Ajax avec XMLHttpRequest

Antoine JAMIN

Juillet 2016

Langages et notions abordés : PHP, HTML, SQL, Javascript, Ajax et XMLHttpRequest.

Asynchronous **JA**vascript and **X**ML ou Ajax est le nom caractérisant toutes les techniques permettant l'échange de données en tâche de fond par le biais de fonctions javascript permettant la manipulation de données au format XML.

L'avantage de cette méthode c'est que la page web continue de fonctionner pendant que la requête est faite et qu'elle traitera la réponse quand elle la recevra. Ceci apporte un véritable dynamisme à une page web.

Ce tutoriel s'appui sur les travaux suivants :

- `http://www.xul.fr/xml-ajax.html`
- **Sébastien DE LA MARCK et Johann PARDANAUD**, Dynamiser vos sites web avec Javascript, juin 2012.

Dans ce tutoriel nous verrons dans un premier temps quelques fonctions JavaScript essentielles pour utiliser les objets XHR. Puis dans un second temps, un exemple utilisant un objet XHR pour afficher des données contenues dans une base de données à partir d'une sélection.

1 Les Fonctions Javascript pour les objets XMLHttpRequest

Dans cette partie nous allons détailler les grandes fonctions javascript permettant d'utiliser le protocole XMLHttpRequest.

open(méthode, url, AS, login, mdp) : préparation pour l'envoi des requêtes.

- méthode : GET, POST ou HEAD.
- url : Addresse de l'endroit où envoyer la requête.
- AS (facultatif) : booléen qui fixe si la requête est de type asynchrone ou non.
- login (facultatif) : permet de saisir le nom d'utilisateur pour l'identification (.htaccess).
- mdp (facultatif) : permet de saisir le mot de passe du login pour l'identification (.htaccess).

send() : envoi de requête par la méthode définie auparavant à l'aide de la fonction open().

encodeURIComponent(valeur) : encode une valeur au format autorisé dans une URL, il est donc conseillé d'utiliser cette fonction sur les variables que l'on souhaite transmettre.

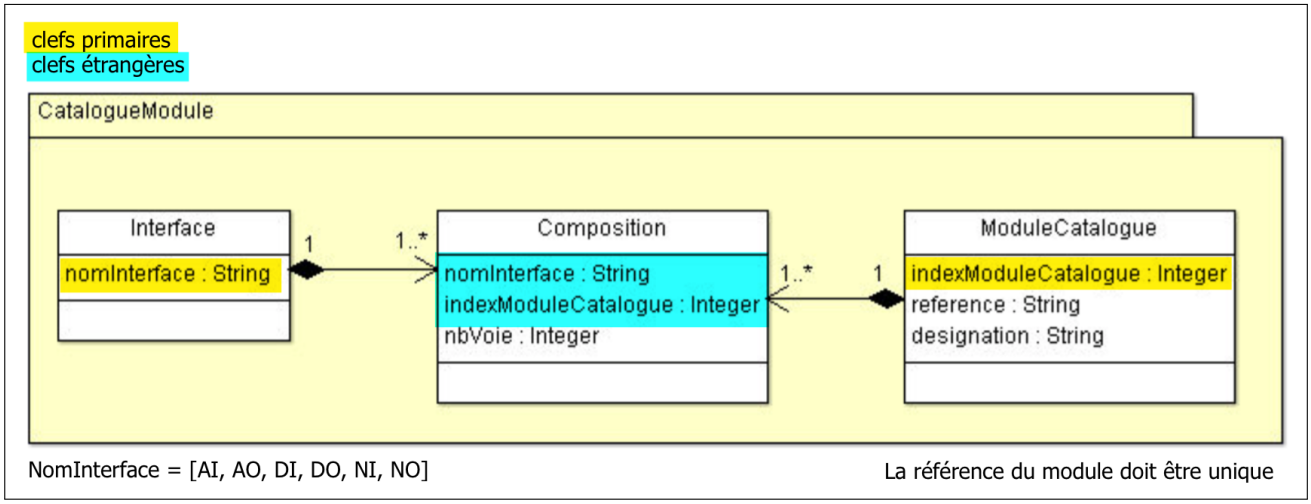
setRequestHeader(header, valeur) : autorise la modification d'un en-tête.
- header : en-tête de la page concernée.
- valeur : valeur à attribuer à l'en-tête.

addEventListener(event, function) : permet d'associer un évènement à une fonction. Uniquement exploitable quand on utilise des requêtes de type asynchrone.

