidget(1,1,v2Field);
lesFields.setWidget(2,1,resultField);
resultField.setEnabled(false);
```


Mise en place des boutons

```
final Button addButton = new Button("+");
addButton.setWidth("4em");
final Button mulButton = new Button("*");
mulButton.setWidth("4em");
lesBoutons.setSpacing(10);
lesBoutons.add(addButton);
lesBoutons.add(mulButton);
```

- Insertion dans la page HTML

```
RootPanel.get().add(mainPanel);
```

Déclaration du serveur coté client

```
private final CalculetteServiceAsync
  calculetteService = GWT
          .create(CalculetteService.class);
```


Mise en place des objets de rappel pour l'additionneur

```
class AddHandler implements ClickHandler {
  public void onClick(ClickEvent event) {
    sendValuesToServer();
  }
  private void sendValuesToServer() {
    int nb1 = Integer.parseInt(v1Field.getText());
    int nb2 = Integer.parseInt(v2Field.getText());
    calculetteService.addServer(nb1, nb2,
      new AsyncCallback<Integer>() {...
    });
  }
}
```


Mise en place des objets de rappel - suite

```
new AsyncCallback<Integer>() {
   public void onFailure(Throwable caught) {
   }
   public void onSuccess(Integer result) {

  resultField.setText(result.toString());
   }
}
```


Ajout des deux objets de rappel

```
addButton.addClickHandler(new AddHandler());
mulButton.addClickHandler(new MulHandler());
```


Implementation de l'interface coté serveur

```
public class CalculetteServiceImpl
   extends RemoteServiceServlet
   implements CalculetteService {
  public Integer addServer(int nb1, int nb2) {
     return nb1+nb2;
  }
  public Integer mulServer(int nb1, int nb2) {
     return nb1*nb2;
  }
}
```


VALIDATION

Verifier (1)

```
public class MaxlengthVerifier {
  private int taille;
  public MaxlengthVerifier(int taille) {
     this.taille = taille;
  }
  public boolean isValid(String name) {
     if (name == null || name.length()==0) {
          return false;
     }
     return name.length() <= taille;
  }
}
```


Verifier (2)

```
private final Label errorLabel = new Label();
private static final MaxlengthVerifier verif25
  = new MaxlengthVerifier(25);

errorLabel.setText("");
if (!verif25.isValid(titreField.getText())) {
  errorLabel.setText("svp entre 1 et 25
  lettres dans le titre");
  return false;
}
```


Verifier (3)

- Mettre la procédure dans un répertoire `shared`
- Dans le fichier `monFic.gwt.xml`
  - `<source path="client"/>`
  - `<source path="shared"/>`

Démonstration

...

Notion de session

- On utilise les servlets qui implémentent les RPC

```
HttpServletRequest request =
  this.getThreadLocalRequest();
HTTPSession session =
  request.getSession();
o = session.getAttribute(key)
session.setAttribute(key, o);
```


Conclusion