74. AJAX

Niveau : Supérieur

Le terme AJAX est l'acronyme de "Asynchronous JavaScript and XML", utilisé pour la première fois par Jesse James Garrett dans son article " AJAX: A New Approach to Web Applications". Ce terme s'est depuis popularisé.

Cependant AJAX est un acronyme qui reflète relativement mal ce qu'est AJAX en réalité : AJAX est un concept qui n'est pas lié particulièrement à un langage de développement et à un format d'échange de données. Pour faciliter la portabilité, la mise en oeuvre courante d'AJAX fait appel aux technologies Javascript et XML.

AJAX n'est pas une technologie mais plutôt une architecture technique qui permet d'interroger un serveur de manière asynchrone pour obtenir des informations permettant de mettre à jour dynamiquement la page HTML en manipulant son arbre DOM via DHTML.

AJAX est donc un modèle technique dont la mise en oeuvre intègre généralement plusieurs technologies :

- Une page web faisant usage d'AJAX (HTML/CSS/JavaScript)
- Une communication asynchrone avec un serveur pour obtenir des données
- Une manipulation de l'arbre DOM de la page pour permettre sa mise à jour (DHTML)
- Une utilisation d'un langage de script (JavaScript généralement) pour réaliser les différentes actions côté client

Historiquement, les développeurs d'applications web concentrent leurs efforts sur la partie métier et la persistance des données. La partie IHM est souvent délaissée essentiellement en invoquant les limitations de la technologie HTML. La maturité des technologies misent en oeuvre par le DHTML permettent maintenant de développer des applications plus riches et plus dynamiques.

Le but principal d'AJAX est d'éviter le rechargement complet d'une page pour n'en mettre qu'une partie à jour. Ceci permet donc d'améliorer l'interactivité et le dynamisme de l'application web qui le met en oeuvre.

Le fait de pouvoir opérer des actions asynchrones côté serveur et des mises à jour partielles d'une page web permet d'offrir de nombreuses possibilités de fonctionnalités :

- Rafraîchissement de données : par exemple rafraîchir le contenu d'une liste lors d'une pagination
- Auto-complétion d'une zone de saisie
- Validation de données en temps réel
- Modifier les données dans une table sans utiliser une page dédiée pour faire la mise à jour. Lors du clic sur un bouton modifier, il est possible de transformer les zones d'affichage en zones de saisie, d'utiliser AJAX pour envoyer une requête de mise à jour à la validation coté serveur et de réafficher les données modifiées à la place des zones de saisies
- ...

Les utilisations d'AJAX sont donc nombreuses mais cette liste n'est pas exhaustive : cependant elle permet déjà de comprendre qu'AJAX peut rendre les applications web plus dynamiques et interactives.

Les technologies requises pour mettre en oeuvre AJAX sont disponibles depuis plusieurs années (les concepts proposés par AJAX ne sont pas récents puisque Microsoft proposait déjà une solution équivalente dans Internet Explorer 5).

La mise à disposition de ces concepts dans la plupart des navigateurs récents permet à AJAX de connaître un énorme engouement essentiellement justifié par les fonctionnalités proposées et par des mises en oeuvres à succès des sites tels que Google Gmail, Google Suggest ou Google GMaps. Cet engouement va jusqu'à qualifier de façon générale l'utilisation d'AJAX et quelques autres concepts sous le terme Web 2.0.

Le succès d'AJAX est assuré par le fait qu'il apporte aux utilisateurs d'applications web des fonctionnalités manquantes déjà bien connues dans les applications de type standalone. La mise à jour dynamique de la page apporte aux utilisateurs une convivialité et une rapidité dans les applications web.

L'accroissement de l'utilisation d'AJAX permet de voir apparaître des frameworks qui facilitent sa mise en oeuvre et son intégration dans les

applications. Un de ces framework, le framework DWR, est présenté dans ce chapitre.

Le Java BluePrints de Sun recense les meilleures pratiques d'utilisation d'AJAX avec J2EE : chaque référence propose une description, une solution de conception et un exemple de code fonctionnel mettant en oeuvre la solution. Ces références sont actuellement l'auto-complétion, une barre de progression et la validation des données d'un formulaire.

AJAX est aussi en cours d'intégration dans les Java Server Faces.

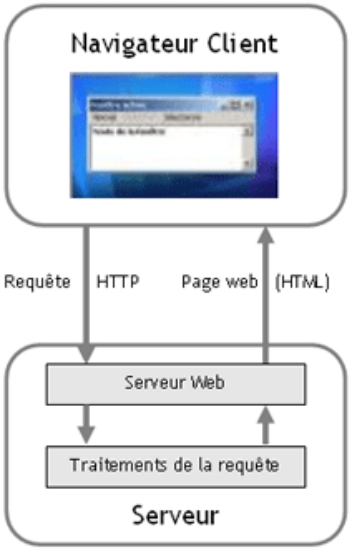
Ce chapitre contient plusieurs sections :

74.1. La présentation d'AJAX

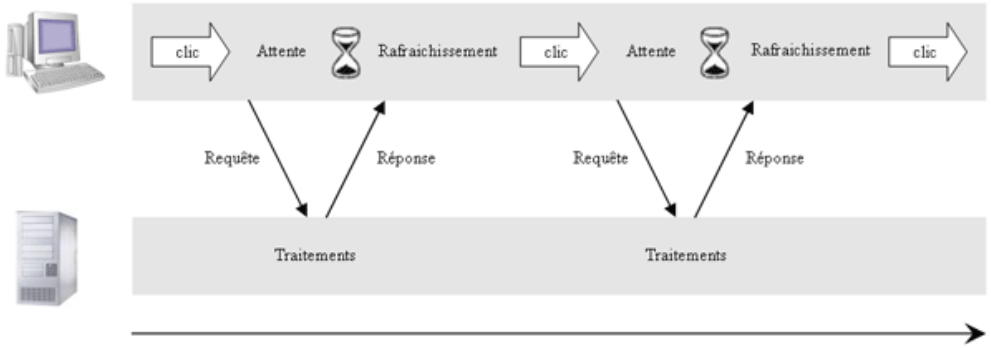
Traditionnellement les pages HTML ont besoin d'être entièrement rafraîchies dès lors qu'une simple portion de la page doit être rafraîchie. Ce mode de fonctionnement possède plusieurs inconvénients :

- limite les temps de réponse de l'application,
- augmente la consommation de bande passante et de ressources côté serveur
- perte du contexte lié au protocole http (utilisation de mécanismes tels que les cookies ou la session pour conserver un état)

Dans une application web, les échanges entre le client et le serveur sont opérés de manière synchrone. Chaque appel nécessitant un traitement côté serveur impose un rafraîchissement complet de la page. Le mode de fonctionnement d'une application web classique est donc le suivant :



Avant AJAX, les applications web fonctionnaient sur un mode soumission/attente/rafraîchissement total de la page. Chaque appel au serveur retourne la totalité de la page qui est donc entièrement reconstruite et retournée au navigateur pour affichage. Durant cet échange, l'utilisateur est obligé d'attendre la réponse du serveur ce qui implique au mieux un clignotement lors du rafraîchissement de la page ou l'affichage d'une page blanche en fonction du temps de traitement de la requête par le serveur.

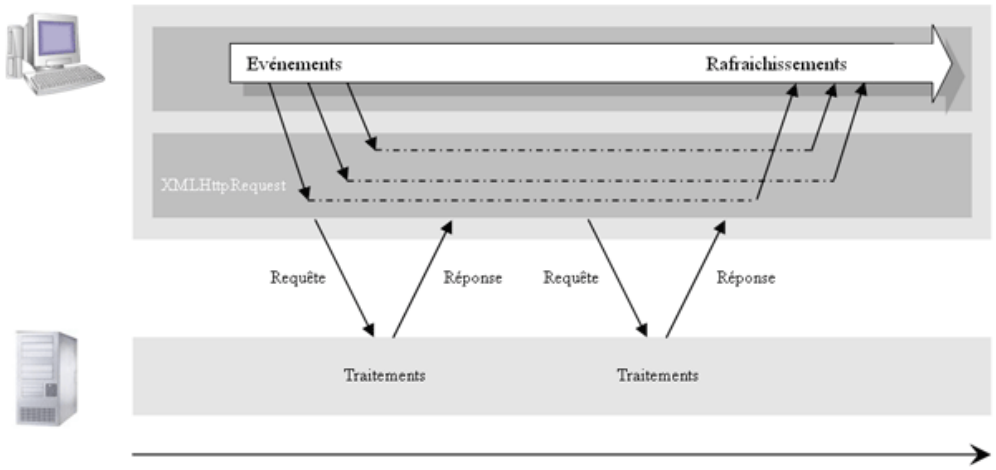


AJAX repose essentiellement sur un échange asynchrone entre le client et le serveur ce qui évite aux utilisateurs d'avoir un temps d'attente obligatoire entre leur action et la réponse correspondante tant qu'ils restent dans la même page. Ce mode de communication et le rafraîchissement partiel de la page en fonction des données reçues en réponse du serveur permettent d'avoir une meilleure réactivité aux actions de l'utilisateur.

Avec AJAX :

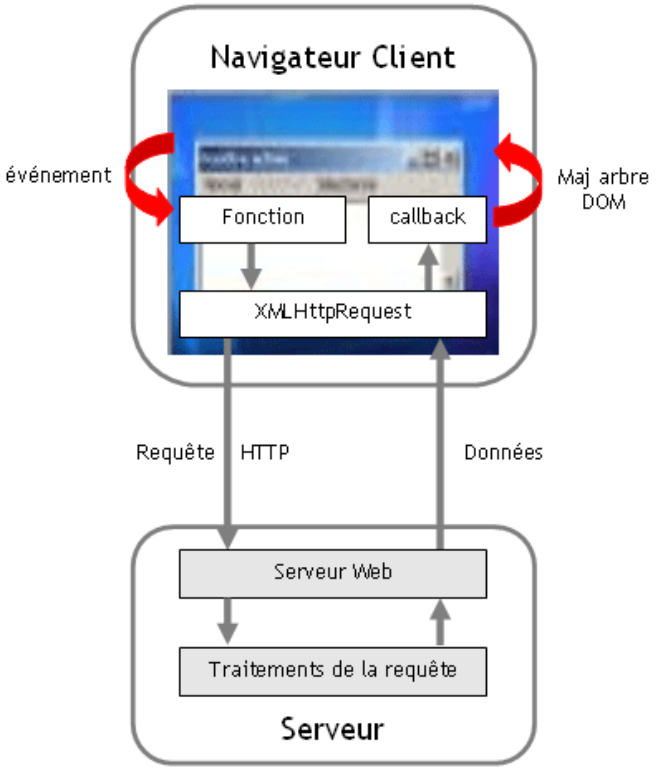
- Le rafraîchissement partiel d'une page remplace le rafraîchissement systématique total de la page
- La communication asynchrone remplace la communication synchrone entre le client et le serveur. Ceci permet d'améliorer l'interactivité entre

l'utilisateur et l'application



L'utilisation d'AJAX dans une application web permet des communications asynchrones et à l'utilisateur de rester dans la page courante. La mise à jour dynamique de la page en fonction de la réponse permet de rendre ses traitements transparents pour l'utilisateur et surtout de lui donner une impression de fluidité.

Le mode de fonctionnement d'une application web utilisant AJAX est le suivant :



Tous ces traitements sont déclenchés par un événement utilisateur sur un composant (clic, changement d'une valeur, perte du focus, ...) ou système (timer, ...) dans la page.