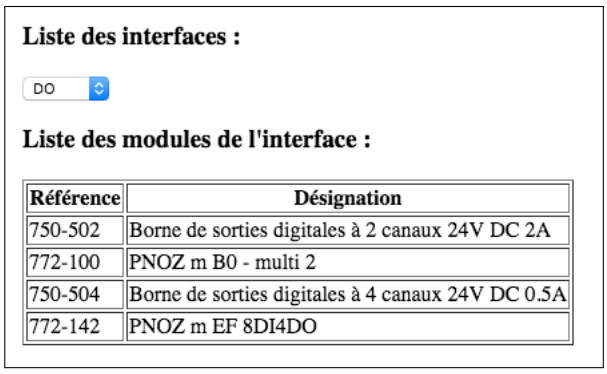
2 Exemple d'utilisation

Cet exemple permet d'afficher tous les modules (carte d'entrées/sorties) appartenant à l'interface (Analog/Digital Input/Output et Autres) sélectionnée par le biais d'un menu déroulant.

L'intégralité de cet exemple se fait avec le diagramme UML suivant :



Le résultat attendu ressemblera à ceci :



2.1 SQL : Base de données

Dans un premier temps il nous faut construire les 3 tables et leurs intéractions.

```
1 -- Creation de la base de donnees
2 CREATE DATABASE IF NOT EXISTS cataloguemodule ;
3
4 -- Creation de la table Composition
5 CREATE TABLE IF NOT EXISTS cataloguemodule.composition (
6   nomInterface varchar(50) NOT NULL,
7   indexModuleCatalogue int(10) unsigned NOT NULL,
8   nbVoie int(11) DEFAULT NULL,
9   PRIMARY KEY (nomInterface,indexModuleCatalogue),
```

```sql
  KEY fkCompositionModuleCatalogue (indexModuleCatalogue)
);

-- Creation de la table InterfaceModule
CREATE TABLE IF NOT EXISTS cataloguemodule.interfacemodule (
  nomInterface varchar(50) NOT NULL DEFAULT '',
  PRIMARY KEY (nomInterface)
);

-- Creation de la table ModuleCatalogue
CREATE TABLE IF NOT EXISTS cataloguemodule.modulecatalogue (
  indexModuleCatalogue int(10) unsigned NOT NULL AUTO_INCREMENT,
  reference varchar(50) NOT NULL,
  designation varchar(100) DEFAULT NULL,
  idTradDesignation int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (indexModuleCatalogue),
  UNIQUE KEY reference (reference),
  UNIQUE KEY idTradDesignation (idTradDesignation)
);


-- Declaration des interactions entre les tables
ALTER TABLE cataloguemodule.composition
  ADD CONSTRAINT fkCompositionInterface FOREIGN KEY (nomInterface)
   REFERENCES cataloguemodule.interfacemodule (nomInterface),
  ADD CONSTRAINT fkCompositionModuleCatalogue FOREIGN KEY (
   indexModuleCatalogue) REFERENCES cataloguemodule.modulecatalogue (
   indexModuleCatalogue);

-- Insertions dans les tables
INSERT INTO cataloguemodule.interfacemodule (nomInterface) VALUES ('AI'), ('
   AO'), ('Autres'), ('DI'), ('DO');
INSERT INTO cataloguemodule.modulecatalogue (indexModuleCatalogue, reference
   , designation, idTradDesignation) VALUES (1, '750-352', 'Coupleur de bus
   de terrain 24V12', 1), (2, '750-430', '8-channels digital input module 24
   VDC', 2), (3, '750-502', 'Borne de sorties digitales à 2 canaux 24V DC 2A
   ', 3), (4, '772-100', 'PNOZ m B0 - multi 2', 4), (17, '750-402', 'Borne d
   ''entrees digitales à 4 canaux 24V DC', 17), (18, '750-434', 'Borne d''
   entrees digitales à 8 canaux 5-12V DC', 18), (19, '750-469', 'Bornes d''
   entrees à 2 canaux pour thermocouple type K', 19), (20, '750-504', 'Borne
    de sorties digitales à 4 canaux 24V DC 0.5A', 20), (21, '750-550', '
   Borne de sorties analogiques à 2 canaux 0-10V', 21), (22, '750-600', '
   Borne finale de bus', 22), (23, '750-602', 'Borne d''alimentation 24V DC'
   , 23), (24, '750-603', 'Borne de distribution 24V DC', 24), (25, '750-333
   ', 'Coupleur de bus de terrain', 25), (26, '750-466', 'Borne d''entrees
   analogiques à 2 canaux 4-20mA', 26), (27, '750-556', 'Borne de sorties
   analogiques à 2 canaux +/- 10V', 27), (28, '772-142', 'PNOZ m EF 8DI4DO',
    28);
INSERT INTO cataloguemodule.composition (nomInterface, indexModuleCatalogue,
    nbVoie) VALUES ('AI', 19, 2), ('AI', 26, 2), ('AO', 21, 2), ('AO', 27,
   2), ('Autres', 1, NULL), ('Autres', 22, NULL), ('Autres', 23, NULL), ('
   Autres', 24, NULL), ('Autres', 25, NULL), ('DI', 2, 8), ('DI', 4, 20), ('
   DI', 17, 4), ('DI', 18, 8), ('DI', 28, 8), ('DO', 3, 2), ('DO', 4, 4), ('
   DO', 20, 4), ('DO', 28, 4);
```

