Comprendre les mécanismes d'AJAX

par Didier Mouronval (Accueil)

Date de publication : 04 août 2009

Dernière mise à jour : 06 août 2009

AJAX est encore trop souvent mal compris par les débutants. Qu'il s'agisse de sa mise en uvre, de ses capacités ou de sa nature même. Cet article va tenter de dégonfler les différentes baudruches concernant cette "technologie". Nous verrons aussi en quoi AJAX se différencie des différentes techniques d'inclusion de contenu.

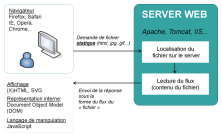
Préambule

Cet article n'a pas pour objet de vous apprendre à utiliser AJAX, mais de vous faire comprendre ses mécanismes. Pour ceux qui souhaitent en savoir plus, je vous invite à consulter :

- **Les tutoriels AJAX** de developpez.com.
- En particulier, pour les débutants et en complément de cet article : **Ajax : Vos premiers pas dans les nouvelles technologies** et **Web 2.0, allez plus loin avec AJAX et XMLHttpRequest**.
- FAQ **La FAQ AJAX**.

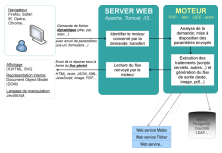
Avant toute chose, il est important de bien comprendre comment est architecturée une communication client / serveur. Il existe deux types de communication entre le client (habituellement, le navigateur) et le serveur, les communications dites "statiques", c'est-à-dire qui se contentent d'envoyer vers le client un flux de données présentes sur le serveur et les communications dites "dynamiques" qui effectuent un traitement sur le serveur pour renvoyer au client le résultat de ce traitement.
Un schéma valant mieux que de longs discours, voici des représentations des architectures client / serveur statiques et dynamiques :

Schéma représentant le fonctionnement d'un site statique :



Relation client / serveur statique

Schéma représentant le fonctionnement d'un site dynamique :



Relation client / serveur dynamique

Je remercie **emmanuel.remy** pour la création et la mise à disposition de ces schémas !

AJAX : késako ?

Ce qui fait généralement le plus peur quand on commence à aborder AJAX, c'est l'apparente complexité pour obtenir une instance *XMLHttpRequest*. *XMLHttpRequest* est la clé de voûte d'une requête AJAX, sans lequel il est impossible de faire quoi que ce soit.
Historiquement, la technologie AJAX est apparue en 1999 avec IE5 ; à l'époque aucune norme AJAX du W3C n'existe et donc aucune notion d'objet *XMLHttpRequest*. Microsoft choisit alors de s'appuyer sur sa technologie *ActiveX MSXML*. Un an et demi plus tard Mozilla propose l'objet *XMLHttpRequest*, dont les noms des fonctions sont calqués sur celles de l'*ActiveX* de IE5 pour conserver un maximum de compatibilité entre les navigateurs. De facto *XMLHttRequest* devient une norme (Safari, Opera, Chrome l'adoptent) et c'est en 2006 que le W3C publie ses premières spécifications. L'objet *XMLHttpRequest* apparait ainsi dans IE7. Une seconde version des spécifications est finalement publiée en 2008.
Toujours est-il que la complexité n'est qu'apparente et que l'implémentation se fait assez facilement (FAQ **voir un exemple**).

Cependant, AJAX n'est rien d'autre qu'un acronyme ! Pour bien le comprendre, décomposons-le :
AJAX : Asynchronous JavaScript And XML, soit en français : JavaScript et XML asynchrones. Cependant, nous allons voir que les termes employés sont parfois là uniquement pour générer l'acronyme !