La partie centrale d'AJAX est un moteur capable de communiquer de façon asynchrone avec un serveur en utilisant le protocole http. Généralement les appels au serveur se font grâce à l'objet Javascript XMLHttpRequest. Cet objet n'est pas défini dans les spécifications courantes de JavaScript mais il est implémenté dans tous les navigateurs récents car il devient un standard de facto. Il est donc important de noter qu'AJAX ne fonctionnera pas sur des navigateurs anciens : ceci est à prendre en compte lors d'une volonté d'utilisation d'AJAX ou lors de sa mise en oeuvre.

L'objet JavaScript XMLHttpRequest occupe donc un rôle majeur dans AJAX puisqu'il assure les communications entre le client et le serveur. Généralement ces communications sont asynchrones pour permettre à l'utilisateur de poursuivre ses activités dans la page.

AJAX nécessite une architecture différente côté serveur : habituellement en réponse à une requête le serveur renvoie le contenu de toute la page. En réponse à une requête faite grâce à AJAX, le serveur doit renvoyer des informations qui seront utilisées côté client par du code JavaScript pour mettre à jour la page. Le format de ces informations est généralement XML mais ce n'est pas une obligation.

Le rafraîchissement partiel d'une page grâce à AJAX permet d'accroître la réactivité de l'IHM mais aussi de diminuer la bande passante et les ressources serveurs consommées lors d'un rafraîchissement complet de la page web.

La mise à jour partielle d'une page en modifiant directement son arbre DOM permet de conserver le contexte de l'état de la page. Les parties inchangées du DOM restent toujours connues et utilisables. Cependant un des effets pervers pour l'utilisateur est l'utilisation du bouton back du navigateur : l'utilisateur est habitué à obtenir l'état précédent de la page dans le cas d'un rafraîchissement à chaque action. Ce nouveau mode peut être déroutant pour l'utilisateur.

74.2. Le détail du mode de fonctionnement

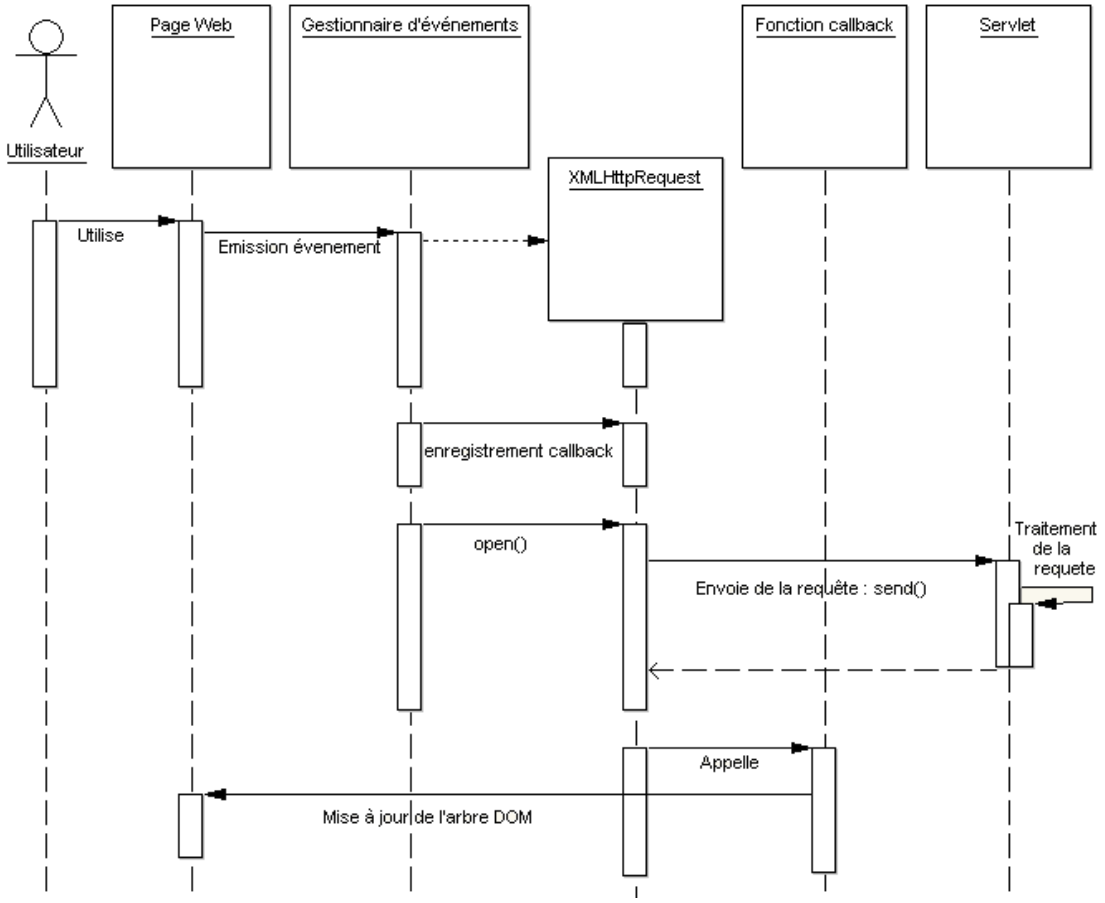
La condition pour utiliser AJAX dans une application web est que le support de JavaScript soit activé dans le navigateur et que celui-ci propose une implémentation de l'objet XMLHttpRequest.

L'objet XMLHttpRequest permet un échange synchrone ou asynchrone avec le serveur en utilisant le protocole HTTP. La requête http envoyée au serveur peut être de type GET ou POST.

Une communication asynchrone permet au navigateur de ne pas bloquer les actions de l'utilisateur en attendant la réponse du serveur. Ainsi, une fonction de type callback est enregistrée pour permettre son appel à la réception de la réponse http.

Côté serveur toute technologie permettant de répondre à une requête http peut être utilisée avec AJAX. J2EE et plus particulièrement les servlets se prêtent particulièrement bien à ces traitements. La requête http est traitée comme toutes les requêtes de ce type. En fonction des paramètres reçus de la requête, des traitements sont exécutés pour générer la réponse http.

A la réception de la réponse par le client, la fonction de type callback est appelée. Elle se charge d'extraire les données de la réponse et de réaliser les traitements de mise à jour de la page web en manipulant son arbre DOM.



Les avantages d'AJAX sont :

- Une économie de ressources côté serveur et de bande passante puisque la page n'est pas systématiquement transmise pour une mise à jour
- Une meilleure réactivité et une meilleure dynamique de l'application web

AJAX possède cependant quelques inconvénients :

- Complexité liée à l'utilisation de plusieurs technologies côté client et serveur
- Utilisation de JavaScript : elle implique la prise en compte des inconvénients de cette technologie : difficulté pour déboguer, différences d'implémentations selon le navigateur, code source visible, ...
- AJAX ne peut être utilisé qu'avec des navigateurs possédant une implémentation de l'objet XMLHttpRequest
- L'objet XMLHttpRequest n'est pas standardisé ce qui nécessite des traitements JavaScript dépendants du navigateur utilisé
- Le changement du mode de fonctionnement des applications web (par exemple : impossible de faire un favori vers une page dans un certain état, le bouton back ne permet plus de réafficher la page dans son état précédent la dernière action, ...)
- La mise en oeuvre de nombreuses fonctionnalités mettant en oeuvre AJAX peut faire rapidement augmenter le nombre de requêtes http à traiter par le serveur
- Le manque de frameworks et d'outils pour faciliter la mise en oeuvre

AJAX possède donc quelques inconvénients qui nécessitent une sérieuse réflexion pour une utilisation intensive dans une application. Un bon compromis est d'utiliser AJAX pour des fonctionnalités permettant une amélioration de l'interactivité entre l'application et l'utilisateur.

Actuellement, AJAX et en particulier l'objet XMLHttpRequest n'est pas un standard. De plus, reposant essentiellement sur JavaScript, son bon fonctionnement ne peut pas être assuré sur tous les navigateurs. Pour ceux avec qui cela peut l'être, le support de JavaScript doit être activé et il est quasiment impératif d'écrire du code dépendant du navigateur utilisé.

Il peut donc être nécessaire de prévoir, lors du développement de l'application, le bon fonctionnement de cette dernière sans utiliser AJAX. Cela permet notamment un fonctionnement correct sur les anciens navigateurs ou sur les navigateurs où le support de JavaScript est désactivé.

Le plus simple pour assurer cette tâche est de détecter au démarrage de l'application si l'objet XMLHttpRequest est utilisable dans le navigateur de l'utilisateur. Dans l'affirmative, l'application renvoie une version avec AJAX de la page sinon une version sans AJAX.

Comme la requête est asynchrone, il peut être important d'informer l'utilisation sur l'état des traitements en cours et surtout sur le succès ou l'échec de leur exécution. Avec un rafraîchissement traditionnel complet de la page c'est facile. En utilisant AJAX, il est nécessaire de faire usage de subtilités d'affichage ou d'effets visuels auxquels l'utilisateur n'est pas forcement habitué. Un exemple concret concerne un bouton de validation : il est utile de modifier le libellé du bouton pour informer l'utilisateur que les traitements sont en cours afin d'éviter qu'il clique plusieurs fois sur le bouton.

Il faut aussi garder à l'esprit que les échanges asynchrones ne garantissent pas que les réponses arrivent dans le même ordre que les requêtes correspondantes sont envoyées. Il est même tout à fait possible de ne jamais recevoir une réponse. Il faut donc être prudent si l'on enchaîne plusieurs requêtes.

74.3. Un exemple simple

Cet exemple va permettre de réaliser une validation côté serveur d'une donnée saisie en temps réel.

Une servlet permettra de réaliser cette validation. La validation proposée est volontairement simpliste et pourrait même être réalisée directement côté client avec du code JavaScript. Il faut cependant comprendre que les traitements de validation pourraient être beaucoup plus complexes avec par exemple une recherche dans une base de données, ce qui justifierait pleinement l'emploi d'une validation côté serveur.

Les actions suivantes sont exécutées dans cet exemple :

- Un événement déclencheur est émis (la saisie d'une donnée par l'utilisateur)
- Création et paramétrage d'un objet de type XMLHttpRequest
- Appel de la servlet par l'objet XMLHttpRequest
- La servlet exécute les traitements de validation et renvoie le résultat en réponse au format XML
- L'objet XMLHttpRequest appelle la fonction d'exploitation de la réponse
- La fonction met à jour l'arbre DOM de la page en fonction des données de la réponse

74.3.1. L'application de tests

La page de test est une JSP qui contient un champ de saisie.