code.sql

2.2 HTML : Page du menu

Le code ci dessous doit être placé entre les deux balises $< body >< /body >$ du code html.
Ce code permet d'afficher la page avec un menu déroulant et une section ($< div >$) contenant le tableau.

```
1  <?php
2    try{
3      $db_catalogmodule=new PDO('mysql:host=localhost;dbname=cataloguemodule;
       charset=utf8','admin','');
4    }catch(Exception $e){
5      die('Erreur : '.$e->getMessage());
6    }
7    $list=$db_catalogmodule->query('SELECT * FROM interfacemodule ORDER BY
       nomInterface');
8    ?>
9
10 <!-- Generation de la liste des interfaces -->
11 <h3>Liste des interfaces :</h3>
12 <select name="interfaces" id="interfaces" onchange="PrintModules()">
13 <?php
14 while($Dinterfaces = $list->fetch()){
15   echo "<option>".$Dinterfaces['nomInterface'];
16   }
17 ?>
18 </select>
19
20
21 <h3>Liste des modules de l'interface :</h3>
22
23 <div id="modules" style="display:inline">
24 </div>
25
26 <script src="index.js" type="text/javascript"></script>
```

index.php

Lignes 1 à 8 : commande php pour se connecter à la base et récupérer toutes les données de la table *interfacemodule* .

Lignes 11 à 18 : permet de générer un menu déroulant contenant tous les noms d'interface contenus dans la table *interfacemodule*. A chaque fois que l'utilisateur modifiera la valeur du menu, la fonction *PrintModules()* s'exécutera.

Ligne 21 à 24 : les balises $< div >< /div >$ permettent d'indiquer l'endroit où devra s'insérer le script généré par la requête XHR.

Ligne 26 : indique que tous les scripts javascript de la page sont dans le fichier index.js.

2.3 Javascript

```
1  function getXhr(){
2    var xhr=null;
3    if(window.XMLHttpRequest) xhr=new XMLHttpRequest();
4    else if(window.ActiveXObject){
5      try{
6        xhr = new ActiveXObject("Msxml2.XMLHTTP");
7      }catch(e){
8        xhr = new ActiveXObject("Microsoft.XMLHTTP");
9      }
```

```
10    }else{
11      alert("Votre navigateur ne supporte pas les objets XMLHttpRequest !!!");
12      xhr=false;
13    }
14    return xhr;
15  }
16
17  function PrintModules(){
18    var xhr=getXhr();
19    xhr.onreadystatechange=function(){
20      if(xhr.readyState==4 && xhr.status==200){
21        selection=xhr.responseText;
22        document.getElementById('modules').innerHTML=selection;
23      }
24    }
25    xhr.open("POST","ajaxindex.php",true);
26    xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
27    sel = document.getElementById('interfaces');
28    idInterface=sel.options[sel.selectedIndex].value;
29    xhr.send("idInterface="+idInterface);
30  }
```

index.js

2.3.1 Fonction getXhr()

Cette fonction permet de créer un objet XMLHttpRequest en prenant en compte la compatibilité du navigateur.

