Initiation à AJAX

AJAX (*Asynchronous Javascript And Xml*) est une méthode permettant d'interroger un serveur http à partir d'un navigateur à l'aide du langage javascript. L'idée de base est d'envoyer une requête au serveur en javascript et d'en récupérer le résultat pour mettre à jour l'affichage de la page.

Remarques :

- Les exemples fournis ont été testés avec Internet Explorer 7, Opera 8 et Firefox 2.
- Le code HTML est volontairement minimaliste…

Merci à Daniel Régnier pour sa précieuse relecture.

1. L'objet XMLHttpRequest

a. Description

Le fonctionnement d'Ajax est basé sur un objet javascript de la classe *XMLHttpRequest*. Cet objet était à l'origine un objet *ActiveX* introduit par Microsoft dans Internet Explorer 5. Les autres navigateur l'ont ensuite implémenté en tant qu'objet javascript, ce que Microsoft a également fait depuis IE7.

Un objet *XMLHttpRequest* permet d'envoyer une requête à un serveur http, de lui transmettre des informations via les méthodes GET ou POST, et de récupérer le résultat qui peut être du XML, du HTML ou du simple texte.

Il convient de remarquer que les navigateurs trop anciens ou n'exécutant pas le code javascript ne donnent pas accès à cette technique.

b. Comment obtenir ce sésame ?

> D'une manière basique, il suffit d'écrire en javascript :

```
requeteHttp = new XMLHttpRequest();
```

> Mais ce serait trop simple…

Ceci ne fonctionnerait pas avec IE5 ou IE6 pour lesquels cet objet est un objet ActiveX.

Pour ces deux navigateurs, il faut :

```
requeteHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

Ou :

```
requeteHttp = new ActiveXObject("Msxml2.XMLHTTP");
```

D'autre part, certains navigateurs basés sur Mozilla (Firefox notamment) exigent que le type de données utilisé par le serveur soit text/xml. On peut forcer ce type à l'aide de la méthode overrideMimeType, qui n'est pas appréciée par tout le monde.

> Au final, il faut écrire :

```
function getRequeteHttp()
{
   var requeteHttp;
   if (window.XMLHttpRequest)
   {  // Mozilla
```

```
      requeteHttp=new XMLHttpRequest();
      if (requeteHttp.overrideMimeType)
      { // problème firefox
        requeteHttp.overrideMimeType('text/xml');
      }
    }
    else
    {
      if (window.ActiveXObject)
      { // C'est Internet explorer < IE7
        try
        {
          requeteHttp=new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e)
        {
          try
          {
            requeteHttp=new ActiveXObject("Microsoft.XMLHTTP");
          }
          catch(e)
          {
            requeteHttp=null;
          }
        }
      }
    }
    return requeteHttp;
}
```

2. Techniques de base

a. Premier exemple

L'exemple 1 permet simplement de mettre en œuvre le mécanisme.

- La première page (page1.html) contient la fonction *getRequeteHttp* vue précédemment et affiche un lien qui provoque l'exécution de la requête vers le serveur.
- La seconde page (page2.html) ne contient en fait que le mot "coucou".
- Lorsque l'utilisateur clique sur le lien de page1.html, la requête est envoyée au serveur par une fonction javascript, celui-ci se contente de retourner le mot "coucou" sur le flux http et la fonction affiche le résultat.

Page1.html

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{
   // voir paragraphe 1
}
function envoyerRequete(url)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url,false);
     requeteHttp.send(null);
     if (requeteHttp.readyState==4)
     { // la requête est achevée, le résultat a été transmis
       if (requeteHttp.status==200)
       { // la requête s'est correctement déroulée
         // (pourrait être 404 pour non trouvé par exemple)
         alert(requeteHttp.responseText);
       }
       else
       {
         alert("La requête ne s'est pas correctement exécutée");
       }
     }
   }
   return;
}
</script>
</head>
<body>
   <a href="javascript:envoyerRequete('page2.html');">requete</a>
</body>
</html>
```

Page2.html

```
Coucou
```

Remarque : cette page ne contient que cela.

Autre exemple

Dans l'exemple 2, on remplace le contenu de page2.html par :

```
<html>
<body>
   <p>Ceci est la deuxi&egrave;me page.</p>
</body>
</html>
```

Et le traitement du résultat de la requête par :

```
function envoyerRequete(url)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url,false);
     requeteHttp.send(null);
     if (requeteHttp.readyState==4)
     { // la requête est achevée, le résultat a été transmis
       if (requeteHttp.status==200)
       { // la requête s'est correctement déroulée
         // (pourrait être 404 pour non trouvé par exemple)
         document.write(requeteHttp.responseText);
       }
       else
       {
         alert("La requête ne s'est pas correctement exécutée");
       }
     }
   }
   return;
}
```

On obtient le remplacement de la page1 par la page2 dans le navigateur. Totalement inutile me direz-vous, un simple lien fait la même chose, c'est vrai…

b. Principe de fonctionnement

Lorsque l'utilisateur clique sur le lien, la fonction *envoyerRequete* est appelée et effectue les traitements suivants :

- Obtention d'un objet *XMLHttpRequest* à l'aide de la fonction *getRequeteHttp*.
- Ouverture de la requête (méthode *open*) en paramétrant :
  - la méthode d'envoi de données au serveur (nous y reviendrons),
  - l'url de la page appelée (celle-ci doit se trouver sur le même domaine),
  - le mode d'appel, synchrone ou asynchrone (nous y reviendrons).
- Envoi de la requête (méthode *send*) en lui passant d'éventuelles informations destinées au serveur (nous y reviendrons).
- Une fois la requête exécutée par le serveur, la fonction teste l'état d'avancement (propriété *status*), 4 indique une requête terminée (0 : non initialisée, 1 : en cours de chargement, 2 : chargée, 3 : interaction en cours).
- La propriété *status* permet de savoir si tout s'est bien déroulé (valeur 200). L'exemple 404 pour une page non trouvée permet de comprendre la signification de cet indicateur.
- Le résultat est ensuite récupéré via la propriété *responseText* (il est également possible d'exploiter la propriété *responseXML* pour obtenir la réponse sous la forme un document XML que l'on peut ensuite analyser à l'aide des routines DOM).

c. Que peut faire le serveur ?

Nous avons vu qu'il peut retourner un simple texte comme "coucou", ou une page html complète. Il peut également retourner un document XML conforme au DOM (Document Object Model) qui peut ensuite être exploité en javascript. Il pourrait également retourner un code javascript qui serait exécuté à l'aide de la fonction *eval* (voir exemple 3 fourni).

L'important est de comprendre que quelle que soit la nature de ce qu'il retourne, il le fait par n'importe quel moyen valide : fichier (ce que nous avons fait), mais aussi PHP, ASP, etc… En fait tout ce qui permet d'envoyer de l'information sur un flux http.

