AJAX : Premiers Pas

Asynchronous JavaScript and XML
A nous la liiiiiberté !

Depuis quelques temps, lorsque l'on s'intéresse au Développement Web, on entend beaucoup parler d'une idée, que certains élévent au rang de technologie, nommée "AJAX".
Cet article va nous permettre de voir ce qu'est AJAX, quels sont ses principales caractéristiques, et comment écrire une application simple utilisant de l'AJAX pour répondre aux attentes de l'utilisateur.

Ceux d'entre vous qui savent déjà ce qui se cache derrière le terme d'Ajax, et souhaitent directement apprendre à développer une application basée sur des fonctionnalités Ajax, voudront peut-être sauter la première partie de cet article, qui, en quelques mots, présente ce qu'est Ajax et ce qu'il peut avoir d'intéressant pour votre site.

Avant de commencer, j'aimerais attirer votre attention sur le fait que cet article demande quelque pré-requis : notamment, puisque présenter ces méthodes et langages n'est pas mon but ici, j'ai considéré que vous connaissez les bases de la programmation en JavaScript, que vous savez ce qu'est HTTP et son fonctionnement de base (types de requêtes, codes erreurs), et que vous connaissez le HTML.
Pour les exemples de programmes côté serveur que je présente, j'ai utilisé le langage PHP, en raison de sa relative simplicité et de sa disponibilité sur bon nombre d'offres d'hébergement accessibles à des particuliers.
Si vous ne maîtrisez pas au moins les bases correspondant à chacun de ces domaines, je ne peux que vous encourager à vous documenter un peu : en développement Web, les informations que vous découvrirez de la sorte vous serviront certainement un jour ou l'autre.

Sommaire

I:\ AJAX ?

Pour commencer, avant d'apprendre à ajouter des fonctionnalités AJAX à une application, nous allons voir ce qui est désigné par ce terme, et quels sont les principaux avantages et inconvénients de cette technologie.

A: Présentation

"Asynchronous JavaScript and XML" ; voila ce que signifie le terme "AJAX" (ou "Ajax", selon les goûts, devrais-je dire).

Ce terme a été défini par Jesse James Garrett, dans son article du 18 février 2005 intitulé Ajax: A New Approach to Web Applications [en].
Etant donné la date de parution de cet article, et l'attention que Ajax a su attirer sur lui depuis, on peut croire qu'il s'agit d'une nouvelle idée révolutionnaire... Mais non !

En fait, l'idée qui se cache derrière le terme d'Ajax est la possibilité, depuis une page Web, d'effectuer des appels à des programmes sur un serveur, et de ne mettre à jour qu'une partie de la page avec le résultat du programme - et non l'intégralité de la page, comme on a l'habitude de le voir sur la plupart de sites Web.
Des techniques de "Remote Scripting" existent depuis plus longtemps que cela, basées soit sur des applets JAVA, soit sur l'emploi d'iframes... Mais elles n'ont pas rencontré de réel succès.
En fait, ce n'est que relativement récemment que des sites potentiellement fortement visités, et se basant sur des fonctionnalités Ajax, ont vu le jour. Je pense notamment à des services tels gmail, qui dispose du poids marketing de google, ou Flickr. Le succès de sites tels ceux-ci, et l'aspect pratique de leur interface, a sans nul doute grandement contribué à celui de l'idée qui se cache derrière le terme d'Ajax : ces sites ont démontré que l'idée d'une application entièrement basée sur le Web est viable - du moins, dans une certaine mesure.

J'insiste sur le fait qu'Ajax en soit n'est pas une technologie (même si ça fait bonne impression de dire que c'en est une), mais, plutôt, un regroupement de plusieurs technologies, et leur emploi de manière judicieuse. En somme, lorsque l'on parle d'Ajax, on pense généralement à l'utilisation combinée des notions suivantes :

- HTML, XHTML, XML : Pour créer une page Web.

- CSS : Pour la présentation de cette page Web.
- JavaScript pour l'aspect dynamique au sein du navigateur, utilisant le DOM (Document Object Model) pour interagir avec les données présentées.
- Et, pour finir, une méthode permettant de faire des appels depuis la page Web au serveur, généralement de manière asynchrone ; en général, c'est une instance d'objet XMLHttpRequest qui est utilisée.

De ces technologies, seule la dernière est "nouvelle" (ou plutôt, seule la dernière est réellement exploitée depuis peu) ; tous les développeurs Web, du débutant au plus expérimenté, connaissent les premières.
Cela aussi est une raison qui explique le succès d'Ajax : développer un site dynamique demande juste d'apprendre à utiliser l'objet XMLHttpRequest ; pour le reste, il suffit de réutiliser ce que l'on sait déjà.
Notamment, il n'est pas nécessaire d'apprendre à programmer dans un langage tel JAVA, d'une certaine complexité par rapport à ce que l'on attend ici, et qui demande des bases solides en programmation.

J'aimerais tout de même, avant de poursuivre, attirer votre attention sur un point : le développement d'une application Web basée sur des fonctionnalités Ajax reste du développement. Cela sous-entend que, même si un débuttant parvient à programmer quelque chose qui fonctionne à peu près, c'est en pratiquant que l'on apprend à développer des applications robustes, ergonomiques, et efficaces ; cela ne vient pas en cinq minutes !
Je présenterai, à la fin de cet article, quelques uns des problèmes qui peuvent se poser lorsque l'on se prépare à développer une application conséquente en Ajax ; jettez-y un coup d'oeil, vous verrez que tout n'est pas rose...

B: Intérêt

J'avais prévu, au départ, d'intituler cette partie "Avantages" ; mais, en réfléchissant, je me suis dit que c'était trop subjectif, et que le terme d'intérêts est probablement plus adapté.
En effet, ce qui doit vous pousser à adopter une nouvelle technologie - ou, dans notre cas, un groupement particulier de technologies - n'est pas la liste des "ooohhhh Ajax fait ça ou ça c'est merveilleux", mais plutôt ce que vous, votre site, et vos visiteurs, peuvent en retirer ! En somme, en quoi est-ce qu'il peut être intéressant d'intégrer des fonctionnalités Ajax à votre site.