- **Asynchrone** : il s'agit de la capacité de communiquer avec une ressource coté serveur sans figer le navigateur (c'est-à-dire que le reste de la page reste actif), il est à noter que bien que l'aspect asynchrone d'une requête Ajax en fait souvent un avantage non négligeable, cela ne reste qu'une option, il est tout à fait possible de créer des requêtes synchrones, qui bloqueront donc votre page tant que le serveur n'aura pas répondu.
- **JavaScript** : et bien oui, AJAX n'est rien d'autre que l'utilisation de l'objet (ou ActiveX) XMLHttpRequest de JavaScript. Tout ce que vous aurez à savoir pour créer une requête Ajax efficace, ce seront les différentes propriétés de cet objet.
- **Et XML** : si le XML se justifiait à l'époque des premières évocations d'AJAX (la mode était au XML et les formats alternatifs comme JSON n'existaient pas), il est aujourd'hui controversé. Toujours est-il que le retour d'une requête AJAX est supposé être soit du texte, soit du XML, on comprend donc bien que ce X est surtout là pour l'acronyme !

Mécanisme de création d'une page HTML

Avant de rentrer plus en détail dans le déroulement d'une requête AJAX, il est important d'avoir une idée claire du déroulement de création d'une page Web et en particulier de la communication client / serveur.
Dans cet article, j'ai pris comme référence de langage serveur le PHP. Sachez que c'est un choix arbitraire et que tout ce qui suit est valable **quel que soit** le langage serveur utilisé !

Lorsqu'un internaute demande une page Web via son navigateur, il va chercher un fichier sur un serveur. Une connexion est donc ouverte vers le serveur pour pouvoir communiquer avec lui. Si le fichier en question est identifié comme du HTML (par son extension essentiellement), alors, il "comprend" que le résultat sera directement exploitable par le navigateur et renvoie donc le contenu du fichier au navigateur qui ferme la connexion.
En revanche, s'il comprend que le contenu est du PHP (j'insiste, ou tout autre langage serveur), alors, il sait qu'il doit d'abord interpréter le code contenu dans le fichier pour pouvoir renvoyer au navigateurs du code HTML compréhensible par le navigateur, que l'on appelle du coup "généré". Bien entendu, quand je dis que le code généré sera compris par le navigateur, cela implique que le développeur a bien codé sa page !
Une fois le code interprété, le résultat est envoyé vers le navigateur qui l'affiche, sans savoir qu'il s'agit d'un document PHP, ce qui du reste ne servirait à rien car un navigateur ne sait pas évaluer du code PHP, puis ferme la connexion.

Interaction PHP / JavaScript

Les notions abordées ici se limitent aux cas des requêtes HTTP de type GET ou POST (les plus courantes), s'il existe des méthodes autorisant les connexions dites persistantes, elles dépassent largement le cadre de cet article.

JavaScript (dans le cadre d'une requête AJAX) est lui interprété sur le navigateur.
La conséquence du fonctionnement décrit précédemment est que les interpréteurs PHP et JavaScript ne fonctionnent **jamais** en même temps !
En effet, lorsque le serveur interprète le PHP, le document HTML n'existe pas, il n'existera qu'une fois le contenu reçu et affiché sur le navigateur, donc une fois la connexion client / serveur fermée. A l'inverse, lorsque l'interpréteur JavaScript entre en action, c'est-à-dire dès que le navigateur aura rencontré une balise *<script>* alors le document PHP n'existe plus.

Il est donc absolument impossible de faire interagir PHP et JavaScript !

Comment ça impossible !?

Je vous entends déjà pester ! Comment ça c'est impossible, je l'ai pourtant déjà fait !
Et bien non, c'est faux ! Vous avez déjà certainement transmis des données du client (navigateur) vers le serveur, en effet, JavaScript est exécuté coté client, PHP coté serveur, le client et le serveur savent communiquer entre eux et se transmettre des données, donc JavaScript est capable de dire au navigateur de transmettre des données au serveur puis le serveur est capable de dire à PHP qu'il a reçu telles données du navigateur. De même, PHP est capable de

dire au serveur d'envoyer des données au navigateur qui les transmettra à JavaScript, mais à **aucun moment** PHP et JavaScript n'ont communiqué entre eux.
Vous trouvez peut-être que je chipote, mais cela a une importance majeure : l'intérêt de faire interagir les deux serait d'échanger des données typées (un nombre, un tableau, un booléen, un objet etc.) mais le client et le serveur ne sont pas des langages de programmation et ne connaissent pas ce genre de données ! Le protocole HTTP impose que les seules informations qui peuvent transiter et être exploitées soient au format texte, éventuellement sous forme de texte structuré (XML par exemple) reconnu à la fois par JavaScript et PHP bref côté client et côté serveur.