Le script PHP ci-dessous (exemple 4) retourne la date courante sur le serveur et un message de bienvenue adapté à l'heure courante.

```
<?php
   $jma=date('d/m/Y');
   if (date('G')>20)
   {
     echo $jma.' Bonsoir.';
   }
   else
   {
     echo $jma.' Bonjour';
   }
?>
```

Ce script peut être par exemple appelé par la page suivante (page1.html) :

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{
   // comme d'habitude
}
function envoyerRequete(url)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url,false);
     requeteHttp.send(null);
     if (requeteHttp.readyState==4)
     {
       if (requeteHttp.status==200)
       {
         traiterReponse(requeteHttp.responseText);
         // voir page suivante
       }
       else
       {
         alert("La requête ne s'est pas correctement exécutée");
       }
     }
   }
   return;
}
```

```
function traiterReponse(reponse)
{
   alert(reponse);
}
</script>
</head>
<body>
   <a href="javascript:envoyerRequete('voirDate.php');">Voir la date</a>
</body>
</html>
```

La fonction *traiterReponse* ne change rien à l'affaire…

d. Enfin un exemple intéressant (à la page 6 !!!)

Tout ce que nous avons fait jusque là, nous aurions pu le faire sans Ajax… L'exemple 5 va enfin nous montrer l'intérêt de la technique.

La page ci-dessous (page1.php) contient un certain nombre d'informations :

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{ // comme d'habitude
}
function envoyerRequete(url)
{ // comme d'habitude
}
function traiterReponse(reponse)
{
   document.getElementById("heure").innerHTML=reponse;
}
</script>
</head>
<body>
Page appel&eacute;e à : <?php echo date('h:i:s'); ?><br />
Ca va sauter dans 10 minutes, surveillez l'heure...<br />
<img src="bombe.gif"><br />
Il y a plein de choses ici !<br />
Il y a plein de choses ici !<br />
Il y a plein de choses ici !<br />
Il y a plein de choses ici !<br />
<form>
<a href="javascript:envoyerRequete('majHeure.php');">Actualiser l'heure ci-
dessous</a><br />
Heure du serveur :
<span id="heure"><?php echo date('h:i:s'); ?></span><br />
Nom : <input type="text"><br />
Pr&eacute;nom : <input type="text"><br />
</form></body>
</html>
```

Le code PHP mis en gras permet d'afficher l'heure du serveur. Le script PHP ci dessous (majHeure.php) rafraichit cette information.

```
<?php
   header("Cache-Control: no-cache, must-revalidate");
   echo date('h:i:s');
?>
```

La première ligne oblige le navigateur à exécuter le script à chaque appel au lieu d'utiliser l'information présente dans son cache.

Cette fois, Ajax a servi a quelque chose :

- Pour afficher l'heure du serveur, on est bien obligé de la lui demander.
- Il doit donc y avoir une requête à chaque clic sur le lien.
- Classiquement, nous aurions dû recharger toute la page pour rafraichir l'heure.
- Ici, le trafic (et donc le temps de réponse) a été limité au strict nécessaire.

3. Un peu plus loin…

a. Appel asynchrone

Jusqu'ici, nous avons réalisé un appel synchrone au serveur dans la fonction *envoyerRequete* :

```
requeteHttp.open('GET',url,false);
requeteHttp.send(null);
if (requeteHttp.readyState==4)
{
   ...
}
```

Dans le code ci-dessus, la fonction javascript envoie la requête à l'aide de la méthode *send*, puis attend la réponse du serveur. Que se passe-t-il si la réponse est longue à venir ou pire, n'arrive jamais ? Le navigateur est figé…

Une autre solution est d'envoyer la requête en indiquant le nom de la fonction à appeler lorsque le résultat sera arrivé (fonction de callback), et donc de ne pas attendre le résultat pour continuer.

Le code de la fonction envoyerRequete devient (exemple 6) :

```
function envoyerRequete(url)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url,true);
     requeteHttp.onreadystatechange=recevoirReponse;
     requeteHttp.send(null);
   }
   return;
}
function recevoirReponse()
{
   alert('Mais elle est où cette réponse ?');
}
```

La première ligne en gras montre que le troisième argument de la méthode *open* a changé. Il indique maintenant l'utilisation d'un appel asynchrone.

Que se passe-t-il lorsque l'on exécute cet exemple ?

- On ouvre l'objet *requeHttp* en précisant un appel asynchrone.
- On indique le nom de la fonction de rappel (callback).
- La requête est envoyée, l'exécution de la fonction *envoyerRequete* se termine, le navigateur n'est pas figé.
- Lorsque l'événement *readystatechange* se produit (c'est à dire à chaque fois que la valeur de la propriété *status* de l'objet *requeteHttp* change), la fonction *recevoirReponse* est appelée par le navigateur.
- On obtient donc 4 fois le message, l'état passant successivement par les valeurs 0, 1, 2, 3 et 4 (voir "principe de fonctionnement" page 4).

Deux choses sont à remarquer :

- La réponse n'est réellement arrivée que lorsque l'état vaut 4.
- Il est impossible d'accèder à cette réponse puisque l'objet *requeteHttp* est local à la fonction *envoyerRequete*.

Il faut donc passer l'objet *requeteHttp* en paramètre à la fonction *recevoirReponse*, ce qui est impossible avec cette syntaxe. Il faut utiliser une "fonction anonyme", c'est-à-dire une fonction définie à la volée à l'aide de la syntaxe suivante (une autre solution serait d'utiliser une variable globale) :

```
requeteHttp.onreadystatechange=function() {//code de la fonction};
```

L'exemple 7 utilise cette technique pour traiter correctement cette fois l'appel asynchrone :

```
function envoyerRequete(url)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url,true);
     requeteHttp.onreadystatechange=
                              function(){recevoirReponse(requeteHttp);};
     requeteHttp.send(null);
   }
   return;
}
function recevoirReponse(requeteHttp)
{
   if (requeteHttp.readyState==4)
   {
     if (requeteHttp.status==200)
     {
       traiterReponse(requeteHttp.responseText);
     }
     else
     {
       alert("La requête ne s'est pas correctement exécutée");
     }
   }
}
function traiterReponse(reponse)
{
   document.getElementById("heure").innerHTML=reponse;
}
```

b. Transmettre de l'information (méthode GET)

Envoyer une requête à un serveur http en javascript, c'est bien.
Le faire en mode asynchrone, c'est mieux…
Mais lui transmettre en plus de l'information, c'est le top !

