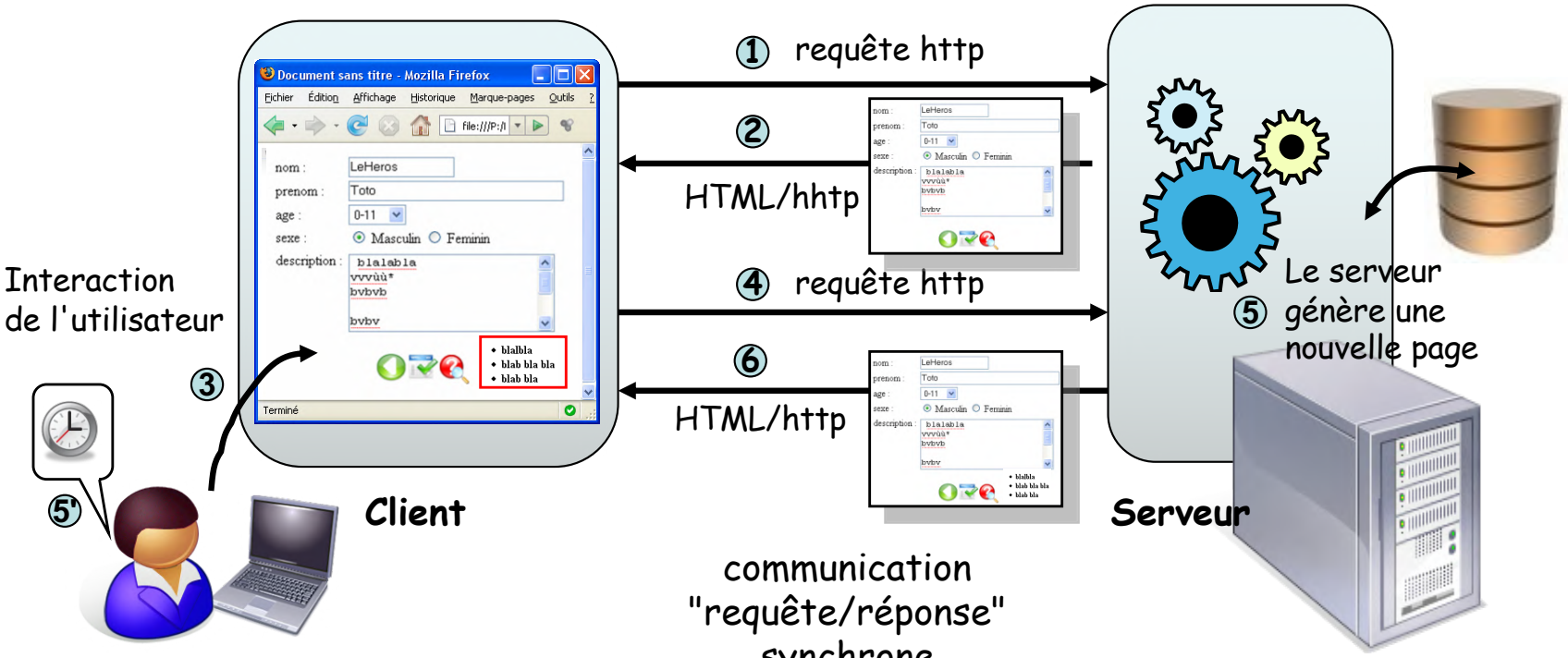
Web 2.0
Introduction à Ajax et à quelques technologies AJAX/Java
DWR (Direct Web toolkit)
Google Web Toolkit (GWT)

Caractéristiques des applications Web "Classiques"

- Navigateur outil générique d'affichage : il n'a aucune intelligence de l'application
- Logique de navigation sous forme d'enchaînement de pages est déterminée par le serveur.



Limites des applications Web "Classiques"

- Ergonomie en retrait
  - Contrainte par HTML
    - Ensemble limité de widgets
  - Pas de retour immédiat aux activités de l'utilisateur
    - L'utilisateur doit attendre la page suivante générée par le serveur
  - Interruption des activités de l'utilisateur
    - L'utilisateur ne peut effectuer d'autres opérations pendant qu'il attend une réponse
  - Perte du contexte opérationnel suite au rafraîchissement
    - Perte de la position de scrolling dans la page
    - Le cerveau doit réanalyser entièrement toute nouvelle page

Remédier aux limites du Web "Classique"

- Animation des écrans prise en charge du côté client
  - Animation d'un écran assurée par du code exécuté sur le navigateur
    - → limite les échanges navigateur/serveur web
    - → possibilité d'augmenter l'interactivité et de réaliser des comportements ergonomiques plus évolués
- Optimisation des échanges navigateur/serveur
  - **Communication asynchrone**
    - Lorsqu'une requête est émise par le client, celui-ci reprend la main immédiatement
  - Echange des données plutôt que de la présentation **(une fois la page initiale chargée)**

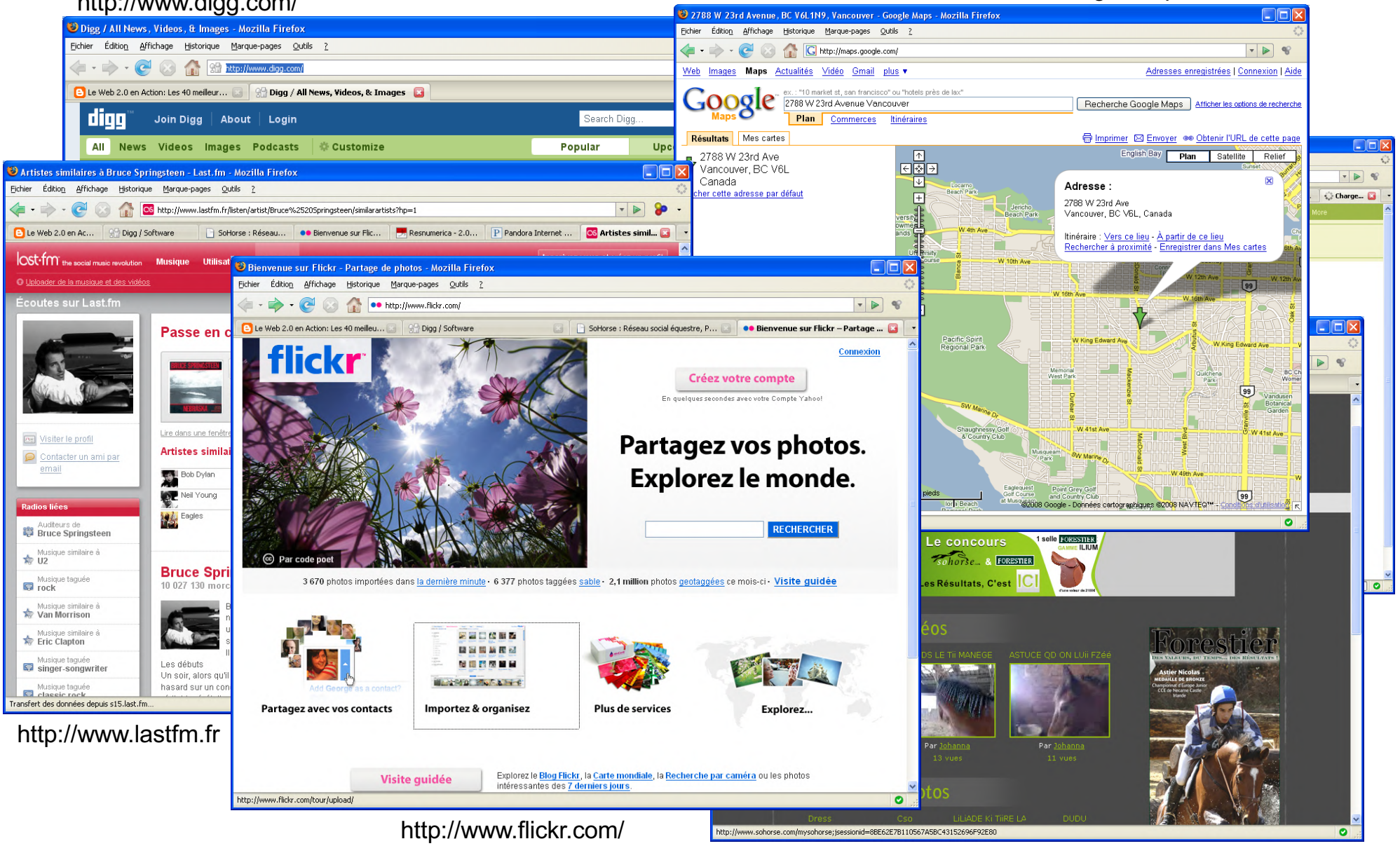
→Technologies RIA (Rich Internet Application)

→Web 2.0 versus Web 1.0

Exemples de sites web 2.0

http://www.digg.com/

Google maps



http://www.lastfm.fr

http://www.flickr.com/

Technologies RIA (Rich Internet Applications)

- De nombreuses technologies (pas toutes récentes)
  - Applets (1995)
    - Code java
  - DHTML (Dynamic HTML)
    - Javascript (1995) + DOM + CSS
  - Flash/FLEX (1996, 2004) Macromedia-Adobe
    - MXML + ActionScript
  - Silverligth (2006) Microsoft
    - XAML + ECMACScript
  - JavaFX (2008) Sun …
  - AJAX : Asynchronous Javascript And XML
    - AJAX = Javascript + XmlhttpRequest
      - Code client écrit en Javascript
      - Communications avec le serveur réalisées à l'aide de l'objet Javascript **`XmlhttpRequest`**

Technologie anciennes :
Javascript 1995
XmlHttpRequest : 1999 dans IE5

Ce qui est nouveau est leur utilisation conjointe

Technologies utilisées en AJAX

AJAX = Javascript + XmlhttpRequest

plus précisément

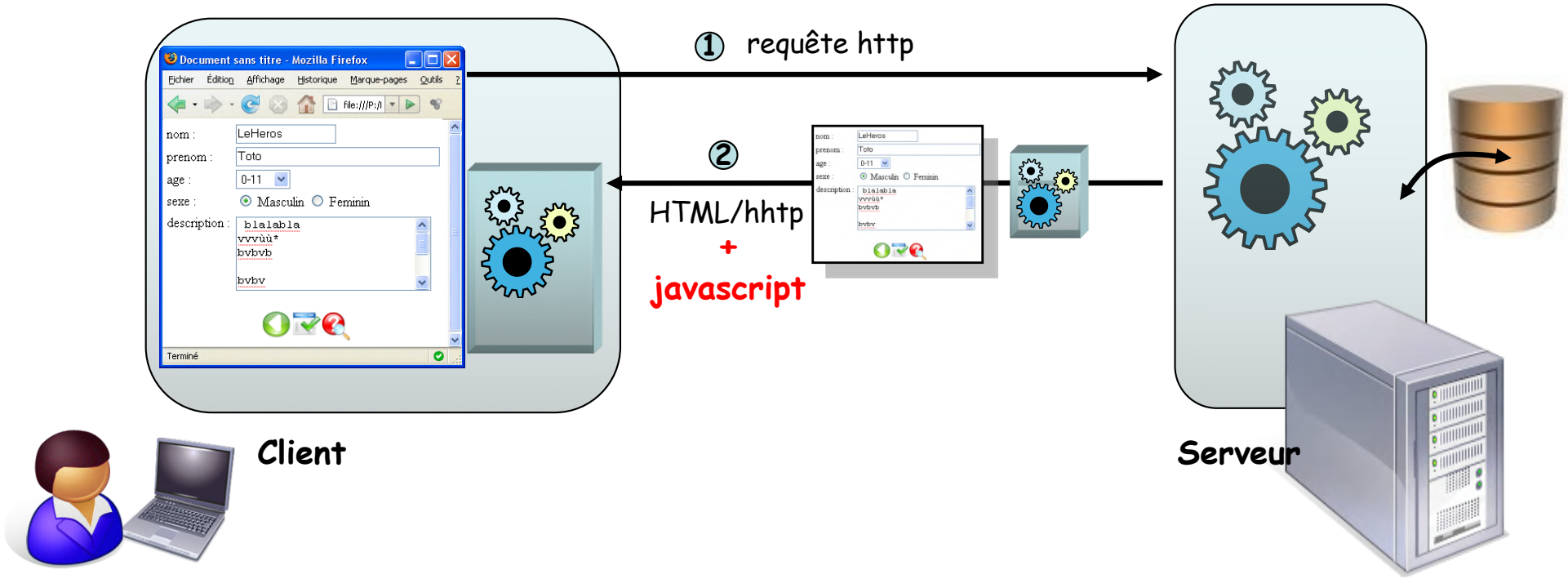
AJAX = DHTML (DOM + CSS + Javascript) + XmlhttpRequest

- Javascript
  - **Langage de script orienté objet et faiblement typé**
  - **Fonctions javascript invoquées lorsque intervient un événement sur la page**
  - **"Glue" pour tout le fonctionnement d'AJAX**
- DOM (Document Object Model)
  - **API pour accéder à des documents structurés**
  - **Représente la structure de documents XML et HTML**
- CSS (Cascading Style Sheets)
  - **Permet un séparation claire du contenu et de la forme de la présentation**
  - **Peut être modifié par le code Javascript**
- XmlhttpRequest
  - **Objet Javascript qui assure une interaction asynchrone avec le serveur**

Application Web AJAX

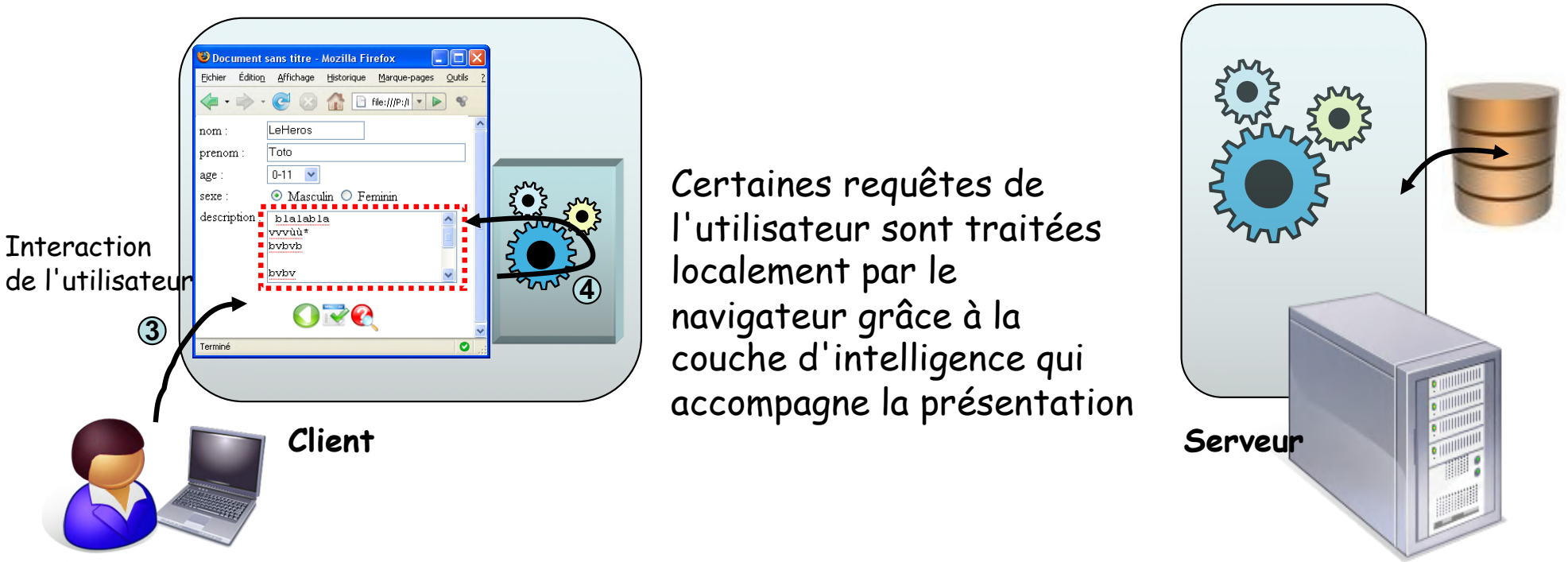
- Une partie de l'intelligence fonctionnelle de l'application est déportée vers le navigateur

1er échange similaire au web "classique" : le serveur envoie une page au client mais en y embarquant de l'intelligence (code javascript)



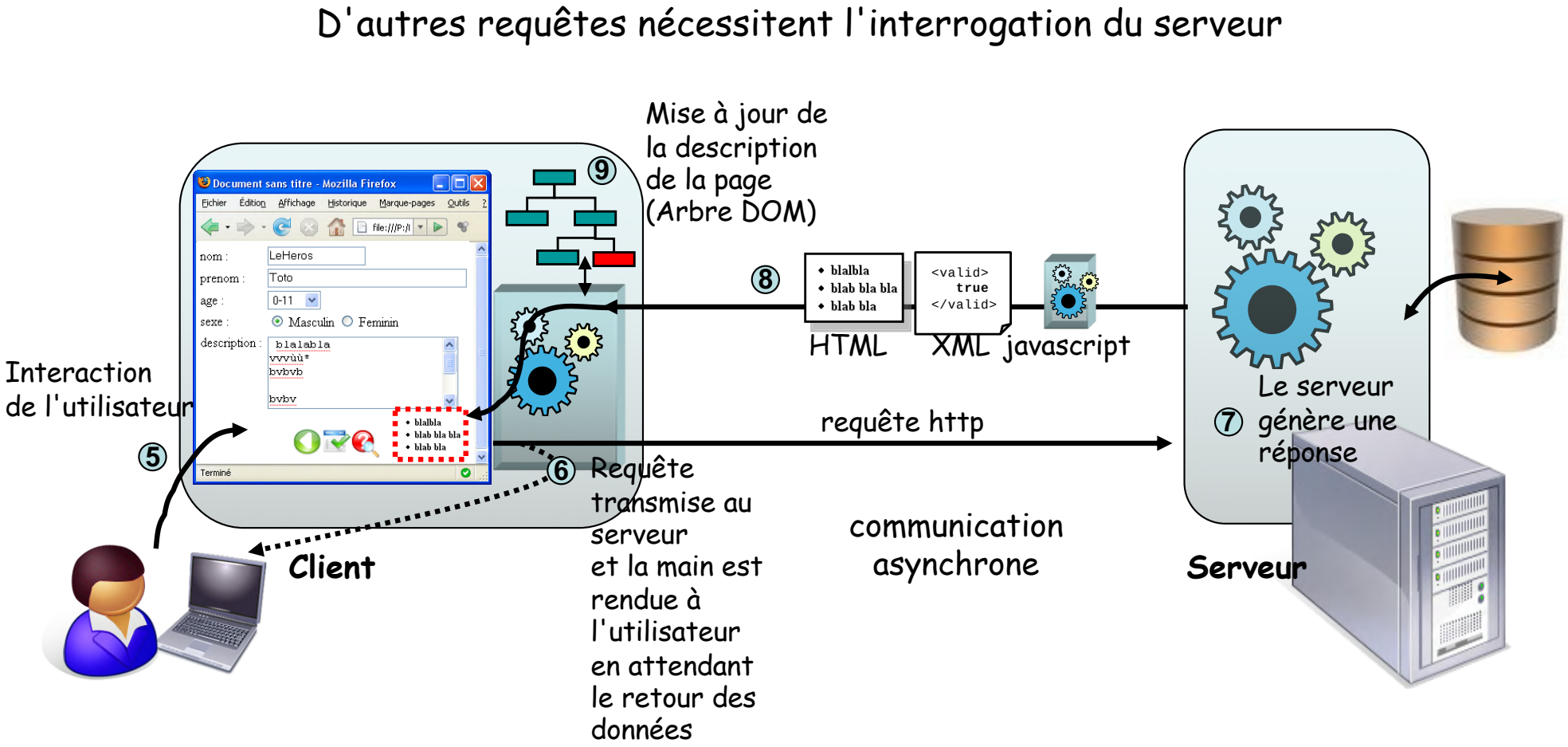
Application Web AJAX

- Une partie de l'intelligence fonctionnelle de l'application est déportée vers le navigateur



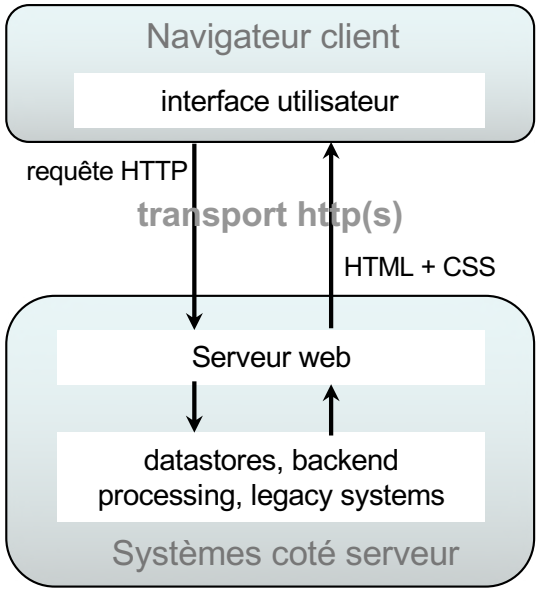
Application Web AJAX

- Une partie de l'intelligence fonctionnelle de l'application est déportée vers le navigateur