La première condition (ligne 3) permet de tester si le navigateur et compatible directement (mozilla, chrome, ...), la deuxième (lignes 4 à 9) teste si le navigateur accepte les objets de type ActiveX (internet explorer, ...) et si il n'y a aucune compatibilité (lignes 10 à 13) on prévient l'utilisateur.

2.3.2 Fonction PrintModules()

Cette fonction permet de gérer l'envoi de requête par le biais de la méthode POST à la page gérant le code d'affichage du tableau des modules.

Ligne 18 : Création et initialisation d'un objet de type XHR par le biais de la fonction *getXhr()*.

Lignes 19 à 24 : A chaque changement de statuts (propriété onreadystatechange) de l'objet XHR, on appelle une fonction qui écrit dans le $< div\ id = "modules" >$ de *index.php*.
Le paramètre *readyState* d'un objet XHR admet plusieurs états :

- 0 : non initialisé.
- 1 : connexion établie
- 2 : requête reçue
- 3 : réponse en cours
- 4 : terminé

Le paramètre *status* d'un objet XHR admet plusieurs états :

- 200 : OK
- 404 : page non trouvée

Le paramètre *responseText* contient les données de la page *ajaxindex.php* à insérer dans le $< div\ id = "modules" >$ de la page *index.php* par le biais du paramètre *innerHTML*.

Lignes 25 et 26 : prépare l'envoi de requêtes avec la méthode POST par le biais de la page *ajaxindex.php* en mode asynchrone et autorise la modification de l'en-tête.

Ligne 27 : récupère l'objet *interfaces* de la page *index.php*.

Ligne 28 : récupère la valeur de l'objet sélectionné dans la liste *interfaces* de la page *index.php*.

Ligne 29 : envoi d'une requête de type POST à la page *ajaxindex.php*, la valeur de l'objet sélectionné dans la liste *interfaces* de la page *index.php*.

2.4 PHP/HTML : Données à afficher

L'objectif de ce code est d'afficher un tableau avec toutes les données appartenant à l'interface sélectionnée.

```
1  <?php
2    //En tete du tableau
3    echo "<table border='1'><thead><tr><th>Reference</th><th>Designation</th
      ></tr></thead><tbody>";
4    try{
5      $db_catalogmodule=new PDO('mysql:host=localhost;dbname=cataloguemodule;
     charset=utf8','admin','');
6      $LComposition=$db_catalogmodule->query("SELECT * FROM composition WHERE
     nomInterface='".$_POST['idInterface']."'");
7    }catch(Exception $e){
8      die('Erreur : '.$e->getMessage());
9    }
10   while ($DComposition=$LComposition->fetch()) {
11     try{
12       $LModule=$db_catalogmodule->query("SELECT * FROM modulecatalogue WHERE
     indexModuleCatalogue='".$DComposition['indexModuleCatalogue']."'");
13     }catch(Exception $e){
14       die('Erreur : '.$e->getMessage());
15     }
16     $DModule=$LModule->fetch();
17     //generation de toutes les lignes du tableau de chaque module present
     dans chaque composition avec l'interface selectionnee
18     echo "<tr><td>".$DModule['reference']."</td><td>".$DModule['designation'
     ]."</td></td></tr>";
19   }
20 ?>
```

ajaxindex.php

Ligne 3 : création du tableau et de sa première ligne.

Lignes 4 à 9 : Connexion à la base de donnée *cataloguemodule* et sélection de tous les paramètres de la table *composition* qui contiennent le *nomInterface* transmis par la requête POST.

Lignes 10 à 19 : Pour chaque donnée de la table *composition* ayant pour *nomInterface* transmis par la requête POST, on sélectionne le module concerné par cette composition.

On génère ensuite les lignes du tableau module par module.