Le concept d'AJAX

Introduction

AJAX est l'acronyme d'*Asynchronous JavaScript And XML*, autrement dit *JavaScript Et XML Asynchrones*.

AJAX n'est ni une technologie ni un langage de programmation ; AJAX est un concept de programmation Web reposant sur plusieurs technologies comme le JavaScript et le XML – d'où le nom *AJAX*. A l'heure actuelle, le XML tend à être délaissé au profit du JSON, ce qui explique que certains puristes utilisent l'acronyme *AJAJ* – dont la prononciation laisse plutôt à désirer.

L'idée même d'AJAX est de faire communiquer une page Web avec un serveur Web sans occasionner le rechargement de la page. C'est la raison pour laquelle JavaScript est utilisé, car c'est lui qui va se charger d'*établir la connexion* entre la page Web et le serveur.

Contrairement à ce qui est souvent dit, le principe de fonctionnement d'AJAX a toujours existé, et ce par le biais de certaines astuces JavaScript, comme l'ajout d'un élément `<script />` après le chargement de la page. Mais il a fallu attendre l'arrivée de l'objet XMLHttpRequest pour que l'utilisation de l'AJAX se démocratise. L'objet XMLHttpRequest est un objet natif JavaScript, développé à l'origine en tant qu'ActiveX dans Internet Explorer, qui facilite grandement la communication JavaScript – Serveur.

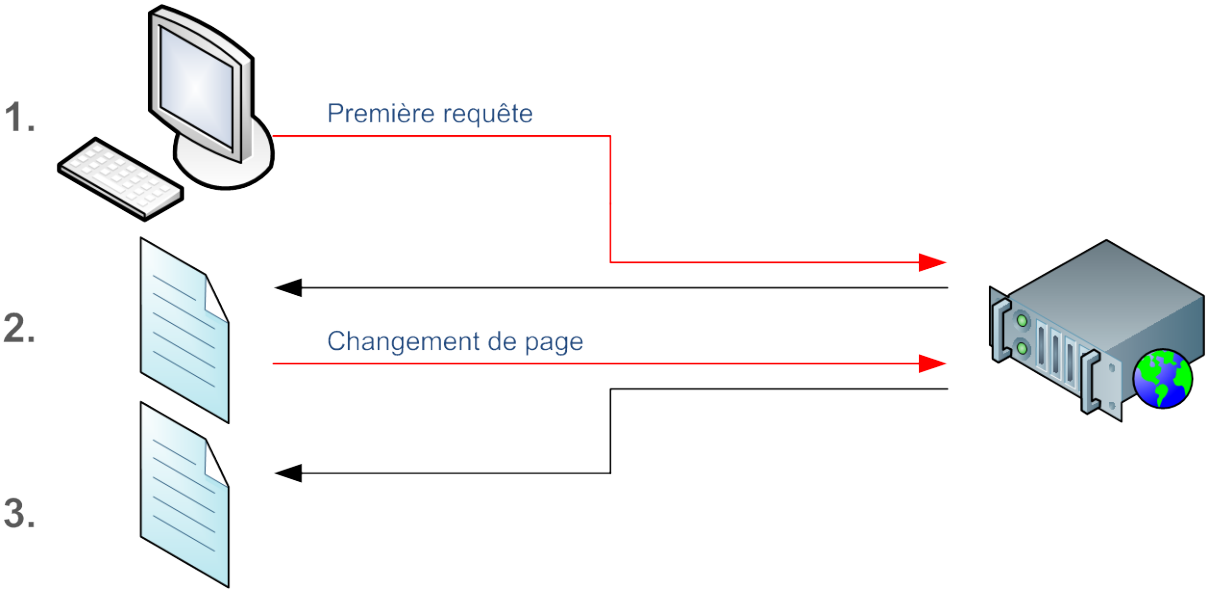
Afin de bien introduire le principe de l'AJAX, voici une comparaison du fonctionnement d'un site Web dit *traditionnel* et d'une application Web mettant en œuvre AJAX.

Page Web et Application Web

Différencions tout d'abord deux côtés applicatifs : le navigateur, à gauche, et le serveur, à droite. A ce petit schéma on va ajouter un pendule qui balance entre les deux côtés.

Site Web traditionnel

Tout d'abord le navigateur envoie une requête – via une URL – au serveur. Le serveur répond en renvoyant au navigateur le code HTML de la page ainsi que tout ce qui lui est associé comme les scripts JavaScript, les images ou les éventuels médias et autres objets embarqués – donc la réponse du serveur est beaucoup plus volumineuse que la requête. Le navigateur affiche la page et l'utilisateur peut la parcourir quelques instants avant de cliquer sur un lien hypertexte qui enverra une nouvelle requête au serveur qui lui-même renverra le HTML correspondant... et ainsi de suite.