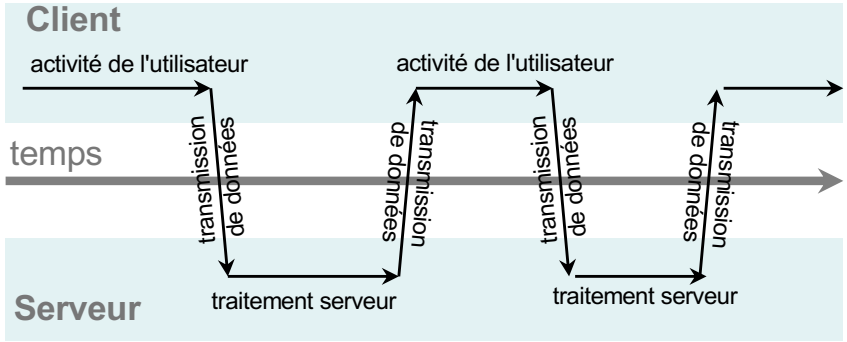


Modèles des applications WEB

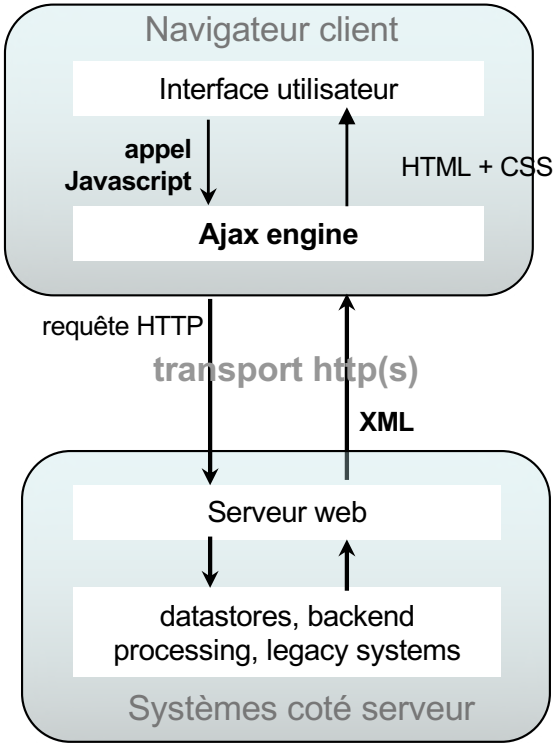
Application Web "classique"



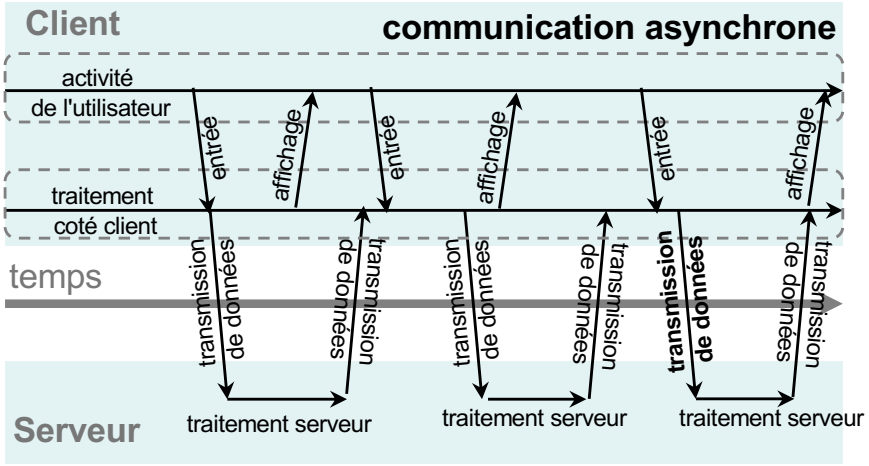
communication synchrone



Application Web AJAX



communication asynchrone



Modèles des applications WEB

Application Web "classique"

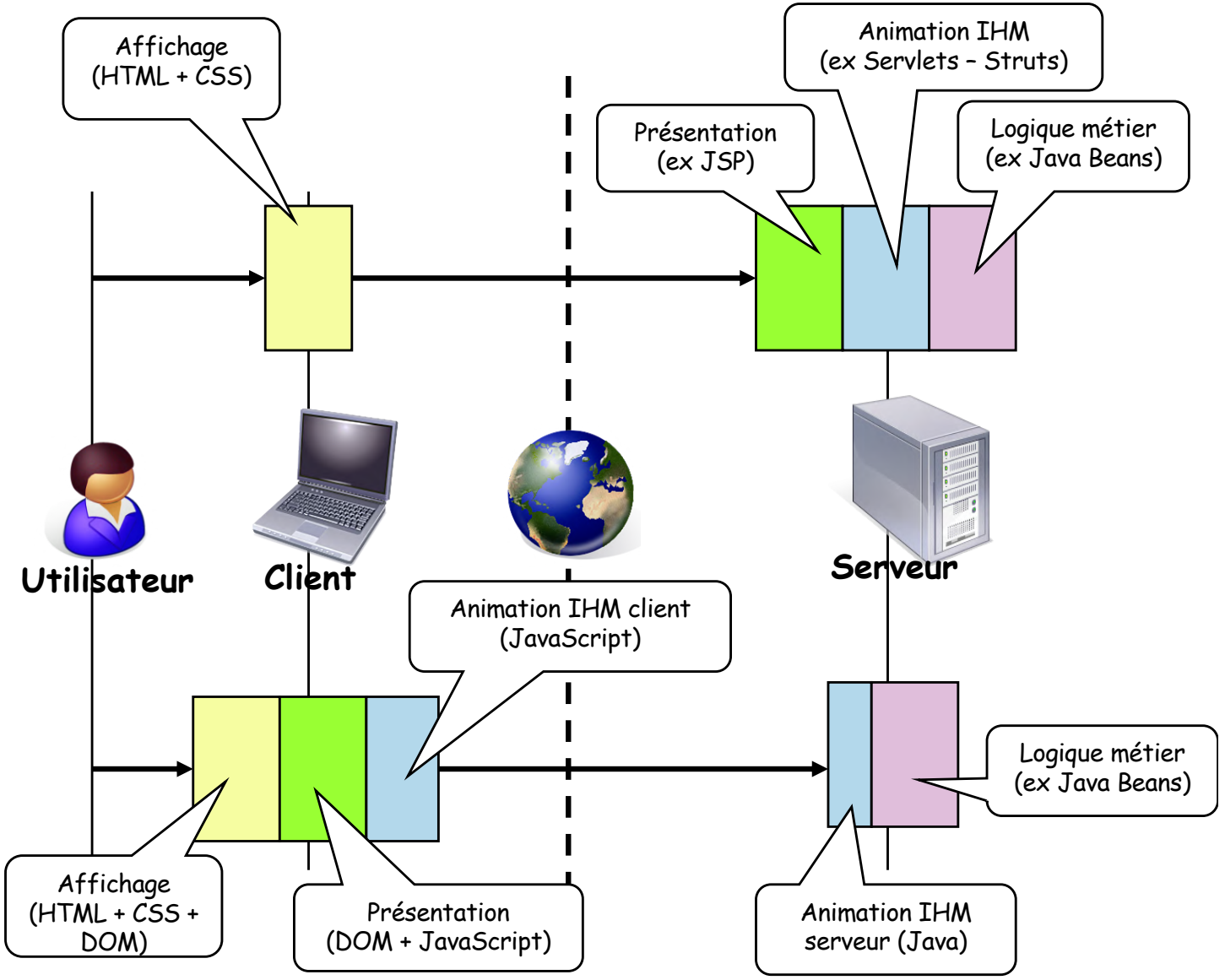
- Peu de javascript
- Charge serveur très importante
- Ergonomie faible

Application Web AJAX Léger

- Cohabitation entre technologie "classique" (ASP, PHP, JSP) et AJAX
- Charge serveur importante
- Ergonomie améliorée

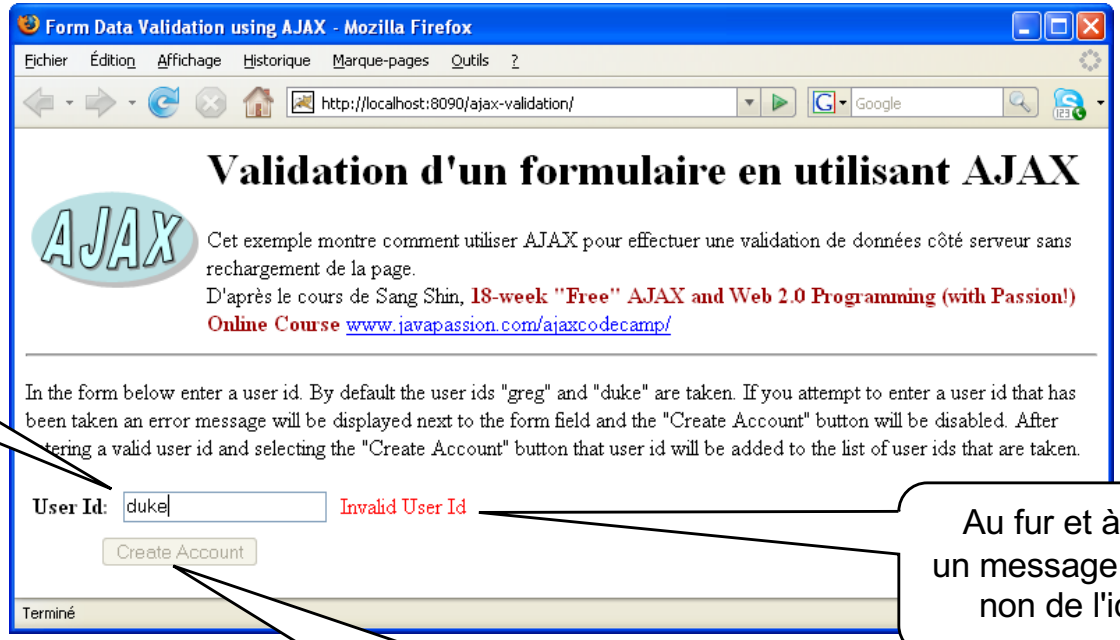
Application Web AJAX complet

- Client en javascript
- Charge serveur modérée
- Ergonomie supérieure



Anatomie d'une interaction AJAX

- Exemple d'après : *AJAX Basics and Development Tools* de Sang Shin (sang.shin@sun.com, Sun Microsystems) www.javapassion.com/ajaxcodecamp

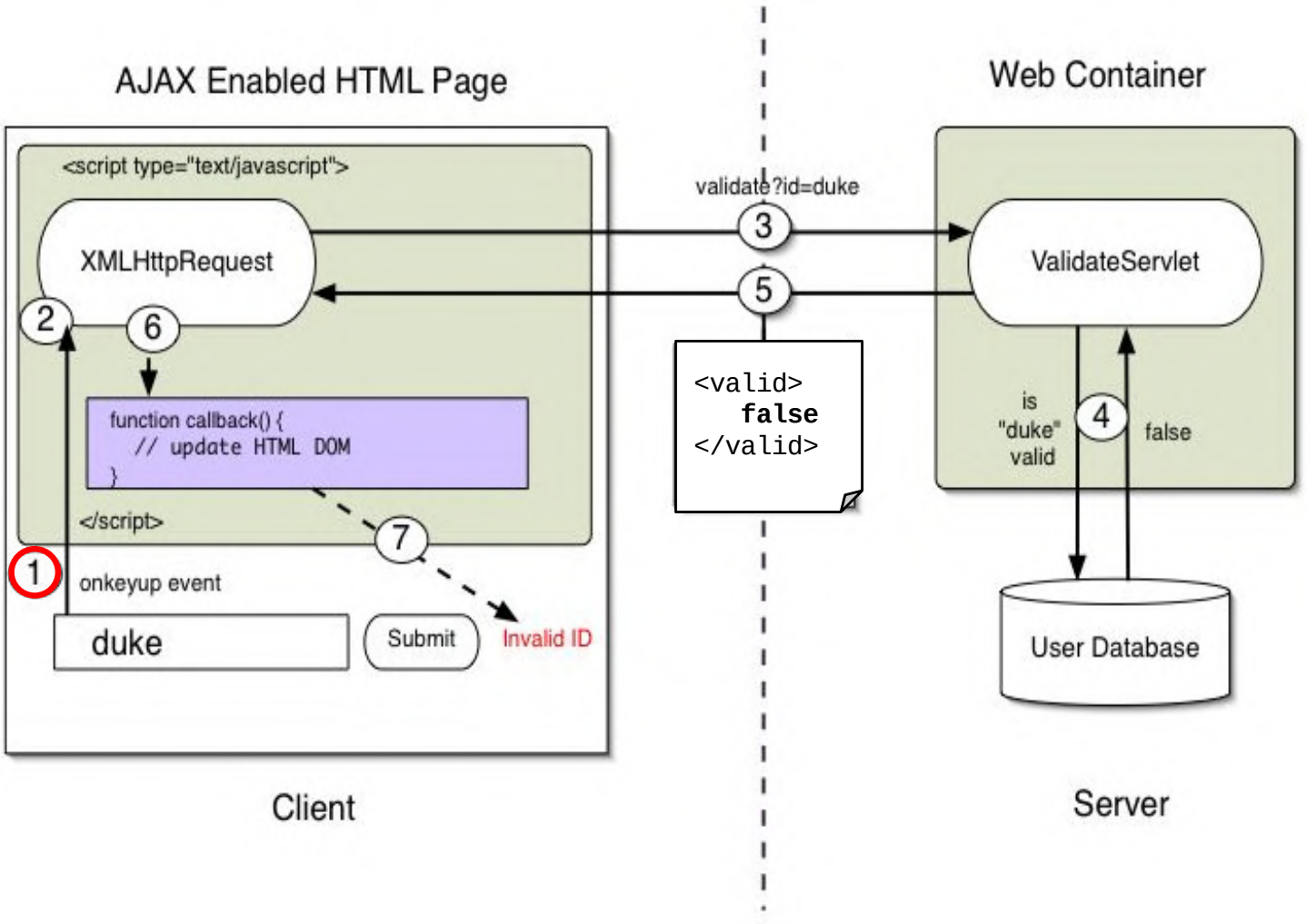


L'utilisateur saisit un identifiant

Au fur et à mesure de la frappe un message indiquant la validité ou non de l'identifiant est affiché

Le bouton de création n'est activé que si l'identifiant est valide (n'est pas déjà utilisé)

Anatomie d'une interaction AJAX



① Événement sur le client → appel d'une fonction javascript

② Création et configuration d'un objet XMLHttpRequest

③ L' objet XMLHttpRequest fait une requête asynchrone

④ Le servlet valide l'identifiant soumis

⑤ Le servlet retourne un document XML contenant le résultat de la validation

⑥ L'objet XMLHttpRequest appelle la fonction callback() et traite ce résultat

⑦ Mise à jour de la page HTML (DOM)

Anatomie d'une interaction AJAX

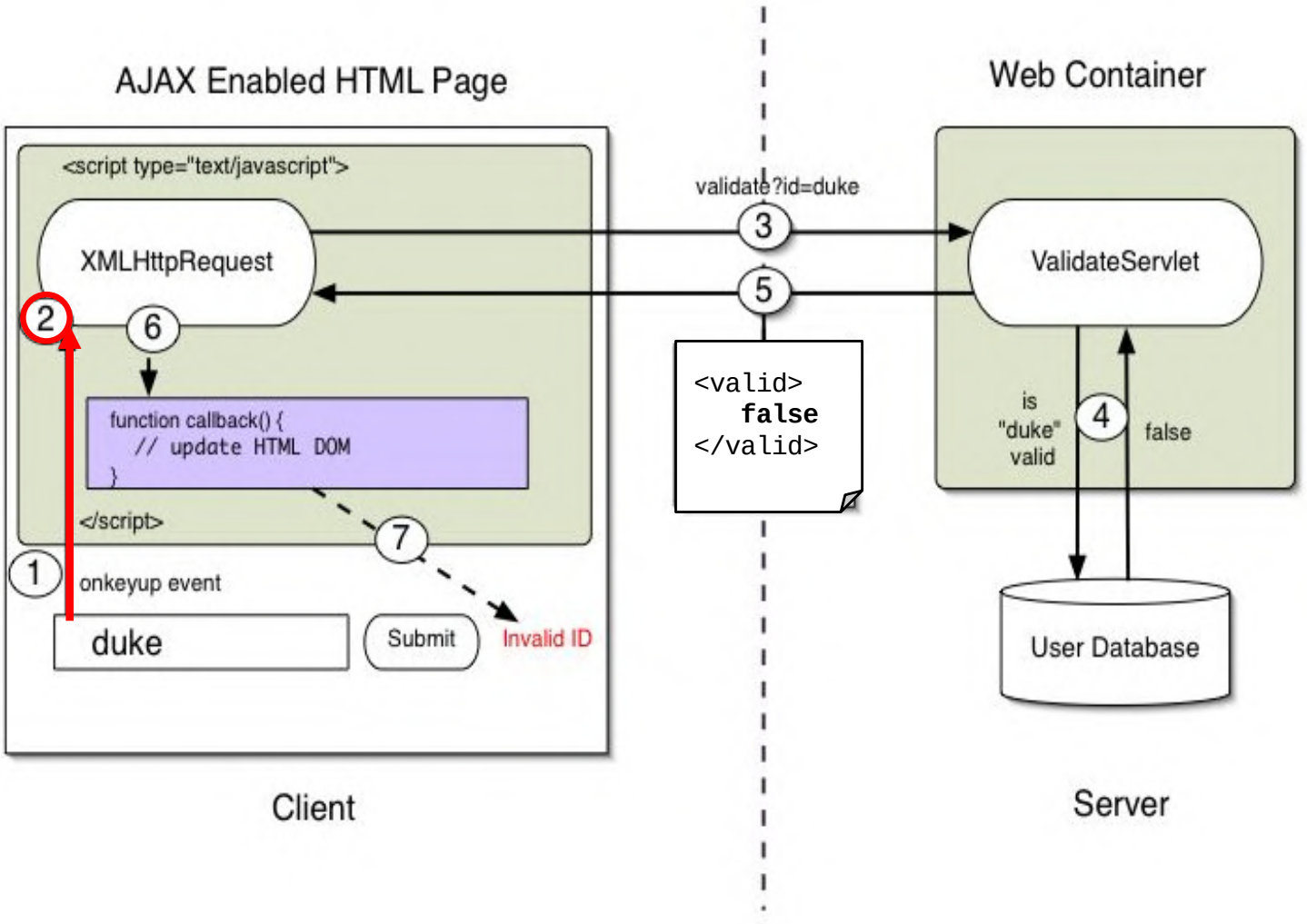
1 Gestion des événements dans le formulaire HTML

La fonction Javascript validateUserId est associée au champ de saisie de texte "userid" pour la gestion des événements de type onkeyup : validateUserId est appelée chaque fois que l'utilisateur tape une lettre dans le champ de saisie.


```
<form name="updateAccount" action="validate" method="post">
    <input type="hidden" name="action" value="create"/>
    <table border="0" cellpadding="5" cellspacing="0">
        <tr>
            <td><b>User Id:</b></td>
            <td>
                <input type="text" size="20" id="userid" name="id" onkeyup="validateUserId()">
            </td>
            <td>
                <div id="userIdMessage"></div>
            </td>
        </tr>
        <tr>
            <td align="right" colspan="2">
                <div align="center">
                  <input id="submit_btn" type="Submit" value="Create Account">
                </div></td>
            <td></td>
        </tr>
    </table>
</form>
```

L'élément div d'id userIdMessage spécifie la position où sera affiché le message de validation de l'entrée

Anatomie d'une interaction AJAX



① Événement sur le client → appel d'une fonction javascript

② Création et configuration d'un objet XMLHttpRequest

③ L' objet XMLHttpRequest fait une requête asynchrone

④ Le servlet valide l'identifiant soumis

⑤ Le servlet retourne un document XML contenant le résultat de la validation

⑥ L'objet XMLHttpRequest appelle la fonction callback() et traite ce résultat

⑦ Mise à jour de la page HTML (DOM)

Anatomie d'une interaction AJAX

Coté client :

la fonction JavaScript invoqué à chaque événement "keyup"sur le champ de saisie

2

```
var req;

function validateUserId() {

    if (window.XMLHttpRequest) {
          req = new XMLHttpRequest();
   } else if (window.ActiveXObject) {
          isIE = true;
          req = new ActiveXObject("Microsoft.XMLHTTP");
   }

   req.onreadystatechange = processRequest;

   if (!target)
       target = document.getElementById("userid");
   var url = "validate?id=" + escape(target.value);

   req.open("GET", url, true);


}
```

Création et configuration d'un objet `XMLHttpRequest`

Selon le navigateur l'objet XMLHttpRequest est crée différemment

fonction callback : fonction Javascript (voir plus loin) qui sera invoquée lorsque le serveur aura fini de traiter la requête :

En Javascript les fonctions sont des objets et peuvent être manipulées en tant que tels