Exemple :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Test validation AJAX</title>
<script type="text/javascript">
<!--
var requete;

function valider() {
   var donnees = document.getElementById("donnees");
   var url = "valider?valeur=" + escape(donnees.value);
   if (window.XMLHttpRequest) {
       requete = new XMLHttpRequest();
   } else if (window.ActiveXObject) {
       requete = new ActiveXObject("Microsoft.XMLHTTP");
   }
   requete.open("GET", url, true);
   requete.onreadystatechange = majIHM;
   requete.send(null);
}

function majIHM() {
  var message = "";

  if (requete.readyState == 4) {
    if (requete.status == 200) {
      // exploitation des données de la réponse
      var messageTag = requete.responseXML.getElementsByTagName("message")[0];
      message = messageTag.childNodes[0].nodeValue;
      mdiv = document.getElementById("validationMessage");
      if (message == "invalide") {
         mdiv.innerHTML = "<img src='images/invalide.gif'>";
      } else {
         mdiv.innerHTML = "<img src='images/valide.gif'>";
      }
    }
  }
```

```
}

//-->
</script>
</head>
<body>
<table>
    <tr>
        <td>Valeur :</td>
        <td nowrap><input type="text" id="donnees" name="donnees" size="30"
            onkeyup="valider();"></td>
        <td>
        <div id="validationMessage"></div>
        </td>
    </tr>
</table>
</body>
</html>
```

Le code JavaScript est détaillé dans les sections suivantes.

L'application contient aussi une servlet qui sera détaillée dans une des sections suivantes.

Le descripteur de déploiement de l'application contient la déclaration de la servlet.

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
      "web-app_2_2.dtd">
<web-app>
    <display-name>Test de validation avec AJAX</display-name>
    <servlet>
        <servlet-name>ValiderServlet</servlet-name>
        <display-name>ValiderServlet</display-name>
        <description>Validation de données</description>
        <servlet-class>
        com.jmd.test.ajax.ValiderServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ValiderServlet</servlet-name>
        <url-pattern>/valider</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

74.3.2. La prise en compte de l'événement déclencheur

Un événement onkeyup est associé à la zone de saisie des données. Cet événement va appeler la fonction JavaScript valider().

Exemple :

```
<input type="text" id="donnees" name="donnees" size="30"
               onkeyup="valider();">
```

Ainsi la fonction sera appelée à chaque fois que l'utilisateur saisit un caractère.

74.3.3. La création d'un objet de type XMLHttpRequest pour appeler la servlet

La fonction JavaScript valider() va réaliser les traitements de la validation des données.

Exemple :

```
var requete;

function valider() {
   var donnees = document.getElementById("donnees");
   var url = "valider?valeur=" + escape(donnees.value);
   if (window.XMLHttpRequest) {
      requete = new XMLHttpRequest();
      requete.open("GET", url, true);
```

```
09.         requete.onreadystatechange = majIHM;
10.         requete.send(null);
11.     } else if (window.ActiveXObject) {
12.         requete = new ActiveXObject("Microsoft.XMLHTTP");
13.         if (requete) {
14.             requete.open("GET", url, true);
15.             requete.onreadystatechange = majIHM;
16.             requete.send();
17.         }
18.     } else {
19.         alert("Le navigateur ne supporte pas la technologie AJAX");
20.     }
21. }
```

Elle réalise les traitements suivants :

- récupère les données saisies
- détermine l'url d'appel de la servlet en passant en paramètre les données. Ces données sont encodées selon la norme http grâce à la fonction escape().
- instancie une requête de type XMLHttpRequest en fonction du navigateur utilisé
- associe à la requête l'url et la fonction à exécuter à la réponse
- exécute la requête

Comme dans de nombreux usages courants de JavaScript, des traitements dépendants du navigateur cible sont nécessaires à l'exécution. Dans le cas de l'instanciation de l'objet XMLHttpRequest, celui-ci est un ActiveX sous Internet Explorer et un objet natif sur les autres navigateurs qui le supportent.

La signature de la méthode open de l'objet XMLHttpRequest est XMLHttpRequest.open(String method, String URL, boolean asynchronous).

Le premier paramètre est le type de requête http réalisé par la requête (GET ou POST)

Le second paramètre est l'url utilisée par la requête.

Le troisième paramètre est un booléen qui précise si la requête doit être effectuée de façon asynchrone. Si la valeur passée est true alors une fonction de type callback doit être associée à l'événement onreadystatechange de la requête. La fonction précisée sera alors exécutée à la réception de la réponse.

La méthode send() permet d'exécuter la requête http en fonction des paramètres de l'objet XMLHttpRequest.

Pour une requête de type GET, il suffit de passer null comme paramètre de la méthode send().

Pour une requête de type POST, il faut préciser le Content-Type dans l'en-tête de la requête et fournir les paramètres de la fonction send().

Exemple :

```
1. requete.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
2. requete.send("valeur=" + escape(donnees.value));
```

L'objet XMLHttpRequest possède les méthodes suivantes :

Méthode	Rôle
abort()	Abandon de la requête
getAllResponseHeaders()	Renvoie une chaîne contenant les en-têtes http de la réponse
getResponseHeader(nom)	Renvoie la valeur de l'en-tête dont le nom est fourni en paramètre
setTimeouts(duree)	Précise la durée maximale pour l'obtention de la réponse
setRequestHeader(nom, valeur)	Précise la valeur de l'en-tête dont le nom est fourni en paramètre
open(méthode, url, [asynchrone[, utilisateur[, motdepasse]]]	Prépare une requête en précisant la méthode (Get ou Post), l'url, un booléen optionnel qui précise si l'appel doit être asynchrone et le user et/ou le mot de passe optionnel
send(data)	Envoi de la requête au serveur

L'objet XMLHttpRequest possède les propriétés suivantes :

Propriété	Rôle
onreadystatchange	Précise la fonction de type callback qui est appelée lorsque la valeur de la propriété readyState change
readyState	L'état de la requête : 0 = uninitialized 1 = loading 2 = loaded 3 = interactive 4 = complete
responseText	Le contenu de la réponse au format texte
responseXML	Le contenu de la réponse au format XML

status	Le code retour http de la réponse
statusText	La description du code retour http de la réponse

Il peut être intéressant d'utiliser une fonction JavaScript qui va générer une chaîne de caractères contenant le nom et la valeur de chacun des éléments d'un formulaire.

Exemple :

```
01. function getFormAsString(nomFormulaire){
02.
03.   resultat ="";
04.   formElements=document.forms[nomFormulaire].elements;
05.
06.   for(var i=0; i<formElements.length; i++ ){
07.     if (i > 0) {
08.       resultat+="&";
09.     }
10.     resultat+=escape(formElements[i].name)+"="
11.       +escape(formElements[i].value);
12.  }
13.
14.  return resultat;
15. }
```

Ceci facilite la génération d'une url qui aurait besoin de toutes les valeurs d'un formulaire.

74.3.4. L'exécution des traitements et le renvoi de la réponse par la servlet

La servlet associée à l'URI "/valider" est exécutée par le conteneur web en réponse à la requête.

Exemple :

```
01. package com.jmd.test.ajax;
02.
03. import java.io.IOException;
04. import javax.servlet.ServletException;
05. import javax.servlet.http.HttpServletRequest;
06. import javax.servlet.http.HttpServletResponse;
07.
08. /**
09.  * Servlet ValiderServlet
10.  *
11.  */
12.  public class ValiderServlet extends javax.servlet.http.HttpServlet
13.                              implements javax.servlet.Servlet {
14.
15.   /* (non-Java-doc)
16.      * @see javax.servlet.http.HttpServlet#HttpServlet()
17.      */
18.     public ValiderServlet() {
19.         super();
20.     }
21.
22.     /* (non-Java-doc)
23.      * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request,
24.      *                                           HttpServletResponse response)
25.      */
26.     protected void doGet(HttpServletRequest request, HttpServletResponse response)
27.                             throws ServletException, IOException {
28.     String resultat = "invalide";
29.     String valeur = request.getParameter("valeur");
30.
31.     response.setContentType("text/xml");
32.     response.setHeader("Cache-Control", "no-cache");
33.
34.     if ((valeur != null) && valeur.startsWith("X")) {
35.       resultat = "valide";
36.     }
37.
38.     response.getWriter().write("<message>"+resultat+"</message>");
39.   }
40. }
```

La validation est assurée si la valeur fournie commence par un caractère "X".

La servlet renvoie simplement un texte indiquant l'état de la validation réalisée dans une balise message.

Il est important que le type Mime retourné dans la réponse soit de type "text/xml".

Il est préférable de supprimer la mise en cache de la réponse par le navigateur. Cette suppression est obligatoire si une même requête peut renvoyer une réponse différente lors de plusieurs appels.

74.3.5. L'exploitation de la réponse

L'objet XMLHttpRequest appelle la fonction de type callback majIHM() à chaque fois que la propriété readyState change de valeur.

La fonction majIHM() commence donc par vérifier la valeur de la propriété readyState. Si celle-ci vaut 4 alors l'exécution de la requête est complète.

Dans ce cas, il faut vérifier le code retour de la réponse http. La valeur 200 indique que la requête a été correctement traitée.

Exemple :

```
01. function majIHM() {
02.   var message = "";
03.
04.   if (requete.readyState == 4) {
05.     if (requete.status == 200) {
06.       // exploitation des données de la réponse
07.       // ...
08.     } else {
09.        alert('Une erreur est survenue lors de la mise à jour de la page');
10.     }
11.   }
12. }
```

En utilisant la valeur de la réponse, la fonction modifie alors le contenu de la page en mettant à jour son arbre DOM. Cette valeur au format XML est obtenue en utilisant la fonction responseXML de l'instance de XMLHttpRequest. La valeur au format texte brut peut être obtenue en utilisant la fonction responseText.

Il est alors possible d'exploiter les données de la réponse.

Exemple :