Communications serveur -> client

Il est donc possible de faire parvenir au client ou au serveur des données de type textuel.
Pour passer de PHP à JavaScript, la solution est plutôt simple. Puisque c'est le navigateur qui créera le contexte JavaScript et qu'il lui suffit de rencontrer une balise *<script>* pour savoir qu'il faut interpréter son contenu comme du JavaScript (à condition bien sûr, que l'attribut *type* soit bien renseigné), il n'y a qu'à insérer ces balises dans la page générée. Voici un exemple pour créer un tableau JavaScript depuis un tableau PHP :

```
<?php
$tableau = array('toto', 'tata', 5);
echo '<script type="text/javascript">
var tableau = ["'.$tableau[0].'","'.$tableau[1].'",'.$tableau[2].'];
</script>';
?>
```

Vous noterez bien que l'on ne transmet pas directement le tableau, on construit un tableau JavaScript depuis les éléments d'un tableau PHP.
De même, concernant la valeur numérique, elle n'est pas insérée comme étant un entier. Elle est ajoutée sous forme de texte formaté de façon à ce que JavaScript puisse interpréter qu'il s'agit d'un nombre.


***Attention**, dans le cadre d'Ajax, le code JavaScript ne sera pas interprété, nous verrons pourquoi plus tard.*

Concernant la communication entre JavaScript et PHP, les modalités sont plus complexes et dépendent de différents cas, c'est ce que nous allons détailler maintenant.

Communications client -> serveur classique

J'entends par communication client / serveur classique la navigation sans utiliser AJAX.
Il existe trois façons de communiquer entre le navigateur et le serveur sans utiliser AJAX.

Les formulaires

Lorsque l'on soumet un formulaire, le navigateur récupère les valeurs des éléments du formulaire soumis possédant un attribut *name* puis transmet ces couples nom / valeur à la page définie par l'attribut *action* via une requête HTTP soit en les ajoutant à l'URL soit dans le corps de la requête selon la valeur de l'attribut *method* ("GET" dans le premier cas, "POST" dans le second).
PHP reçoit ces paramètres dans les tableaux *$_GET* ou *$_POST*. Les valeurs sont alors nécessairement de type chaîne. Il ne peut pas en être autrement pour deux raisons :

- Comme nous l'avons déjà évoqué, le protocole ne connait pas les types.
- Il est nécessaire de passer par une couche supplémentaire, le HTML, pour transmettre les données, or HTML n'accepte pas les variables et ne reconnait donc pas les types non plus.

PHP peut alors construire une nouvelle page en prenant ces informations en compte puis renvoyer un nouveau document HTML au navigateur. Le navigateur recharge alors la page pour afficher le nouveau contenu.

Vous noterez toutefois que les champs de formulaires peuvent êtres remplis en JavaScript. Il est donc possible de transmettre des représentations textuelles des variables JavaScript à PHP (voire de les réintégrer dans la nouvelle page). Mais il s'agira tout de même d'un contexte JavaScript différent.

Les liens

Comme pour les formulaires, il est possible de transmettre des données depuis un lien. Cependant, nous sommes ici limités à la méthode GET, c'est-à-dire à les faire passer par l'URL. Ces paramètres sont à inclure dans l'URL après un point d'interrogation ("?") sous forme de couples nom / valeur séparés par le caractère "&" :

```
www.developpez.com?toto=tata&auteur=bovino
```

Là encore, nous sommes limités à des données textuelles, mais nous pouvons ajouter avec JavaScript des valeurs à transmettre :

```
<a href="www.developpez.com" id="test">Lien</a>
<script type="text/javascript">
 var variable = 'bovino';
 var parametres = '?toto=tata&auteur=" + variable;
 document.getElementById('test').href += parametres;
</script>
```

Vous constaterez que la couche HTML est ici aussi nécessaire pour communiquer avec le serveur.