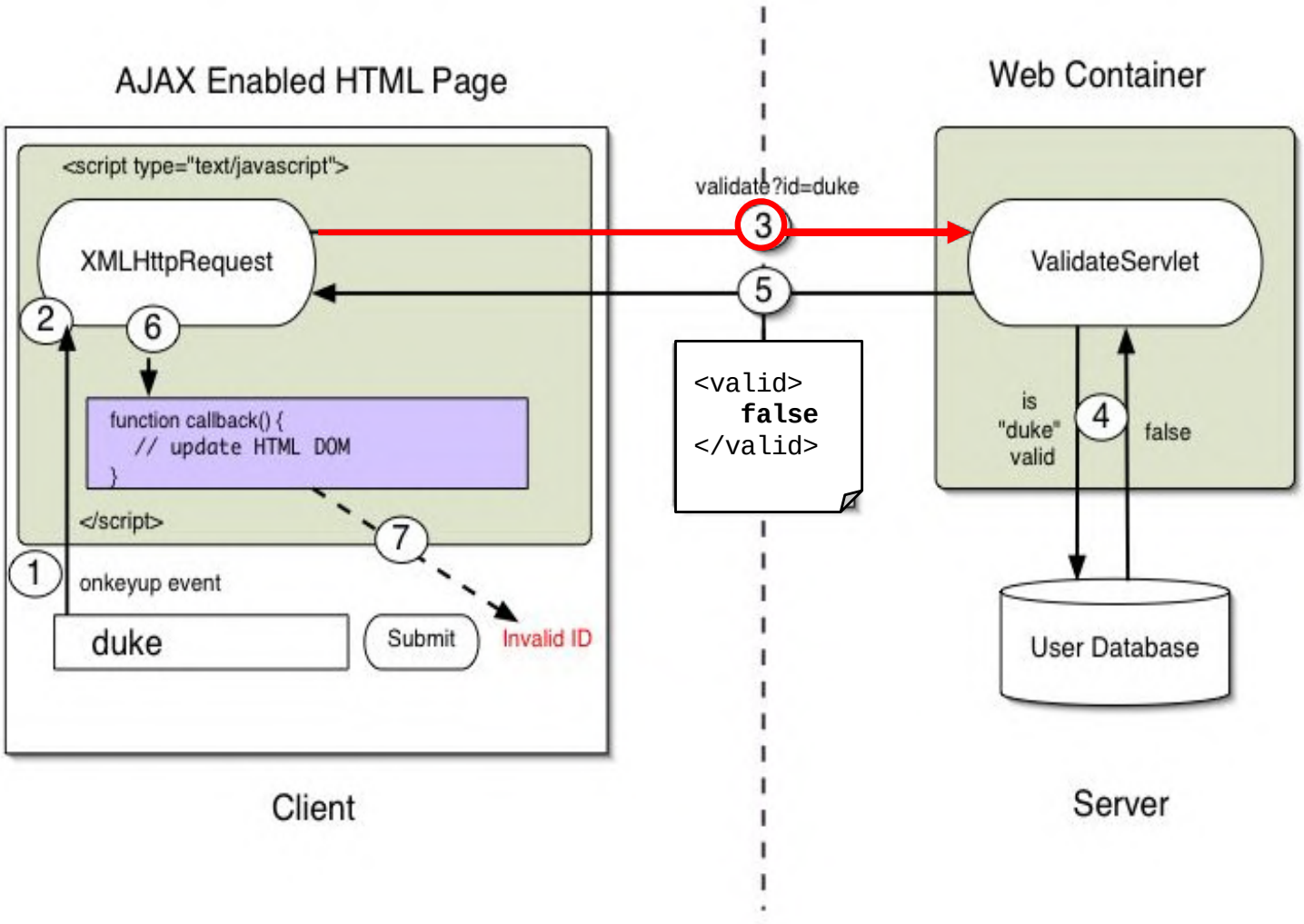
Récupération de la valeur userid tapée par l'utilisateur (via API DOM)

```
<input type="text" size="20" id="userid"
    name="id" onkeyup="validateUserId()">
```

et construction de l'url du composant serveur qui sera invoqué

L'appel sera asynchrone

Anatomie d'une interaction AJAX



① Événement sur le client → appel d'une fonction javascript

② Création et configuration d'un objet XMLHttpRequest

③ L' objet XMLHttpRequest fait une requête asynchrone

④ Le servlet valide l'identifiant soumis

⑤ Le servlet retourne un document XML contenant le résultat de la validation

⑥ L'objet XMLHttpRequest appelle la fonction callback() et traite ce résultat

⑦ Mise à jour de la page HTML (DOM)

Anatomie d'une interaction AJAX

Coté client :

la fonction JavaScript invoqué à chaque événement "keyup"sur le champ de saisie

```
var req;

function validateUserId() {

    if (window.XMLHttpRequest) {
          req = new XMLHttpRequest();
   } else if (window.ActiveXObject) {
          isIE = true;
          req = new ActiveXObject("Microsoft.XMLHTTP");
   }

   req.onreadystatechange = processRequest;

   if (!target)
       target = document.getElementById("userid");
   var url = "validate?id=" + escape(target.value);

   req.open("GET", url, true);

    req.send(null);
}
```

2

Création et configuration d'un objet `XMLHttpRequest`

Selon le navigateur l'objet XMLHttpRequest est crée différemment

fonction callback : fonction Javascript (voir plus loin) qui sera invoquée lorsque le serveur aura fini de traiter la requête :

En Javascript les fonctions sont des objets et peuvent être manipulées en tant que tels

Récupération de la valeur userid tapée par l'utilisateur (via API DOM)

```
<input type="text" size="20" id="userid"
    name="id" onkeyup="validateUserId()">
```

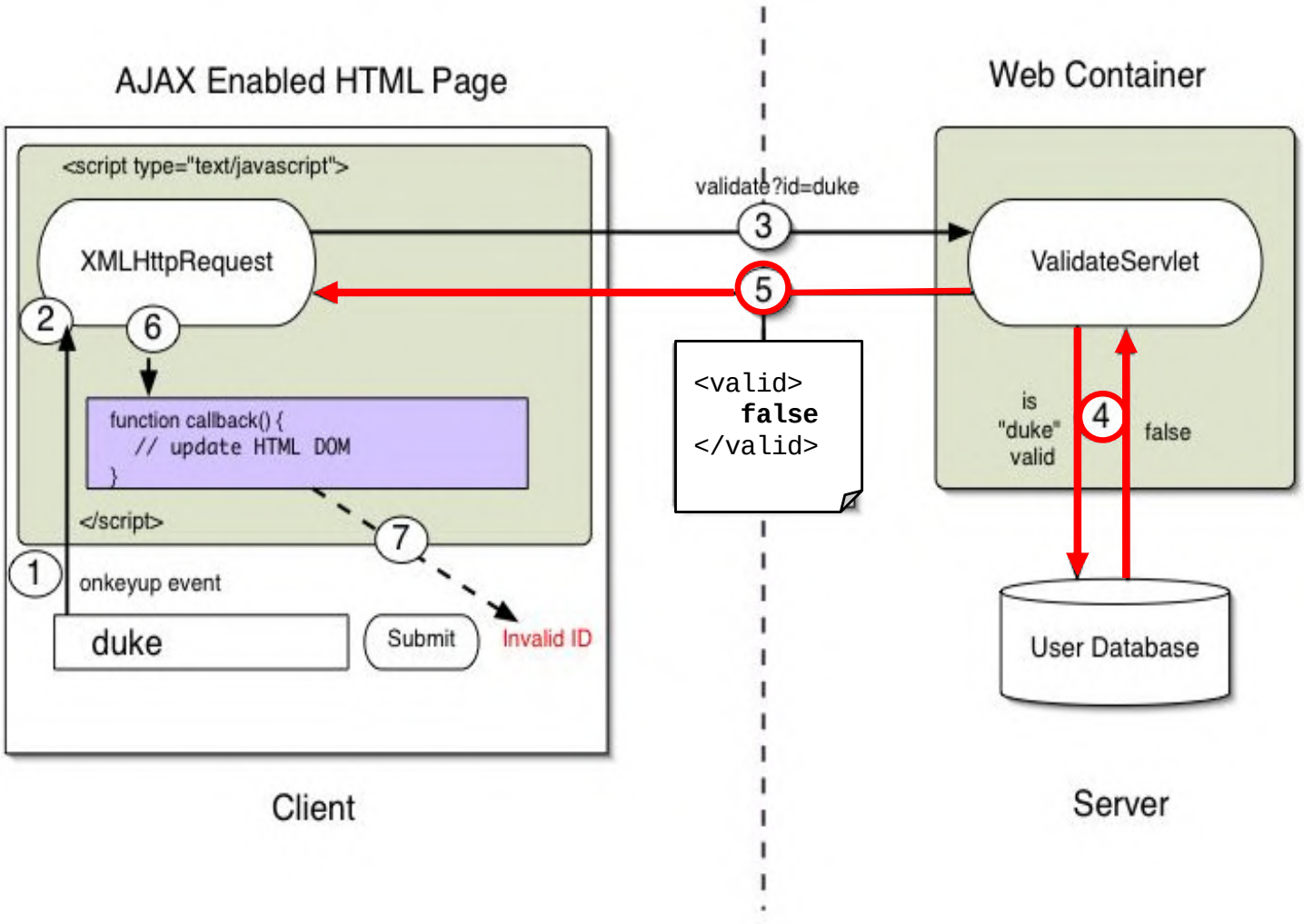
et construction de l'url du composant serveur qui sera invoqué

L'appel sera asynchrone

3

L' objet `XMLHttpRequest` effectue une requête asynchrone

Anatomie d'une interaction AJAX



① Événement sur le client → appel d'une fonction javascript

② Création et configuration d'un objet XMLHttpRequest

③ L' objet XMLHttpRequest fait une requête asynchrone

④ Le servlet valide l'identifiant soumis

⑤ Le servlet retourne un document XML contenant le résultat de la validation

⑥ L'objet XMLHttpRequest appelle la fonction callback() et traite ce résultat

⑦ Mise à jour de la page HTML (DOM)

Anatomie d'une interaction AJAX

Coté Serveur :

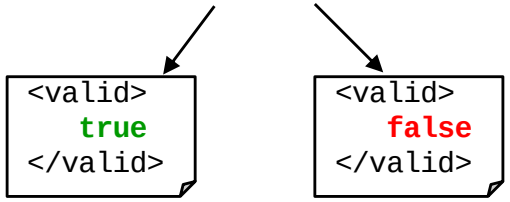
la servlet traitant la requête GET émise par la fonction JavaScript `validateUuserId`

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse  response)
        throws IOException, ServletException {

    String targetId = request.getParameter("id");

    if ((targetId != null) &&
          LoginManager.validateUserId(targetId.trim())) {

        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>true</valid>");
    } else {

        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>false</valid>");
    }
}
```

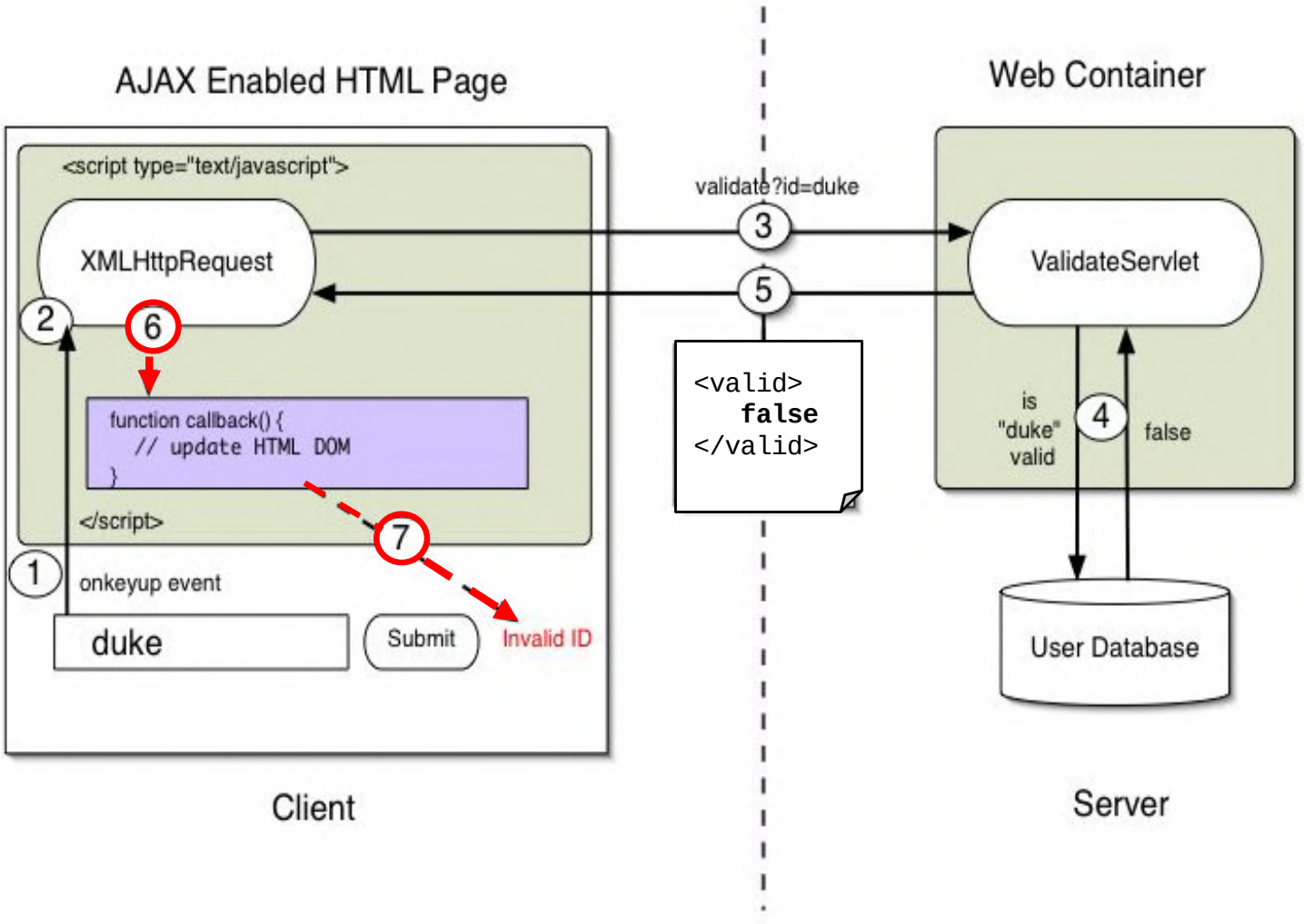
4 **Le servlet valide l'identifiant soumis**

S'agit-il d'un identifiant déjà utilisé ?

5 **Le servlet retourne un document XML contenant le résultat de la validation**



Anatomie d'une interaction AJAX



① Événement sur le client → appel d'une fonction javascript

② Création et configuration d'un objet XMLHttpRequest

③ L' objet XMLHttpRequest fait une requête asynchrone

④ Le servlet valide l'identifiant soumis

⑤ Le servlet retourne un document XML contenant le résultat de la validation

⑥ L'objet XMLHttpRequest appelle la fonction callback() qui traite ce résultat

⑦ Mise à jour de la page HTML (DOM)

Anatomie d'une interaction AJAX

Coté client :

6 L'objet `XMLHttpRequest` appelle la fonction callback() et traite ce résultat

```
function processRequest() {

   if (req.readyState == 4) {
      if (req.status == 200) {


         var message = req.responseXML.
               getElementsByTagName("valid")[0].
               childNodes[0].nodeValue;


         setMessageUsingDOM(message);


          var submitBtn = document.getElementById("submit_btn");
          if (message == "false") {
               submitBtn.disabled = true;
          } else {
               submitBtn.disabled = false;
          }
      }
   }
}
```

```
<valid>
   false
</valid>
```

Cette fonction est invoquée chaque fois que le champ readyState de l'objet XMLHttpRequest est modifié

readyState == 4 et status = 200 indiquent que la réponse a été correctement reçue par le client

Extraction de la valeur true ou false des données retournées par le serveur

7 Mise à jour de la page HTML (DOM)

Affiche dans la zone prévue à cet effet la validité ou non de l'identificateur fourni

Active ou désactive le bouton de soumission du formulaire selon validité de l'identificateur fourni

Anatomie d'une interaction AJAX

7 Mise à jour de la page HTML (DOM)

```
<td>
   <div id="userIdMessage"></div>
</td>
```

Invalid User Id

```
function setMessageUsingDOM(message) {

   var userMessageElement = document.getElementById("userIdMessage");

   var messageText;
   if (message == "false") {
      userMessageElement.style.color = "red";
      messageText = "Invalid User Id";
   } else {
      userMessageElement.style.color = "green";
      messageText = "Valid User Id";
   }

   var messageBody = document.createTextNode(messageText);