```
01. function majIHM() {
02.   var message = "";
03.
04.   if (requete.readyState == 4) {
05.     if (requete.status == 200) {
06.       // exploitation des données de la réponse
07.       var messageTag = requete.responseXML.getElementsByTagName("message")[0];
08.       message = messageTag.childNodes[0].nodeValue;
09.       mdiv = document.getElementById("validationMessage");
10.       if (message == "invalide") {
11.           mdiv.innerHTML = "<img src='images/invalide.gif'>";
12.       } else {
13.           mdiv.innerHTML = "<img src='images/valide.gif'>";
14.       }
15.     } else {
16.       alert('Une erreur est survenue lors de la mise à jour de la page.'+
17.             '\n\nCode retour = '+requete.statusText);
18.     }
19.   }
20. }
```

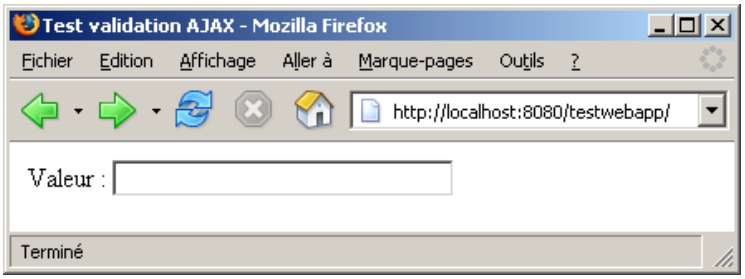
Il est aussi possible que la réponse contienne directement du code HTML à afficher. Il suffit simplement d'affecter le résultat de la réponse au format texte à la propriété innerHTML de l'élément de la page à rafraîchir.

Exemple :

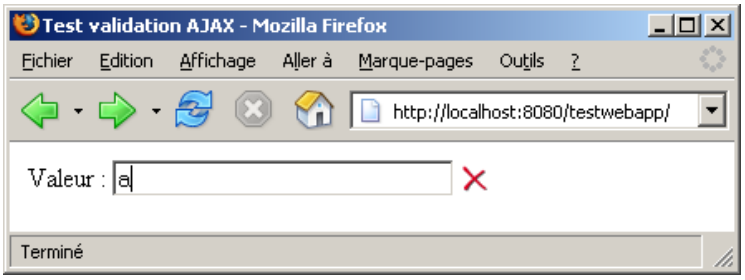
```
01. function majIHM() {
02.   if (requete.readyState == 4) {
03.     if (requete.status == 200) {
04.       document.getElementById("validationMessage").innerHTML = requete.responseText;
05.     } else {
06.       alert('Une erreur est survenue lors de la mise à jour de la page.'+
07.             '\n\nCode retour = '+requete.statusText);
08.   }
09. }
```

74.3.6. L'exécution de l'application

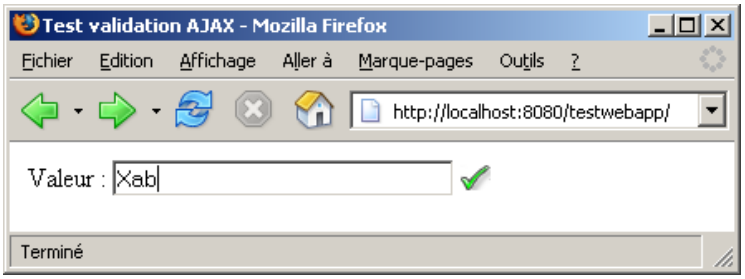
La page de test s'affiche au lancement de l'application



La saisie d'un caractère déclenche la validation



L'icône dépend du résultat de la validation.



74.4. Des frameworks pour mettre en oeuvre AJAX

La mise en oeuvre directe de l'objet XmlHttpRequest est relativement lourde (nécessite l'écriture de nombreuses lignes de code), fastidieuse (pas facile à déboguer) et souvent répétitive. La mise en oeuvre de plusieurs technologies côté client et serveur peut engendrer de nombreuses difficultés notamment dans le code JavaScript (débogage difficile, gestion de la compatibilité du support par les navigateurs, ...).

Aussi de nombreux frameworks commencent à voir le jour pour faciliter le travail des développeurs. Cette section va détailler l'utilisation du framework DWR et proposer une liste non exhaustive d'autres frameworks.

74.4.1. Direct Web Remoting (DWR)

DWR (Direct Web Remoting) est une bibliothèque open source Java dont le but est de faciliter la mise en oeuvre d'AJAX dans les applications Java.

DWR se charge de générer le code JavaScript permettant l'appel à des objets Java de type bean qu'il suffit d'écrire. Sa devise est "Easy AJAX for Java".

DWR encapsule les interactions entre le code JavaScript côté client et les objets Java côté serveur : ceci rend transparent l'appel de ces objets côté client.

La mise en oeuvre de DWR côté serveur est facile :

- Ajouter le fichier dwr.jar au classpath de l'application
- Configurer une servlet dédiée aux traitements des requêtes dans le fichier web.xml
- Ecrire les beans qui seront utilisés dans les pages
- Définir ces beans dans un fichier de configuration de DWR

La mise en oeuvre côté client nécessite d'inclure des bibliothèques JavaScript générées dynamiquement par la servlet de DWR. Il est alors possible d'utiliser les fonctions JavaScript générées pour appeler les méthodes des beans configurés côté serveur.

DWR s'intègre facilement dans une application web puisqu'il repose sur une servlet. Elle s'intégrera plus particulièrement avec les applications mettant en oeuvre le framework Spring dont elle propose un support. DWR est aussi inclus dans le framework WebWork depuis sa version 2.2.

DWR fournit aussi une bibliothèque JavaScript proposant des fonctions de manipulations courantes en DHTML : modifier le contenu des conteneurs <DIV> ou <SPAN>, remplir une liste déroulante avec des valeurs, etc ...

DWR est une solution qui encapsule l'appel de méthodes de simples objets de type Javabean exécutés sur le serveur dans du code JavaScript généré dynamiquement. Le grand intérêt est de masquer toute la complexité de l'utilisation de l'objet XMLHttpRequest et de simplifier à l'extrême le code à développer côté serveur.

DWR se compose de deux parties :

- Du code JavaScript qui envoie des requêtes à la servlet et met à jour la page à partir des données de la réponse
- Une servlet qui traite les requêtes reçues et renvoie une réponse au navigateur

Côté serveur, une servlet est déployée dans l'application web. Cette servlet a deux rôles principaux :

1. Elle permet de générer dynamiquement des bibliothèques de code JavaScript. Deux de celles-ci sont à usage général. Une bibliothèque de code est générée pour chaque bean défini dans la configuration de DWR
2. Elle permet de traiter les requêtes émises par le code JavaScript générés pour appeler la méthode d'un bean

DWR génère dynamiquement le code JavaScript à partir des Javabeans configurés dans un fichier de paramètres en utilisant l'introspection. Ce code se charge d'encapsuler les appels aux méthodes du bean, ceci incluant la conversion du format des données de JavaScript vers Java et vice versa. Ce mécanisme est donc similaire à d'autres solutions de type RPC (remote procedure call).

Une fonction de type callback est précisée à DWR pour être exécutée par un bean à la réception de la réponse à la requête.

DWR facilite donc la mise en oeuvre d'AJAX avec Java côté serveur : il se charge de toute l'intégration de Javabeans pour permettre leur appel côté client de manière transparente.

Le site officiel de DWR est à l'url : http://directwebremoting.org/dwr/

La documentation de ce projet est particulièrement riche et de nombreux exemples sont fournis sur le site.

La version utilisée dans cette section est la version 1.1.1. Elle nécessite un JDK 1.3 et conteneur web supportant la version 2.2 de l'API servlet.

74.4.1.1. Un exemple de mise en oeuvre de DWR

Il faut télécharger le fichier dwr.jar sur le site officiel de DWR et l'ajouter dans le répertoire WEB-INF/Lib de l'application web qui va utiliser la bibliothèque.

Il faut ensuite déclarer dans le fichier de déploiement de l'application web.xml la servlet qui sera utilisée par DWR. Il faut déclarer la servlet et définir son mapping :

Exemple :

```
01. <servlet>
02.     <servlet-name>dwr-invoker</servlet-name>
03.     <display-name>DWR Servlet</display-name>
04.     <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
05.
06.     <init-param>
07.         <param-name>debug</param-name>
08.         <param-value>true</param-value>
09.     </init-param>
10. </servlet>
11.
12. <servlet-mapping>
13.     <servlet-name>dwr-invoker</servlet-name>
14.     <url-pattern>/dwr/*</url-pattern>
15. </servlet-mapping>
```

Il faut créer un fichier de configuration pour DWR nommé dwr.xml dans le répertoire WEB-INF de l'application

Exemple :