Les redirections JavaScript

L'objet *location* de JavaScript comprend quatre méthodes permettant de faire une redirection. Il est donc possible d'utiliser ces méthodes pour transmettre des paramètres.

méthode	description
location.href	Il ne s'agit en fait pas d'une méthode mais d'une propriété qui fait référence à l'URL de la page. Elle permet de récupérer ou d'affecter l'URL de la page.
location.assign()	Cette méthode charge l'URL passée en argument avec inscription dans l'historique.
location.replace()	Cette méthode charge l'URL passée en argument sans inscription dans l'historique.
location.reload()	Cette méthode recharge la page en cours, elle n'est pas utile pour transférer des paramètres JavaScript.

Exemples

```
location.href = 'www.developpez.com?toto=tata&auteur=bovino';
location.assign('www.developpez.com?toto=tata&auteur=bovino');
location.replace('www.developpez.com?toto=tata&auteur=bovino');
```

Il faut noter qu'ici, c'est JavaScript qui appelle directement la requête HTTP, il n'y a plus d'intermédiaire HTML, en revanche, c'est le moteur HTML qui reçoit la réponse du serveur.

Le point commun de toutes ces méthodes est que l'on envoie des informations au serveur qui renvoie un document complet. Il y a donc nécessairement rechargement de la page. AJAX va nous permettre de modifier uniquement des

portions de page lorsque le rechargement complet n'est pas nécessaire. Nous allons voir ce que cela change par rapport aux modèles précédents.

Le cas AJAX

Premier principe

L'utilité essentielle d'AJAX est la capacité d'utiliser des informations présentes sur le serveur dans un script JavaScript sans avoir à recharger la page. Les exemples typiques d'utilisation sont nombreux, par exemple :

- Modifier une partie de la page sans avoir à la recharger entièrement comme c'était le cas dans les techniques présentées auparavant.
- Enregistrer à la volée des informations sur le serveur.
- Vérifier la validité et/ou la disponibilité (pseudos, mots de passe...) de champs de formulaires avant de soumettre ceux-ci (ATTENTION : cela n'exonère pas des vérifications coté serveur !)
- Proposer des suggestions lors du remplissage d'un champ de formulaire (autocomplétion).
- Etc.

*La première conséquence est donc que lors d'une requête AJAX, le serveur **ne doit pas** renvoyer un document HTML.*

En effet, dans les communications client / serveur sans utiliser AJAX, on parle de navigation, c'est-à-dire que le serveur doit renvoyer une page HTML complète au navigateur qui se charge ensuite de l'afficher. Le navigateur a besoin d'une page complète (et si possible valide) pour l'afficher correctement.
A l'inverse, dans le cas d'AJAX, nous souhaitons juste mettre à jour une partie de la page, y insérer une page HTML complète constituerait donc une erreur grave de conception et risquerait de provoquer des erreurs d'affichage.
Il est cependant possible de créer coté serveur un fragment HTML (par exemple le contenu HTML que l'on veut insérer et qui peut lui-même comporter des balises. Mais ce fragment ne sera pas interprété comme du HTML mais comme du texte simple.

Que doit-on donc faire côté serveur ?

JavaScript ne pourra interpréter la réponse que comme du texte. Cependant, la méthode de création est identique à celle utilisée pour la création d'une page. En PHP, le résultat retourné par le serveur sera celui qui aura été affiché avec des instructions *echo* par exemple. Le script PHP devra donc organiser le résultat obtenu de façon a être facilement exploitable par JavaScript. Il existe différentes techniques pour cela :

- Renvoyer du texte simple, par exemple si vous avez juste un message à renvoyer.

```
echo 'Ce login est invalide';
```

- Un document XML.