   if (userMessageElement.childNodes[0]) {
       userMessageElement.replaceChild(
              messageBody, userMessageElement.childNodes[0]);
   } else {
        userMessageElement.appendChild(messageBody);
   }
}
```

Récupération de l'objet DOM correspondant à la zone du message grâce à l'id inséré dans le code HTML

Préparation du message

Création du message
Si il existe déjà un message le remplace par le nouveau, sinon le rajoute

Comment faire de l'AJAX ?

- Une multitude de solutions pour faire de l'AJAX. Plus de 210 outils dénombrés en mai 2007
  (source http://ajaxian.com/archives/210-ajax-frameworks-and-counting)

Pure Javascript

Multipurpose	37
Remoting	19
Graphics and Effects	6
Flash	3
Logging	5
XML	6
Specialised	3
Subtotal	*79*

Server-Side

4D	1
C++	1
Coldfusion	4
Eiffel	0
DotNet (+ASP/C*)	19
Java	44
Lisp	1
Lotus Notes	2
Multi-Language	11
Perl	2
PHP	38
Python	5
Ruby	1
Smalltalk	1
Tcl	1
Subtotal	*131*

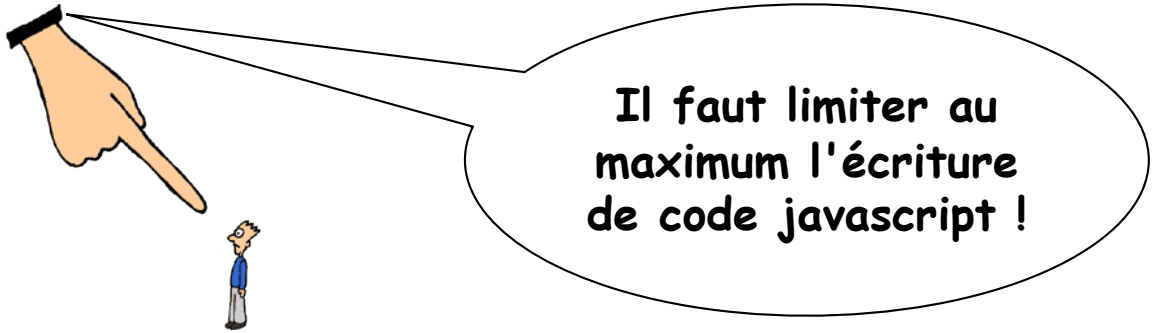
Moralité :

"If you're still rolling your own XMLHttpRequests or visual effects, now might be a good time to start investigating the alternatives".
Michael Mahemoff , Ajaxian, mai 2007

Développer en AJAX

- L'un des gros problèmes d'AJAX est le développement Javascript
  - Test et débogage limités
  - Portabilité difficile, il faut différentes versions des fonctions selon les navigateurs

Un outil pour FireFox 



- 1ère solution : utilisation des bibliothèques javascript
  - Prototype, Script.aculo.us, DOJO, Yahoo UI, JQuery
- 2ème solution : Utilisation des bibliothèque de tags, composants Struts ou JSF "ajaxifiés"
  - Les composants génèrent du javascript (AjaxTags, AjaxAnywhereRichFace, ajax4JSF, IceFaces…)
- 3ème solution : traduction d'un langage en Javascript
  - GWT (Google Web Toolkit) java → javascript

Introduction à DWR (Direct Web Remoting)

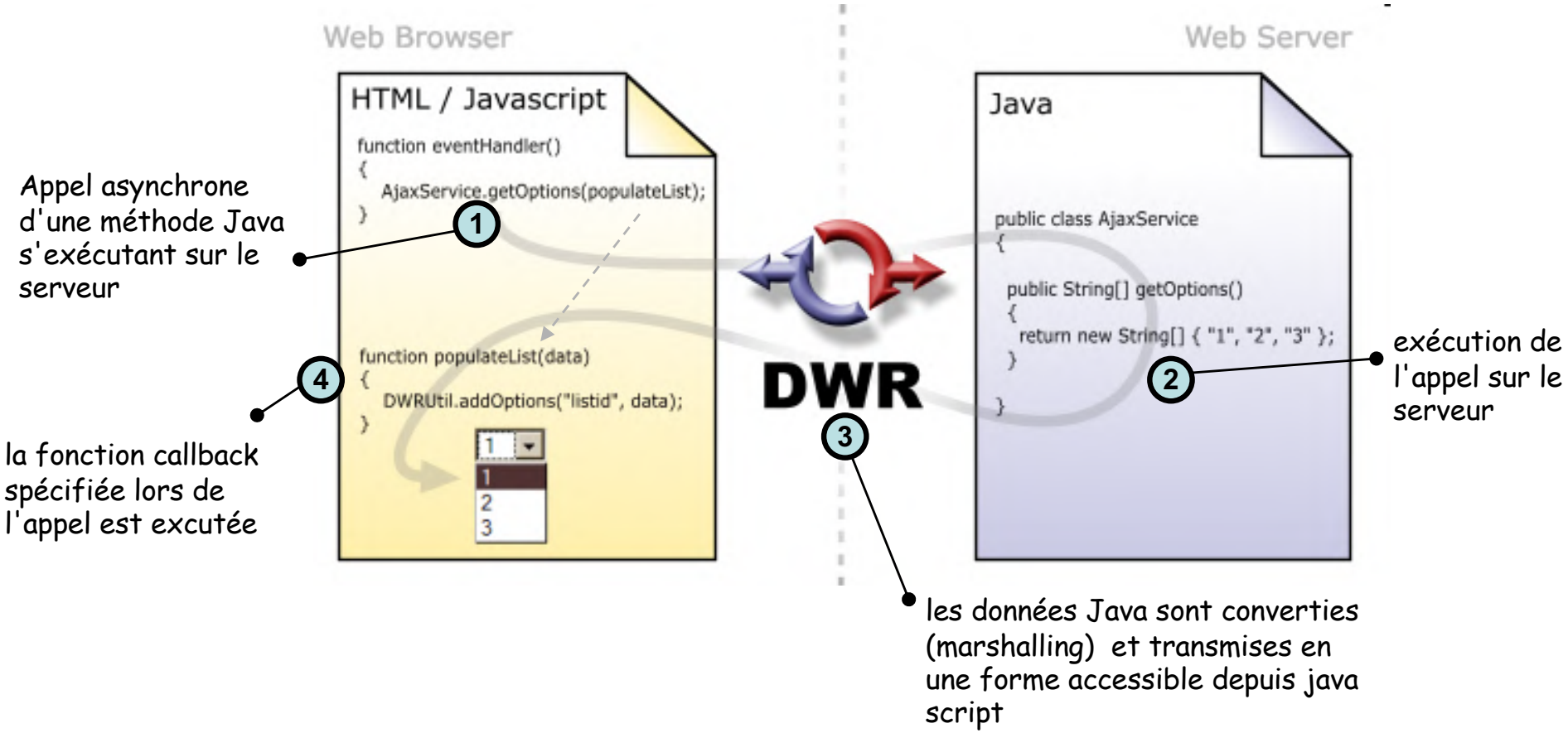
http://directwebremoting.org/

DWR c'est quoi ?

- DWR (Direct Web Remoting) bibliothèque open-source destinée à faciliter l'écriture d'applications AJAX
  - pas vraiment un technologie client ni une technologie serveur mais le pont ("glue") entre les deux.
    - masque les couches de "bas niveau" pour les échanges entre le client et le serveur via l'objet XMLHttpRequest
  - prend en charge les communications entre javascript (client) et java (serveur)
    - le code javascript peut invoquer "directement" des méthodes java s'exécutant depuis le serveur
    - reverse AJAX : code java s'exécutant sur le serveur peut utiliser l'API client pour mettre à jour le contenu du navigateur
  - quelques fonctions javascript utilitaires

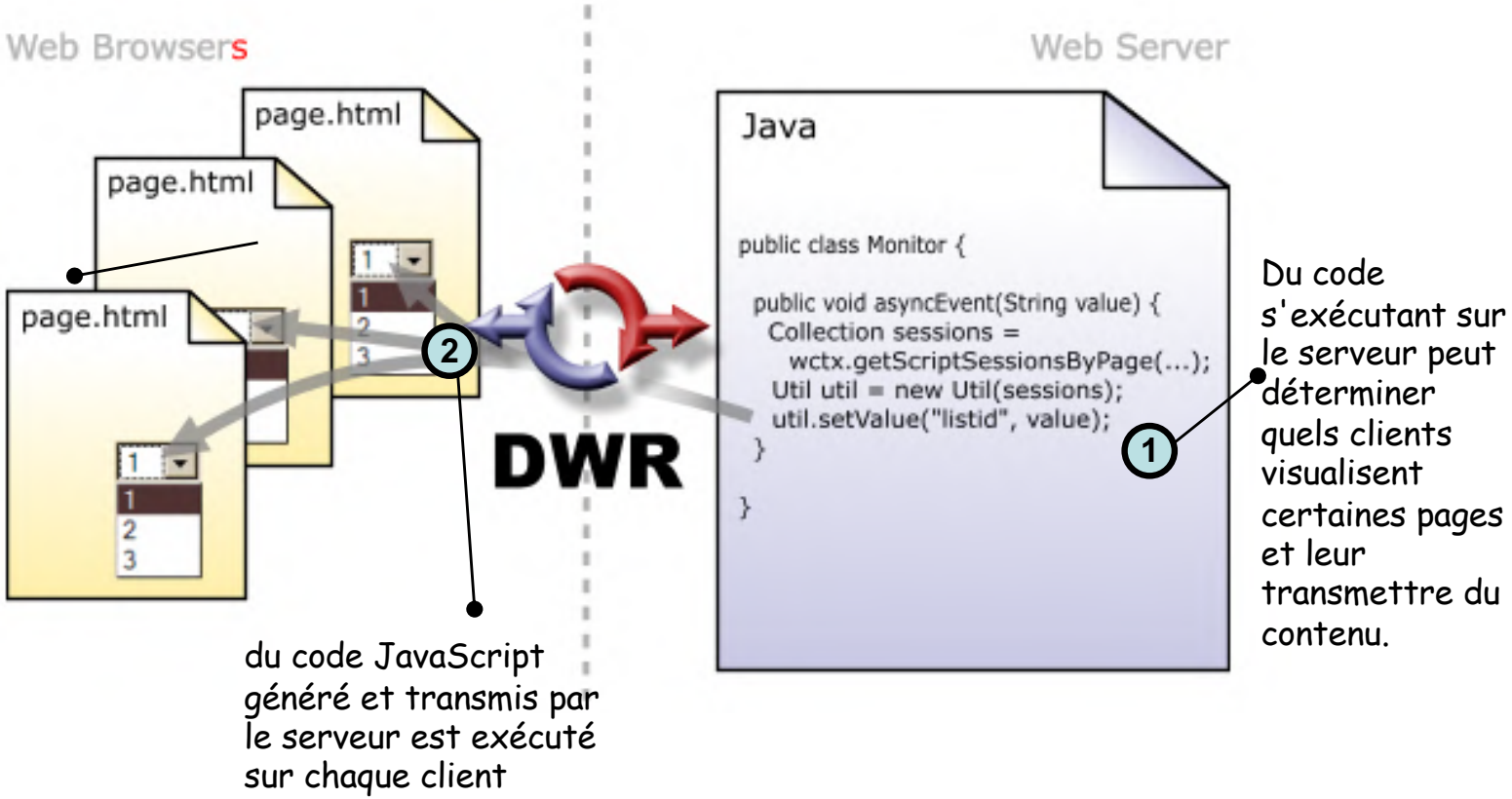
DWR : principe de fonctionnement

- AJAX



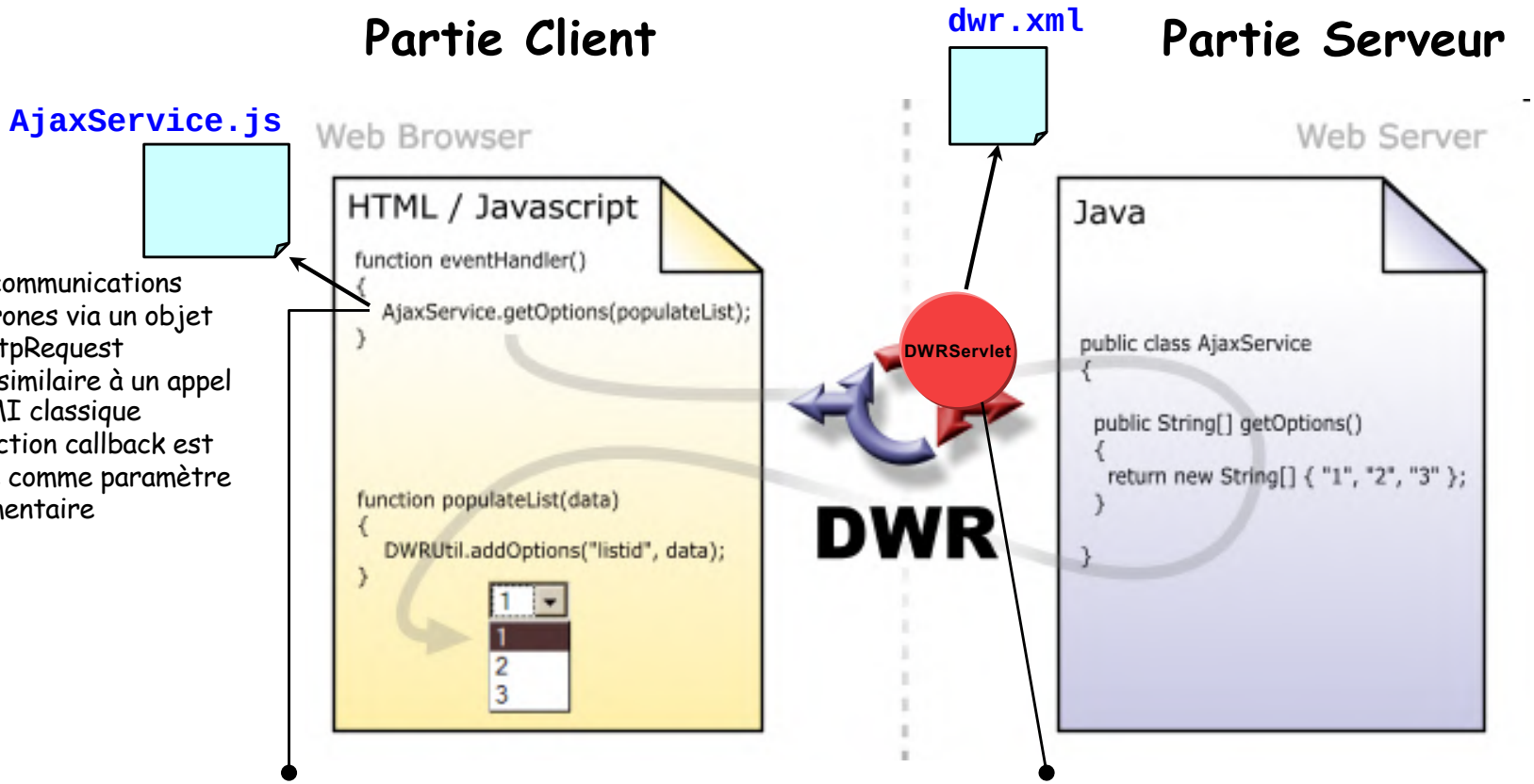
DWR : principe de fonctionnement

- Reverse Ajax



DWR : principe de fonctionnement

- DWR constitué de deux parties



• gère communications asynchrones via un objet XMLHttpRequest
• appel similaire à un appel RPC/RMI classique
• la fonction callback est ajoutée comme paramètre supplémentaire

Code JavaScript s'exécutant dans le navigateur, il envoie les requêtes et met à jour dynamiquement la page web.
DWR génère automatiquement une classe JavaScript à partir de la classe Java côté serveur qui encapsule les appels aux méthodes de la classe Java (AjaxService.js)

`org.directwebremoting.servlet.DwrServlet` traite les requêtes DWR et envoie les réponses au navigateur
Le servlet délègue les requêtes à une classe Java spécifiée dans le fichier de configuration **`dwr.xml`**

DWR effectue la conversion des paramètres et des valeur de retour de JavaScript vers java et inversement

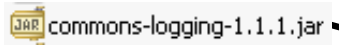
Construction d'une application basée sur DWR

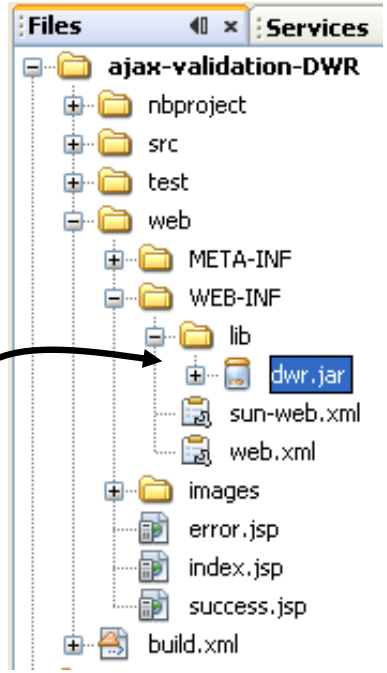
- Les étapes pour la mise en œuvre d'une application basée sur DWR sont les suivantes :
  1. Copier le fichier **dwr.jar** dans le répertoire **WEB-INF/lib** de l'application web
  2. Editer le fichier de configuration **web.xml**
     - spécification du servlet DWRServlet
  3. Création d'un fichier **dwr.xml** dans le répertoire **WEB-INF**
     - spécification des classes Java et des méthodes que l'on veut exposer (rendre accessibles) côté client
  4. Ecriture du code JavaScript côté client invoquant le code Java distant (syntaxe proche de RCP/RMI)
  5. Construction, déploiement, test de l'application

Construction d'une application basée sur DWR

- Etape 1 :Copie de **dwr.jar** dans WEB-INF/lib
  - téléchargeable depuis http://directwebremoting.org/download
  - version
    - 3.0rc2 (Release Candidate 2) en cours de développement
    - 2.07 (Dernière version stable)
  - contient le runtime DWR, en particulier le servlet **`DWRServlet`**

Nécessaire aussi de télécharger la librairie Commons-Logging du projet Apache http://commons.apache.org/logging/

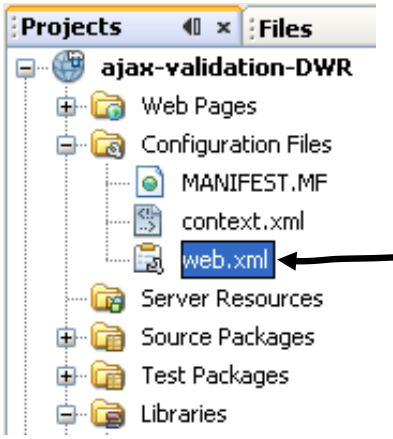




Construction d'une application basée sur DWR

- Etape 2 : éditer le fichier **WEB-INF/web.xml**

ajouter au fichier **web.xml** la déclaration du servlet DWR



web.xml

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-class>
  <init-param>
     <param-name>debug</param-name>
     <param-value>true</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

classe du servlet DWR

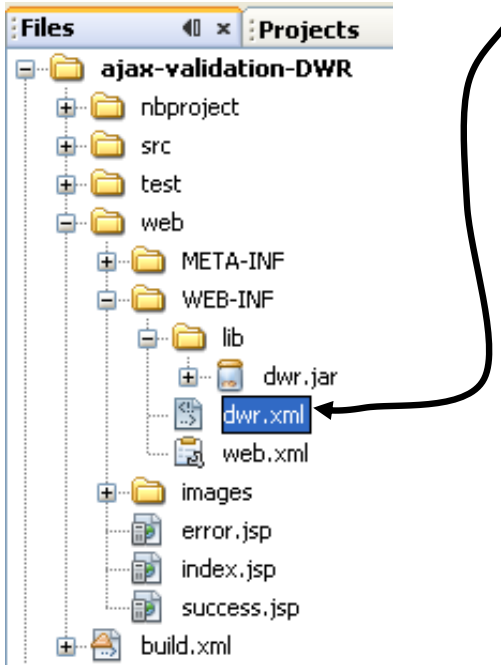
à utiliser en phase de développement, mais doit être retiré à la mise en production

toutes les urls préfixées par **dwr** seront redirigées vers le servlet DWR

Construction d'une application basée sur DWR

- Etape 3 : créer le fichier **dwr.xml** dans **WEB-INF**

le fichier **dwr.xml** définit pour quelles classes Java et quelles méthodes de ces classes DWR doit créer du code JavaScript qui permettra de les invoquer depuis le poste client



dwr.xml élément allow spécifie les classes Java que DWR expose au client

```
<!DOCTYPE dwr PUBLIC
   '-//GetAhead Limited//DTD Direct Web Remoting 3.0//EN'
   'http://getahead.org/dwr/dwr30.dtd'>
<dwr>
    <allow>

        <create creator="new" javascript="JSLoginManager">
          <param name="class" value="dwrdemo.services.LoginManager"/>
        </create>


    </allow>
</dwr>
```

nom de la classe JavaScript (Client)

fully qualified name de la classe Java (Serveur)

Par défaut, toutes les méthodes publiques de la classe Java sont exposées côté client. Il est préférable d'exposer de manière sélective les méthodes d'une classe Java (directive **include** où **exclude** dans l'élément create) .

```
<include method="validateUserId"/>
```

JavaScript au contraire de Java ne supporte pas la surcharge des méthodes.
Si deux fonctions sont déclarées avec le même nom la plus récente écrase la précédente.

Dans le fichier dwr.xml on ne précise que le nom de la méthode exposée.
Si elle est surchargée, DWR publie la première méthode trouvée correspondant au nom indiqué.

Construction d'une application basée sur DWR

- attributs de l'élément **`create`**

```
<create creator="new" javascript="JSLoginManager">
  <param name="class" value="dwrdemo.services.LoginManager"/>
  <include method="validateUserId"/>
</create>
```

javascript : nom par lequel la classe java est accessible depuis le code JavaScript

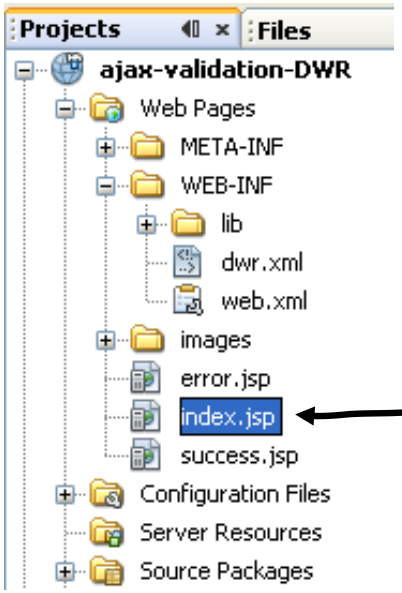
creator : indique la manière dont DWR doit procéder pour obtenir une instance de la classe java
- new : appel du constructeur par défaut
- none : pas de création d'objet (ex classe qui n'a que des méthodes statiques où quand on veut utiliser un objet déjà défini dans la portée spécifiée)
- spring : accès à des Beans au travers du framework Spring
- jsf: utilise un objet JSF (Java Server Faces)
- struts : utilise un FormBean de Struts
- scripted : utilise un langage de script (BeanShell, Groovy…) via BSF (Bean Scripting Framework)
- possibilité d'ajouter ses propres créateurs

scope : indique dans quelle portée l'objet Java est créé. Par défaut page, autres options : request, session, application. Le nom associé à l'objet est le nom spécifié par l'attribut javascript

si il n'existe pas déjà dans la portée spécifiée l'objet Java est instancié dès que le proxy JavaScript tente d'invoquer l'une de ses méthodes

Construction d'une application basée sur DWR

- Etape 4a : écrire le code JavaScript qui côté client invoque les méthodes de la classe Java



inclure le code des bibliothèques JavaScript de DWR

index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
...
<script type='text/javascript' src='dwr/engine.js'></script>
<script type='text/javascript' src='dwr/util.js'></script>
<script type='text/javascript' src='dwr/interface/JSLoginManager.js'></script>
...
</html>
```

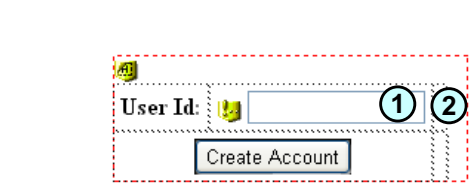
inclure le code la classe JavaScript générée par DWR pour accéder aux méthodes de la classe Java **LoginManager**

les chemins sont soit des chemins relatifs soit des chemins absolus avec comme racine le contexte de l'application web

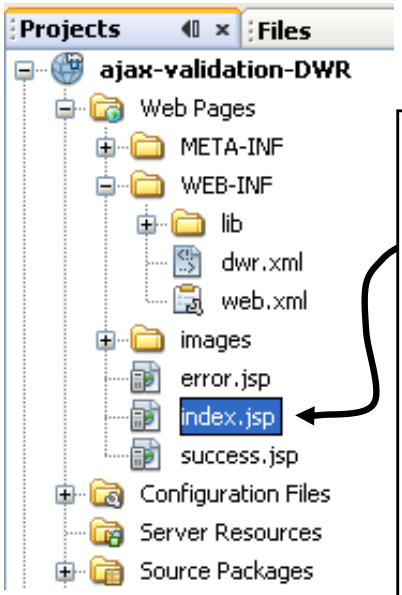
```
<script src='/[YOUR-WEBAPP]/dwr/interface/[YOUR-SCRIPT].js'></script>
<script src='/[YOUR-WEBAPP]/dwr/engine.js'></script>
```

Construction d'une application basée sur DWR

- Etape 4b : écrire le code JavaScript qui côté client invoque les méthodes de la classe Java




```
<form name="updateAccount" action="validate" method="post">
    <input type="hidden" name="action" value="create"/>
    <table border="0" cellpadding="5" cellspacing="0">
        <tr>
            <td><b>User Id:</b></td>
            <td>
                <input type="text" size="20" id="userid" name="id" onkeyup="validateUserId()">
            </td>
            <td>
                <div id="userIdMessage"></div>
            </td>
        </tr>
        <tr>
            <td align="right" colspan="2">
```




```
<script type='text/javascript'>

function validateUserId() {
  var userId = dwr.util.getValue("userid");

  JSLoginManager.validateUserId(userId, userIdValidated );
}

function userIdValidated(data) {

    if (! data) {
       dwr.util.setValue("userIdMessage","Invalid User Id");
    } else {
       dwr.util.setValue("userIdMessage","Valid User Id");
    }

  }
</script>
```

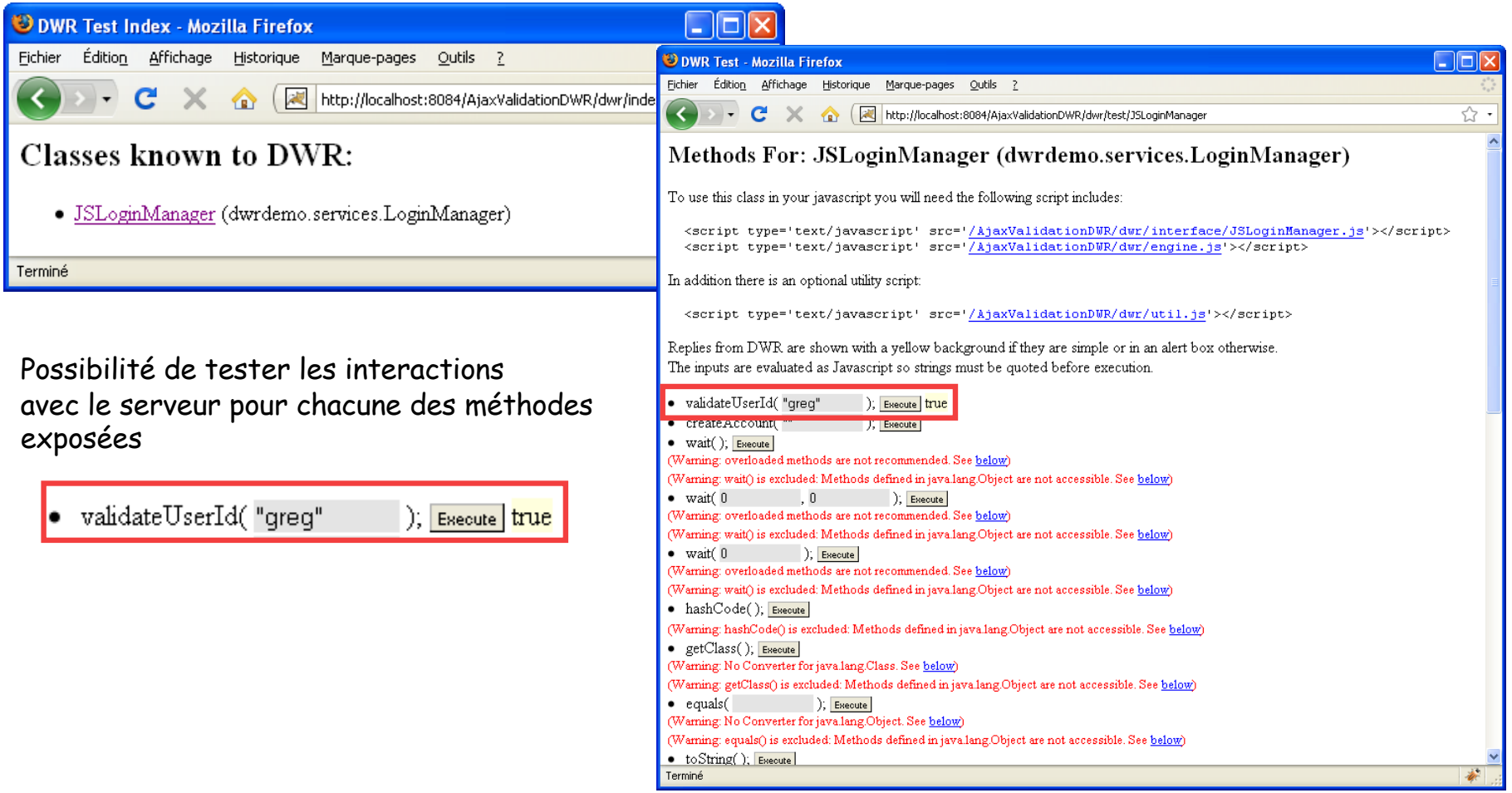
Récupère la valeur du champ de saisie et invoque la méthode **validateUserId(userId)** de la classe **LoginManager** sur le serveur.