Avant tout, comme je l'ai déjà mentionné plus haut, Ajax permet de réaliser, depuis la page web affichée par le navigateur, des appels au serveur - de manière asynchrone.
Ce que cela signifie, c'est que vous pouvez ne modifier le contenu que d'une partie de la page, selon les actions effectuées par le personne visitant votre site. Cela signifie aussi que vous pouvez enregistrer sur un serveur (en base de données, par exemple) les préférences d'un utilisateur, ou lui présenter des informations, le tout sans qu'il n'ait à recharger la page !

Par exemple, lorsqu'un utilisateur saisit des données dans un champs de formulaire, vous pouvez en temps réel lui afficher des informations correspondant à sa saisie à côté du champ de saisie.
Une idée dans ce genre serait un champ de formulaire où le visiteur de votre site doit saisir un nom

de ville ; au fur et à mesure de sa saisie, vous pouvez lui proposer une liste de villes dont le nom commence par ce qu'il a saisi, à partir de la liste de noms de ville que vous avez en base de données.

Pour résumer, Ajax permet de rendre un site encore plus dynamique : après les sites disposant de capacités dynamiques côté serveur (génération de pages sur le serveur en intégrant des données dynamiques, mais pages statiques côté client - on peut notamment penser à tout ce qui est pages en PHP, ASP, ... Forums, Sites de nouvelles, ...), on assiste maintenant à l'apparition de sites disposant de capacités dynamiques côté client.
La visite d'un site n'est plus le passage d'une page statique à une autre page statique, mais à présent véritablement l'interaction entre un utilisateur, et une page, qui évolue en fonction de ses besoins, ses attentes, et ses préférences.

A présent, à vous de déterminer si votre site et vos utilisateurs ont besoin de ce genre de fonctionnalités.

II:\ Fonctionnement

Maintenant que nous avons vu, dans les grandes lignes, ce qu'est AJAX, nous allons pouvoir passer à la pratique.
Nous commencerons par le côté client ; après quoi nous passerons au programme chargé de répondre aux requêtes de celui-ci, sur le serveur.

Du côté client, au sein du navigateur de l'internaute visitant votre site, donc, il faut deux éléments :

- Une page HTML (ou XHTML, ou ce que vous voulez).
- Un script programmé en JavaScript.
- Eventuellement, pour arranger un peu l'aspect graphique de la page, une feuille de style CSS.

Si votre script ou votre feuille de style ne doivent servir que pour une page, vous pouvez directement les y intégrer ; sinon, vous serez probablement tentés de les mémoriser au sein d'un fichier externe, qui pourra être mis en cache par le client, et qui vous facilitera les mises à jour.

A: Une page HTML

Tout d'abord, après avoir, bien entendu, réfléchit à une vue d'ensemble de l'application, afin que vous sachiez où vous allez, il va vous falloir écrire une page HTML - ou, pourquoi pas, XHTML, ce qui ne serait pas plus mal ; vous auriez la possibilité de la manipuler plus facilement en cas de besoin.
Naturellement, cette page HTML peut être générée dynamiquement côté serveur, que ce soit en PHP, ASP, Perl, ... enfin, tout ce que vous voulez : à ce niveau là, il s'agit uniquement de générer une page HTML.

Par contre, il faut que vous pensiez à intégrer plusieurs éléments à votre page...

Tout d'abord, il faut qu'elle présente un moyen de déclencher les appels Ajax. Cela peut se faire en insérant un lien, qui appelle à l'événement onclick une fonction JavaScript ; comme ceci, par exemple :

```
<a href="" onclick="gestionClic(); return false;">Cliquez ici !</a>
```

Notez que nous retournons false après l'appel de la fonction, afin que le clic sur le lien ne soit pas transmis au navigateur, qui essayerait alors de charger l'URL donnée en href (Ici, l'URL étant vide, la page serait rechargée, ce qui n'est pas l'effet désiré).
Naturellement, vous pouvez choisir de déclencher un appel Ajax sur l'événement que vous voulez, sur l'élément que vous voulez ; par exemple, en cas de saisie dans un champ de formulaire, de survol d'une image, ... A vous de voir ce qui répond le mieux à vos besoins, et qui serait le plus intuitif pour vos visiteurs.

Si vous intégrer des fonctionnalités Ajax à votre site, cela peut être pour afficher à l'utilisateur des informations au fur et à mesure qu'il navigue, sans avoir à recharger l'intégralité de la page.
Afficher des informations implique qu'il fut prévoir un endroit au sein de la page pour cela, d'une part ; et qu'il faut que vous puissiez accèder à cet "endroit" depuis votre JavaScript, d'autre part.
Pour cela, il vous faut un élément (un div, un span, une image, un lien ; ce que vous voulez, selon ce que vous avez l'intention d'afficher), auquel vous associez un identifiant unique ; comme ceci, par exemple :

```
<span id="identifiantunique">
blah blah
</span>
```

Au chargement de la page, c'est "blah blah" qui sera affiché ; et vous aurez la possibilité, en JavaScript, d'accèder à l'élément via son identifiant "identifiantunique", afin de remplacer son contenu par autre chose - typiquement, le résultat de votre appel Ajax, ou les informations que vous extrayerez du résultat de l'appel en question.

B: Un peu de JavaScript

Le point central d'une application Web basée sur des fonctionnalités Ajax est le script côté client, en JavaScript, qui va permettre de lier la page HTML au programme sur le serveur.
Une requête Ajax implique plusieurs étapes ; en généralisant, pour un appel simple, on en compte trois :

- La création d'un object XMLHttpRequest
- L'ouverture d'une connexion et l'envoi de la requête
- Et le traitement du résultat de la requête

Point par point, nous allons voir les opérations de base à effectuer pour chacune de ces étapes.