*Attention, pour que la réponse du serveur puisse être correctement interprétée, il est important de le préciser dans le header du script PHP et que le XML retourné soit valide.
Il est aussi à noter que le format XML est considéré comme plus sécurisé que les autres modes d'échange.*

- Un objet JSON (FAQ **JavaScript Object Notation**).

S'il est un langage de balises permettant de structurer parfaitement les données, XML génère en contrepartie des échanges conséquents de données souvent inadaptées et qui nécessitent d'être ensuite analysées ("*parsées*") dès réception. Le format d'échange JSON est ainsi naturellement apparu ; il présente l'avantage d'organiser les données

de façon comparable, mais beaucoup plus légère, en utilisant la syntaxe des objets JavaScript, et de les rendre directement utilisables :

```
{nom1: valeur1,
nom2: valeur2,
nom3: {
 sousnom1: sousvaleur1,
 sousnom2: sousvaleur2
}
}
```

On utilise des couples nom / valeur, chaque couple est séparé par une virgule (mais il ne faut pas de virgule après le dernier), les noms et les valeurs sont séparés par deux points (caractère ":"). Les valeurs peuvent elles-mêmes être des objets.

Le format JSON offre plusieurs avantages :
- Il est léger et facilement manipulable.
- Il est normalisé, cela signifie que n'importe qu'elle application supportant ce format peut accéder à son contenu sans avoir à ajouter quoi que ce soit.

En revanche, il présente aussi un inconvénient : s'agissant au final d'une donnée typée, il ne pourra pas être envoyé en tant que tel mais sous forme de texte, il faudra donc l'évaluer avec JavaScript avant de l'utiliser.

- Un système de pseudo XML en séparant les différents résultats avec des séparateurs prédéfinis.

```
$nom1 = $valeur1;
$nom2 = $valeur2;
$nom3 = $valeur3;
echo $nom1.'<>'.$valeur1.'<arg>'.$nom2.'<>'.$valeur2.'<arg>'.$nom3.'<>'.$valeur3;
```

Les avantages de ce type de format sont : - Il est relativement léger et facile à mettre en □uvre. - Il est directement exploitable en JavaScript et chaque élément facilement récupérable au moyen d'appels à la méthode *split()* de l'objet *String*.

Les inconvénients :
- Il n'est pas normalisé, il est donc nécessaire de bien documenter sa structure.
- Il devient vite lourd à gérer en cas de structure trop complexe (imbrications multiples par exemple).

Etapes de la requête

Une requête AJAX se décompose en différentes étapes :

- Créer un objet XMLHttpRequest (**Voir un exemple dans la FAQ**).
- Définir les actions à réaliser lors des différentes réponses du serveur (méthode onreadystatechange()).
- Récupérer les éventuels paramètres à envoyer.
- Ouvrir une connexion avec le serveur (méthode open()).
- Envoyer la requête au serveur (méthode send()).

Vous constaterez qu'avec AJAX, c'est JavaScript qui envoie et qui reçoit les données. Il s'agit d'une différence essentielle avec la navigation classique.
En effet, nous avons vu que c'est le moteur HTML qui insère le JavaScript dans le contexte du document et cela lorsqu'il rencontre une balise <*script*>. Donc, comme le moteur HTML n'intervient pas dans une requête AJAX, ce qui signifie que :

Il n'est pas possible de récupérer du JavaScript avec AJAX.