L'exemple 8 montre cette possibilité. Il utilise la base de données mysql nommée "ajax" :

```
Categorie (ca_id, ca_libelle)
   Clé primaire (ca_id)

Produit (pr_id, pr_libelle, pr_prix, pr_categorie)
   Clé primaire (pr_id)
   Clé étrangère (pr_categorie) en référence à categorie(ca_id)
```

Cette base peut être constituée à l'aide du script "creerAjax.sql", après création de la base de données via phpMyAdmin par exemple.

La page principale (page1.php) affiche la liste des catégories de produits. Lorsque l'utilisateur sélectionne une catégorie, une requête retournant le nombre de produits de cette catégorie est envoyée au serveur. Le résultat est affiché sur la page.

page1.php

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{ // idem
}
function envoyerRequete(url,idCateg)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url + '?categ=' + escape(idCateg),true);
     requeteHttp.onreadystatechange=
                              function() {recevoirReponse(requeteHttp);};
     requeteHttp.send(null);
   }
   return;
}
function recevoirReponse(requeteHttp)
{ // idem
}
function traiterReponse(reponse)
{
   document.getElementById("nbPdt").innerHTML=reponse;
}
</script>
</head>
```

La première ligne en gras montre le passage d'information à l'aide de la méthode GET. Il serait possible de passer plusieurs valeurs : `monScript.php?nom=dupont&prenom=paul`. L'utilisation de la fonction *escape* permet de s'affranchir des problèmes liés aux caractères spéciaux.

Le corps du document est présenté page suivante.

```
<body>
    <form name="categorie">
       <select name="categ" size="4"
         onchange="javascript:envoyerRequete('getNbProduits.php',this.value)">
         <?php
           $cnx=mysql_connect('localhost','root','');
           mysql_select_db('ajax',$cnx);
           $req=mysql_query('select * from categorie');
           $ligne=mysql_fetch_assoc($req);
           while($ligne)
           {
             echo '<option value='.$ligne['ca_id'].'>'.$ligne['ca_libelle'];
             $ligne=mysql_fetch_assoc($req);
           }
           mysql_close($cnx);
         ?>
       </select>
    </form>
    <br />
    Nombre de produits : <span id="nbPdt">0</span>
    </body>
    </html>
```

La construction de la liste est classique. Sur l'événement *change*, la valeur sélectionnée est transmise à la fonction *envoyerRequete*.

getNbProduits.php

```
    <?php
       header("Cache-Control: no-cache, must-revalidate");
       $cnx=mysql_connect('localhost','root','');
       mysql_select_db('ajax',$cnx);
       $req=mysql_query('select count(*) as nb from produit
                                    where pr_categorie='.$_GET['categ']);
       $res=mysql_fetch_assoc($req);
       echo $res['nb'];
       mysql_close($cnx);
    ?>
```

La première ligne n'est plus nécessaire si la base de données n'est pas modifiée par ailleurs. En effet, le code de catégorie change d'un appel à l'autre et ce changement force la réexécution de la requête HTTP. C'est d'ailleurs une technique possible pour forcer le navigateur à ne pas utiliser son cache : transmettre systématiquement un paramètre généré aléatoirement à la requête HTTP.

On pourrait également écrire l'appel de la fonction *envoyerRequete* en lui passant directement le paramètre : *onchange="javascript:envoyerRequete('getNbProduits.php?categ='+escape(this.value))"*.

c. Transmettre de l'information (méthode POST)

Pour utiliser la méthode POST, il est nécessaire de paramétrer l'entête utilisé par la requête :

```
    requeteHttp.open('POST',url,true);
    requeteHttp.setRequestHeader('Content-Type',
                                    'application/x-www-form-urlencoded');
```

Et de transmettre la ou les valeur(s) lors de l'envoi :

```
    requeteHttp.send('categ=' + escape(idCateg));
```

Il est possible de transmettre plusieurs valeurs avec la syntaxe habituelle :

```
    requeteHttp.send('nom=dupont&prenom=paul');
```

L'exemple 9 présente une transmission d'information à l'aide de la méthode POST et montre la possibilité de créer des pages très réactives.

L'utilisateur saisit le libellé d'un produit. La page est mise à jour à chaque caractère entré pour afficher les produits dont le début du libellé correspond à la saisie.

page1.php

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{ // idem
function envoyerRequete(url, debut)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('POST',url,true);
     requeteHttp.onreadystatechange=
                            function() {recevoirReponse(requeteHttp);};
     requeteHttp.setRequestHeader('Content-Type',
                            'application/x-www-form-urlencoded');
     requeteHttp.send('debutLib=' + escape(debut));
   }
   return;
}
function recevoirReponse(requeteHttp)
{ // idem
}
function traiterReponse(reponse)
{
   var i,nb,selectPdt;
   var produits=reponse.split('/');
   nb=produits.length;
   selectPdt=document.getElementById("listePdt");
   selectPdt.length=nb;
   for (i=0; i<nb; i++)
   {
     selectPdt.options[i].text=produits[i];
   }
}
</script>
</head>
<body>
Libell&eacute; :
<input type="text"
        onkeyup="javascript:envoyerRequete('getProduits.php',this.value)">
<br />
Liste des produits : <br />
<select id="listePdt" size="4">
</select>
</body>
</html>
```

La fonction *envoyerRequete* transmet le début du libellé à l'aide de la méthode POST. La fonction *traiterReponse* reconstruit dynamiquement la liste des produits après avoir "découpé" la réponse (liste de libellés séparés par le caractère '/'). La mise à jour a lieu à chaque entrée de caractères.

getProduits.php

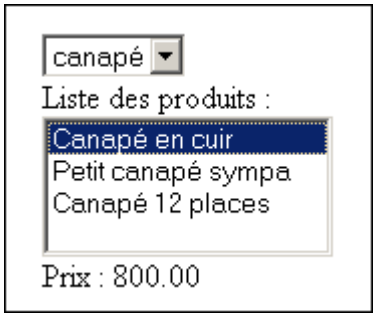
```
<?php
   header("Cache-Control: no-cache, must-revalidate");
   header('Content-Type: text/plain; charset=ISO-8859-1');
   $cnx=mysql_connect('localhost','root','');
   mysql_select_db('ajax',$cnx);
   $req=mysql_query("select * from produit
                     where pr_libelle like '".$_POST['debutLib']."%'");
   $pdt=mysql_fetch_assoc($req);
   if ($pdt)
   {
     $resultat=$pdt['pr_libelle'];
     $pdt=mysql_fetch_assoc($req);
   }
   else
   {
     $resultat='';
   }
   while($pdt)
   {
     $resultat=$resultat.'/'.$pdt['pr_libelle'];
     $pdt=mysql_fetch_assoc($req);
   }
   mysql_close($cnx);
   echo $resultat;
?>
```

La première ligne en gras permet de spécifier l'encodage des caractères.
La seconde ligne en gras montre la construction de la réponse.