1: XMLHttpRequest

Une requête Ajax se fait à l'aide d'un objet XMLHttpRequest. C'est cette classe qui permet de gérer l'ouverture de la connexion, l'envoi de la requête, et la réception de la réponse.
La classe XMLHttpRequest est intégrée en natif dans tous les navigateurs récents ou mis à jour fréquemment (Mozilla Firefox, Safari, Konqueror, ...) ; par contre, sous Internet Explorer, elle n'est présente que sous la forme d'un contrôle ActiveX. Nous n'avons pas encore écrit la moindre ligne de Javascript qu'il nous faut déjà prendre en compte les incompatibilités entre les différents navigateurs...

Sous Internet Explorer, l'instanciation d'un objet de type XMLHttpRequest se fait de la manière suivante :

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

Sous les autres navigateurs, on utilise ceci :

```
http = new XMLHttpRequest();
```

Par commodité, on peut être tenté d'écrire une fonction qui détecte le navigateur, et qui retourne un objet XMLHttpRequest construit comme il faut :

```
function createRequestObject()
{
    var http;
    if(window.XMLHttpRequest)
    { // Mozilla, Safari, ...
        http = new XMLHttpRequest();
    }
    else if(window.ActiveXObject)
    { // Internet Explorer
        http = new ActiveXObject("Microsoft.XMLHTTP");
    }
    return http;
}
```

Vous noterez que nous n'avons pas géré le cas où l'utilisateur utilise un navigateur qui ne fourni pas la classe XMLHttRequest (on peut notamment penser aux navigateurs non-graphiques) ; à vous de choisir que faire si une telle situation se présente ; est-ce que votre site sera inutilisable ? Ou prévoyez-vous de développer une application qui se dégrade gracieusement (qui continue de fonctionner "pas trop mal" alors qu'un composant nécessaire n'est pas disponible) ?

2: Envoi d'une requête

Une fois que nous avons un objet de type XMLHttpRequest, obtenu par exemple en utilisant la fonction que j'ai donné juste au-dessus, nous allons pouvoir envoyer une requête au serveur.
Cela nécessite deux étapes de préparation, puis l'envoi en lui-même.

Tout d'abord, nous devons indiquer à l'objet XMLHttpRequest la méthode qui devra être utilisée pour la requête (get, post, head, put, ... - en général, nous utiliserons du GET ; mais, après, ça dépend des besoins de votre application, bien entendu).
Puis nous indiquons l'adresse du programme chargé de traiter notre requête, ainsi que, éventuellement, la liste des paramètres que nous souhaitons passer à celui-ci, dans le cas où la requête n'est pas effectuée en mode POST (voir plus bas dans le cas contraire).
Et enfin, nous indiquons si la requête doit être asynchrone (true) ou non (false). Généralement, comme le A de **A**jax l'indique, nous travaillons avec des requêtes asynchrones : elles se font en arrière-plan, sans bloquer la navigation ; de la sorte, votre visiteur peut continuer à utiliser l'application même pendant que la requête se fait, ce qui peut être particulièrement appréciable lorsqu'il faut un certain temps pour traiter sa requête.
Pour cela, nous utilisons la méthode open :

```
objetXMLHttpRequest.open(methode, urlEtParams, async);
```

Par exemple :

```
http.open('get', './serveur.cgi', true);
```

Pour préparer une requête GET vers la page 'serveur.cgi', en mode asynchrone.

Ensuite, il nous faut indiquer à l'objet de type XMLHttpRequest quelle fonction de notre script devra être appelée lorsque quelque chose en rapport avec notre requête se passera (chargement, chargée, terminée, ...).
Pour cela, on renseigne la propriété onreadystatechange de l'objet. Par exemple, comme ceci :

```
http.onreadystatechange = handleAJAXReturn;
```

De la sorte, à chaque fois que l'état de l'objet changera, la fonction handleAJAXReturn sera appelée, ce qui nous permettra de prendre en compte le changement d'état, et, si nous le désirons, d'effectuer un traitement en réponse.

Et enfin, nous pouvons effectuer la requête ; pour cela, nous utilisons la méthode send de l'objet XMLHttpRequest.
Notez que si le requête que vous effectuer est en mode POST, c'est la méthode send qui prend en paramètre la liste des paramètres que vous souhaitez passer à votre programme sur le serveur, sous forme d'une chaîne de requête (de la forme `'a=123&b=345&c=coucou'`) ; dans le cas contraire, passer `null` convient tout à fait.
Par exemple :

```
http.send(null);
```

Une fois cette instruction effectuée, la requête Ajax s'effectue, et vous pourrez, comme nous allons le voir, récupérer l'état de celle-ci, ainsi que, une fois terminée, son résultat.

Notez qu'il peut être utile pour vos utilisateurs, surtout lorsqu'une requête met du temps à s'effectuer, d'afficher un message indiquant que l'information demandée est en cours de chargement, afin qu'ils ne se demandent pas si leur demande a bien été prise en compte ou non.

Par exemple, au moment d'appeler la méthode `send`, vous pouvez afficher un message "Chargement..." dans un coin de l'écran (ce que fait gmail, par exemple, et qui est très appréciable - d'autant plus que ceratins traitements mettent un temps perceptible à s'effectuer), ou à l'endroit où le résultat de la requête sera affiché une fois qu'elle sera terminée.
Naturellement, ce n'est pas obligatoire, et n'est pas nécessaire dans le cas d'une requête n'affichant aucune information en retour (encore que...), mais il est toujours bon d'indiquer à l'utilisateur que quelque chose se passe, surtout lorsque l'on développe une application Web qui ne corresponde pas aux habitudes de navigation acquises ces dix dernières années.

3: Traitement du résultat

Et maintenant, nous allons pouvoir apprendre à traiter les différents événements que l'on peut choisir de prendre en compte pendant l'exécution d'une requête Ajax.
Vous vous souvenez que nous avons, avant l'envoi de notre requête, spécifié une fonction qui devait être appelée à chaque changement d'état de notre objet ? Et bien, c'est elle qui va nous intéresser, à présent.