```
01. <!DOCTYPE dwr PUBLIC
02.     "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
03.     "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
04. <dwr>
05.   <allow>
06.     <create creator="new" javascript="JDate">
07.       <param name="class" value="java.util.Date"/>
08.     </create>
09.   </allow>
10. </dwr>
```

Ce fichier permet de déclarer à DWR la liste des beans qu'il devra encapsuler pour des appels en JavaScript. Dans l'exemple, c'est la classe java.util.Date fournie dans l'API standard qui est utilisée.

Le creator de type "new" instancie la classe en utilisant le constructeur sans argument. L'attribut javascript permet de préciser le nom de l'objet JavaScript qui sera utilisé côté client.

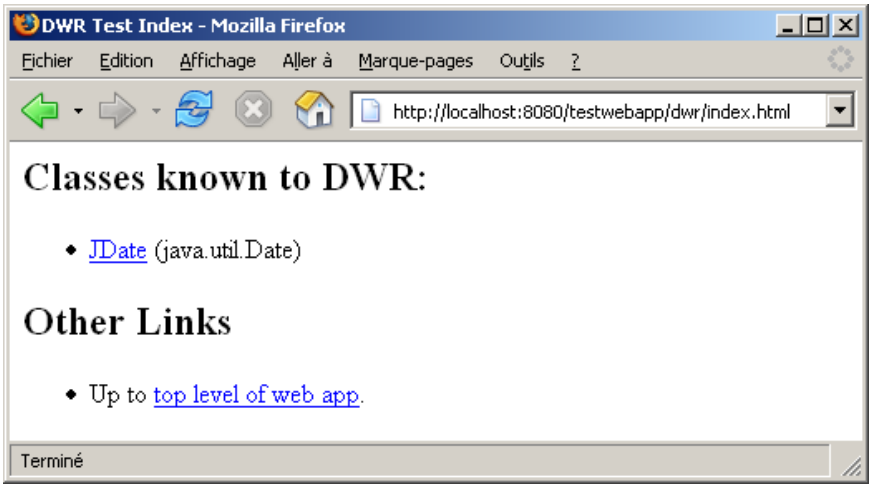
Le tag param avec l'attribut name ayant pour valeur class permet de préciser le nom pleinement qualifié du Bean à encapsuler.

DWR possède quelques restrictions :

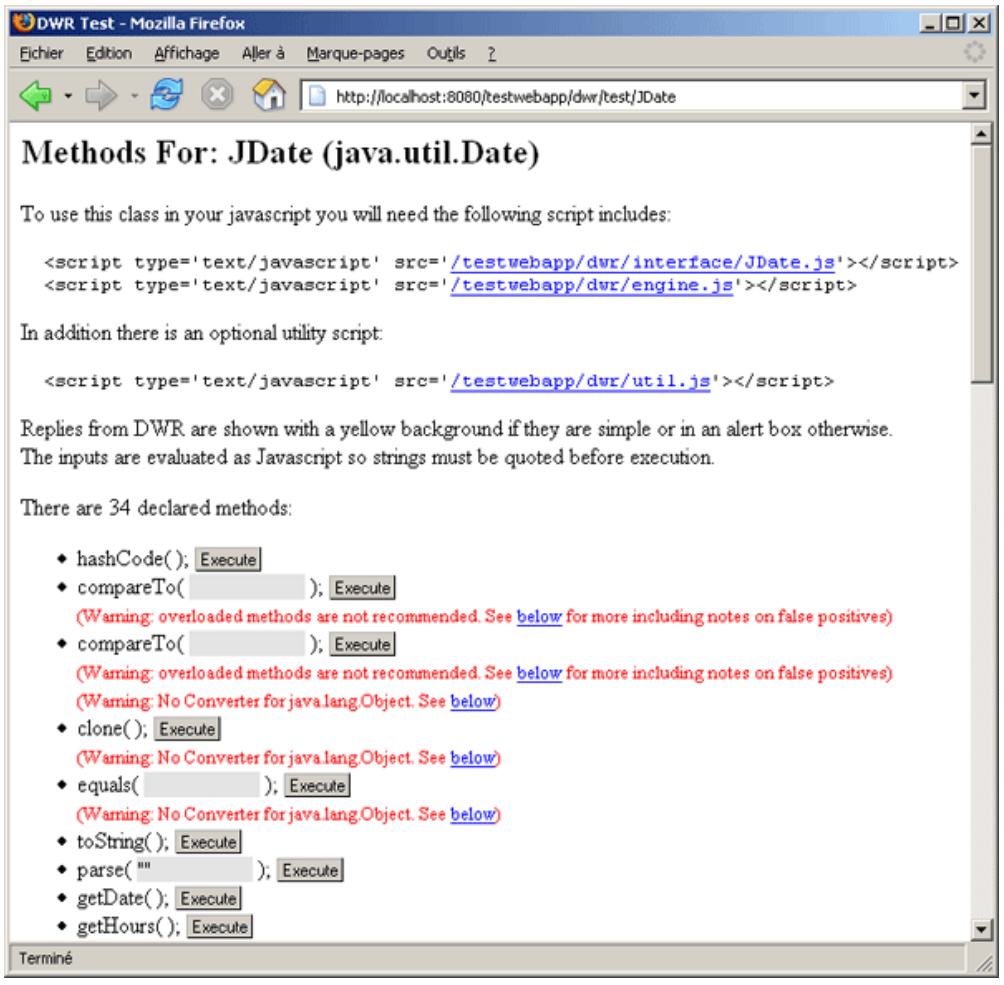
- Il ne faut surtout pas utiliser comme noms de méthodes dans les beans exposés des mots réservés en JavaScript. Un exemple courant est le mot delete
- Il faut éviter l'utilisation de méthodes surchargées

Par défaut, DWR encapsule toutes les méthodes public de la classe définie. Il est donc nécessaire de limiter les méthodes utilisables par DWR à celles requises par les besoins de l'application soit dans la définition des membres de la classe soit dans le fichier de configuration de DWR.

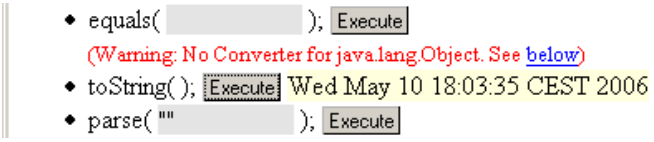
Il suffit alors de lancer l'application et d'ouvrir un navigateur sur l'url de l'application en ajoutant /dwr



Cette page liste tous les beans qui sont encapsulés par DWR. Il suffit de cliquer sur le lien d'un bean pour voir afficher une page de test de ce bean. Cette page génère dynamiquement une liste de toutes les méthodes pouvant être appelées en utilisant DWR.

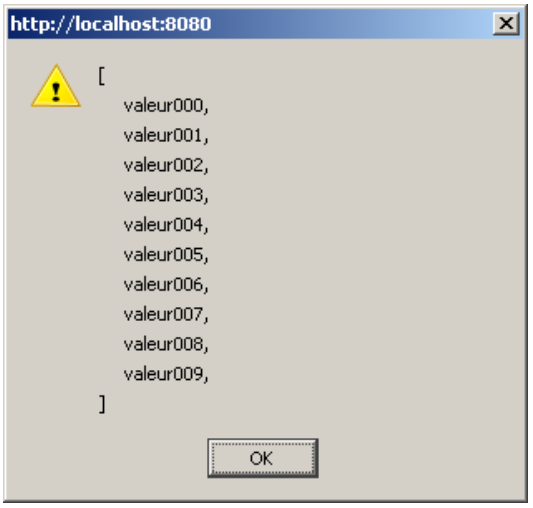


Pour exécuter dynamiquement une méthode sans paramètre, il suffit de simplement cliquer sur le bouton "Execute" de la méthode correspondante.

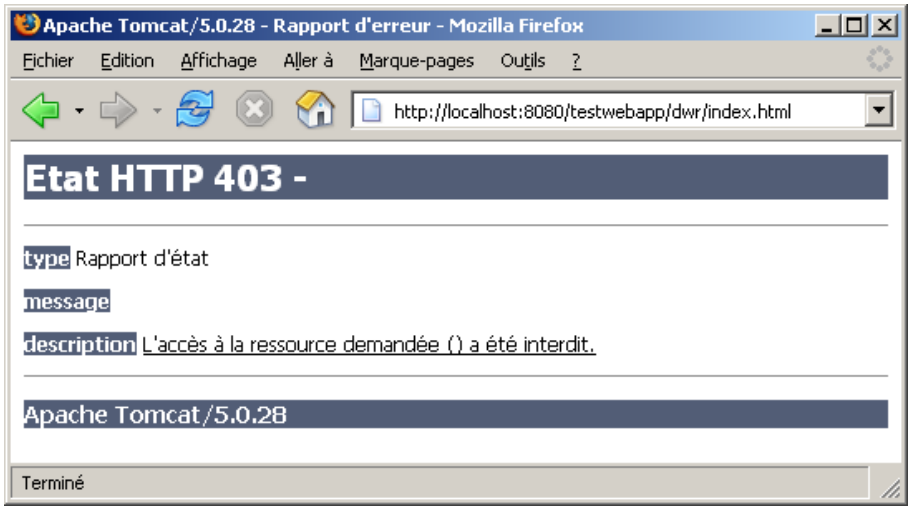


Pour exécuter dynamiquement une méthode avec paramètres, il suffit de saisir leurs valeurs dans leurs zones respectives et de cliquer sur le bouton "Execute".

Si la valeur retournée par la méthode n'est pas une valeur simple alors le résultat est affiché dans une boîte de dialogue.



Si le paramètre debug de la servlet DWR est à false, il n'est pas possible d'accéder à ses fonctionnalités de tests.



Ce mode debug proposé par DWR est particulièrement utile lors de la phase de développement pour vérifier toutes les méthodes qui sont prises en compte par DWR et les tester. Pour des raisons de sécurité, il est fortement déconseillé de l'autoriser dans un contexte de production.

Pour permettre l'utilisation des scripts générés, il suffit de faire un copier/coller dans la partie en-tête de la page HTML des tags <SCRIPT> proposés dans la page de tests de DWR.

Exemple :

```
1. <script type='text/javascript' src='/testwebapp/dwr/interface/JDate.js'></script>
2. <script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
3. <script type='text/javascript' src='/testwebapp/dwr/util.js'></script>
```

Remarque : il est possible d'utiliser un chemin relatif plutôt qu'un chemin absolu pour ces ressources.

74.4.1.2. Le fichier DWR.xml

Le fichier dwr.xml permet de configurer DWR. Il est généralement placé dans le répertoire WEB-INF de l'application web exécutant DWR.

Le fichier dwr.xml a la structure suivante :

Exemple :

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">

<dwr>

  <init>
    <creator id="..." class="..."/>
    <converter id="..." class="..."/>
  </init>

  <allow>
    <create creator="..." javascript="..."/>
    <convert converter="..." match="..."/>
  </allow>

  <signatures>
    ...
  </signatures>

</dwr>
```

Le tag optionnel <init> permet de déclarer ses propres créateurs et convertisseurs. Généralement, ce tag n'est pas utilisé car les créateurs et convertisseurs fournis en standard sont suffisants.

Le tag <allow> permet de définir les objets qui seront utilisés par DWR.

Le tag <create> permet de préciser la façon dont un objet va être instancié. Chaque classe qui pourra être appelée par DWR doit être déclarée avec un tel tag. Ce tag possède la structure suivante :

Exemple :

```
<allow>
  <create creator="..." javascript="..." scope="...">
    <param name="..." value="..."/>
    <auth method="..." role="..."/>
    <exclude method="..."/>
    <include method="..."/>
  </create>
  ...
</allow>
```

Les tags fils <param>, <auth>, <exclude>, <include> sont optionnels

La déclaration d'au moins un créateur est obligatoire. Il existe plusieurs types de créateurs spécifiés par l'attribut creator du tag fils <create> :

Type de créateur	Rôle
new	Instancie l'objet avec l'opérateur new
null	Ne crée aucune instance. Ceci est utile si la ou les méthodes utilisées sont statiques
scripted	Instancie l'objet en utilisant un script via BSF
spring	Le framework Spring est responsable de l'instanciation de l'objet
jsf	Utilise des objets de JSF
struts	Utilise des ActionForms de Struts
pageflow	Permet l'action au PageFlow de Beehive ou WebLogic

L'attribut javascript permet de donner le nom de l'objet Javascript. Il ne faut pas utiliser comme valeur un mot réservé de JavaScript.

L'attribut optionnel scope permet de préciser la portée du bean. Les valeurs possibles sont : application, session, request et page. Sa valeur par défaut est page.

Le tag <param> permet de fournir des paramètres au créateur. Par exemple, avec le creator new, il est nécessaire de fournir en paramètre le nom pleinement qualifié de la classe à instancier

Exemple :

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
  <allow>
    <create creator="new" javascript="JDate">
      <param name="class" value="java.util.Date"/>
    </create>
    <create creator="new" javascript="TestDWR">
      <param name="class" value="com.jmd.test.ajax.dwr.TestDWR"/>
    </create>
  </allow>
```

```
</dwr>
```

Avec le fichier de configuration dwr.xml, DWR propose un mécanisme qui permet de limiter les méthodes qui lui seront accessibles. Les tags <include> et <exclude> permettent respectivement d'autoriser ou d'exclure l'utilisation d'une liste de méthodes. Ces deux tags sont mutuellement exclusifs, en l'absence de l'un deux, toutes les méthodes sont utilisables.

Le tag <auth> permet de gérer la sécurité d'accès en utilisant les rôles J2EE de l'application : DWR propose donc la prise en compte des rôles J2EE définis dans le conteneur web pour restreindre l'accès à certaines classes.

Le tag <converter> permet de préciser la façon dont un objet utilisé en paramètre ou en type de retour va être converti. Un convertisseur assure la transformation des données entre le format des objets client (Javascript) et serveur (Java).

Chaque bean utilisé en tant que paramètre doit être déclaré dans un tel tag. Par défaut, l'utilisation du tag <converter> est inutile pour les primitives, les wrappers de ces primitives (Integer, Float, ...), les classes String et java.util.Date, les tableaux de ces types, les collections (List, Set, Map, ...) et certains objets de manipulation XML issus de DOM, JDOM et DOM4J.

Les convertisseurs Bean et Objet fournis en standard doivent être explicitement utilisés dans le fichier dwr.xml pour des raisons de sécurité.

Exemple :

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
  <allow>
    <create creator="new" javascript="TestDWR">
      <param name="class" value="com.jmd.test.ajax.dwr.TestDWR"/>
    </create>

    <convert converter="bean" match="com.jmd.test.ajax.dwr.Personne"/>
  </allow>
</dwr>
```

Il est possible d'utiliser le caractère joker *

Exemple :

```
<convert converter="bean" match="com.jmd.test.ajax.dwr.*"/>
<convert converter="bean" match="*"/>
```

Le convertisseur Bean permet de convertir un Bean en un tableau associatif JavaScript et vice versa en utilisant les mécanismes d'introspection.

Exemple :

```
public class Personne {
  public void setNom(String nom) { ... }
  public void setTaille(int taille) { ... }
  // ...
}
```

L'appel d'une méthode acceptant la classe Personne en paramètre peut se faire de la manière suivante dans la partie cliente :

Exemple :

```
var personne = { nom:"Test", taille:33 };
TestDWR.setPersonne(personne);
```

Il est possible de restreindre l'accès à certaines propriétés d'un bean dans son convertisseur.

Exemple :

```
<convert converter="bean" match="com.jmd.test.ajax.dwr.Personne"/>
  <param name="exclude" value="dateNaissance, taille"/>
</convert>
```

Exemple :

```
<convert converter="bean" match="com.jmd.test.ajax.dwr.Personne"/>
  <param name="include" value="nom, prenom"/>
</convert>
```

L'utilisation de ce dernier exemple est recommandée.

Le convertisseur Objet est similaire mais il utilise directement les membres plutôt que de passer par les getter/setter.

Il possède un paramètre force qui permet d'autoriser l'accès aux membres privés de l'objet par introspection.

Exemple :

```
<convert converter="object" match="com.jmd.test.ajax.dwr.Personne"/>
  <param name="force" value="true"/>
</convert>
```

74.4.1.3. Les scripts engine.js et util.js

Pour utiliser ces deux bibliothèques, il est nécessaire de les déclarer dans chaque page utilisant DWR.