4. Un exemple complet

a. Résultat à obtenir

L'idée est de créer un écran de recherche d'un produit :



b. Technique classique

La solution classique consiste à :

- Charger une première fois la liste des catégories, la liste des produits de la première catégorie et le prix du premier produit.
- Recharger entièrement la page lorsque l'utilisateur sélectionne une catégorie,
- Recharger entièrement la page lorsque l'utilisateur sélectionne un produit.

Il est évident que cette solution entraine un trafic (donc une lenteur) inutile…

c. Un solution un peu lourde

Il est possible de faire mieux en constituant en php dès le premier chargement :

- Un tableau javascript contenant toutes les informations concernant les catégories et les produits.
- Un ensemble de fonctions javascript destinées à mettre à jour l'affichage.

Mais cette solution est un peu lourde et conduit à créer des pages HTML assez volumineuses.

d. Merci Ajax !

Avec Ajax, la solution est simple… Seules les informations nécessaires sont rechargées, la page est plus réactive, le trafic limité et la charge du SGBD optimisée. L'exemple 10 implémente cette solution.

page1.php

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{ // idem
}

function envoyerRequeteListeProduits(idCateg)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('POST','getProduits.php',true);
     requeteHttp.onreadystatechange=
                        function() {recevoirListeProduits(requeteHttp);};
     requeteHttp.setRequestHeader('Content-Type',
                                    'application/x-www-form-urlencoded');
     requeteHttp.send('categ=' + escape(idCateg));
   }
   return;
}
```

```
function recevoirListeProduits(requeteHttp)
{
   if (requeteHttp.readyState==4)
   {
     if (requeteHttp.status==200)
     {
       var selectPdt=document.getElementById("listePdt");
       if(requeteHttp.responseText=='')
       {
         selectPdt.length=0;
         document.getElementById("prix").innerHTML='';
       }
       else
       {
         var produits,i,nb,pdt;
         produits=requeteHttp.responseText.split('/');
         nb=produits.length;
         selectPdt.length=nb;
         for (i=0; i<nb; i++)
         {
           pdt=produits[i].split('*');
           selectPdt.options[i].value=pdt[0];
           selectPdt.options[i].text=pdt[1];
         }
         selectPdt.options[0].selected='selected';
         envoyerRequeteProduit(selectPdt.options[0].value);
       }
     }
     else
     {
       alert("La requête ne s'est pas correctement exécutée");
     }
   }
}
```

// La ligne en gras est destinée à mettre à jour l'affichage du prix du produit sélectionné.

```
function envoyerRequeteProduit(idPdt)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('POST','getInfoProduit.php',true);
     requeteHttp.onreadystatechange=
                        function() {recevoirInfoProduit(requeteHttp);};
     requeteHttp.setRequestHeader('Content-Type',
                                     'application/x-www-form-urlencoded');
     requeteHttp.send('pdt=' + escape(idPdt));
   }
}
```

```
function recevoirInfoProduit(requeteHttp)
{
   if (requeteHttp.readyState==4)
   {
     if (requeteHttp.status==200)
     {
       document.getElementById("prix").innerHTML=requeteHttp.responseText;
     }
     else
     {
       alert("La requête ne s'est pas correctement exécutée");
     }
   }
}
</script>
</head>
<body>
<?php
   echo '<select name="idCateg"
          onchange="javascript:envoyerRequeteListeProduits(this.value)">';
   $cnx=mysql_connect('localhost','root','');
   mysql_select_db('ajax',$cnx);
   $req=mysql_query('select * from categorie');
   $ligne=mysql_fetch_assoc($req);
   if ($ligne)
   {
     $categ1=$ligne['ca_id'];
   }
   else
   {
     $categ1=null;
   }
   while($ligne)
   {
     echo '<option value='.$ligne['ca_id'].'>'.$ligne['ca_libelle'];
     $ligne=mysql_fetch_assoc($req);
   }
   echo '</select>';
   echo '<br />';
```

```
    echo 'Liste des produits : <br />';
       echo '<select id="listePdt" size="4"
             onchange="javascript:envoyerRequeteProduit(this.value)">';
       if ($categ1!=null)
       {
         $req=mysql_query('select * from produit
                           where pr_categorie='.$categ1);
         $ligne=mysql_fetch_assoc($req);
         if ($ligne)
         {
           echo '<option selected
                         value='.$ligne['pr_id'].'>'.$ligne['pr_libelle'];
           $prix1=$ligne['pr_prix'];
           $ligne=mysql_fetch_assoc($req);
         }
         else
         {
           $prix1='';
         }
         while($ligne)
         {
           echo '<option value='.$ligne['pr_id'].'>'.$ligne['pr_libelle'];
           $ligne=mysql_fetch_assoc($req);
         }
       }
       echo '</select>';
       echo '<br />';
       echo 'Prix : <span id="prix">'.$prix1.'</span>';
       mysql_close($cnx);
    ?>
    </body>
    </html>
```

getProduits.php

```
    <?php
       header("Cache-Control: no-cache, must-revalidate");
       header('Content-Type: text/plain; charset=ISO-8859-1');
       $cnx=mysql_connect('localhost','root','');
       mysql_select_db('ajax',$cnx);
       $req=mysql_query('select * from produit
                         where pr_categorie='.$_POST['categ']);
       $pdt=mysql_fetch_assoc($req);
       if ($pdt)
       {
         $resultat=$pdt['pr_id'].'*'.$pdt['pr_libelle'];
         $pdt=mysql_fetch_assoc($req);
       }
       else
       {
         $resultat='';
       }
       while($pdt)
       {
         $resultat=$resultat.'/'.$pdt['pr_id'].'*'.$pdt['pr_libelle'];
         $pdt=mysql_fetch_assoc($req);
       }
       mysql_close($cnx);
       echo $resultat;
    ?>
```

getInfoProduit.php