L'objet XMLHttpRequest fournit une propriété indiquant l'état dans lequel il se trouve ; cette propriété est `readyState` ; elle peut prendre plusieurs valeurs, qui correspondent chacune à un état de l'objet :

- 0 : Non initialisé
- 1 : En cours de chargement (requête en train de s'effectuer ; en attente de la réponse du programme sur le serveur)
- 2 : Chargé (la réponse est arrivée)
- 3 : Interactif (la réponse est en cours de traitement)
- 4 : Terminé (la requête est terminée, nous pouvons utiliser le résultat)

Si nous ne sommes intéressés que par le résultat, c'est le passage à l'état 4 qui nous intéresse.

Notre fonction de gestion de changement d'état peut donc ressembler à ceci :

```
function handleAJAXReturn()
{
    if(http.readyState == 4)
    {
        // Utilisation du résultat
    }
}
```

Cela dit, de cette manière, nous allons utiliser le résultat quel qu'il soit ; même si, notamment, une erreur s'est produite lors de la requête (que ce soit une erreur au niveau du serveur (erreur 500 par exemple), l'adresse de notre programme qui soit invalide (erreur 404 notamment), ou autre).
Pour éviter cela, il peut être judicieu de vérifier si le code HTTP retourné par le serveur correspond à "OK" (qui a pour code 200). Pour cela, on peut modifier notre fonction et lui rajouter la sécurité suivante :

```
function handleAJAXReturn()
```

```
{
    if(http.readyState == 4)
    {
        if(http.status == 200)
        {
            // Utilisation du résultat
        }
    }
}
```

Maintenant, voyons comment on peut utiliser le résultat...

Dans le cas où le serveur a envoyé une réponse en XML, celle-ci est accessible via la propriété `responseXML` de l'objet XMLHttpRequest qui a servi à effectuer la requête.
JavaScript vous fourni ensuite (quitte à utiliser une librairie qui améliore la compatibilité entre les différents navigateurs) la possibilité de manipuler les données de ce XML via le DOM.

Dans le cas où la réponse est du texte brut (ou si vous voulez y accéder sous forme de texte brut), que ce soit du texte, du HTML, du JSON, ..., vous pouvez utiliser la propriété `responseText` de l'objet XMLHttpRequest.
Par exemple :

```
alert(http.responseText);
```

pour l'afficher (d'une façon très peu agréable pour l'utilisateur, mais pratique pour le débuggage ^^).

C: Un programme sur le Serveur

Du côté du serveur, il vous faut un programme capable de répondre aux requêtes effectuées par le client. Ici encore, ce programme peut être développé dans le langage que vous voulez (selon ce que vous propose votre hébergeur, cela va de soit) : PHP, ASP, C, ...
Il faut juste que votre programme soit à même de reconnaitre les paramètres qui peuvent éventuellement être passés au moment de la requête, et qu'il sache générer une réponse.

Si la réponse attendue est du texte brut, vous n'avez pas grand chose à faire : il vous suffit d'écrire sur la sortie standard, et c'est ceci qui sera envoyé au client.
Par exemple (en PHP) :

```
<?php
    echo 'Hello World !';
?>
```

En somme, nous ne faisons ici rien de plus que de générer dynamiquement du contenu, qui pourra être affiché à partir du JavaScript, y compris sans aucune modification : il nous est même possible, pourquoi pas, de générer directement du HTML !

Si la réponse attendue doit être au format XML, ce n'est pas bien complexe non plus ; il nous

faut juste penser à préciser que le Content-Type est "application/xml" (et non "text/html" comme c'est généralement le cas par défaut - du moins en PHP).

```php
<?php
    header('Content-Type: application/xml');
    echo '<?xml version="1.0" ?>';
    echo '<racine>';
    echo 'Hello World !';
    echo '</racine>';
?>
```

Naturellement, libre à vous d'utiliser des API XML (et je vous encourage à le faire, d'ailleurs, si vous avez à manipuler des données complexes ou en grandes quantités) si vous en avez à disposition, plutôt que d'écrire vous-même le XML à la main.

Pour finir, j'aimerais émettre une recommandation : pour éviter que les données retournées par la requête Ajax ne soient mises en cache (que ce soit par votre navigateur - j'ai eu le problème sous Internet Explorer, notamment - ou un serveur proxy), et, donc, que l'utilisateur ait l'impression que rien ne se passe puisque les données affichées restent les mêmes, il peut être judicieu de demander, via les en-têtes HTTP de votre réponse, que la page ne soit pas mise en cache.
En PHP, on peut utiliser cette portion de code :

```php
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache");
```

(D'après la page 'header' de la documentation de PHP [fr])

III:\ Un exemple d'application AJAX

Maintenant que nous avons vu la théorie et que nous savons comment effectuer un simple appel Ajax, pourquoi ne pas passer à un petit exemple complet ?
Après tout, un peu de mis en application ne peut que nous faire du bien.

A: Présentation de notre application

L'application que je vais ici utiliser comme exemple est extrêmement simple : nous avons une page web qui contient un lien vers un site, et, à côté, un nombre de clics sur ce lien. A chaque fois que le lien est cliqué, nous voulons que le compteur soit incrémenté de 1, et réaffiché, sans que toute la page ne soit rechargée.
Vous pouvez voir cette (mini-)application sur cette page.
Je l'ai aussi intégrée ci-dessous dans une iframe (pour ceux qui ne les ont pas désactivées) afin que vous n'ayez pas à ouvrir une autre fenêtre :

Comme plus haut, nous allons commencer par voir la page client, puis le JavaScript, pour finir par le programme côté serveur.

B: Page cliente

Comme vous pouvez vous en douter à la lecture de la description de notre application, la page cliente va être plutôt simple : quelques lignes de XHTML, un lien, et un appel de fonction JavaScript sur l'événement onclick correspondant à ce lien. D'autre part, un élément span doté d'un identifiant, qui nous servira pour afficher le résultat de la requête Ajax.