La fonction **userIdValidated()** est une fonction callback, elle met à jour le message dans la division **userIdMessage**

Construction d'une application basée sur DWR

- Etape 5 : construire, déployer, tester le code

DWR crée une page de test des différentes classes de l'application exposées coté client
http://localhost:8084/<Your-Application-Context>/dwr



Possibilité de tester les interactions avec le serveur pour chacune des méthodes exposées

Gestion des appels asynchrones

```
<script type='text/javascript'>

  function validateUserId() {
    var userId = dwr.util.getValue("userid");

    JSLoginManager.validateUserId(userId, userIdValidated );
  }

  function userIdValidated(data) {

      if (! data) {
         dwr.util.setValue("userIdMessage","Invalid User Id");
      } else {
         dwr.util.setValue("userIdMessage","Valid User Id");
      }

  }
</script>
```

① un paramètre supplémentaire : la fonction callback qui sera invoquée lorsque l'exécution de la méthode côté serveur sera terminée et la réponse retournée.

La fonction `userIdValidated()` est une fonction callback, elle met à jour le message dans la division **`userIdMessage`**

② La fonction callback peut être écrite sous forme "in-line"

```
<script type='text/javascript'>

  function validateUserId() {
    var userId = dwr.util.getValue("userid");

    JSLoginManager.validateUserId(userId, function (data) {
       if (! data) {
         dwr.util.setValue("userIdMessage","Invalid User Id");
       } else {
         dwr.util.setValue("userIdMessage","Valid User Id");
       }
     });
  }
</script>
```

Gestion des appels asynchrones

- Un troisième syntaxe alternative est d'utiliser un objet méta-data

```
<script type='text/javascript'>

  function validateUserId() {
    var userId = dwr.util.getValue("userid");

    JSLoginManager.validateUserId(userId,
       function (data) {
         if (! data) {
           dwr.util.setValue("userIdMessage","Invalid User Id");
         } else {
           dwr.util.setValue
         }
       }
    );
  }
</script>
```

La fonction callback peut être écrite sous forme "in-line"

l'objet méta-data spécifie une fonction callback

avantage : permet de spécifier des options supplémentaires lors de l'appel du callback

timeout

gestionnaire d'erreur

objet méta-data

```
<script type='text/javascript'>

  function validateUserId() {
    var userId = dwr.util.getValue("userid");

    JSLoginManager.validateUserId(userId,
    {
      callback:function(data){
        if (! data) {
          dwr.util.setValue("userIdMessage","Invalid User Id");
        } else {
        dwr.util.setValue("userIdMessage","Valid User Id");
      }

      timeout:5000,

      errorHandler:function(message) { alert("Oops: " + message); }

    }
    );
  }
</script>
```

Convertisseurs

- **Convertisseurs** (converters) se chargent de la conversion (marshalling) des données entre le client (JavaScript) et le serveur (Java)

- DWR propose un certain de nombre de convertisseurs
  - **Basic Converters** : int, char, float, double, boolean …,Integer, Float, Double, Boolean…, java.lang.String
  - **Date Converter** : java.util.Date, java.sql.Date, java.sql.Time, java.sql.TimeStamp
  - **Array** et **Collection Converters** : tableaux et Collections (Lists, Sets, Maps, Iterators, etc) des types précédents
  - **DOM Converter** : objets DOM (par ex. Element and Document) des API DOM, XOM, JDOM and DOM4J
  - **Bean** et **Object Converter** : objets Java Beans
    - convertissent un POJO en un tableau associatif JavaScript et inversement
    - Bean Converter : utilise getters et setters, Object Converter accède directement aux attributs
- Possibilité de créer ses propres convertisseurs.
  - rarement nécessaire

Convertisseurs

- Activation des convertisseurs
  - Activation implicite (activés par défaut, pas de déclaration à faire)
    - **Basic Converters**
    - **Date Converter**
    - **Array** et **Collection Converters**
    - **DOM Converter**
  - Activation explicite
    - **Bean** et **Object Converter**
    - DWR vérifie qu'il a l'autorisation avant de toucher (et d'exposer) votre propre code
    - élément **<converter…>** dans la section **<allow></allow>** du fichier **dwr.xml**

Convertisseurs

- Activation explicite d'un convertisseur pour un Java Bean

classe exposée via DWR

EtudiantDAO
public **Etudiant** loadEtudiant(int num) public void saveEtudiant(**Etudiant** e) …

Java Bean

Etudiant
private int numero private String nom private String prenom private int age
public int getAge() public void setAge(int age) …

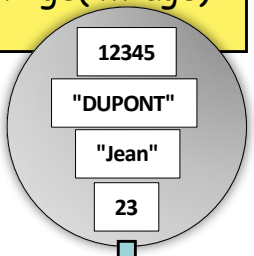
fichier de configuration **dwr.xml**

```
<!DOCTYPE dwr PUBLIC
   '-//GetAhead Limited//DTD Direct Web Remoting 3.0//EN'
   'http://getahead.org/dwr/dwr30.dtd'>
<dwr>
    <allow>

        <create creator="new" javascript="JSEtudiantDAO">
          <param name="class" value="pl2.notes.EtudiantDAO"/>
        </create>

        <convert converter="bean" match="pl2.notes.model.Etudiant">
            <param name="include" value="numero,nom,prenom"/>
        </convert>

    </allow>
</dwr>
```

on peut grâce au paramètre include spécifier les propriétés exposés

data paramètre de la fonction callback correspond à un objet **Etudiant**

converti vers JavaScript par DWR via convertisseur *bean*



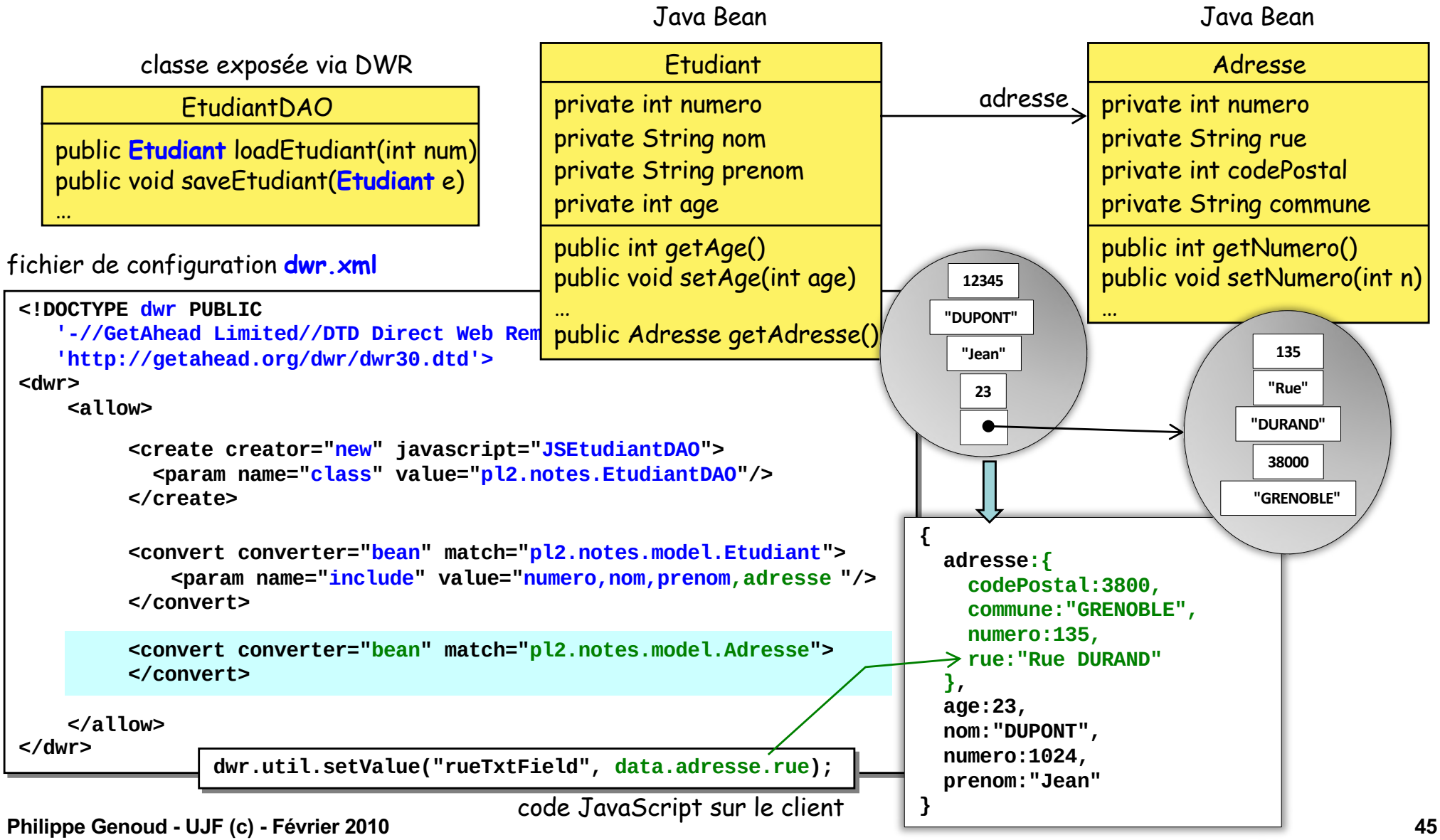

```
{
  nom:"DUPONT",
  numero:1024,
  prenom:"Jean"
}
```

```
JSEtudiantDAO.loadEtudiant(etudID,
      function(data) {
            if (data != null) {
               dwr.util.setValue("nomTxtField", data.nom);
               dwr.util.setValue("prenomTxtField", data.prenom);
…              …
            }
            else {
              …
            };
      }
);
```

code JavaScript sur le client

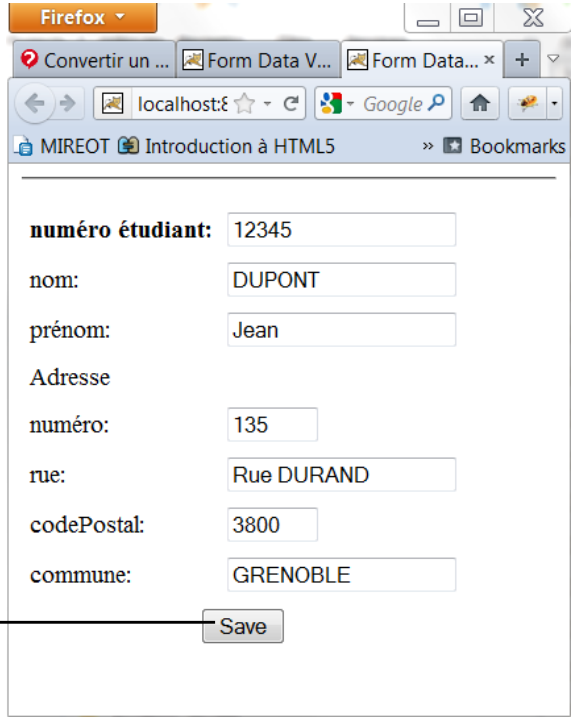
Convertisseurs

- Activation explicite d'un convertisseur pour un Java Bean



Convertisseur

- transmission d'objets javascript vers Java



code HTML

```
<input type="text"  size="20" id="numeroEtud"
       onchange="loadEtudiant()"/>

<input type="text" id="numRue" value="" size="5" />
<input type="text"  id="nomRue" value="" size="20" />

<input id="saveEtud" type="submit"
       value="Save" onclick="saveEtudiantInfo();">
```

code javaScript

```
function saveEtudiantInfo() {
     var e = {
          numero: parseInt(dwr.util.getValue("numeroEtud")),
          nom: dwr.util.getValue("nom"),
          prenom:dwr.util.getValue("prenom"),
          adresse: {
               numero:parseInt(dwr.util.getValue("numRue")),
               rue:dwr.util.getValue("nomRue"),
               codePostal:parseInt(dwr.util.getValue("codePostal")),
               commune: dwr.util.getValue("commune")
          }
     };
     JSEtudiantManager.saveEtudiant(e,etudiantSaved);
}

function etudiantSaved() {
     alert("etudiant sauvegardé");
}
```

EtudiantDAO

public **Etudiant** loadEtudiant(int num)
public void saveEtudiant(Etudiant e)
...

classe exposée via DWR

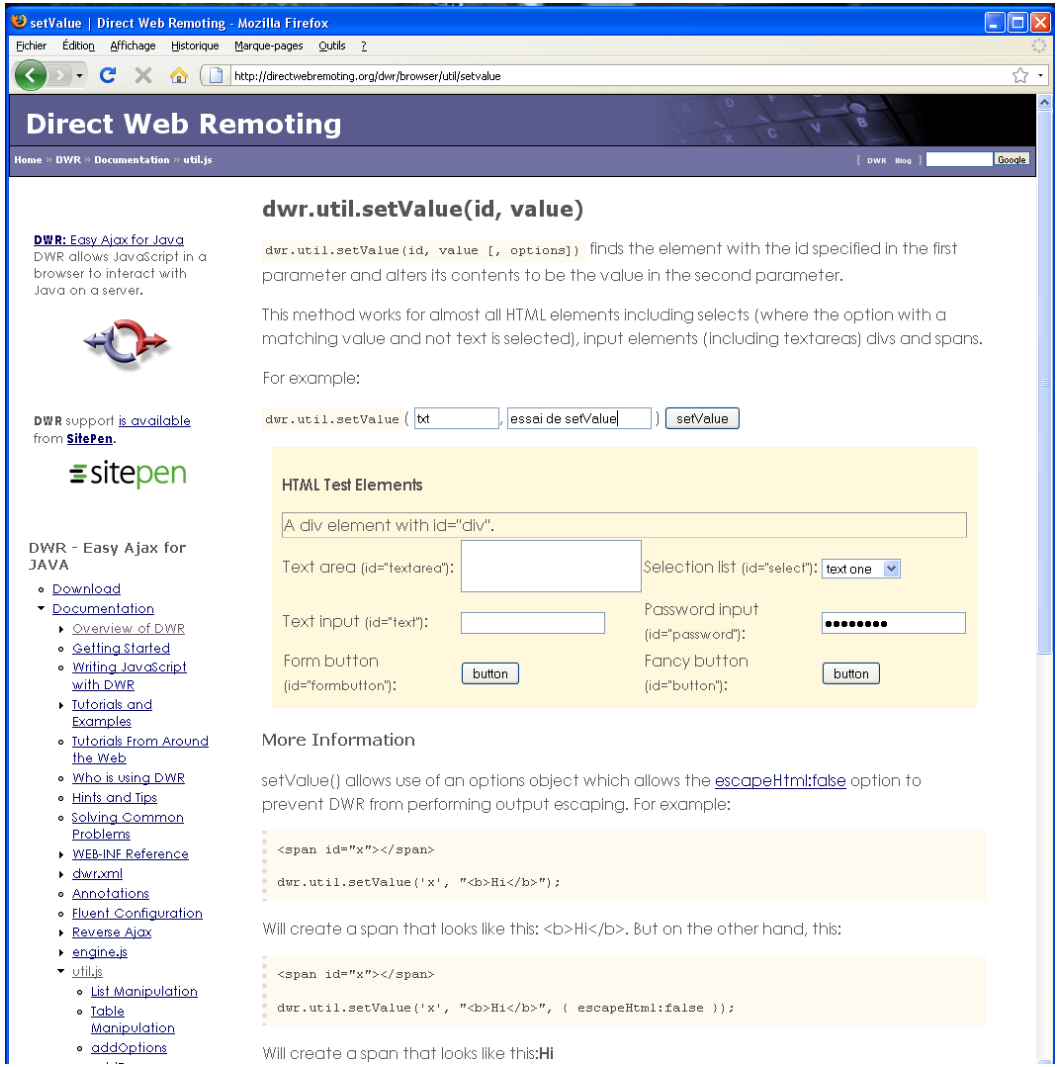
Fonctions utilitaires : util.js

- **util.js** : un ensemble de fonctions utilitaires JavaScript pour faciliter la mise à jour des pages
  - peut être utilisé indépendamment de DWR
- liste des fonctions
  - récupération d'un élément de la page : **byId**
  - consultation/modification de la valeur d'un élément : **getValue**, **getValues**, **setValue**, **setValues**
  - **getText** : donne le texte (et non la valeur) associé à un élément
  - manipulation de tables : **addRows** and **removeAllRows**
  - manipulation de listes : **addOptions** and **removeAllOptions**
  - **toDescriptiveString** : une alternative à toString
  - **onReturn** : permet de gérer la touche return supportée différemment selon les navigateurs
  - **selectRange** : permet de gérer sélection d'une partie d'un text input supportée différemment selon les navigateurs
  - **useLoadingMessage**
- toutes le fonctions sont préfixées par **dwr.util.**

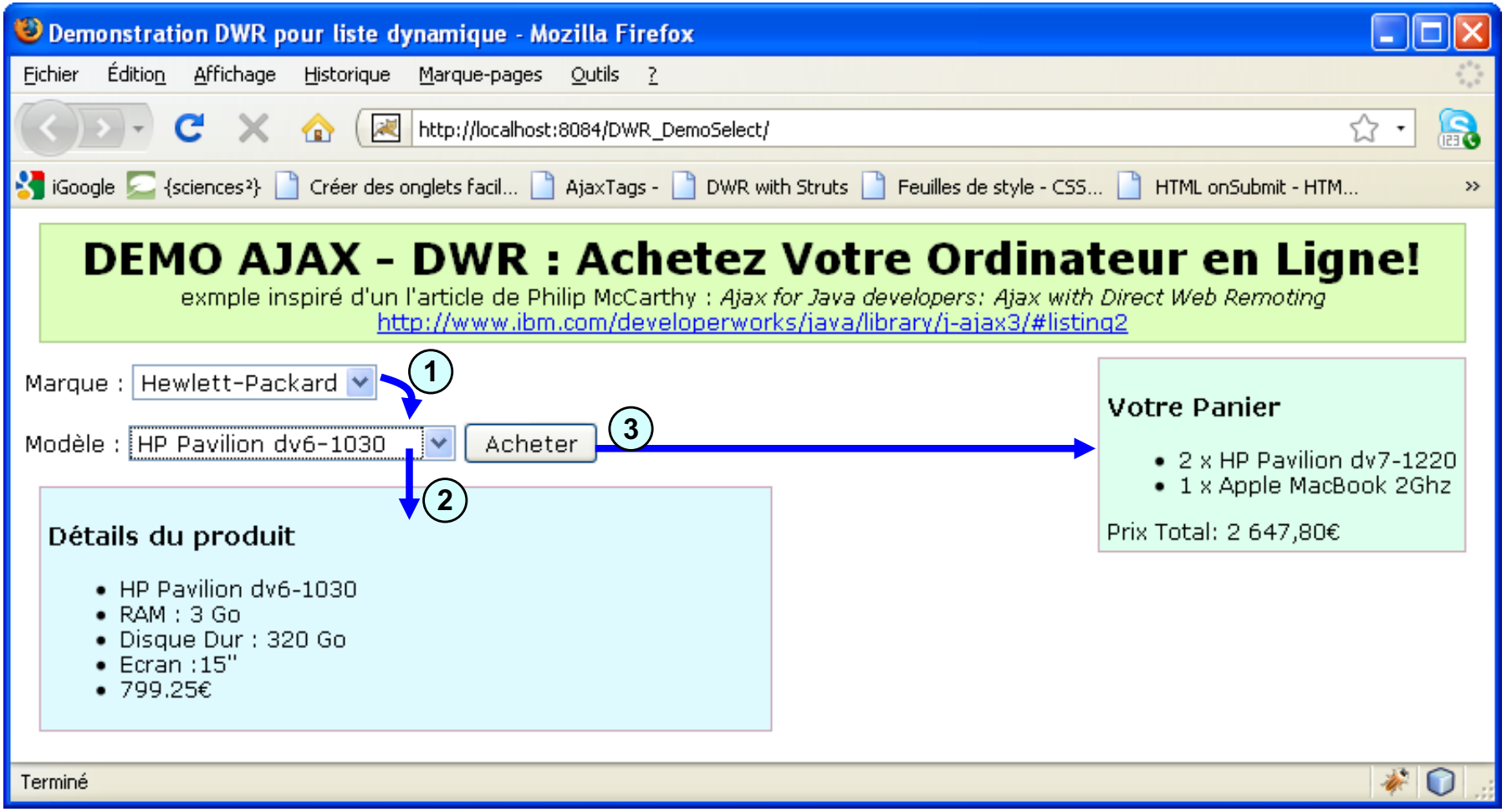
Fonctions utilitaires : util.js

- **dwr.util.getValue(id)**
  - récupère de manière transparente la valeur d'un élément HTML
    - on n'a pas à se soucier de savoir si l'élément est un champ text, un liste de sélection ou un div…
- **dwr.util.setValue(id,value)**
  - affecte de manière transparente une valeur à un élément HTML

•de manière générale pour expérimenter avec les fonctions util de DWR : http://directwebremoting.org/dwr/browser/util