```
<?php
   header("Cache-Control: no-cache, must-revalidate");
   header('Content-Type: text/plain; charset=ISO-8859-1');
   $cnx=mysql_connect('localhost','root','');
   mysql_select_db('ajax',$cnx);
   $req=mysql_query('select pr_prix from produit
                        where pr_id='.$_POST['pdt']);
   $pdt=mysql_fetch_assoc($req);
   mysql_close($cnx);
   echo $pdt['pr_prix'];
?>
```

5. Et le X dans tout cela ?

a. Introduction

J'y viens…

Le X d'ajax signifie XML. Document Object Model (DOM) est une norme permettant de manipuler des documents XML. Il est possible de récupérer le résultat d'une requête ajax sous la forme d'un document XML et de l'exploiter à l'aide de DOM.

b. Structure d'un document XML selon DOM

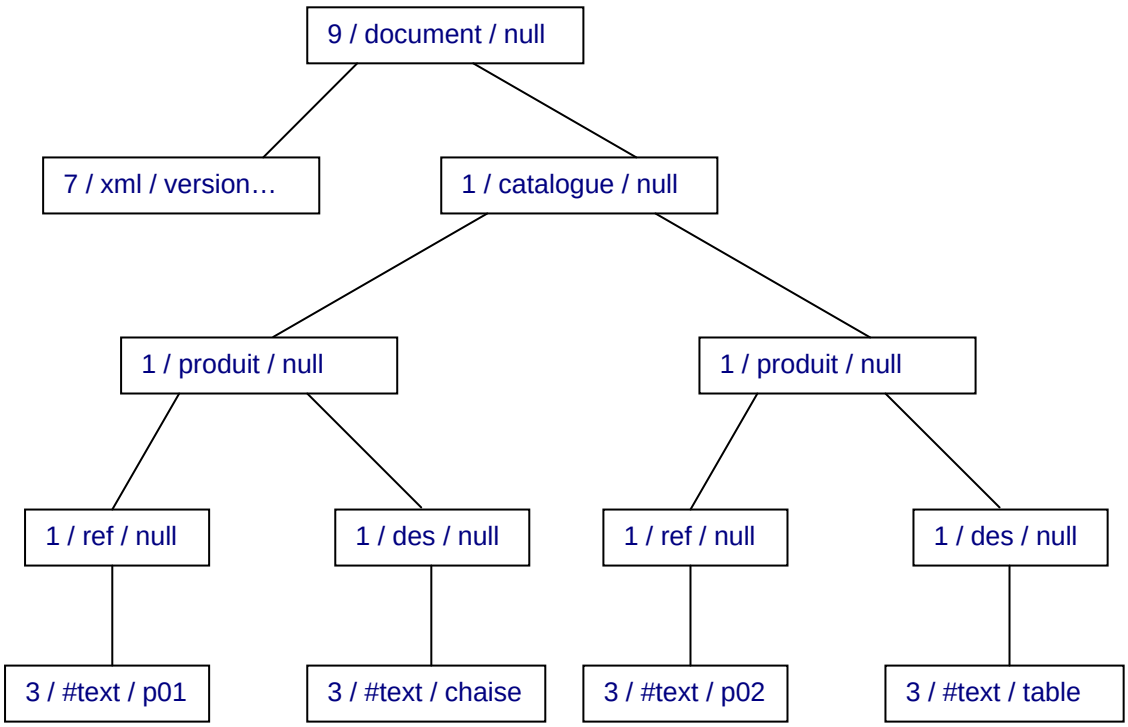
Considérons le fichier XML *donnees.xml* (répertoire *Exemple11*) ci-dessous :

donnees.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<catalogue><produit><ref>p01</ref><des>chaise</des></produit><produit><ref>
p02</ref><des>table</des></produit></catalogue>
```

Représentation DOM

L'arbre ci-dessous représente ce document en indiquant pour chaque nœud son type, son nom et sa valeur (*nodeType*, *nodeName* et *nodeValue*).



6. Présentation de DOM

a. Introduction

Les principaux objets (ce sont en réalité des interfaces) exploitables à l'aide de DOM sont : Node, NodeList, Attr, Element, Document et NamedNodeMap. Seuls les éléments les plus importants à mes yeux sont présentés ici.
Pour une description plus complète, voir *http://xmlfr.org/w3c/TR/REC-DOM-Level-1/level-one-core.html*.

b. Types de nœuds

1 : élément XML
2 : attribut
3 : texte
9 : Document

c. Node

Représente un nœud XML.

> Propriétés :

Nom	Type	Modifiable	Commentaire
nodeType	Entier	Non	Le type du nœud
nodeName	Chaîne	Non	Le nom du nœud
nodeValue	Chaîne	Oui	La valeur du nœud
parentNode	Node	Non	Le nœud parent
childNodes	NodeList	Non	La liste des nœuds enfants
firstChild	Node	Non	Le premier nœud enfant
lastChild	Node	Non	Le dernier nœud enfant
attributes	NamedNodeMap	Non	La liste des attributs d'un élément

> Méthodes

- booléen hasChildNodes()

Retourne vrai si le nœud courant (celui avec lequel on appelle la méthode) possède des nœuds enfants.

- Node insertBefore(Node nouveau, Node positionNoeud)

Insère le nœud *nouveau* avant le nœud *positionNoeud* dans la liste des enfants du nœud courant. Retourne le nœud inséré.

- Node appendChild(Node nouveau)

Ajoute le nœud *nouveau* à la fin des enfants du nœud courant.

- Node removeChild(Node ancien)

Supprime le nœud *ancien* de la liste des enfants du nœud courant. Retourne le nœud supprimé.

- Node replaceChild(Node nouveau, Node ancien)

Remplace le nœud *ancien* par le nœud *nouveau* dans la liste des enfants du nœud courant. Si le nœud *nouveau* appartenait déjà à l'arbre formé par le document XML, il se trouve en fait déplacé. Retourne le nœud *ancien*.

- Node cloneNode(booléen enProfondeur)

Retourne une copie du nœud courant. Si *enProfondeur* vaut vrai, la fonction retourne l'ensemble du sous arbre représenté par le nœud.

d. NodeList

Représente une liste ordonnée de nœuds.

> Propriétés :

Nom	Type	Modifiable	Commentaire
length	entier	Non	Le nombre de nœuds de la liste

> Méthodes

- Node item(entier pos)

Retourne le nœud à la position *pos* dans la liste. Le premier nœud de la liste est à la position 0.

e. Attr

Représente un attribut XML, hérite de Node.

> Propriétés :

Nom	Type	Modifiable	Commentaire
name	Chaîne	Non	Le nom de l'attribut
value	Chaîne	Oui	La valeur de l'attribut

f. Element

Représente un élément XML, hérite de Node.

> Propriétés :

Nom	Type	Modifiable	Commentaire
tagName	Chaîne	Non	Le nom de l'élément (attribut ID)

> Méthodes

- Chaîne getAttribute(chaîne nom)

Retourne la valeur de l'attribut *nom* de l'élément courant.

- Attr getAttributeNode(chaîne nom)

Retourne l'objet attribut (et pas seulement sa valeur) nommé *nom* de l'élément courant.

- void setAttribute(chaîne nom, chaîne valeur)

Si l'attribut *nom* existe, sa valeur devient *valeur*. S'il n'existe pas, il est ajouté aux attributs de l'élément courant.

- Attr setAttributeNode(Attr nouveau)

Si l'élément courant ne possède pas d'attribut de même nom que l'attribut *nouveau*, cet attribut est ajouté et la fonction retourne null. Si l'élément courant possède déjà un attribut de même nom, il est remplacé par l'attribut *nouveau* et l'ancien attribut est retourné par la fonction.