Exemple :

```
<script type='text/javascript' src='/[WEB-APP]/dwr/engine.js'></script>
<script type='text/javascript' src='/[WEB-APP]/dwr/util.js'></script>
```

Le fichier engine.js est la partie principale côté JavaScript puisqu'il assure toute la gestion de la communication avec le serveur.

Certaines options de paramétrage peuvent être configurées en utilisant la fonction DWREngine.setX().

Il est possible de regrouper plusieurs communications en une seule en utilisant les fonctions DWREngine.beginBatch() et DWREngine.endBatch(). Lors de l'appel de cette dernière, les appels sont réalisés vers le serveur. Ce regroupement permet de réduire le nombre d'objets XMLHttpRequest créés et le nombre de requêtes envoyées au serveur.

Le fichier util.js propose des fonctions utilitaires pour faciliter la mise à jour dynamique de la page. Ces fonctions ne sont pas dépendantes d'autres éléments de DWR.

Fonction	Rôle
$(id)	Encapsuler un appel à la fonction document.getElementById() comme dans la bibliothèque Prototype
addOptions	Ajouter des éléments dans une liste ou un tag ou
removeAllOptions	Supprimer tous les éléments d'une liste ou d'un tag ou
addRows	Ajouter des lignes dans un tableau
removeAllRows	Supprimer toutes les lignes dans un tableau
getText	Renvoyer la valeur sélectionnée dans une liste
getValue	Renvoyer la valeur d'un élément HTML
getValues	Obtenir les valeurs de plusieurs éléments fournis sous la forme d'un ensemble de paires clé:valeur_vide dont la clé est l'id de l'élément à traiter
onReturn	Gérer l'appui sur la touche return avec un support multi-navigateur
selectRange(id, debut,fin)	Gérer une sélection dans une zone de texte avec un support multi-navigateur
setValue(id,value)	Mettre à jour la valeur d'un élément
setValues	Mettre à jour les valeurs de plusieurs éléments fournis sous la forme d'un ensemble de paires clé:valeur dont la clé est l'id de l'élément à modifier
toDescriptiveString(id, level)	Afficher des informations sur un objet avec un niveau de détail (0, 1ou 2)
useLoadingMessage	Mettre en place un message de chargement lors des échanges avec le serveur

74.4.1.4. Les scripts client générés

DWR assure un mapping entre les méthodes des objets Java et les fonctions JavaScript générées. Chaque objet Java est mappé sur un objet JavaScript dont le nom correspond à la valeur de l'attribut javascript du creator correspondant dans le fichier de configuration de DWR.

Le nom des méthodes est conservé comme nom de fonction dans le code JavaScript. Le premier paramètre de toutes les fonctions générées par DWR est la fonction de type callback qui sera exécutée à la réception de la réponse. Les éventuels autres paramètres correspondent à leurs équivalents dans le code Java.

DWR s'occupe de transformer un objet Java en paramètre ou en résultat en un équivalent dans le code JavaScript. Par exemple, une collection Java est transformée en un tableau d'objets JavaScript de façon transparente, l'utilisation des objets Java est donc nettement facilitée.

L'utilisation de la bibliothèque util.js peut être particulièrement pratique pour faciliter l'exploitation des données retournées et utilisées par les fonctions générées.

Des exemples d'utilisation sont fournis dans les sections d'exemples suivantes.

74.4.1.5. Un exemple pour obtenir le contenu d'une page

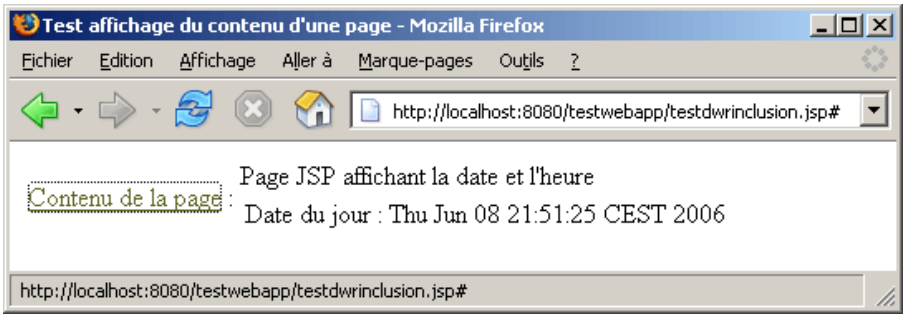
Il est possible qu'une méthode d'un bean renvoie le contenu d'une JSP en utilisant l'objet uk.ltd.getahead.dwr.ExecutionContext. Cet objet permet d'obtenir le contenu d'une url donnée.

Exemple : la JSP dont le contenu sera retourné

```
Page JSP affichant la date et l'heure
<table>
  <tr>
    <td>Date du jour :</td>
    <td nowrap><%=new java.util.Date()%></td>
  </tr>
</table>
```

Exemple :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Test affichage du contenu d'une page</title>
  <script type='text/javascript'
    src='/testwebapp/dwr/interface/TestDWR.js'></script>
  <script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
  <script type='text/javascript' src='/testwebapp/dwr/util.js'></script>

  <script type='text/javascript'>
  <!--

  function inclusion() {
    TestDWR.getContenuPage(afficherInclusion);
  }

  function afficherInclusion(data) {
    DWRUtil.setValue("inclusion", data);
  }

  function init() {
    DWRUtil.useLoadingMessage();
  }

  -->
  </script>
</head>
<body onload="init();">

<table>
  <tr>
    <td><a href="#" onclick="inclusion()">Contenu de la page</a> :</td>
    <td nowrap>
    <div id="inclusion"></div>
    </td>
  </tr>
</table>

</body>
</html>
```



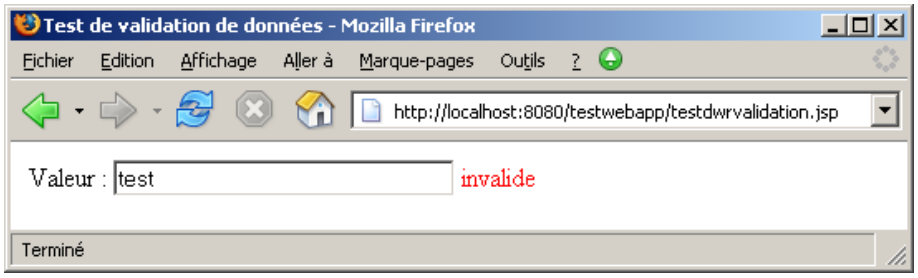
Lors d'un clic sur le lien, le contenu de la JSP est affiché dans le calque.

74.4.1.6. Un exemple pour valider des données

Dans cet exemple, à chaque saisie dans la zone de texte, le contenu est validé à la volée par un appel à une méthode d'un bean.

Exemple :

```
01. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02.     pageEncoding="ISO-8859-1"%>
03. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
04. <html>
05. <head>
06.   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
07.   <title>Test de validation de données</title>
08.   <script type='text/javascript' src='/testwebapp/dwr/interface/TestDWR.js'></script>
09.   <script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
10.   <script type='text/javascript' src='/testwebapp/dwr/util.js'></script>
11.
12.   <script type='text/javascript'>
13.   <!--
14.   function valider() {
15.     TestDWR.validerValeur(afficherValidation, $("donnees").value);
16.   }
17.
18.   function afficherValidation(data) {
19.     DWRUtil.setValue("validationMessage",data);
20.     if (data == "valide") {
21.       $("validationMessage").style.color='#00FF00';
22.     } else {
23.       $("validationMessage").style.color='#FF0000';
24.     }
25.   }
26.
27.   function init() {
28.     DWRUtil.useLoadingMessage();
29.   }
30.   -->
31.   </script>
32. </head>
33. <body onload="init();">
34.
35. <table>
36.     <tr>
37.         <td>Valeur :</td>
38.         <td nowrap><input type="text" id="donnees" name="donnees" size="30"
39.             onkeyup="valider();"></td>
40.         <td>
41.         <div id="validationMessage"></div>
42.         </td>
43.     </tr>
44. </table>
45.
46. </body>
47. </html>
```

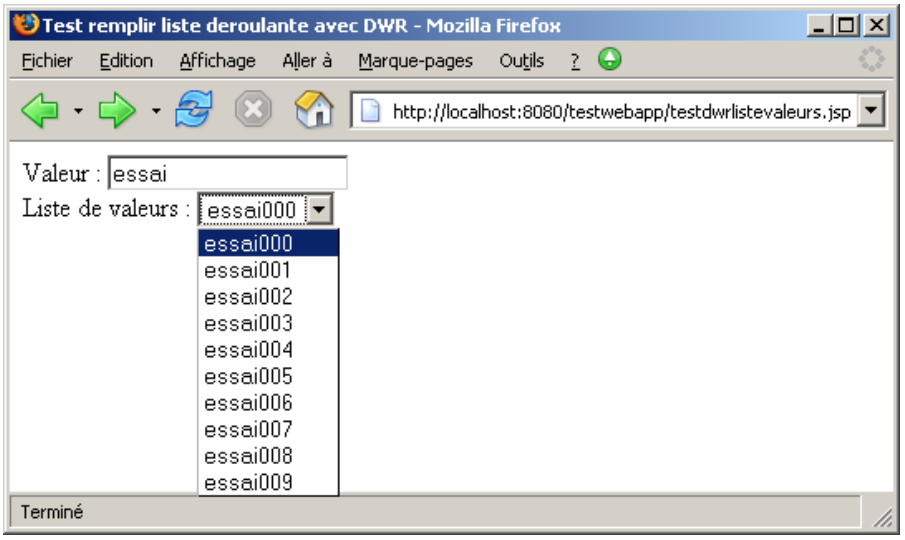


Exemple :

```
01. public String validerValeur(String valeur) {
02.
03.   String resultat = "invalide";
04.
05.   if ((valeur != null) && valeur.startsWith("X")) {
06.     resultat = "valide";
07.   }
08.
09.   return resultat;
10. }
```

74.4.1.7. Un exemple pour remplir dynamiquement une liste déroulante

Cet exemple va remplir dynamiquement le contenu d'une liste déroulante en fonction de la valeur d'une zone de saisie.



Côté serveur la méthode getListeValeurs() du bean est appelée pour obtenir les valeurs de la liste déroulante. Elle attend en paramètre une chaîne de caractères et renvoie un tableau de chaînes de caractères.

Exemple :

```
01. package com.jmd.test.ajax.dwr;
02.
03. public class TestDWR {
04.
05.   public String[] getListeValeurs(String valeur)
06.   {
07.     String[] resultat = new String[10];
08.
09.     for(int i = 0 ; i <10;i++ ) {
10.       resultat[i] = valeur+"00"+i;
11.     }
12.
13.     return resultat;
14.   }
15.
16. }
```

La page de l'application est composée d'une zone de saisie et d'une liste déroulante.

Exemple :

```
01. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02.     pageEncoding="ISO-8859-1"%>
03. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
04. <html>
05. <head>
06. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
07. <title>Test remplir liste deroulante avec DWR</title>
08. <script type='text/javascript' src='/testwebapp/dwr/interface/TestDWR.js'></script>
09. <script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
10. <script type='text/javascript' src='/testwebapp/dwr/util.js'></script>
11.
12. <script type='text/javascript'>
13. <!--
14. function rafraichirListeValeurs() {
15.   TestDWR.getListeValeurs(remplirListeValeurs, $("valeur").value);
```