On peut cependant noter un choix que j'ai été amené à faire au cours du développement de cette application : au chargement de la page, on peut (même plus que ça) vouloir afficher le nombre de fois que le lien a été cliqué, même si le visiteur actuel n'a pas encore cliqué dessus.
Dans ce cas de figure, nous avons deux solutions : nous pouvons effectuer un appel Ajax (voire même, des appels Ajax) au chargement de la page (événement onload de l'élément body) pour récupérer les données, et les afficher en JavaScript, ou nous pouvons intégrer les données à la page côté serveur, en utilisant un langage permettant de générer dynamiquement des pages.

Dans le premier cas, nous nous simplifions la vie, en tant que développeur : les appels Ajax (ou des appels approchants) doivent de toute façon être écrits pour assurer le bon fonctionnement du site ; alors, pourquoi ne pas les réutiliser aussi pour le premier affichage ?
Cela dit, une telle approche présente deux inconvénients majeurs : tout d'abord, la page peut s'afficher pendant quelques temps avant que les informations ne viennent s'y ajouter (le temps que les requêtes Ajax soient faites, les réponses reçues, et les résultats traités). D'autre part, cela revient à effectuer plus de requêtes (une pour le chargement de la page ; une ou plusieurs pour le chargement des données ; au minimum, on multiplie par deux le nombre de requêtes nécessaires au premier affichage d'une page - à vous de voir ce que ça représente comme charge pour votre serveur), ce qui peut éventuellement poser des problèmes de charge du serveur.

Dans le second cas, cela nous demande plus de temps en développement : la page doit être générée par programmation, ce qui implique du code à écrire (PHP ou autres, notamment).
Par contre, cela entraine moins de requêtes effectuées auprès du serveur, et une utilisation moindre de la bande passante.
Mais le principal avantage, à mon avis, de cette approche, est que la page affichée contiendra toujours des informations, même pour les utilisateurs ayant désactivé le JavaScript ou utilisant un navigateur en mode texte n'intégrant pas de JavaScript. Et ça, ça me parait être un point capital : avec cette méthode, tous les visiteurs pourront savoir combien de fois un lien a été cliqué (Vous vous souvenez, plus haut, je parlais de la capacité d'une application Web à se "dégrader gracieusement" ? Voila un excellent exemple à ce propos.). Voila la raison qui fait que j'ai tendance à privilégier cette approche lorsque je le peux, comme dans l'exemple que je vous présente...

Je vous laisse jeter un coup d'oeil ; rien de bien compliqué :

```
<?php echo '<?xml version="1.0" encoding="iso-8859-15"?>'; ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>AJAX : Exemple de client</title>
<script type="text/javascript" src="./exemple_1.js"></script>
</head>
<body>
<p>
<a href="" onclick="gestionClic(); return false;">Cliquez ici !</a>
(Nombre de clics : <span id="nbr_clics">
<?php
    $str = @file_get_contents('./exemple_1.data');
    if($str !== FALSE)
        echo unserialize($str);
    else
        echo 0;
?>
</span>)
</p>
</body>
</html>
```

Exemple Complet

Pour expliquer brièvement la portion de code PHP : le nombre de clics sur le lien est mémorisé sous forme sérialisée dans un fichier ; pour l'obtenir, il nous suffit de lire le contenu du fichier, de le dé-sérialiser, et de l'afficher ; en pensant à afficher 0 si le fichier n'a pas pu être lu (si le lien n'a jamais été cliqué, par exemple).
Naturellement, nous aurions pu utiliser une autre méthode, telle un enregistrement en base de données, pour mémoriser le nombre de clics ; mais pourquoi faire compliqué pour un simple petit exemple ?

C: JavaScript client

Nous allons maintenant passer à la partie JavaScript côté client. Nous ne verrons rien de nouveau ici, étant donné que nous ne ferons qu'utiliser ce que nous avons appris un peu plus haut :

```
var http; // Notre objet XMLHttpRequest

function createRequestObject()
{
    var http;
    if(window.XMLHttpRequest)
    { // Mozilla, Safari, ...
        http = new XMLHttpRequest();
    }
    else if(window.ActiveXObject)
    { // Internet Explorer
        http = new ActiveXObject("Microsoft.XMLHTTP");
```

```
    }
    return http;
}

function gestionClic()
{
    document.getElementById('nbr_clics').innerHTML = '<em>Chargement...</em>';
    http = createRequestObject();
    http.open('get', './exemple_1-serveur.php5', true);
    http.onreadystatechange = handleAJAXReturn;
    http.send(null);
}

function handleAJAXReturn()
{
    if(http.readyState == 4)
    {
        if(http.status == 200)
        {
            document.getElementById('nbr_clics').innerHTML = http.responseText;
        }
        else
        {
            document.getElementById('nbr_clics').innerHTML = "<strong>N/A</strong>";
        }
    }
}
```

Exemple Complet

Comme vous pouvez le constater, lorsque l'utilisateur cliquera sur le lien, la fonction `gestionClic` sera appelée.
Elle va commencer par créer un objet XMLHttpRequest, puis elle demandera le chargement de la page './exemple_1-serveur.php5', en mode asynchrone. C'est la fonction `handleAJAXReturn` qui sera chargée de gérer les changement d'état de l'objet.

Lorsque l'objet passe en état terminé (code 4), on vérifie que la page a bien été trouvée ; si oui, on affiche le nombre de clics au sein de l'élément prévu à cet effet dans la page HTML ; sinon, on affiche un message d'erreur.

Vous remarquerez que, pour cet exemple, j'ai choisi d'afficher un message de chargement à l'endroit où le résultat doit ensuite être affiché ; pas forcément l'emplacement le plus judicieux dans tous les cas, mais, pour cette application, ce n'était pas inadapté.
D'autre part, c'est au moment de réaliser l'appel Ajax que j'ai affiché ce message ; j'aurais aussi pu, évidemment, l'afficher au passage à l'état chargement (code 1), dans ma fonction `handleAJAXReturn`.