- void removeAttribute(chaîne nom)

Supprime l'attribut *nom* des attributs de l'élément courant.

- Attr removeAttributeNode(Attr ancien)

Supprime l'attribut *ancien* des attributs de l'élément courant. Retourne l'attribut supprimé.

- NodeList getElementsByTagName(chaîne nom)

Retourne la liste de tous les éléments de nom *nom* enfants de l'élément courant. Si *nom* vaut " * ", tous les éléments enfants sont retournés.

g. Document

Représente un document XML. Hérite de Node.

> Propriétés :

Nom	Type	Modifiable	Commentaire
documentElement	Element	Non	L'élément racine du document

> Méthodes

- Element createElement(chaîne nom)

Crée un nouvel élement de nom *nom* et le retourne.

- Text createTextNode(chaîne valeur)

Crée un nouveau nœud texte de contenu *valeur*. Le type *Text* est dérivé de *Node*.

- Attr createAttribut(chaîne nom)

Crée un nouvel attribut de nom *nom*.

- NodeList getElementsByTagName(chaîne nom)

Retourne la liste de tous les éléments de nom *nom* enfants du document courant. Si *nom* vaut " * ", tous les éléments enfants sont retournés.

h. NameNodeMap

Représente une collection non ordonnée de nœuds XML.

> Propriétés :

Nom	Type	Modifiable	Commentaire
length	entier	Non	Le nombre de nœuds de la collection

> Méthodes

- Node item(entier pos)

Retourne le nœud à la position *pos* dans la liste. Le premier nœud de la liste est à la position 0. Contrairement au type NodeList, l'ordre des nœuds n'est pas défini par DOM, mais il est possible de les obtenir tous en balayant la collection de la position 0 à la position length-1.

- Node getNamedItem(chaîne nom)

Retourne le nœud XML de nom *nom* s'il appartient à la collection, null sinon.

- Node setNamedItem(Node nouveau)

Ajoute le nœud *nouveau* à la collection et retourne le nœud ajouté. Si un nœud de même nom existe déjà dans la collection il est remplacé par le nœud *nouveau*.

- Node removeNamedItem(chaîne ancien)

Retire le nœud de nom *ancien* de la collection et le retourne.

7. Mise en œuvre de DOM

a. Premier exemple

Dans l'exemple 11, nous exploitons un fichier *donnees.xml* contenant un catalogue de produits. La requête envoyée de manière asynchrone retourne ce fichier qui est ensuite exploré via DOM pour en afficher le contenu.

donnees.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<catalogue><produit><ref>p01</ref><des>chaise</des></produit><produit><ref>
p02</ref><des>table</des></produit></catalogue>
```

Remarque : il n'y a aucun retour à la ligne dans le contenu XML du fichier.

page1.html

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{ // comme d'habitude
}
function envoyerRequete(url)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('GET',url,true);
     requeteHttp.onreadystatechange=
                              function() {recevoirReponse(requeteHttp);};
     requeteHttp.send(null);
   }
   return;
}

function recevoirReponse(requeteHttp)
{
   if (requeteHttp.readyState==4)
   {
     if (requeteHttp.status==200)
     {
       traiterReponse(requeteHttp.responseXML);
     }
     else
     {
       alert("La requête ne s'est pas correctement exécutée");
     }
   }
}
```

```
function traiterReponse(reponseXML)
{
    var pdts=reponseXML.getElementsByTagName("produit");
    var i,pdt;
    for (i=0;i<pdts.length;i++)
    {
      pdt=pdts.item(i);
      alert('ref : ' + pdt.childNodes.item(0).firstChild.nodeValue);
      alert('des : ' + pdt.childNodes.item(1).firstChild.nodeValue);
    }
}
</script>
</head>
<body>
    <a href="javascript:envoyerRequete('donnees.xml');">
      envoyer requete ajax
    </a>
</body>
</html>
```

Fonctionnement :

- La ligne en gras dans la fonction *recevoirReponse* montre que l'on va traiter cette fois le résultat sous la forme d'un document XML et non comme un texte.
- Dans la fonction *traiterReponse*, la variable *pdts* reçoit la liste des nœuds XML correspondants à un élément *produit*.
- On explore ensuite cette liste dont la taille est donnée par la propriété *length*.
- Pour chaque élément *produit*, on récupère la liste des nœuds enfants (*childNodes*) et on exploite le fait que le premier nœud enfant *item(0)* correspond à la référence du produit, le second *item(1)* à sa désignation.
- Le texte contenu dans ces deux nœuds est contenu dans leur premier (et seul) nœud enfant (propriété *firstChild*).

b. Problèmes des éléments vides

Si le document XML contient des retours à la ligne, il y aura des éléments vides de type texte. Le problème, c'est que les navigateurs les traitent différemment : IE les ignore, Mozilla les prend en compte…

Pour résoudre ce problème, la meilleure solution est de ne prendre en compte que les nœuds correpondant à des éléments à l'aide de la méthode *getElementsByTagName* de chaque élément *produit* obtenu.

L'exemple 12 met en œuvre cette technique.

donnees.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<catalogue>
    <produit>
      <ref>p01</ref>
      <des>chaise</des>
    </produit>
    <produit>
      <ref>p02</ref>
      <des>table</des>
    </produit>
</catalogue>
```

Page1.html

```
function traiterReponse(reponseXML)
{
   var pdts=reponseXML.getElementsByTagName("produit");
   var i,j,pdt,elements,nom,valeur;
   i=0;
   for (i=0;i<pdts.length;i++)
   {
     pdt=pdts.item(i);
     elements=pdt.getElementsByTagName("*");
     for(j=0;j<elements.length;j++)
     {
       nom=elements.item(j).nodeName;
       valeur=elements.item(j).firstChild.nodeValue;
       alert(nom + ':' + valeur);
     }
   }
}
```

- La première ligne en gras montre la récupération des éléments enfants de l'élement *produit*.
- La seconde montre l'accès au nœud de type *Text* enfant du nœud *ref* ou *des*.

Autre solution

Une autre manière de résoudre ce problème est de "passer" les nœuds vides dans la liste des nœuds explorés en prenant en compte la propriété *nodeType* :

- Les nœuds vides sont des nœuds texte (*nodeType* = 3).
- Pour les nœuds correspondant à un élément, la propriété *nodeType* est à 1.

Le fichier *page2.html* (répertoire *Exemple12*) présenté page suivante implémente cette autre solution.

Page2.html