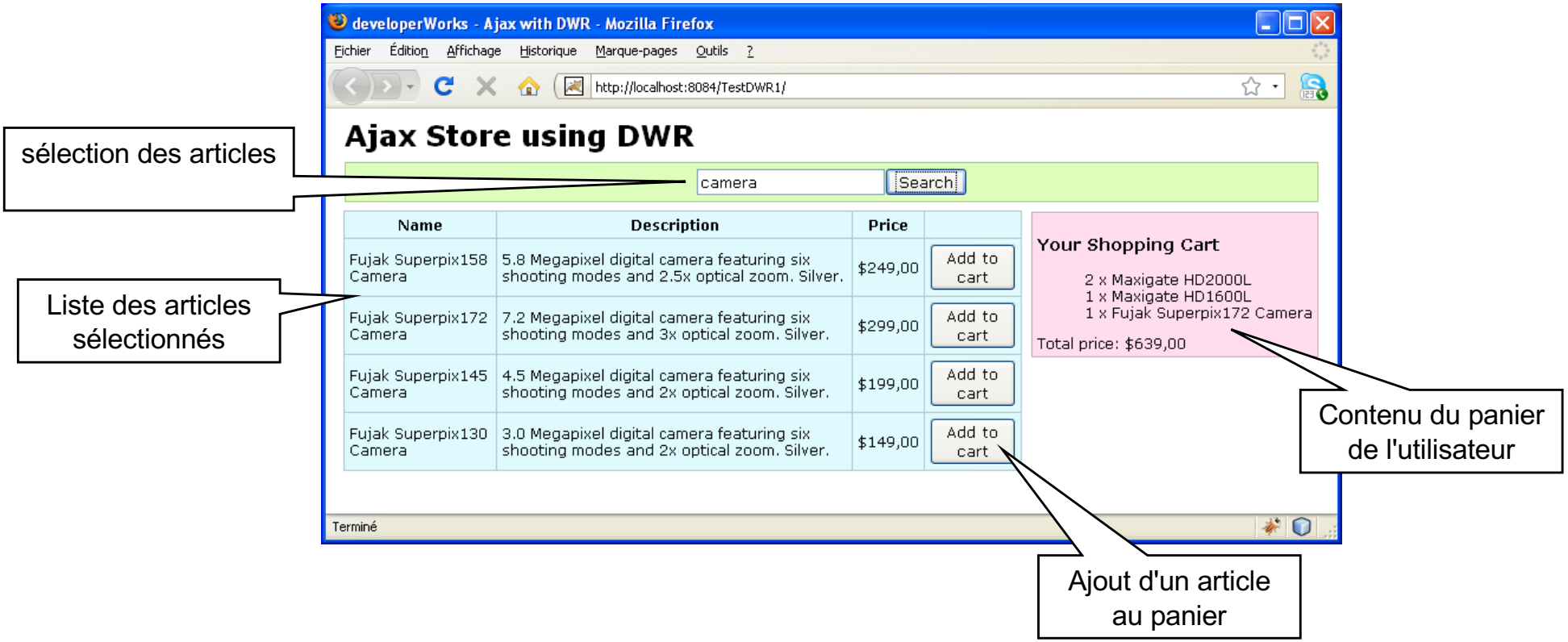


Exemple d'application AJAX-DWR



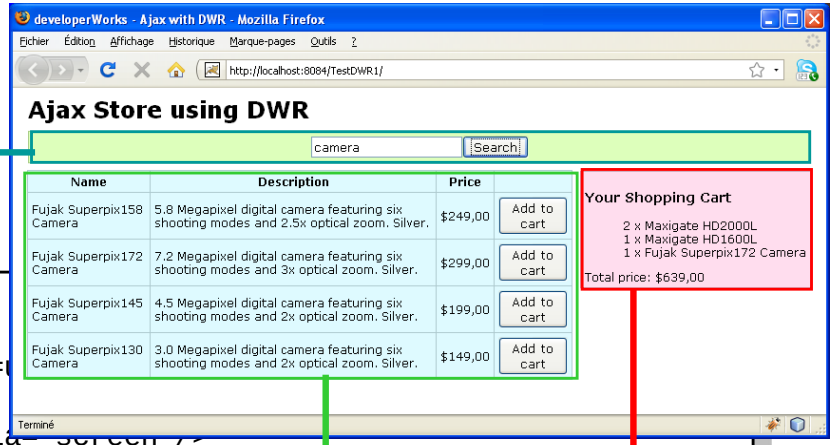
Exemple d'application DWR

- d'après : *Ajax for Java developers: Ajax with Direct Web Remoting*
  Philip McCarthy `http://www.ibm.com/developerworks/java/library/j-ajax3/#listing2`



La page HTML

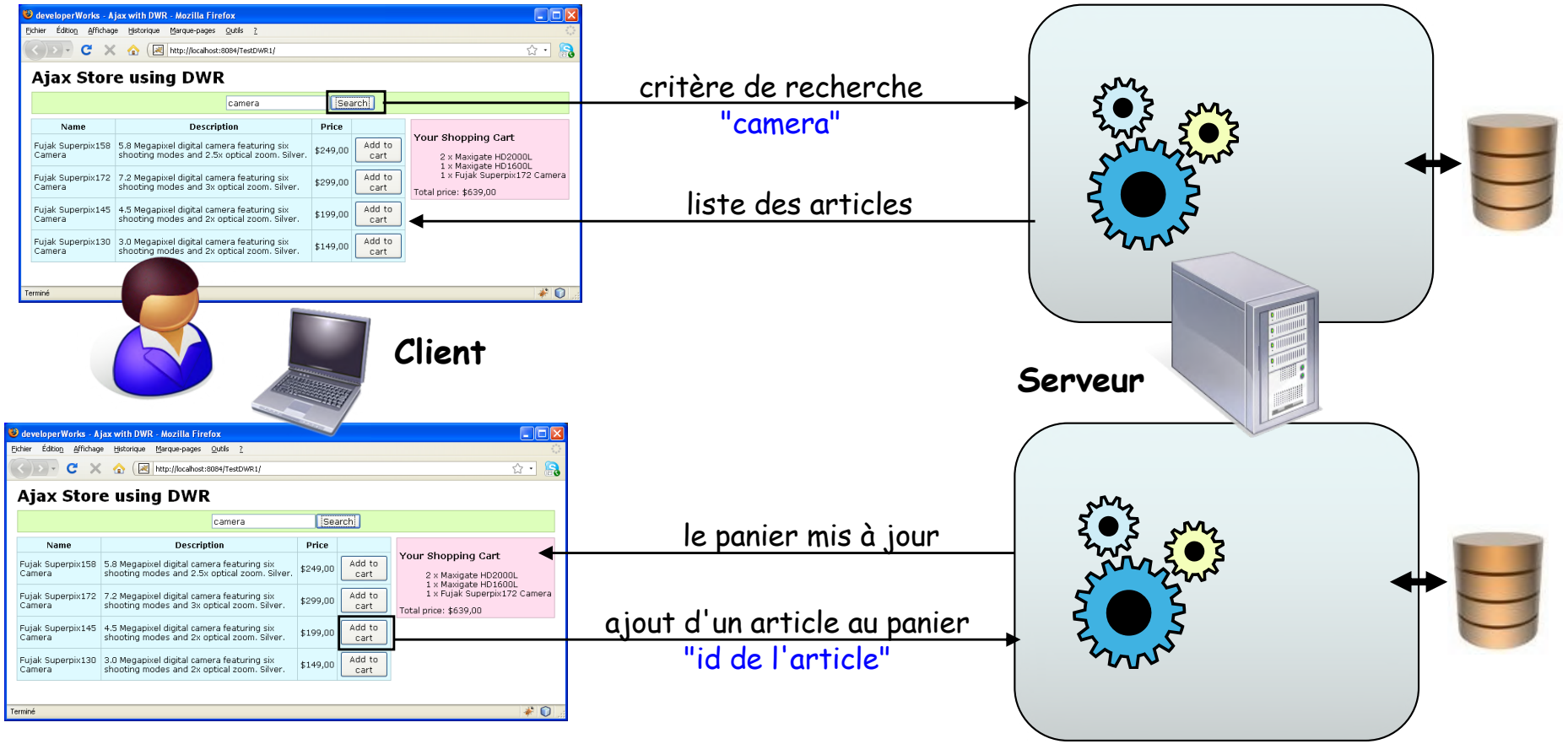
- La structure de la page est définie dans une **page statique** (**.html**)
- Le contenu du catalogue et du panier sont définis **dynamiquement** (**AJAX**)



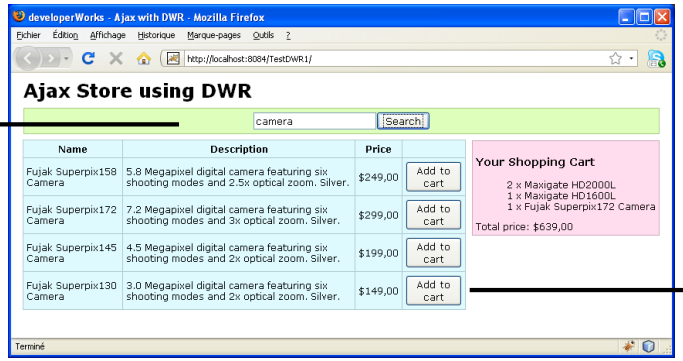
index.html

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=
        <title>developerWorks - Ajax with DWR</title>
        <link rel="stylesheet" type="text/css" href="style.css" media=
    </head>
    <body>
    <h1>Ajax Store using DWR</h1>
    <form id="searchform">
        <input id="searchbox"/><button type="submit" id="searchbtn">Search</button>
    </form>

    <div id="cart">
        <h3>Your Shopping Cart</h3>
        <ul id="contents"></ul>
        Total price: <span id="totalprice"></span>
    </div>

    <div id="catalogue">
        <table>
            <thead>
                <tr><th>Name</th><th>Description</th><th>Price</th><th></th><!-- Add to Cart buttons --></tr>
            </thead>
            <tbody id="items">
            </tbody>
        </table>
    </div>
</html>
```

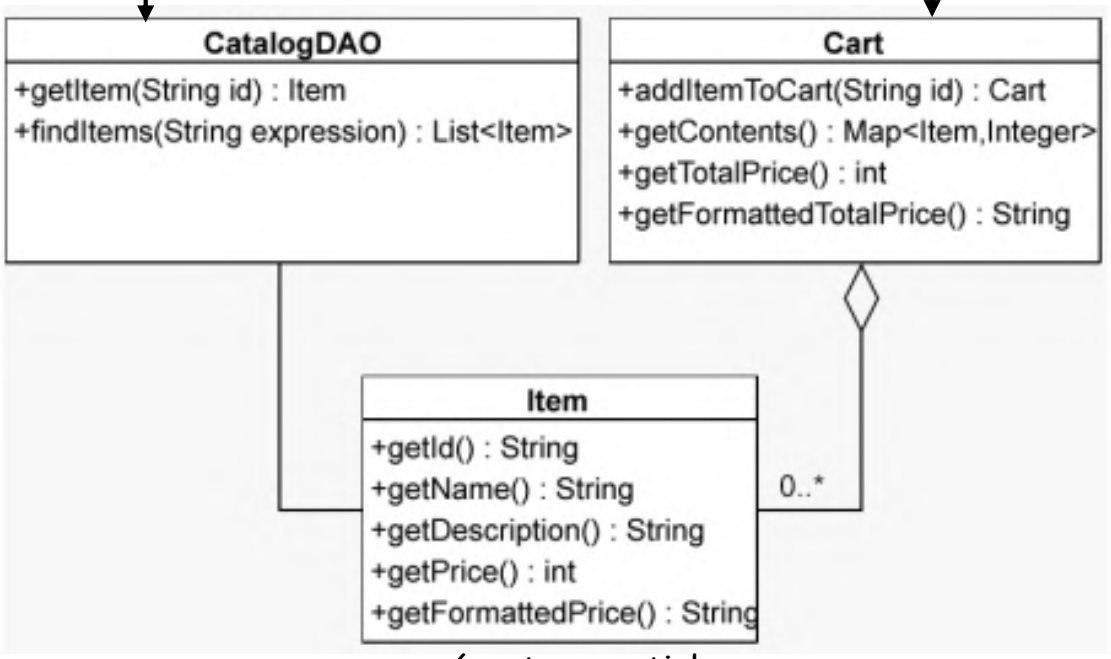
Echanges AJAX avec le serveur



Côté serveur : les classes Java



Data Acces Object pour la recherche des détails des articles dans le SI



le panier de l'utilisateur

représente un article

DWR côté serveur

éléments create : indiquent les classes dont les méthodes sont accessibles depuis JavaScript

éléments convert : indiquent comment les paramètres et types de retour des méthodes Java doivent être convertis depuis/vers JavaScript

fichier dwr.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC '-//GetAhead Limited//DTD Direct Web Remoting 3.0//EN'
       'http://getahead.org/dwr/dwr30.dtd'>
<dwr>
    <allow>
        <create creator="new" javascript="Catalogue">
            <param name="class" value="developerworks.ajax.store.CatalogueDAO"/>
            <include method="getItem"/>
            <include method="findItems"/>
        </create>

        <convert converter="bean" match="developerworks.ajax.store.Item">
            <param name="include" value="id,name,description,formattedPrice"/>
        </convert>

        <create creator="new" scope="session" javascript="Cart">
            <param name="class" value="developerworks.ajax.store.Cart"/>
            <include method="addItemToCart"/>
            <include method="getFormattedTotalPrice"/>
            <include method="getCart"/>
        </create>

        <convert converter="bean" match="developerworks.ajax.store.Cart">
            <param name="include" value="simpleContents,formattedTotalPrice"/>
        </convert>
    </allow>
</dwr>
```

nom utilisé en JavaScript pour accéder à l'objet Java

les méthodes exposées

Spécifie comment la classe CatalogueDAO doit être rendue accessible aux appels AJAX

spécifie comment le type Java Item est converti vers/depuis JavaScript

les membres de la classe Item inclus dans la conversion

DWR côté client

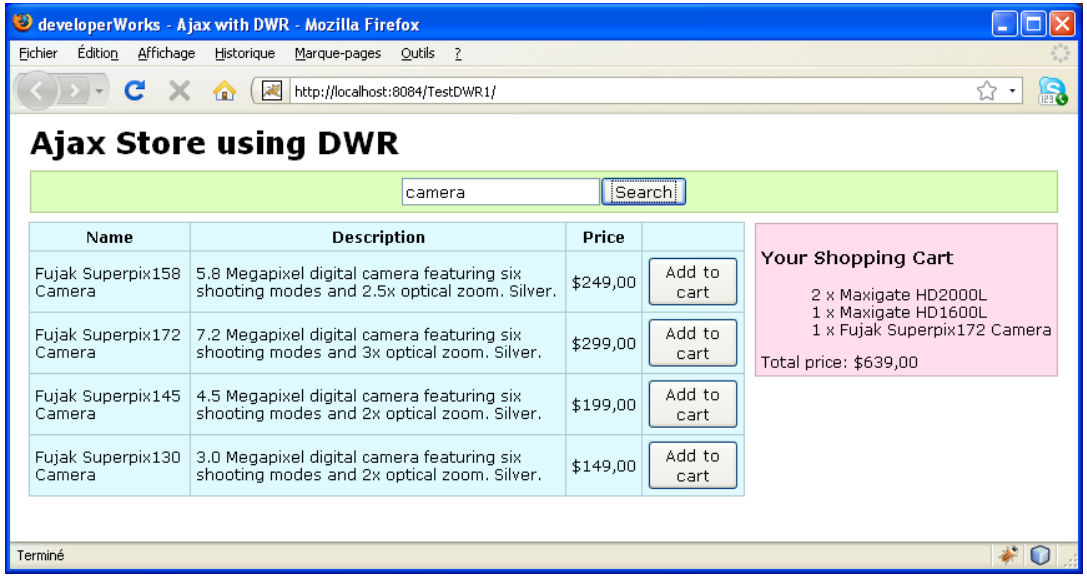
fichier dwr.xml

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>developerWorks - Ajax with DWR</title>
        <link rel="stylesheet" type="text/css" href="style.css" media="screen"/>

        <script type='text/javascript' src='/TestDWR1/dwr/engine.js'></script>
        <script type='text/javascript' src='/TestDWR1/dwr/util.js'></script>

        <script type='text/javascript' src='/TestDWR1/dwr/interface/Catalogue.js'></script>
        <script type='text/javascript' src='/TestDWR1/dwr/interface/Cart.js'></script>

        <script type='text/javascript' src='/TestDWR1/shopping.js'></script>
    </head>
    <body>
    <h1>Ajax Store using DWR</h1>
    <form id="searchform">
        <input id="searchbox"/><button type="submit" id="searchbtn">Search</button>
    </form>
    ...
    ...

</html>
```

Le "moteur" DWR et les méthodes utilitaires

Le code JavaScript pour invoquer les classes Java du serveur

Le code java script pour l'animation de la page index.html

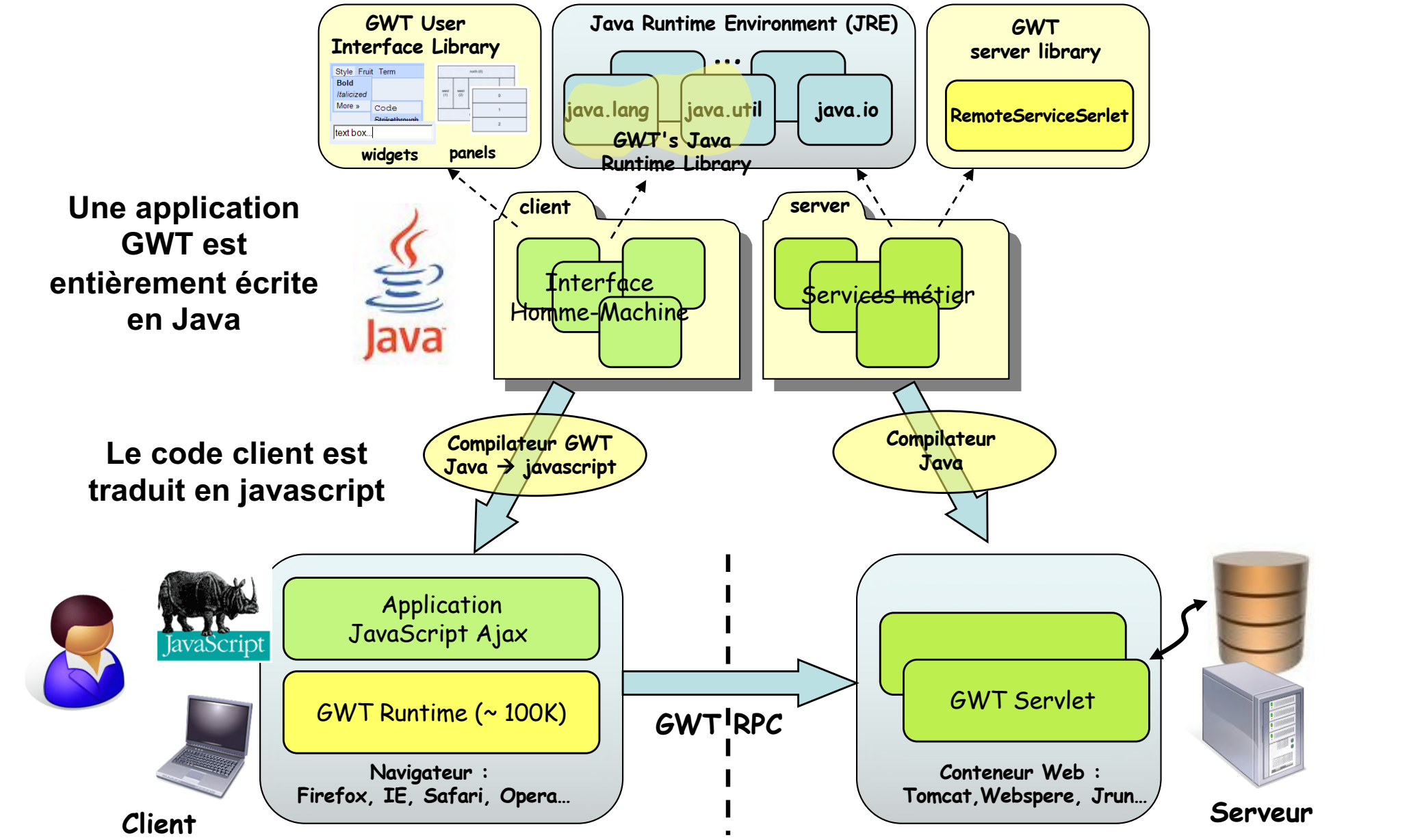



```
function searchFormSubmitHandler() {

  var searchexp = dwr.util.getValue("searchbox");

  // Call remoted DAO method, and specify callback function
  Catalogue.findItems(searchexp, displayItems);