Bon d'accord, j'exagère un peu... Mais il faut bien comprendre que le code JavaScript contenu dans la réponse du serveur est, comme tout le reste, considéré comme du texte, il est donc nécessaire de l'évaluer (avec la méthode *eval()*) avant de pouvoir l'utiliser, tout comme les réponses au format JSON.

```
eval(XMLHttpRequest.responseText);
```

Il est cependant à noter que l'utilisation de *eval()* est plutôt déconseillée en règle générale.
Cette méthode oblige à appeler le moteur JavaScript, ce qui réduit considérablement les performances.
Cette méthode est aussi considérée comme peu sécurisée car elle donne une importance exagérée au texte évalué.
Dans le cadre d'AJAX, on peut se dire (avec bon sens) qu'étant donné la *Same Origin Policy* (SOA : politique de même origine) inhérente à JavaScript et donc à AJAX, les risques sont minimes. C'est logiquement le cas si l'on considère que le développeur est compétent, ce qui n'est pas toujours le cas... mais qu'en est-il si la page appelée par votre requête fonctionne à la manière d'un proxy, qui est encore aujourd'hui la méthode la plus courante pour faire de l'AJAX inter domaine ? Avez-vous vraiment confiance dans les intermédiaires chez lesquels vous récupérez vos données ?
Il faut toutefois être conscient qu'il est impossible d'éviter l'utilisation de *eval()* pour interpréter du JavaScript récupéré par AJAX.
De ce fait, si vous avez à inclure du JavaScript dans un document via AJAX, demandez-vous au préalable si vous ne faites pas de fautes de conception et donc s'il n'y a pas un moyen plus académique de procéder.
D'après mon expérience, les cas où il est impossible de procéder différemment sont assez rares.

Une utilisation détournée de *eval()* serait de passer directement par le DOM, l'intérêt étant de rendre le script accessible dans le contexte global si vous l'évaluez par exemple dans le cadre d'une fonction anonyme de rappel type *callback* :

```
var lafonction = XMLHttpRequest.responseText;
var lescript = document.createElement('script');
lescript.type='text/javascript';
lescript.appendChild(document.createTextNode(lafonction));
document.getElementsByTagName('head')[0].appendChild(lescript);
```

Attention, cette méthode n'a pas été testée avec tous les navigateurs !

Asynchrone ou synchrone ?

Il est aussi important de comprendre à quoi correspondent les notions de requêtes synchrones ou asynchrones.
Le principe est relativement simple et bien compris, mais les implications sont plus difficiles à appréhender.

Lors d'une requête synchrone, la fonction qui la contient bloque son exécution tant que la réponse du serveur n'a pas été reçue. A titre de comparaison, c'est presque le même fonctionnement qu'avec une boîte d'alerte. Les implications sont que de ce fait, aucun code JavaScript ne peut être exécuté pendant ce temps, la page n'est plus interactive.

Lors d'une requête asynchrone au contraire, la requête est lancée mais le code de la fonction continue à s'exécuter.
C'est habituellement un intérêt majeur d'AJAX, en revanche, il est important de notifier au visiteur que l'action qu'il a déclenchée est en cours d'exécution, car rien n'est plus agaçant que de cliquer sur un contrôle devant effectuer une action et de ne rien voir arriver...
Une autre conséquence, au niveau du développeur, est de bien comprendre que pendant le cours de la requête, la fonction qui l'a créée continue son exécution et bien souvent termine son exécution avant que la réponse du serveur ne revienne. Or, quand une fonction termine son exécution, son contexte est détruit.
De ce fait, il est important de savoir que si vous avez besoin des résultats de la requête dans la fonction qui la crée, vous risquez d'être obligé de passer par une requête synchrone.

En général, on préfèrera envisager tous les traitements de retour dans la fonction appelée par la réussite de la requête (ce que l'on appelle habituellement le *callback*) via l'événement *onreadystatechange* en prenant bien soin de rendre disponibles les éléments du contexte requis.

L'aspect asynchrone ne représente que la valeur par défaut d'une requête AJAX, elle est cependant celle que je préconiserais.
Tout comme pour les évaluations de scripts, les cas où l'utilisation de requêtes synchrones est indispensable sont plutôt rares et je préfère tenter de privilégier les actions des utilisateurs au maximum.

Remerciements

Je tiens à remercier **emmanuel.remy** pour ses conseils qui ont permis d'améliorer notablement la qualité de cet article et pour la création des schémas ainsi que **ram-000** pour sa relecture avisée.