D: Programme serveur

Du côté serveur, étant donné la simplicité de notre application, pas grand chose non plus à dire : on lit les données enregistrée dans le fichier (Cf ce que j'ai dit un peu plus haut sur ce point, au

niveau client), on ajoute un au nombre de clics enregistrés, on affiche le résultat, et on le ré-enregistre dans le fichier.
Notez que le script commence par attendre une seconde en ne faisant rien ; ceci pour que l'on ait le temps, côté client, de voir le message de chargement. (Dans une application réelle, il serait bien évidemment ridicule de faire attendre l'utilisateur de la sorte - sauf si vous faites payer vos visiteurs à la durée qu'ils passent sur le site ^^).
Voici le code source du programme (Ici encore, j'ai choisi une implantation en PHP) :

```php
<?php
    sleep(1);

    $nbr = 1;

    $str = @file_get_contents('./exemple_1.data');
    if($str !== FALSE)
        $nbr = unserialize($str)+1;

    header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
    header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
    header("Cache-Control: no-store, no-cache, must-revalidate");
    header("Cache-Control: post-check=0, pre-check=0", false);
    header("Pragma: no-cache");

    echo $nbr;

    file_put_contents('./exemple_1.data', serialize($nbr));
?>
```

Exemple Complet

Vous noterez que, comme évoqué plus haut, j'ai pris soin d'inclure à mon script quelques lignes visant à désactiver la mise en cache.

IV:\ Quelques éléments supplémentaires

Pour terminer cet article, j'aimerais attirer votre attention sur quelques points supplémentaires...

Nous avons vu qu'il pouvait être simple d'intégrer un minimum d'Ajax à une application Web ; mais nous n'avons pas encore vu les différents problèmes que l'utilisation d'Ajax pouvait poser. C'est principalement de ces problèmes, ou de certaines limitations, que je vais parler maintenant, afin de vous montrer que même si Ajax est une méthode de développement qui peut être fort utile, elle n'est pas aussi miraculeuse que certains voudraient le (faire ?) croire.

A: Ajax, Utilisabilité, Navigation

Pour commencer, et pour rejoindre ce que je dis juste au-dessus, AJAX est encore un mot nouveau, pour désigner des idées relativement anciennes, même si elles n'étaient que peu

exploitées.
Cela dit, "AJAX" sonne bien, en terme de marketing ; mais lorsque vous développez un site Web, pensez à vos utilisateurs : est-ce que des fonctionnalités Ajax peuvent leur rendre service ? Et si ce n'est pas votre site Web sur lequel vous travaillez, demandez-vous si intégrer de telles fonctionnalités ne risque pas de vous coûter trop cher...
En somme, veillez à ce que "AJAX" ne soit pas, pour vous ou vos clients, (qu') un buzzword [en] (d'ailleurs, devinez qu'est-ce que l'on retrouve en tête de la Liste de buzzwords de wikipedia [en] au moment où j'écris cet article ?)

J'ai déjà évoqué ce point plus haut, mais je pense qu'en reparler ici ne peut être que bénéfique : AJAX n'est pas **une** technologie, mais un regroupement de plusieurs technologies, dans le but, au départ, d'améliorer l'utilisabilité et l'ergonomie d'un site Web - ou d'une application Web ; vu que l'on peut de plus en plus parler d'applications Web, avec ce genre de méthodes.
Cela dit, cela signifie que vous devez vous assurer que votre site soit toujours utilisable par vos visiteurs, et non uniquement par vous, qui le connaissez.

Depuis les débuts de la démocratisation d'Internet, il y a une dizaine d'années, les visiteurs de sites Web se sont habitués à un principe de navigation en "page par page". Cette idée correspond à plusieurs principes :

- Tout d'abord, lorsqu'un utilisateur clique sur un lien, il a l'habitude de voir la page entière du site disparaitre, et une autre page entière s'afficher - et non que seule une partie de la page soit mise à jour.
- Ce chargement est accompagné, au niveau du navigateur, par une animation qui indique à l'utilisateur que "quelque chose charge" ; en somme, il sait qu'il doit attendre, et que ce qu'il a demandé arrive : il n'a pas à se demander "est-ce que j'ai bien cliqué ?", puisqu'un élément, tel une icone animée ou une barre de chargement lui indique que a requête a été prise en compte.
  A vous, au sein de votre application, d'afficher un message du style "Chargement en cours...", pour que l'utilisateur ne se pose pas de questions. (Sous gmail, par exemple, notez le message en rouge, dans le coin supérieur droit de l'écran)
- Chaque page contient une idée ; et à chaque page est associée une adresse, unique. Autrement dit, à chaque idée correspond une adresse unique. Cela implique plusieurs choses :
  - Lorsqu'un utilisateur saisit une adresse (une URL - Uniform Ressource Location), il s'attend à arriver sur la même information que la personne qui lui a donné cette adresse.
- L'utilisateur peut mettre cette adresse en favoris (bookmark) ; et lorsqu'il cliquera sur ce favoris, ça le ramènera sur la même page, puisqu'une adresse ne correspond qu'à une seule page.
- Le visiteur a la possibilité d'utiliser les boutons précédent/suivant de son navigateur pour naviguer entre les pages qu'il a déjà vu.
  Puisqu'en utilisant AJAX on peut modifier le contenu d'une partie d'une page, cela signifie qu'une page ne contient plus une idée... Donc, à une adresse peuvent être associées plusieurs idées (plusieurs pages, pourrait-on presque dire).
  Les principes énoncés ci-dessus sont donc potentiellement cassés... Des solutions sont envisageables, et je ne peux que vous conseiller d'y réfléchir si vous avez l'intention de

développer une application sérieuse basée sur des fonctionnalités AJax.
En particulier, on peut penser à modifier l'URL au fur et à mesure de nos appels Ajax : en général, il est permis, en JavaScript, de modifier la partie variable de l'adresse ; ce qu'on appelle courramment l'ancre : http://www.monsite.com/mapage.html**#ancre**. Notez que je ne rentrerai pas dans la technique au cours de cet article, qui n'est destiné qu'à vous présenter les bases d'Ajax ; mais on peut trouver des informations à ce sujet sur plusieurs sites, tels celui-ci [en].