  // Return false to suppress form submission
  return false;
}
```

récupération de l'expression de recherche

Introduction à GWT

GWT : grands principes



GWT : grands principes

- Deux modes d'exécution d'une application GWT

- **Hosted :**
  - L'application est exécutée au sein d'une machine virtuelle Java
  - Un navigateur spécial fourni par GWT abrite une machine virtuelle Java capable d'afficher et manipuler les objets graphiques composant l'interface
  - Utilisé en phase de développement : permet le débogage en Java ☺

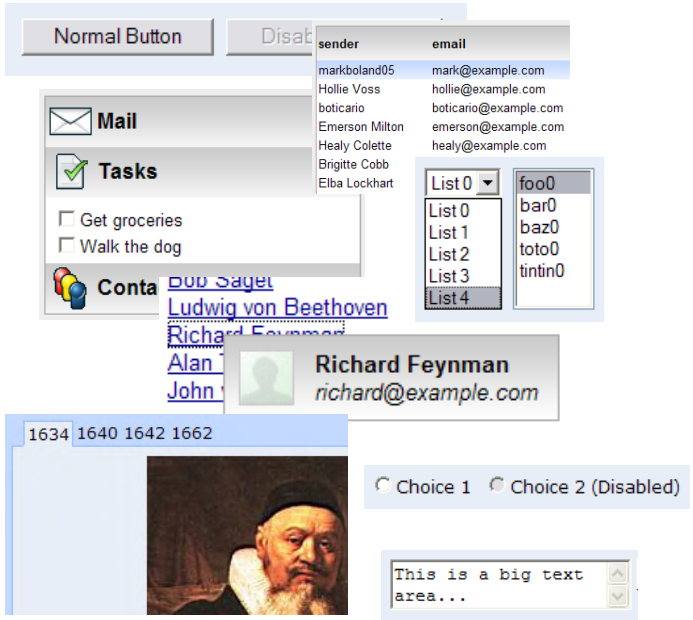
- **Web mode :**
  - L'application est exécutée à partir d'un navigateur web du marché
  - La partie cliente doit être compilée au préalable en javascript
  - Utilisé en production

GWT : grands principes

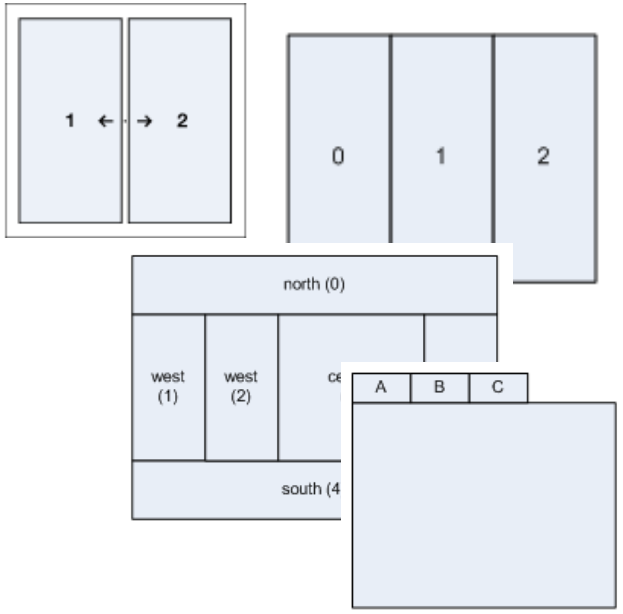
- Retour aux sources de la programmation d'IHM graphiques pour le développement de la partie client.
  - programmation similaire à ce qu'il se fait avec Swing, SWT ou Visual Basic
  - assembler des composants graphiques (widgets)
  - armer des gestionnaires sur les événements reçus par les widgets

Possibilité de définir de nouveaux widgets

ou d'intégrer des frameworks javascript (Dojo, Yahoo UI…)



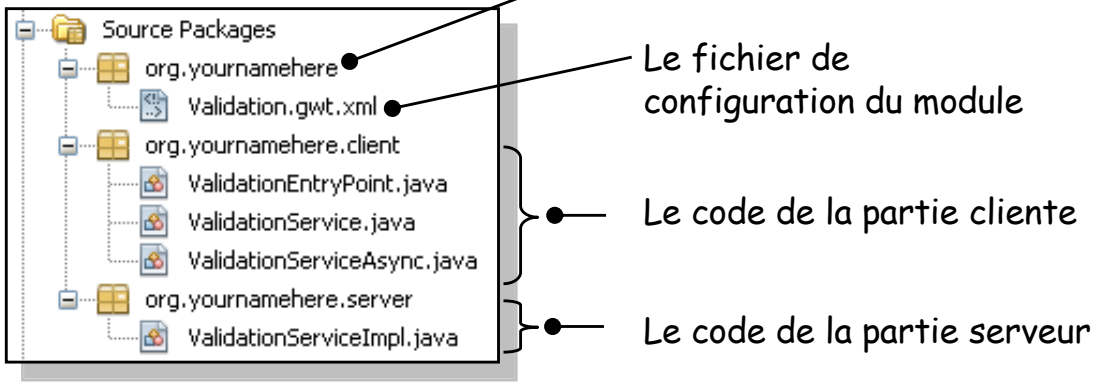
widgets standards GWT



panels

Module GWT

- Module : composant d'IHM de haut niveau défini par GWT



Le nom logique du module est le nom du fichier de configuration (sans le suffixe gwt.xml) préfixé par le nom du package racine

org.yournamehere.Validation

Validation.gwt.xml

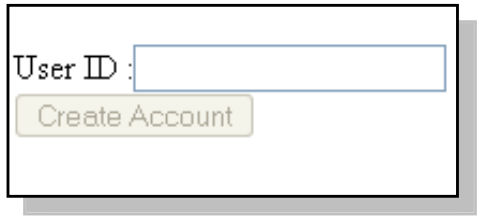
Modules dont hérite ce module

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
     <inherits name="com.google.gwt.user.User"/>
     <entry-point class="org.yournamehere.client.ValidationEntryPoint"/>
</module>
```

Points d'entrée de ce module : classe Java chargée et exécutée a démarrage du module

Création d'un module

- La partie cliente du module implémente l'interface EntryPoint




```
public class ValidationEntryPoint implements EntryPoint {

    public ValidationEntryPoint() {
    }

    public void onModuleLoad() {

        final Label userID = new Label("User ID : ");
        final HTML lblServerReply = new HTML();
        final TextBox txtUserInput = new TextBox();
        final Button button = new Button("Create Account");
        button.setEnabled(false);


        HorizontalPanel panel1 = new HorizontalPanel();
        panel1.add(userID);
        panel1.add(txtUserInput);
        panel1.add(lblServerReply);
        VerticalPanel panel2 = new VerticalPanel();
        panel2.add(panel1);
        panel2.add(button);
        RootPanel.get("slot0").add(panel2);
    }
}
```

Constructeur sans paramètres

Méthode invoquée au chargement du module
construit le contenu du module

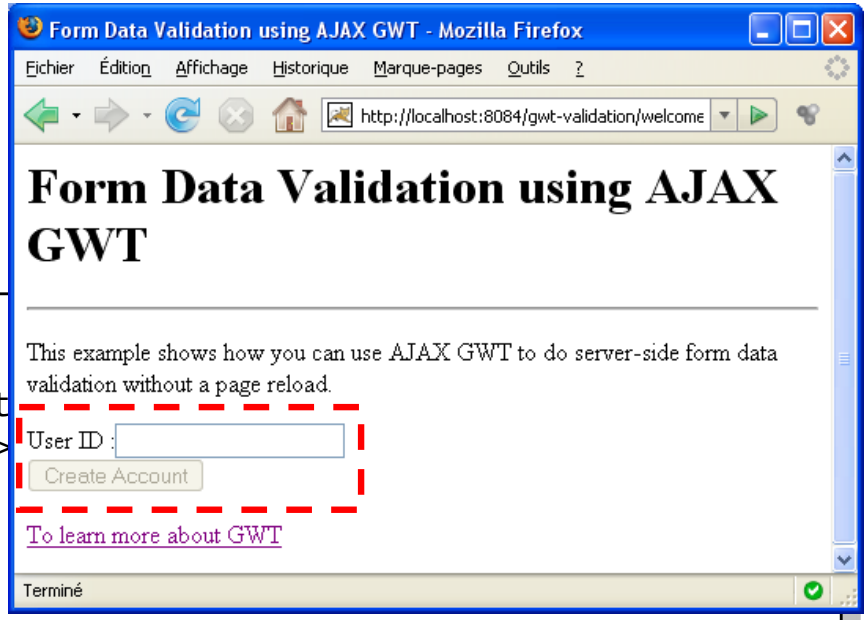
Création des composants
(widgets GWT)

Assemblage des composants

id de l'élément de la page HTML où sera "accroché" le
RootPanel de ce module

Intégration du module dans une page HTML

- Host page : page HTML qui contient l'invocation d'un module GWT



Validation.html

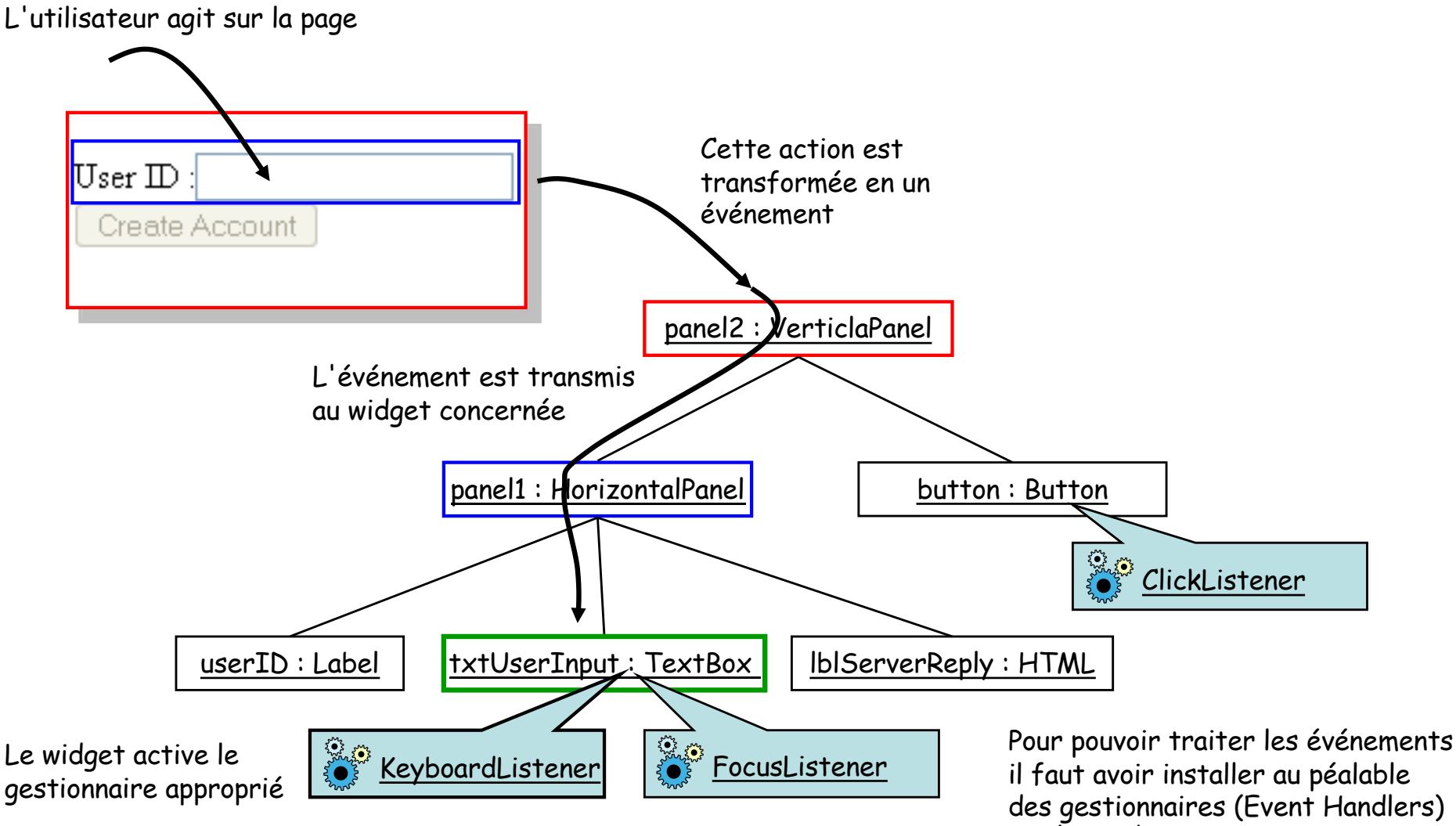
```
<html>
   <head>
        <meta http-equiv="Content-Type" content="text
   <title>Form Data Validation using AJAX GWT</title>
    </head>
    <body>
        <h1>Form Data Validation using AJAX GWT</h1>
        <hr/>
        <p>
            This example shows how you can use AJAX GWT to do server-side
            form data validation without a page reload.
        </p>
        <script language="javascript" src="org.yournamehere.Validation.nocache.js"></script>
        <div id="slot0"></div>
        <p>
           <a href="http://code.google.com/webtoolkit/">To learn more about GWT</a>
        </p>
    </body>
</html>
```

Element HTML auquel sera associé le RootPanel du module

Le nom complètement qualifié du module suivi de nocahe.js

Gestion de l'interaction

- Modèle d'événement similaire à celui des frameworks d'IHM classique (Swing, SWT …)

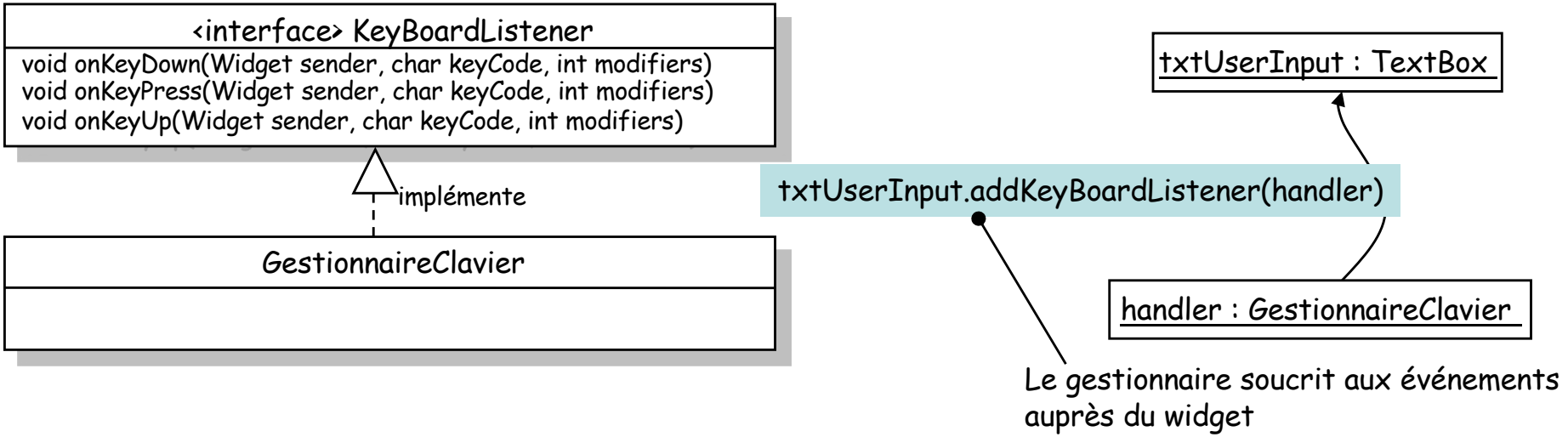


Gestion de l'interaction

- Une interface d'écoute (listener interface) definit une ou plusieurs méthodes que le widget appelle pour annoncer un événement

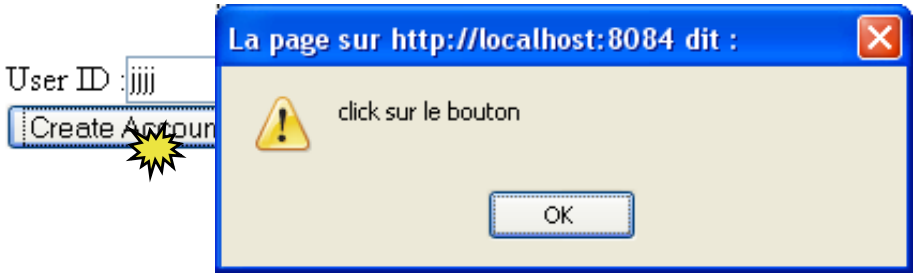
- Un gestionnaire destiné à traiter un ou des événements d'un type particulier doit être défini par une classe qui implémente l'interface d'écoute associée.

- Le gestionnaire doit s'enregistrer (en passant sa référence) auprès du widget pour recevoir ces événements.



Gestion de l'interaction

- Souvent écriture des gestionnaire d'événement à l'aide de classes anonymes internes (inner classes)



<interface> ClickListener
void public void onClick(Widget w)

Une instance d'une classe qui implémente l'interface ClickListener

```
button.addClickListener(new ClickListener() {

   public void onClick(Widget w) {
      alert("click sur le bouton");
   }
} );
```

Le code de cette classe

<interface> KeyBoardListener
void onKeyDown(Widget sender, char keyCode, int modifiers)
void onKeyPress(Widget sender, char keyCode, int modifiers)
void onKeyUp(Widget sender, char keyCode, int modifiers)

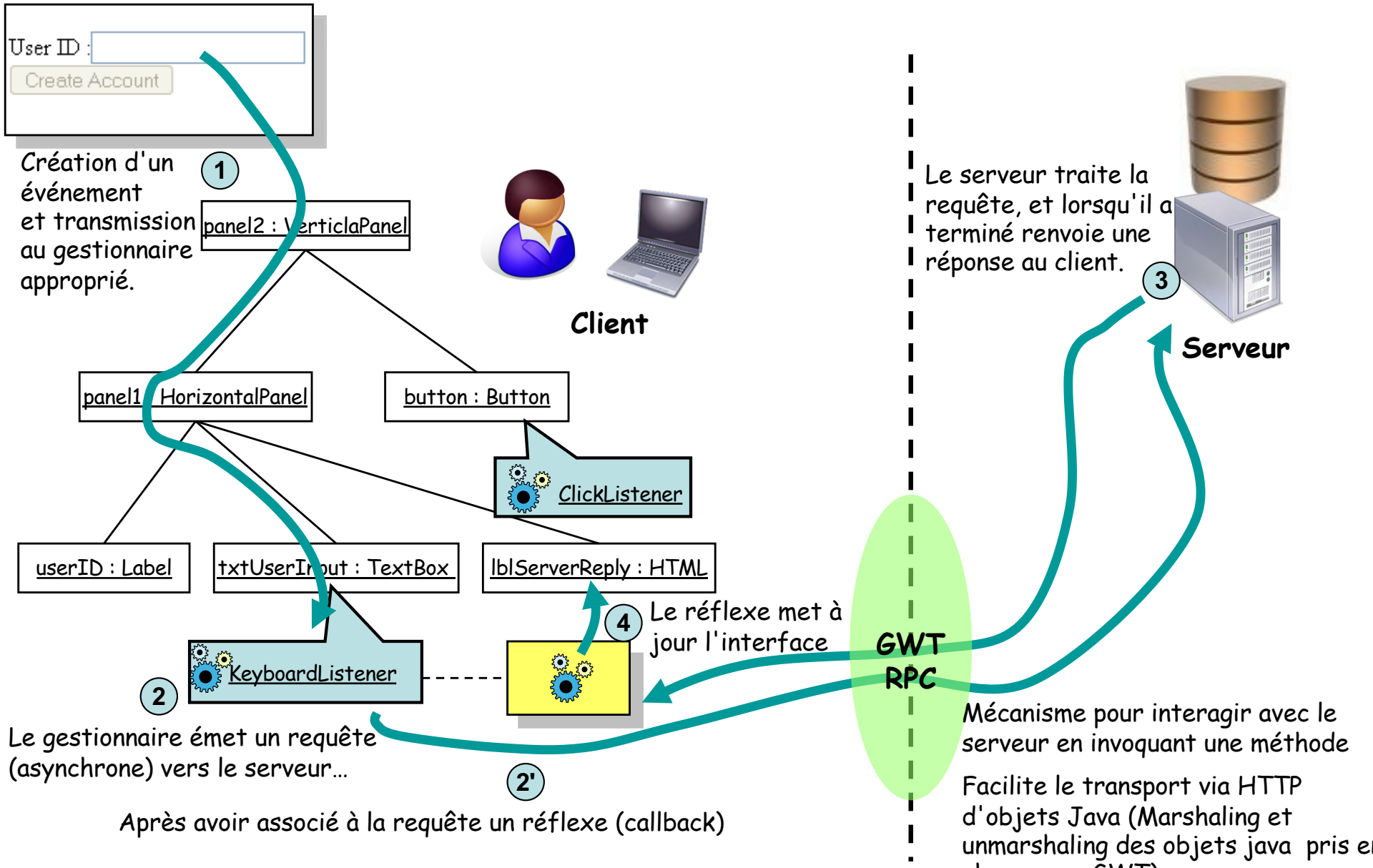
Quand l'interface d'écoute définit plusieurs méthodes il est parfois pratique de passer par une classe Adpater qui implémente déjà cette interface

La classe hérite de KeyboardListenerAdapter

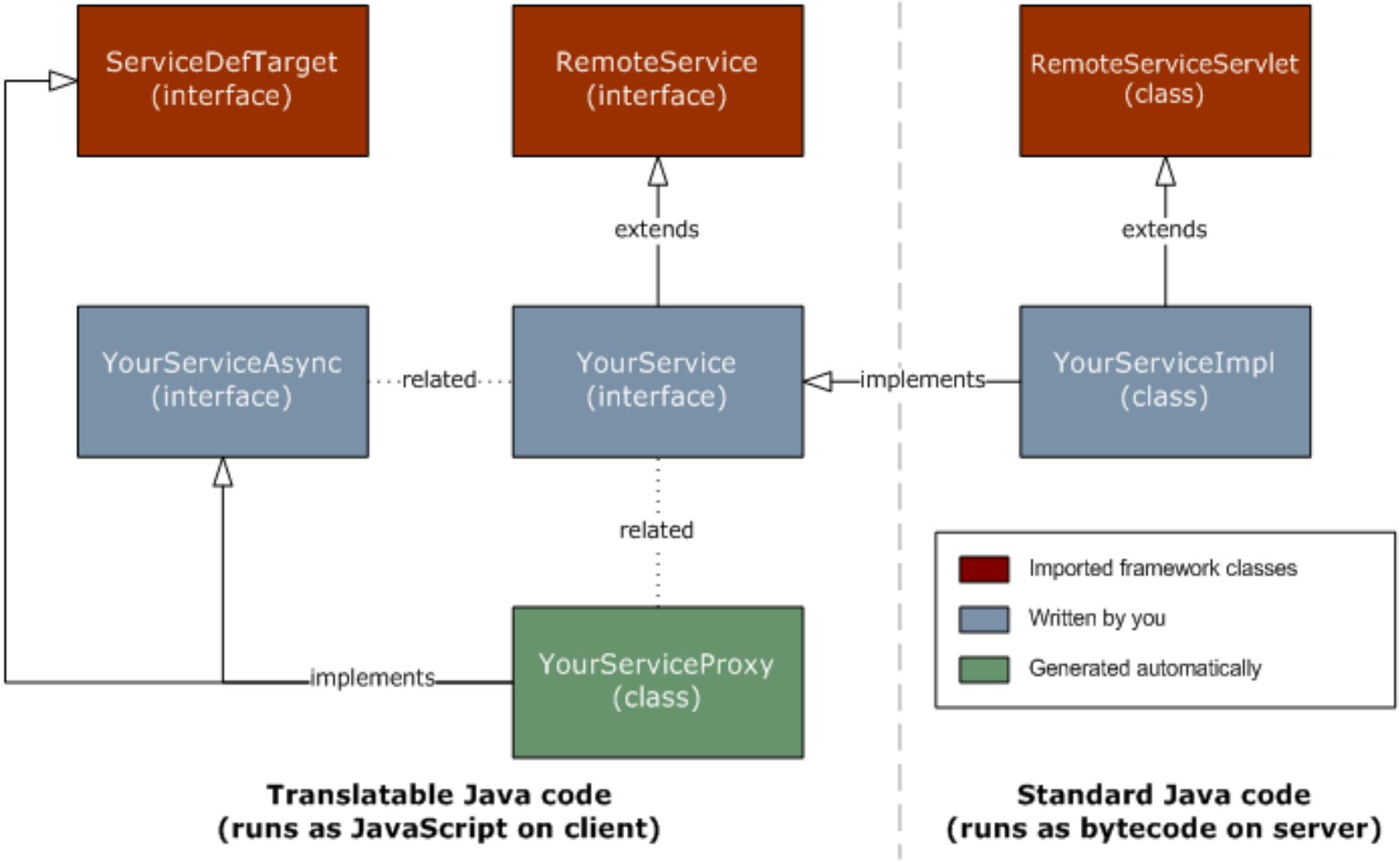
```
txtUserInput.addKeyboardListener(new KeyboardListenerAdapter() {

    public void onKeyUp(Widget sender, char keyCode, int modifiers) {
        alert("le texte tappé est " + txtUserInput.getText());
    }
});
```