```
function traiterReponse(reponseXML)
{
   var pdts=reponseXML.getElementsByTagName("produit");
   var i,j,pdt;
   i=0;
   for (i=0;i<pdts.length;i++)
   {
     pdt=pdts.item(i);
     j=0;
     while(pdt.childNodes.item(j).nodeType!=1) j++; // passer les vides
     alert('ref : ' + pdt.childNodes.item(j).firstChild.nodeValue);
     j++;
     while(pdt.childNodes.item(j).nodeType!=1) j++; // passer les vides
     alert('des : ' + pdt.childNodes.item(j).firstChild.nodeValue);
   }
}
```

Les deux boucles en gras permettent de passer les éléments vides avec les navigateurs basés sur Mozilla.

c. Cas des attributs

Il est également possible de traiter les éléments XML possédant des attributs. L'exemple 13 manipule de tels éléments.

donnees.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<catalogue>
   <produit ref="p01">
     <des>chaise</des>
   </produit>
   <produit ref="p02">
     <des>table</des>
   </produit>
</catalogue>
```

page1.html

```
function traiterReponse(reponseXML)
{
   var pdts=reponseXML.getElementsByTagName("produit");
   var i,j,pdt,ref;
   for (i=0;i<pdts.length;i++)
   {
     pdt=pdts.item(i);
     ref=pdt.attributes.getNamedItem('ref').nodeValue;
     alert('ref : ' + ref);
     j=0;
     while(pdt.childNodes.item(j).nodeType!=1) j++; // passer les vides
     alert('des : ' + pdt.childNodes.item(j).firstChild.nodeValue);
   }
}
```

La ligne en gras montre comment utiliser la propriété *attributes* et la méthode *getNamedItem* pour récupérer la référence du produit.

Remarque : le contenu XML peut bien entendu être généré en PHP. C'est ainsi que la requête adressant le document *donnees.php* du répertoire *Exemple13* peut être traité de la même manière que le fichier *donnees.xml* par la page *page2.html*.

donnees.php

```
<?php
   header("Cache-Control: no-cache, must-revalidate");
   header('Content-Type: text/xml; charset=ISO-8859-1');
   echo '<?xml version="1.0" encoding="ISO-8859-1"?>';
   echo '<catalogue>';
   echo '<produit ref="p01">';
   echo '<des>chaise</des>';
   echo '</produit>';
   echo '<produit ref="p02">';
   echo '<des>table</des>';
   echo '</produit>';
   echo '</catalogue>';
?>
```

page2.html

```
   <a href="javascript:envoyerRequete('donnees.php');">
                                          envoyer requete ajax</a>
```

8. DOM et la page HTML

a. Principe

La variable javascript *document* représente le document HTML affiché par le navigateur. Ce document peut être lui aussi manipulé à l'aide de DOM, ce qui offre de nombreuses possibilités pour rendre les pages réactives. Il est en effet possible de manipuler le document affiché pour le modifier dynamiquement en javascript à l'aide de DOM, l'affichage étant automatiquement mis à jour par le navigateur.

b. Exemple

L'exemple 14 utilise ces possibilités pour construire dynamiquement une liste d'options sur la page HTML en cours d'affichage. La liste est constituée à partir du fichier XML *donnees.xml* récupéré à l'aide d'une requête ajax.

donnees.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<catalogue>
   <produit>
     <ref>p01</ref>
     <des>chaise</des>
   </produit>
   <produit>
     <ref>p02</ref>
     <des>table</des>
   </produit>
</catalogue>
```

page1.html

```
function traiterReponse(reponseXML)
{
   var pdts=reponseXML.getElementsByTagName("produit");
   var bodyHtml=document.getElementsByTagName("body").item(0);
   var listePdt=document.createElement("select");
   listePdt.setAttribute("id","listePdt");
   listePdt.setAttribute("size","4");
   listePdt.onchange=function() {afficher(this.value);};
   var i,j,pdt,option,elements,nom,valeur;
   for (i=0;i<pdts.length;i++)
   {
     option=document.createElement("option");
     pdt=pdts.item(i);
     elements=pdt.getElementsByTagName("*");
     for(j=0;j<elements.length;j++)
     {
       nom=elements.item(j).nodeName;
       valeur=elements.item(j).firstChild.nodeValue;
       if (nom=='ref')
       {
         option.setAttribute("value",valeur);
       }
       else
       {
         option.appendChild(document.createTextNode(valeur));
         listePdt.appendChild(option);
       }
     }
   }
   bodyHtml.appendChild(listePdt);
}
```

```
function afficher(ref)
{
   alert(ref);
}
```

Les lignes en gras dans la fonction *traiterReponse* montrent la construction de la liste et son ajout au document HTML.

Pour la gestion de l'événement onchange, le respect de DOM aurait voulu que l'on écrive :

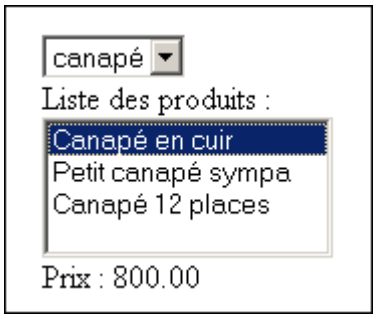
```
listePdt.setAttribute("onchange","javascript:afficher(this.value)");
```

Cette solution ne fonctionne malheureusement pas avec Internet Explorer.

9. Retour sur notre exemple

a. Rappel du résultat attendu

Il s'agit de reproduire la page de recherche d'un produit de l'exemple 10 (paragraphe 4) en utilisant cette fois DOM pour manipuler le résultat de la requête ajax et pour mettre à jour la page HTML.



b. Solution

L'exemple 15 répond à ce cahier des charges :

- Une requête ajax est envoyée au serveur lorsque l'utilisateur sélectionne une catégorie.
- S'il existe des produits dans cette catégorie, la liste des produits est construite en utilisant DOM.
- Une autre requête affiche le prix lorsque l'utilisateur sélectionne un produit dans la liste.

getProduits.php

```
<?php
   header("Cache-Control: no-cache, must-revalidate");
   header('Content-Type: text/xml; charset=ISO-8859-1');
   echo '<?xml version="1.0" encoding="ISO-8859-1"?>';
   echo '<catalogue>';
   $cnx=mysql_connect('localhost','root','');
   mysql_select_db('ajax',$cnx);
   $req=mysql_query('select * from produit
                          where pr_categorie='.$_POST['categ']);
   $pdt=mysql_fetch_assoc($req);
   while($pdt)
   {
     echo '<produit>';
     echo '<ref>'.$pdt['pr_id'].'</ref>';
     echo '<des>'.$pdt['pr_libelle'].'</des>';
     echo '</produit>';
     $pdt=mysql_fetch_assoc($req);
   }
   mysql_close($cnx);
   echo '</catalogue>';
?>
```

getInfoProduit.php