En somme, même si ajouter des fonctionnalités Ajax à votre site n'est pas une tâche bien difficile, vous devez prêter attention à vos utilisateurs, et à leurs habitudes de navigation, sinon, votre site risque de ne pas être facile à utiliser... et de perdre de l'intérêt auprès de vos visiteurs...
Et n'oubliez pas, selon le public que vous visez, qu'il peut être bon de respecter certains points en rapport avec l'accessibilité...

Enfin, pour faire un lien avec la partie suivante, qui parlera des problèmes plus techniques, si je puis dire, n'oubliez pas que Ajax se base sur du Javascript. Cela implique que pour pouvoir utiliser les fonctionnalités Ajax de votre site, vos visiteurs doivent, d'une part, utiliser un navigateur qui le supporte (autrement dit, pas un navigateur en mode texte), et, d'autre part, ne pas l'avoir désactivé.
Il en va de même pour l'objet XMLHtppRequest, qui impose aussi des contraintes en terme de navigateurs : il n'est pas présent sur les navigateurs les plus anciens, d'une part... Et, sous Internet Explorer, il s'agit d'un contrôle ActiveX ; ici encore, il ne faut donc pas que vos visiteurs aient désactivés, pour raison de sécurité, notamment, les contrôles ActiveX - sachant que seuls ceux utilisant Internet Explorer sont concernés par ce problème, les autres navigateurs intégrant cet objet en natif.

B: D'un point de vue plus technique

D'un point de vue plus technique, même si nous avons vu qu'un appel Ajax n'était pas difficile à écrire, plusieurs choses à noter ; certains points dont nous allons parler peuvent vous poser problème, d'autres ne sont regroupés ici qu'à titre d'information... Peut-être que certains points seront approfondis à l'avenir dans un autre article, d'ailleurs...

Nous avons vu juste au-dessus que le fait qu'une application Web Ajax soit construite en JavaScript peut poser problème du fait que l'utilisateur peut le désactiver dans son navigateur - si tant est qu'il utilise un navigateur qui le supporte.
J'ajouterais que le JavaScript est un langage **interprété côté client** ; si vous avez une application de quelques milliers de lignes en JavaScript, il y a fort à parier qu'elle sera horriblement lente - et nous savons tous à quel point les utilisateurs sont peu patients, surtout depuis la démocratisation des accès Internet haut débit...

Un point sensible lorsque l'on utilise l'objet XMLHttpRequest pour effectuer des requêtes asynchrones est que cet objet ne supporte qu'une seule requête à la fois ; dans l'application que nous avons développé plus haut, si vous cliquez une seconde fois sur le lien alors que la réponse à votre premier clic n'est pas encore arrivée, l'application risque de ne pas fonctionner correctement (Notamment, il est possible que le résultat ne soit jamais affiché, l'application ne présentant rien

d'autre que le message de chargemenent, et il est aussi possible, parfois au rechargement de la page, qu'une exception JavaScript soit lancée) ; au besoin, pour tester, n'hésitez pas à augmenter le temps d'attente dans le script qui répond aux requêtes.
Une solution est envisageable pour ce problème : avant d'accepter d'envoyer une requête avec un objet XMLHttpRequest, vérifiez son statut : s'il vaut 0 ou 4, c'est soit que l'objet n'a jamais envoyé de requête, soit qu'il a fini de traiter celle qui était en cours auparavant ; vous pouvez donc le réutiliser pour en envoyer une autre. Sinon, c'est qu'une requête est en cours, et, dans ce cas, mieux vaut ne pas réutiliser cette instance pour le moment.

Un autre problème est que l'objet XMLHttpRequest n'intègre pas de gestion des timeout.
Admettons que l'utilisateur de votre site envoit une requête AJAX à votre serveur, mais que celui-ci soit dans l'impossibilité de lui répondre - ou dans l'impossibilité de le faire dans des délais acceptables...
Si vous n'avez pas pensé à vous-même gérer cette situation, l'utilisateur de votre application peut se trouver face à un message de chargement qui ne disparaitra jamais... Alors qu'il serait bien plus adapté de l'informer, au bout de quelques secondes, qu'il est impossible de répondre à sa demande : au moins, il n'attendrait pas indéfiniment, et saurait que l'opération demandée n'est pas possible pour le moment. (Sans parler du fait qu'il ne se demanderait pas si sa requête a ou non été prise en compte ; après-tout, est-ce que le message de chargement qui ne disparait pas ne serait pas qu'un bug ?)
En somme, à vous d'appeler (grâce à la fonction setTimeout, par exemple), au bout de quelques secondes, une fonction qui annule la requête Ajax et affiche un message à l'utilisateur. Assurez-vous que l'annulation ne sera faite que si la requête était encore en cours ; et, si la requête réussi, pensez à annuler l'appel à la fonction d'annulation (avec clearTimeout), ou vous risquez d'annuler une requête future utilisant le même objet.

Dans le même esprit, les internautes ont pris l'habitude de pouvoir annuler le chargement d'une page si celui-ci prend trop de temps (ou tout simplement s'ils ont fait une erreur), en cliquant sur le bouton "Annuler" (ou équivalent) de leur navigateur.
Si vous voulez fournir ce genre de fonctionnalité à vos utilisateurs, ce sera à vous de l'implémenter ; d'une part, il vous faut un bouton ou un lien qui appelle une fonction JavaScript qui sera chargée de l'annulation. Et, d'autre part, il vous faut cette fonction ; prennez juste garde à ce que l'objet avec lequel vous allez travailler soit bien en train d'effectuer une requête (autrement dit, que son statut ne soit ni 0 ni 4), avant d'essayer d'annuler celle-ci.