La méthode onKeyup est redéfinie

Gestion de l'interaction et communication avec le serveur

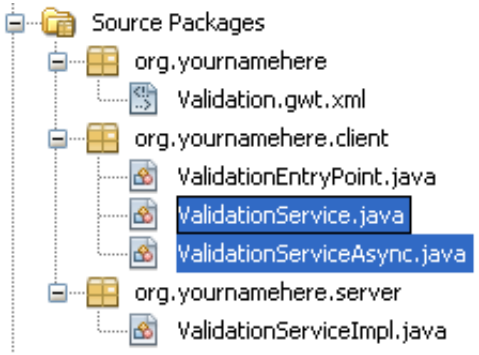


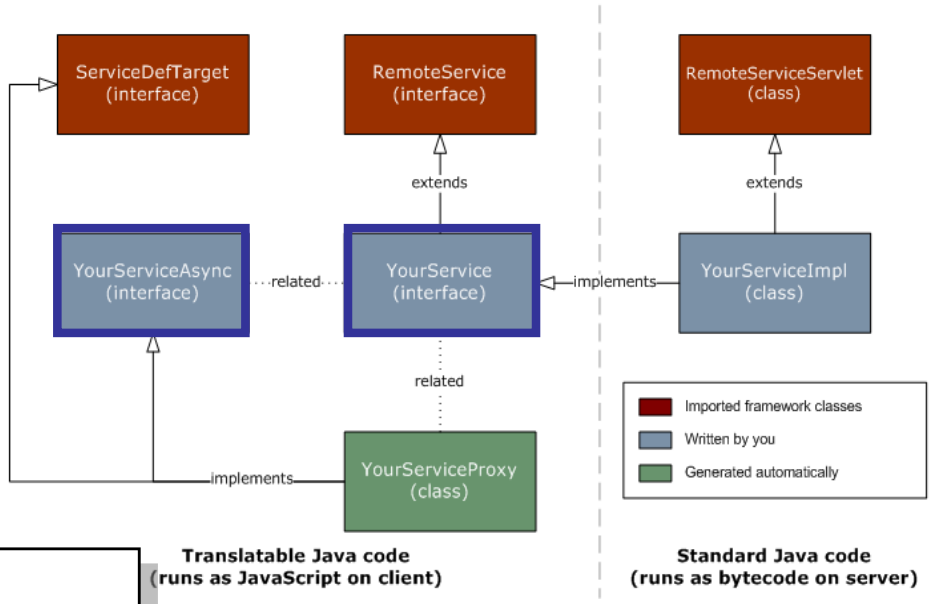
GWT RPC : principes



GWT RPC : mise en oeuvre

1 Ecriture des interfaces de service





ValidationService.java

```
package org.yournamehere.client;

import com.google.gwt.user.client.rpc.RemoteService;

public interface ValidationService extends RemoteService{
    public Boolean validate(String s);
    public Boolean createAccount(String s);
}
```

ValidationServiceAsync.java — Même nom que l'interface synchrone mais on ajoute le suffixe *Async*

```
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface ValidationServiceAsync {
    public void validate(String s, AsyncCallback callback);
    public void createAccount(String s, AsyncCallback callback);
}
```

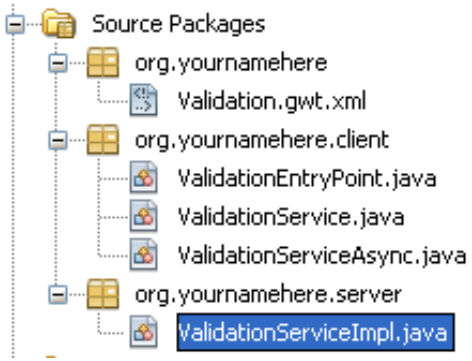
Ajout d'un paramètre de type *AsyncCallback*

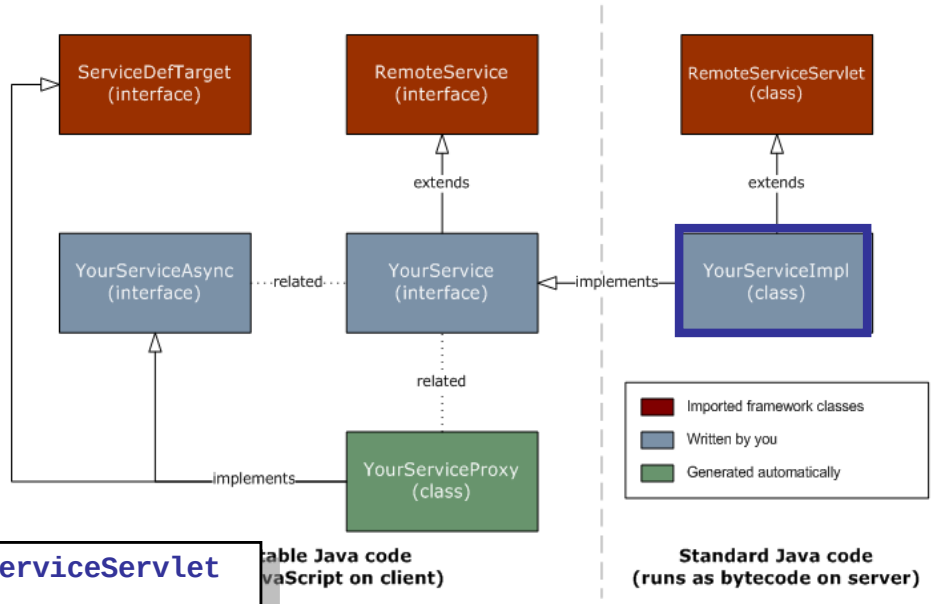
```
void onFailure(java.lang.Throwable caught)
void onSuccess(java.lang.Object result)
```

Le type de retour est toujours *void*

GWT RPC : mise en oeuvre

2 Implémentation du service





`ValidationServiceImpl.java`

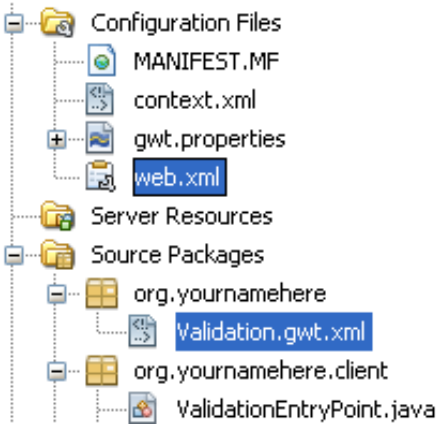
```
public class ValidationServiceImpl extends RemoteServiceServlet
                    implements ValidationService {


    ...
    public Boolean validate(String s) {
        if (! accounts.containsKey(s.trim()))
           return Boolean.TRUE;
        else
           return Boolean.FALSE;
    }

    public Boolean createAccount(String s) {
        if ((s != null) && !accounts.containsKey(s.trim())) {
            accounts.put(s.trim(), "account data");
            return Boolean.TRUE;
        }
        else
           return Boolean.FALSE;
    }
}
```

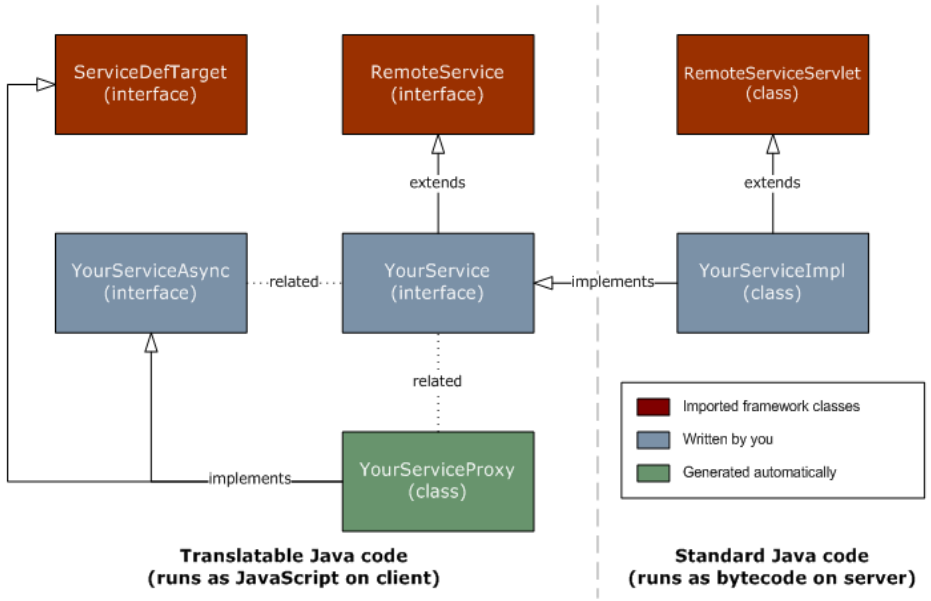
Vérifie qu'il n'existe pas de compte au nom de *s* sur le serveur

Création du compte sur le serveur renvoie True si la création a réussi False sinon

GWT RPC : mise en oeuvre

3 Configuration du service





Validation.gwt.xml

```
<module>
   <inherits name="com.google.gwt.user.User"/>
   <entry-point class="org.yournamehere.client.ValidationEntryPoint"/>
   <servlet path='/validationservice' class='org.yournamehere.server.ValidationServiceImpl'/>
</module>
```

Dans le fichier de configuration du module pour le test en phase de développement (hosted mode)

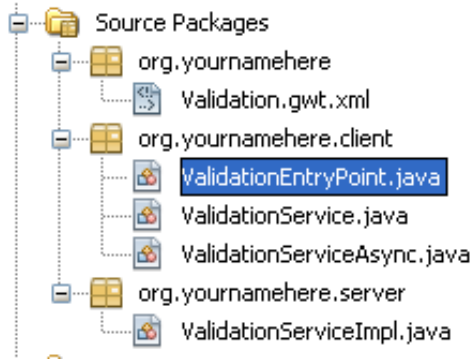
Web.xml

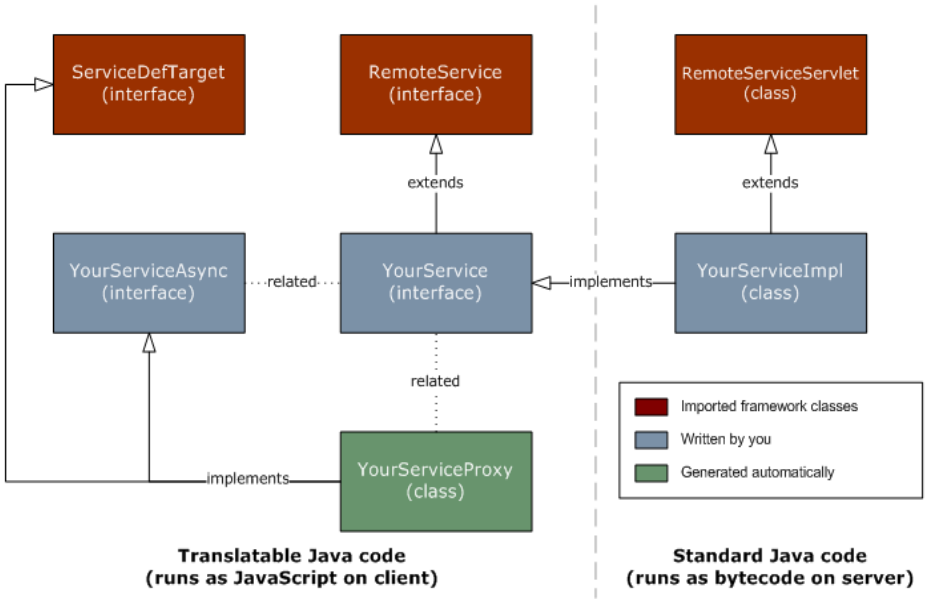
```
<servlet>
   <servlet-name>ValidationService</servlet-name>
   <servlet-class>org.yournamehere.server.ValidationServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>ValidationService</servlet-name>
   <url-pattern>/validationservice</url-pattern>
</servlet-mapping>
```

Dans le fichier web.xml pour le déploiement en production (web mode)

GWT RPC : mise en oeuvre

4 Invoquer le service depuis le client





a) Instancier un proxy client (objet de type ValidationServiceAsync) en utilisant GWT.create()

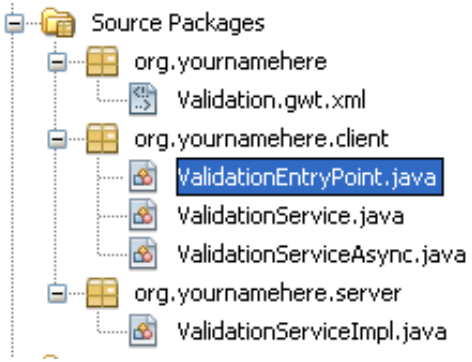
b) Spécifier l'URL point d'entrée pour le proxy en utilisant ServiceDefTarget

c) Créer un objet callback asynchrone qui sera notifié lorsque le RPC sera terminé

d) Faire l'appel depuis le client

GWT RPC : mise en oeuvre

4 Invoquer le service depuis le client



a) Instancier un proxy client (objet de type ValidationServiceAsync)

b) Spécifier l'URL point d'entrée pour le proxy en utilisant ServiceDefTarget

c) Créer un objet callback asynchrone notifié lorsque le RPC sera terminé

d) Faire l'appel depuis le client

ValidationEntryPoint.java

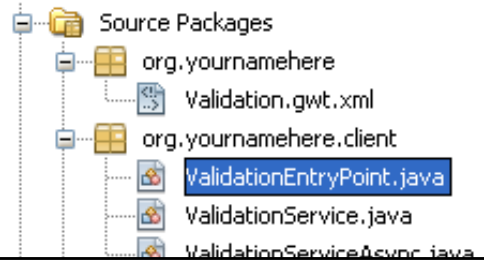
```
public class ValidationEntryPoint implements EntryPoint {

    ...

    public static ValidationServiceAsync getService() {

   (a) ValidationServiceAsync service = (ValidationServiceAsync) GWT.create(ValidationService.class);

   (b) ServiceDefTarget endpoint = (ServiceDefTarget) service;
       String moduleRelativeURL = GWT.getModuleBaseURL() + "validationservice";
       endpoint.setServiceEntryPoint(moduleRelativeURL);

        return service;
    }
}
```

GWT génére le code du proxy pour le service.

URL à laquelle l'implémentation du service s'exécute.

GWT RPC : mise en oeuvre

4 Invoquer le service depuis le client



a) Instancier un proxy client (objet de type ValidationServiceAsync)

b) Spécifier l'URL point d'entrée pour le proxy en utilisant ServiceDefTarget

c) Créer un objet callback asynchrone notifié lorsque le RPC sera terminé

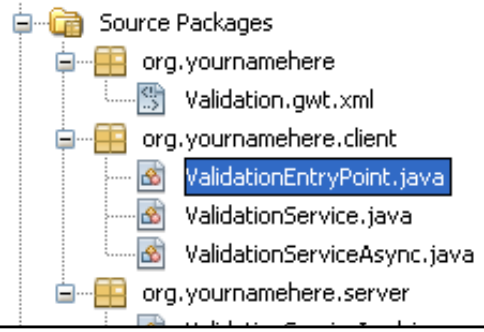
```
public class ValidationEntryPoint implements EntryPoint {

    ...
    public void onModuleLoad() {

        final Label userID = new Label("User ID : ");
        final HTML lblServerReply = new HTML();
        final TextBox txtUserInput = new TextBox();
        final Button button = new Button("Create Account");

        final AsyncCallback callback = new AsyncCallback() {

            public void onSuccess(Object result) {
                boolean res = ((Boolean) result).booleanValue();
                if (res) {
                    lblServerReply.setHTML("<div style=\"color:green\">Identit&eacute; Valide ! </div>");
                    button.setEnabled(true);
                } else {
                    lblServerReply.setHTML("<div style=\"color:red\">Identit&eacute; invalide ! </div>");
                    button.setEnabled(false);
                }
            }

            public void onFailure(Throwable caught) {
                lblServerReply.setText("Communication failed");
            }
        };
}
```

c

Classe interne, ce qui lui permet d'accéder aux variables locales de la méthode onModuleLoad pour la mise à jour de l'interface

GWT RPC : mise en oeuvre

4 Invoquer le service depuis le client



a) Instancier un proxy client (objet de type ValidationServiceAsync)

b) Spécifier l'URL point d'entrée pour le proxy en utilisant ServiceDefTarget

c) Créer un objet callback asynchrone notifié lorsque le RPC sera terminé

d) Faire l'appel depuis le client

```
public class ValidationEntryPoint implements EntryPoint {

    ...
    public void onModuleLoad() {

        final Label userID = new Label("User ID : ");
        final HTML lblServerReply = new HTML();
        final TextBox txtUserInput = new TextBox();
        final Button button = new Button("Create Account");

        final AsyncCallback callback = new AsyncCallback() {

            public void onSuccess(Object result) { ... }

            public void onFailure(Throwable caught) {...}
        };

        txtUserInput.addKeyboardListener(new KeyboardListenerAdapter() {

            public void onKeyUp(Widget sender, char keyCode, int modifiers) {
                getService().validate(txtUserInput.getText(), callback);
            }
        });
        ...
```

d

Gestionnaire d'événements clavier associé au champ de saisie de l'identifiant de l'utilisateur

Appel du serveur à distance. Cet appel est asynchrone, le flot de contrôle continuera immédiatement et plus tard le callback sera invoqué quand le service aura été exécuté.

RPC et exceptions

- Les appels RPC peuvent provoquer de nombreuses erreurs
  - Problème réseau, panne du serveur, erreur lors d'un traitement d'une requête
- GWT permet de traiter ce type de problèmes à l'aide d'exceptions java.

- Les méthodes d'une interface de service peuvent définir des clauses throws

- Les exceptions ainsi déclarées doivent être traitées dans la méthode onFailure(Throwable) de l'objet callback.

- Si un appel de service distant ne peut aboutir (réseau coupé, problème de DNS, arrêt du serveur HTTP) une exception de type InvocationException est passé à la méthode onFailure.

GWT : conclusion

- Les atouts de GWT sont nombreux :
  - Simplicité de mise en œuvre
    - Utilise un paradigme de programmation connu
    - Unifie les technologies nécessaires au développement d'applications WeB
      - Il ne vient pas s'ajouter à la pile des technologies Web/Java (servlets, JSP, JSTL, Struts…) il les remplace.
    - Pas besoin d'apprendre/utiliser javascript
      - Pas besoin de gérer les incompatibilités entre navigateurs (GWT le fait pour vous!)
  - Intégration à la plateforme Java
    - Le contrôle statique des types du langage Java réduit les erreurs de programmation et améliore la productivité
      - Des erreurs JavaScript communes (typos, incompatibilités de types) sont facilement détectées à la compilation plutôt qu'à l'exécution.
    - Le mode hôte facilite la mise au point
    - Utilisation des environnements de développement Java (Eclipse, Netbeans…)
      - Plugin GWTDesigner sous Eclipse

GWT : conclusion

- Les atouts de GWT … suite :
  - Ouverture
    - Facilement extensible
      - De nombreux frameworks opensource complète la palette de composants de base de GWT : GWT ext, MyGWT, GWidget, GWT components
    - Capacité d'intégrer des frameworks javascript externes (JSNI)
      - Ex : projet Tatami ObjetDirect – France Telecom pour encapsulation de DOJO dans des composants GWT
  - Robustesse et stabilité
    - Google prend son temps avant de retirer le Tag beta à une API (GWT 1.4 beta)
  - Pérennité
    - Projet opensource (licence apache 2.0) avec un géant comme principal sponsor
    - Une communauté très active

GWT : conclusion

- Mais encore quelques difficultés :
  - Programmation asynchrone peut être déroutante
    - Un traitement séquentiel doit être découpé en une série de callback
  - Intégration de javascript peut se révéler délicate
  - Tout java n'est pas traductible en javascript
    - Partiellement java.lang et java.util
    - Pas de support pour Java 5
      - Généricité, annotations…
  - Echange des données entre le client et le serveur
    - Sérialisation à la google (interface IsSerializable plus restrictive que java.io.Serializable)
  - Nécessité d'une réflexion sur les architectures des applications GWT
    - Problème plus général à Ajax et RIA