```
}

function remplirListeValeurs(data) {
  DWRUtil.removeAllOptions("listevaleurs");
  DWRUtil.addOptions("listevaleurs", data);
  DWRUtil._selectListItem($("listevaleurs"),$("listevaleurs").options[0].value);
}

function init() {
  DWRUtil.useLoadingMessage();
  rafraichirListeValeurs();
}
-->
</script>
</head>
<body onload="init();">

<p>Valeur : <input type="text" id="valeur"
    onblur="rafraichirListeValeurs();" /><br />
Liste de valeurs : <select id="listevaleurs" style="vertical-align:top;"></select>
</p>

</body>
</html>
```

La fonction init() se charge d'initialiser le contenu de la liste déroulante au chargement de la page.

La fonction rafraichirListeValeurs() est appelée dès que la zone de saisie perd le focus. Elle utilise la fonction JavaScript TestDWR.getListeValeurs() générée par DWR pour appeler la méthode du même nom du bean. Les deux paramètres fournis à cette fonction permettent d'une part de préciser que c'est la fonction remplirListeValeurs() qui fait office de fonction de callback et d'autre part de fournir la valeur de la zone de saisie en paramètre de l'appel de la méthode getListeValeurs() du bean.

La fonction remplirListeValeurs() se charge de vider la liste déroulante, de remplir son contenu avec les données reçues en réponse du serveur (elles sont passées en paramètre de la fonction) et de sélectionner le premier élément de la liste. Pour ces trois actions, trois fonctions issues de la bibliothèque util.js de DWR sont utilisées.

La fonction addOptions() utilise les données passées en paramètre pour remplir la liste.

74.4.1.8. Un exemple pour afficher dynamiquement des informations

L'exemple de cette section va permettre d'afficher dynamiquement les données d'une personne sélectionnée. L'exemple est volontairement simpliste (la liste déroulante des personnes est en dur et les données de la personne sont calculées plutôt qu'extraites d'une base de données). Le but principal de cet exemple est de montrer la facilité d'utilisation des beans mappés par DWR dans le code JavaScript.

Le bean utilisé encapsule les données d'une personne

Exemple :

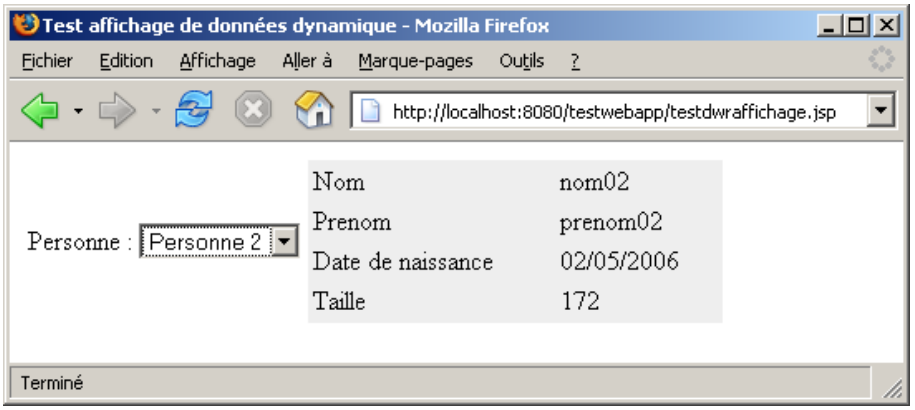
```
package com.jmd.test.ajax.dwr;

import java.util.Date;

public class Personne {
  private String nom;
  private String prenom;
  private String dateNaissance;
  private int taille;

  public Personne() {
    super();
  }

  public Personne(String nom, String prenom, String dateNaissance, int taille) {
    super();
    this.nom = nom;
    this.prenom = prenom;
    this.dateNaissance = dateNaissance;
    this.taille = taille;
  }

  public String getDateNaissance() {
    return dateNaissance;
  }

  public void setDateNaissance(String dateNaissance) {
    this.dateNaissance = dateNaissance;
  }

  public String getNom() {
    return nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
```

```
  }

  public String getPrenom() {
    return prenom;
  }

  public void setPrenom(String prenom) {
    this.prenom = prenom;
  }

  public int getTaille() {
    return taille;
  }

  public void setTaille(int taille) {
    this.taille = taille;
  }

}
```

La page est composée d'une liste déroulante de personnes. Lorsqu'une personne est sélectionnée, les données de cette personne sont demandées au serveur et sont affichées.

Exemple :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Test affichage de données dynamique</title>
<script type='text/javascript' src='/testwebapp/dwr/interface/TestDWR.js'></script>
<script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
<script type='text/javascript' src='/testwebapp/dwr/util.js'></script>

<script type='text/javascript'>
<!--
function rafraichir() {
  TestDWR.getPersonne(afficherPersonne, $("personnes").value);
}

function afficherPersonne(data) {
  DWRUtil.setValue("nomPersonne",data.nom);
  DWRUtil.setValue("prenomPersonne",data.prenom);
  DWRUtil.setValue("datenaissPersonne",data.dateNaissance);
  DWRUtil.setValue("taillePersonne",data.taille);
}

function init() {
  DWRUtil.useLoadingMessage();
}
-->
</script>
</head>
<body onload="init();">

<table>
    <tr>
        <td>Personne :</td>
        <td nowrap><select id="personnes" name="personnes"
            onchange="rafraichir();">
            <option value="1">Personne 1</option>
            <option value="2">Personne 2</option>
            <option value="3">Personne 3</option>
            <option value="4">Personne 4</option>
            </select>
        </td>
        <td>
        <div id="informationPersonne">
        <table bgcolor="#eeeeee" width="250">
        <tr><td>Nom</td><td><span id="nomPersonne"></span></td></tr>
        <tr><td>Prenom</td><td><span id="prenomPersonne"></span></td></tr>
        <tr><td>Date de naissance</td><td><span id="datenaissPersonne"></span></td></tr>
        <tr><td>Taille</td><td><span id="taillePersonne"></span></td></tr>
        </table>
        </div>
        </td>
    </tr>
</table>

</body>
</html>
```



Exemple : le source de la méthode du bean qui recherche les données de la personne

```
1. public Personne getPersonne(String id) {
2.   int valeur = Integer.parseInt(id);
3.   if (valeur < 10) {
4.     id = "0"+id;
5.   }
6.   Personne resultat = new Personne("nom"+id,"prenom"+id,id+"/05/2006",170+valeur);
7.   return resultat;
8. }
```

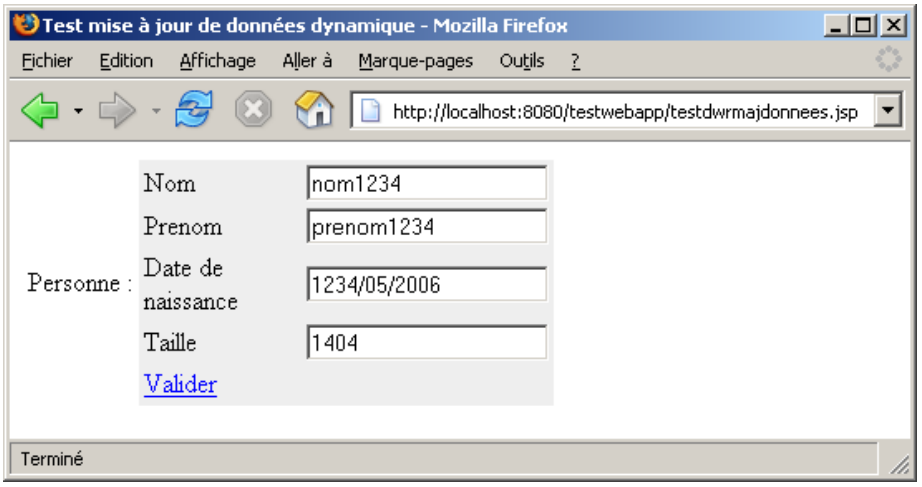
Dans le fichier de configuration dwr.xml, un convertisseur de type bean doit être déclaré pour le bean de type Personne

Exemple :

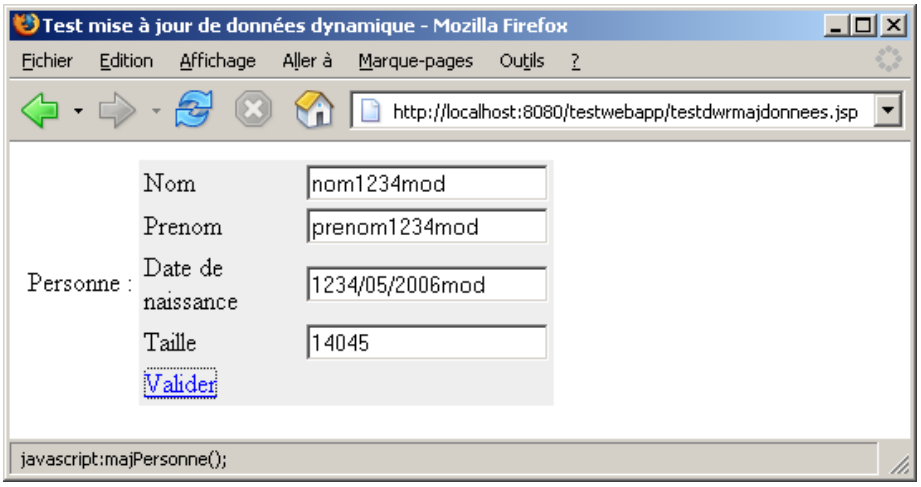
```
1. <allow>
2.   <create creator="new" javascript="TestDWR">
3.     <param name="class" value="com.jmd.test.ajax.dwr.TestDWR"/>
4.   </create>
5.
6.   <convert converter="bean" match="com.jmd.test.ajax.dwr.Personne"/>
7. </allow>
```

74.4.1.9. Un exemple pour mettre à jour des données

Cet exemple va permettre de modifier les données d'une personne.



Il suffit de modifier les données et de cliquer sur le bouton valider



Les données sont envoyées sur le serveur.

Exemple :

```
INFO: Exec[0]: TestDWR.setPersonne()
nom=nom1234mod
prenom=prenom1234mod
datenaiss=1234/05/2006mod
taille14045
```

Exemple : le source de la méthode du bean qui recherche les données de la personne

```
public void setPersonne(Personne personne)
{
  System.out.println("nom="+personne.getNom());
  System.out.println("prenom="+personne.getPrenom());
  System.out.println("datenaiss="+personne.getDateNaissance());
  System.out.println("taille"+personne.getTaille());
    // code pour rendre persistant l'objet fourni en paramètre
}
```

Cette méthode affiche simplement les données reçues. Dans un contexte réel, elle assurerait les traitements pour rendre persistantes leurs modifications.

La page de l'application est la suivante.