```
<?php
   header("Cache-Control: no-cache, must-revalidate");
   header('Content-Type: text/xml; charset=ISO-8859-1');
   echo '<?xml version="1.0" encoding="ISO-8859-1"?>';
   echo '<infos>';
   $cnx=mysql_connect('localhost','root','');
   mysql_select_db('ajax',$cnx);
   $req=mysql_query('select pr_prix from produit
                        where pr_id='.$_POST['pdt']);
   $pdt=mysql_fetch_assoc($req);
   mysql_close($cnx);
   echo '<prix>'.$pdt['pr_prix'].'</prix>';
   echo '</infos>';
?>
```

page1.php

```
<html>
<head>
<script type="text/javascript">
function getRequeteHttp()
{ // comme d'habitude
}

function envoyerRequeteListeProduits(idCateg)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('POST','getProduits.php',true);
     requeteHttp.onreadystatechange=
                        function() {recevoirListeProduits(requeteHttp);};
     requeteHttp.setRequestHeader('Content-Type',
                        'application/x-www-form-urlencoded');
     requeteHttp.send('categ=' + escape(idCateg));
   }
   return;
}
```

```
function recevoirListeProduits(requeteHttp)
{
   if (requeteHttp.readyState==4)
   {
     if (requeteHttp.status==200)
     {
       var pdts=requeteHttp.responseXML.getElementsByTagName("produit");
       var bodyHtml=document.getElementsByTagName("body").item(0);
       var selectPdt=bodyHtml.getElementsByTagName("select").item(1);
       bodyHtml.removeChild(selectPdt);
       selectPdt=document.createElement("select");
       selectPdt.setAttribute("id","listePdt");
       selectPdt.setAttribute("size","4");
       selectPdt.onchange=function() {envoyerRequeteProduit(this.value);};
       var i,j,pdt,option,elements,nom,valeur;
       for (i=0;i<pdts.length;i++)
       {
         option=document.createElement("option");
         if(i==0)
         {
           option.setAttribute("selected","selected");
         }
         pdt=pdts.item(i);
         elements=pdt.getElementsByTagName("*");
         for(j=0;j<elements.length;j++)
         {
           nom=elements.item(j).nodeName;
           valeur=elements.item(j).firstChild.nodeValue;
           if (nom=='ref')
           {
             option.setAttribute("value",valeur);
           }
           else
           {
             option.appendChild(document.createTextNode(valeur));
             selectPdt.appendChild(option);
           }
         }
       }
       var brPrix=bodyHtml.getElementsByTagName("br").item(2);
       bodyHtml.insertBefore(selectPdt,brPrix);
       if(pdts.length>0)
       {
       envoyerRequeteProduit(selectPdt.firstChild.getAttribute("value"));
       }
       else
       {
         document.getElementById("prix").innerHTML='';
       }
     }
     else
     {
       alert("La requête ne s'est pas correctement exécutée");
     }
   }
}
```

```
function envoyerRequeteProduit(idPdt)
{
   var requeteHttp=getRequeteHttp();
   if (requeteHttp==null)
   {
     alert("Impossible d'utiliser Ajax sur ce navigateur");
   }
   else
   {
     requeteHttp.open('POST','getInfoProduit.php',true);
     requeteHttp.onreadystatechange=
                         function() {recevoirInfoProduit(requeteHttp);};
     requeteHttp.setRequestHeader('Content-Type',
                         'application/x-www-form-urlencoded');
     requeteHttp.send('pdt=' + escape(idPdt));
   }
}

function recevoirInfoProduit(requeteHttp)
{
   if (requeteHttp.readyState==4)
   {
     if (requeteHttp.status==200)
     {
       document.getElementById("prix").innerHTML=requeteHttp.responseText;
     }
     else
     {
       alert("La requête ne s'est pas correctement exécutée");
     }
   }
}
</script>
</head>
<body>
<?php
   echo '<select name="idCateg" onchange=
                   "javascript:envoyerRequeteListeProduits(this.value)">';
   $cnx=mysql_connect('localhost','root','');
   mysql_select_db('ajax',$cnx);
   $req=mysql_query('select * from categorie');
   $ligne=mysql_fetch_assoc($req);
   if ($ligne)
   {
     $categ1=$ligne['ca_id'];
     echo '<option selected value='
                             .$ligne['ca_id'].'>'.$ligne['ca_libelle'];
     $ligne=mysql_fetch_assoc($req);
     while($ligne)
     {
       echo '<option value='.$ligne['ca_id'].'>'.$ligne['ca_libelle'];
       $ligne=mysql_fetch_assoc($req);
     }
   }
   else
   {
     $categ1=null;
   }
   echo '</select>';
   echo '<br />';
   echo '<span>Liste des produits : </span><br />';
   echo '<select id="listePdt" size="4" onchange=
                         "javascript:envoyerRequeteProduit(this.value)">';
```

```
    if ($categ1!=null)
    {
      $req=mysql_query('select * from produit
                        where pr_categorie='.$categ1);
      $ligne=mysql_fetch_assoc($req);
      if ($ligne)
      {
        echo '<option selected value='
                        .$ligne['pr_id'].'>'.$ligne['pr_libelle'];
        $prix1=$ligne['pr_prix'];
        $ligne=mysql_fetch_assoc($req);
      }
      else
      {
        $prix1='';
      }
      while($ligne)
      {
        echo '<option value='.$ligne['pr_id'].'>'.$ligne['pr_libelle'];
        $ligne=mysql_fetch_assoc($req);
      }
    }
    echo '</select>';
    echo '<br />';
    echo 'Prix : <span id="prix">'.$prix1.'</span>';
    mysql_close($cnx);
?>
</body>
</html>
```

Conclusion

Une fois bien comprise, la technique ajax est simple à mettre en oeuvre et permet d'obtenir des pages très réactives. Certains y voient le retour en force de javascript…

Pour ce qui est de l'utilisation de DOM, il est certain que les différences de comportement entre navigateurs compliquent un peu les choses.

L'utilisation d'Ajax n'a pas que des avantages...

D'une part, il faut rester conscient que tout ceci ne fonctionne plus si le navigateur est trop ancien ou encore si l'utilisateur a désactivé l'exécution des scripts javascript sur son poste de travail. Il est important d'utiliser la balise HTML *noscript* pour parer à cette éventualité. Bien sûr, le problème ne se pose pas en intranet puisque l'on maîtrise la configuration des postes clients.

D'autre part, on peut estimer qu'une application Web possède 4 composantes clairement identifiées :

- L'information présentée à l'utilisateur décrite en HTML,
- la présentation de cette information exprimée en CSS,
- les données servant de ressources à l'information mémorisées dans une base de données,
- les traitements nécessaires à l'application exécutés par le moteur PHP du serveur http.

Le retour de javascript risque de « casser » cette belle organisation en réintroduisant du traitement sur le poste client.