Pour rentrer un peu plus dans les détails pour les deux derniers points que j'ai mis en évidence, je vous propose de lire cet article, intitulé Async Requests over an Unreliable Network [en], sur ajaxblog.com [en].

(J'ai l'impression que je ne vais jamais arriver au bout de liste de problèmes qui peuvent potentiellement se poser lorsque vous essayez d'ajouter des fonctionnalités Ajax à votre site...)

J'ai déjà évoqué ce point plus haut, mais je pense qu'il peut être bon d'en reparler : il n'est pas possible, en utilisant XMLHttpRequest, d'effectuer des requêtes vers un domaine autre que celui sur lequel votre application est hébergée.
Par exemple, si on remplace le http.open de l'application présentée en exemple plus haut par ceci :

```
http.open('get', 'http://www.google.fr/', true);
```

La console JavaScript affichera un message d'erreur, de la forme "Permission Denied to call method XMLHttpRequest.open" (Sous Firefox 1.5), ou un message d'avertissement de sécurité.
(A moins bien sûr que vous ne soyez développeur chez google ^^)
Cette limitation est probablement présente, à mon avis, pour des questions de sécurité... Même si je dois admettre que, parfois, son absence pourrait simplifier certaines tâches...

Certains vous diront qu'utiliser de l'Ajax au sein de votre site permet d'économiser la bande passante de votre serveur, étant donné qu'il n'aura pas à renvoyer les données de pages en entières aux clients, mais seulement les parties modifiées. Cela dit, les navigateurs intègrent généralement un système de cache ; grâce à lui, une partie des données de la page sont sauvegardées chez le client (les images, notamment), et n'ont donc pas à être rechargées depuis le serveur à chaque affichage de page. En somme, l'éconnomie de bande passante se fait sur le texte contenu dans les pages, ce qui ne représente généralement pas une économie énorme.
D'autre part, on peut avoir tendance, surtout lorsque l'on débutte en Ajax, à vouloir effectuer une requête pour chaque information de la page. Pour reprendre notre exemple d'un peu plus haut, on peut avoir une liste de liens, et, pour chacun d'entre eux, effectuer un appel Ajax au serveur pour savoir combien de fois il a été cliqué... Pourquoi ne pas effectuer une seule requête, qui nous permette d'obtenir le nombre de clics pour tous les liens en une seule fois, et ensuite, analyser ce résultat en JavaScript côté client, pour assigner à chaque lien le nombre de clics (par exemple, en utilisant du XML pour contenir les données) ? De la sorte, nous n'aurions qu'une seule requête.
Voici un exemple simple, mais qui illustre bien un des dangers qui se présente pour votre serveur si vous ne réfléchissez pas un peu avant de développer votre application : ajouter des appels Ajax à votre site, ça peut rendre service à vos utilisateurs, mais envoyer 10 fois plus de requêtes à votre serveur, ça va lui faire mal... et s'il devient surchargé, il aura du mal à répondre aux attentes des utilisateurs, ce qui n'est jamais bon...

C: Aller plus loin avec Ajax

Nous avons vu qu'il était extrêment simple d'ajouter des fonctionnalités Ajax de base à un site Web ; qu'il était un peu plus difficile (mais possible !) d'avoir des fonctionnalités Ajax robustes...
Maintenant, nous allons voir qu'il est possible d'aller un peu plus loin que ce que nous avons fait jusqu'à présent...

Tout d'abord, même si nous avons, dans nos exemples, principalement utilisé du texte en réponse à la requête Ajax, n'oublions pas que le X de AJA**X** signifie XML.
Cela signifie que le serveur peut renvoyer à l'application des données structurées, qui pourront être ensuite traitées côté client, grâce aux capacités qu'à le JavaScript de manipuler le DOM [fr].
Il suffit de récupérer ce XML à l'aide de la propriété `responseXML` de notre instance de XMLHttpRequest.
Cela dit, puisqu'il est possible de renvoyer du texte brut, nous pouvons en fait utiliser n'importe quel type de données, à partir du moment où celles-ci sont représentables sous forme textuelle... On peut par exemple penser à utiliser du JSON (JavaScript Object Notation) [en] à la place du XML ; si vous lisez l'article vers lequel je viens de vous proposer un lien, vous verrez que cela peut présenter des avantages, dans certains cas de figure.

Ecrire quelques appels Ajax à la main est faisable ; la preuve, nous l'avons fait un peu plus haut. Par contre, développer une application Web entière en partant de zéro n'est pas forcément chose aisée : d'une part, il convient de réfléchir à ce que l'on veut faire et comment, d'une part, mais, d'autre part, il faudra, au moment du développement en JavaScript, veiller à tester notre application sous le plus grand nombre de navigateurs grand-public possible.
En effet, il existe toujours quelques incompatibilités, que ce soit en terme de disponibilité de fonctionnalités, ou en terme de méthode d'accès à celles-ci ; nous en avons eu la preuve un peu plus tôt, lorsque nous avons du employer une méthode particulière sous Internet Explorer pour créer un objet XMLHttpRequest.
Pour vous faciliter la tâche, plusieurs collections de classes JavaScript sont disponibles. On parle parfois de Toolkits, Librairies, Bibliothèques ; certains parlent même de framework, ... Je n'en citerai que quelques unes ici, mais vous pouvez jeter un coup d'oeil sur le Toolkit Dojo (Notamment, à titre d'exemple, vous pouvez êtres intéressé par les tabs ou sur une barre des tâches, ou encore sur cet exemple traitant de manipulation de la sélection - n'ayant pas voulu pointer tous les exemples, je ne peux que vous conseiller d'aller voir les autres par vous-même : dans certains cas, ça vaut le déplacement !), ou sur la librairie open-source Rico, ou encore sur Sarissa. A vous de les essayer - ou d'en essayer d'autres - et de voir celle qui convient le mieux à vos besoins.

Liens en rapport avec cette page : Ajax Blog (en), Ajaxian (en), Ajax: A New Approach to Web Applications (en), Wikipedia : Ajax (en).