Exemple :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Test mise à jour de données dynamique</title>
  <script type='text/javascript' src='/testwebapp/dwr/interface/TestDWR.js'></script>
  <script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
  <script type='text/javascript' src='/testwebapp/dwr/util.js'></script>

  <script type='text/javascript'>
  <!--
  var personne;

  function rafraichir() {
    TestDWR.getPersonne(afficherPersonne, "1234");
  }

  function afficherPersonne(data) {
    personne = data;
    DWRUtil.setValues(data);
  }

  function majPersonne()
  {
    DWRUtil.getValues(personne);
    TestDWR.setPersonne(personne);
  }

  function init() {
    DWRUtil.useLoadingMessage();
    rafraichir();
  }

  -->
```

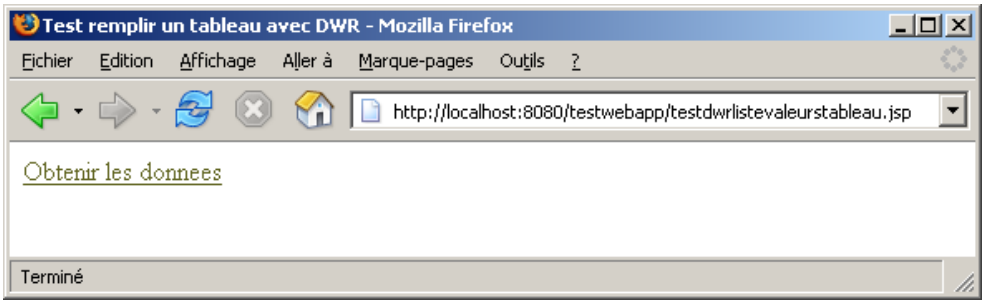
```
  </script>
</head>
<body onload="init();">

<table>
  <tr>
    <td>Personne :</td>
    <td>
    <div id="informationPersonne">
    <table bgcolor="#eeeeee" width="250">
    <tr><td>Nom</td><td><input type="text" id="nom"></td></tr>
    <tr><td>Prenom</td><td><input type="text" id="prenom"></td></tr>
    <tr><td>Date de naissance</td><td><input type="text" id="dateNaissance"></td></tr>
    <tr><td>Taille</td><td><input type="text" id="taille"></td></tr>
    <tr><td colspan="2"><a href="javascript:majPersonne();">Valider</a></td></tr>
    </table>
    </div>
    </td>
    </tr>
</table>

</body>
</html>
```

Cet exemple utilise les fonctions getValues() et setValues() qui mappent automatiquement les propriétés d'un objet avec les objets de l'arbre DOM dont l'id correspond.

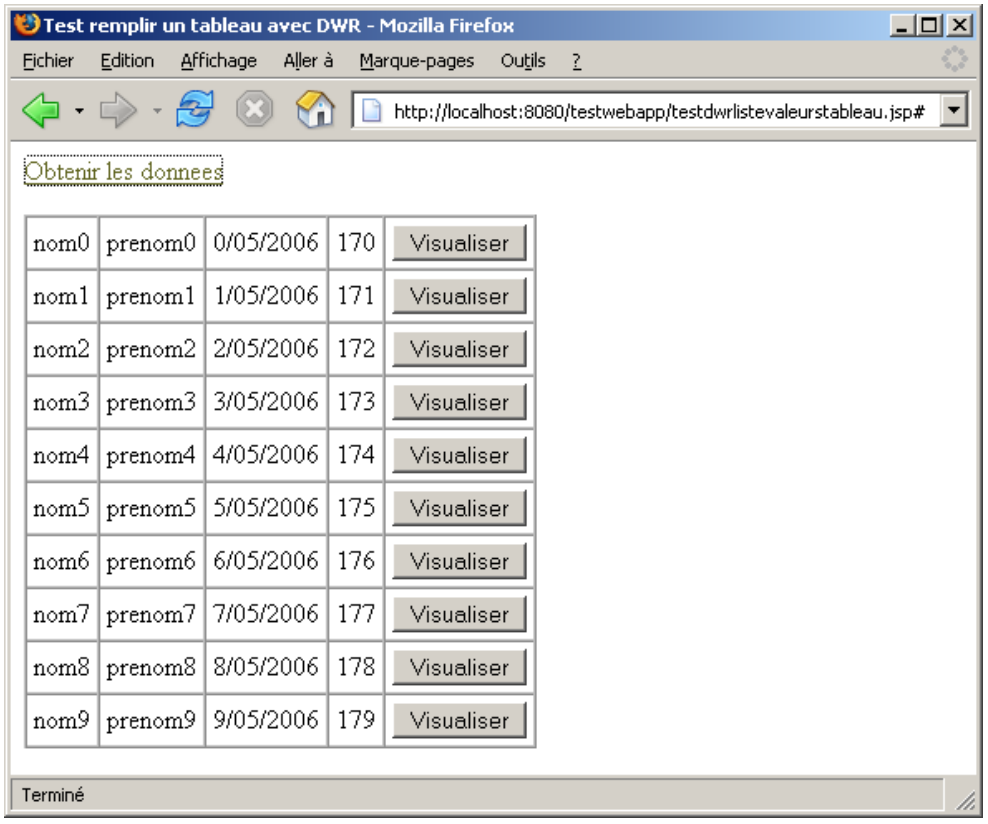
Remarque : il est important que l'objet personne qui encapsule les données de la personne soit correctement initialisé, ce qui est fait au chargement des données de la personne.

74.4.1.10. Un exemple pour remplir dynamiquement un tableau de données

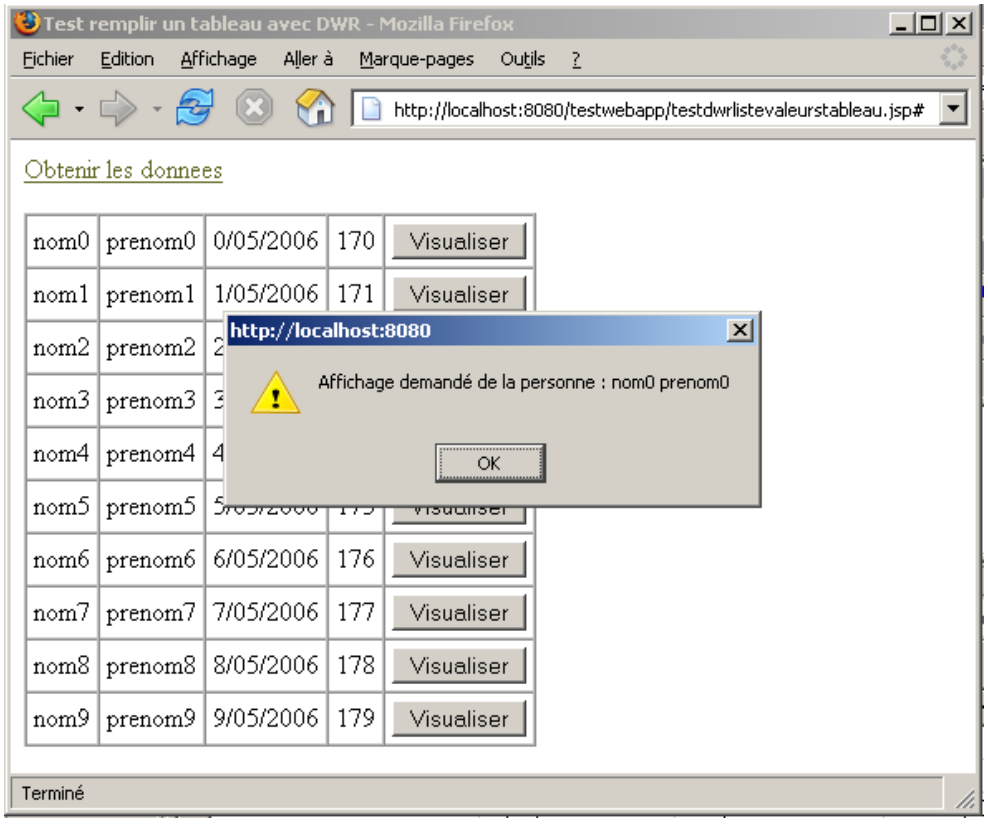
Cet exemple va remplir dynamiquement le contenu d'un tableau avec une collection d'objets.



Un clic sur le lien permet d'afficher le tableau avec les données retournées par le serveur.



Un clic sur le bouton "visualiser" affiche un message avec le nom et la personne concernée.



Côté serveur la méthode getPersonnes() du bean est appelée pour obtenir la liste des personnes sous la forme d'une collection d'objets de type Personne.

Exemple :

```
01. public List getPersonnes() {
02.   List resultat = new ArrayList();
03.   Personne personne = null;
04.
05.   for (int i = 0; i<10 ; i++) {
06.     personne = new Personne("nom"+i,"prenom"+i,i+"/05/2006",170+i);
07.     resultat.add(personne);
08.   }
09.   return resultat;
10. }
```

La page de l'application est composée d'un calque contenant un tableau.

Exemple :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Test remplir un tableau avec DWR</title>
  <script type='text/javascript' src='/testwebapp/dwr/interface/TestDWR.js'></script>
  <script type='text/javascript' src='/testwebapp/dwr/engine.js'></script>
  <script type='text/javascript' src='/testwebapp/dwr/util.js'></script>

  <script type='text/javascript'>
  <!--
  function rafraichirListeValeurs() {
    TestDWR.getPersonnes(remplirListeValeurs);
  }

  function remplirListeValeurs(data) {

    DWRUtil.removeAllRows("tableau");

    if (data.length == 0) {
      alert("");
      $("donnees").style.visibility = "hidden";
    } else {
      DWRUtil.addRows("tableau",data,cellulesFonctions);
      $("donnees").style.visibility = "visible";
    }
  }

  // tableau des fonctions permettant d'assurer le rendu des différentes cellules du tableau
  var cellulesFonctions = [
    function(item) { return item.nom; },
    function(item) { return item.prenom; },
    function(item) { return item.dateNaissance; },
    function(item) { return item.taille; },
    function(item) {
      var btn = document.createElement("button");
      btn.innerHTML = "Visualiser";
      btn.itemId = item.nom+" "+item.prenom;
      btn.onclick = afficherPersonne;
      return btn;
    }
  ];


  function afficherPersonne() {
    alert("Affichage demandé de la personne : "+this.itemId);
  }

  function init() {
    DWRUtil.useLoadingMessage();
  }
  -->
  </script>
</head>
<body onload="init();">

<p><a href="#" onclick="rafraichirListeValeurs()">Obtenir les donnees</a></p>
<div id="donnees">
  <table id="tableau" border="1" cellpadding="4" cellspacing="0"></table>
</div>
</body>
</html>
```

Cet exemple met en oeuvre les fonctions de manipulation de tableaux de la bibliothèque util.js notamment la fonction DWRUtil.addRows("tableau",data,cellulesFonctions) qui permet d'ajouter un ensemble de lignes à un tableau HTML.

Elle attend en paramètre l'id du tableau à modifier, les données à utiliser et un tableau de fonctions qui vont définir le rendu de chaque cellule d'une ligne du tableau. Ces fonctions peuvent simplement retourner la valeur d'une propriété de l'objet courant ou renvoyer des objets plus complexes comme un bouton.

Remarque : pour les boutons générés, il serait préférable d'utiliser des id mieux adaptés comme un suffixe et un identifiant uniqueque de concaténer les noms et prénoms. Ici, le but est de rendre l'exemple le plus possible.

74.4.2. D'autres frameworks

Ils existent de nombreux autres frameworks permettant de mettre en oeuvre AJAX, dont voici une liste non exhaustive :

Framework	Url
AjaxAnywhere	http://ajaxanywhere.sourceforge.net
AjaxTags	http://ajaxtags.sourceforge.net/
ajax4suggest	http://sourceforge.net/projects/ajax4suggest
Echo 2	http://echo.nextapp.com/site/echo2
jWic	http://www.jwic.de