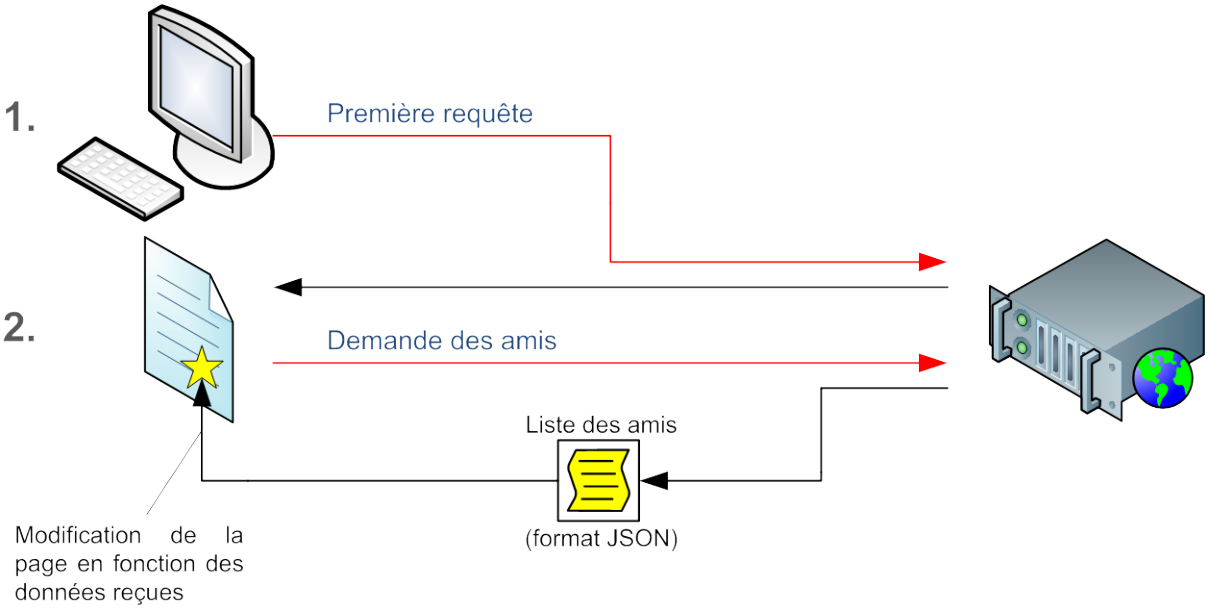
D'un point de vue purement pratique, c'est assez aberrant comme principe car c'est extrêmement inefficace et couteux puisque le serveur va à chaque fois renvoyer tout, ce qui prend du temps et ce qui occasionne une charge pour le serveur.

Ce principe de fonctionnement montre que le navigateur n'intervient pas dans le processus si ce n'est que pour afficher la page. Le gros du travail se fait du côté du serveur. Le pendule balance donc du côté du serveur.

Application AJAX

Voyons ce même schéma du point de vue d'AJAX. Quand on utilise le concept d'AJAX dans une page Web, on parle d'application Web (ou application AJAX).

La première requête est la même – ça on ne sait rien y faire. La différence va résider dans le fait que quand l'utilisateur cliquera sur un lien – ou un autre élément cliquable – la page ne se rechargera pas et le navigateur enverra une requête au serveur, lequel renverra les données demandées dans un format léger – comme le format JSON. Dans ce cas, le serveur n'a renvoyé qu'un minimum de données ce qui est beaucoup plus léger et donc plus rapide. Le navigateur, par le biais de JavaScript, peut alors mettre à jour une petite partie de la page avec les données reçues du serveur.



Dans ce cas, le pendule est au centre. Le serveur et le navigateur travaillent pour proposer une solution optimale. C'est l'exemple parfait de l'équilibre d'une application AJAX

Il faut cependant faire attention à ne pas mésinterpréter ce schéma, car il ne s'agit en aucun cas de remplacer la page entière, ce qui reviendrait à recharger la page. Si on se réfère au pendule, il va balancer du côté du serveur, ce qui montre qu'il y a un problème, et dans ce cas AJAX n'est pas utilisé correctement.

C'est donc la raison pour laquelle il est totalement absurde de créer un site Web entièrement en AJAX. Ce n'est pas possible car AJAX ne doit jamais se subtiliser à la méthode traditionnelle. AJAX doit être utilisé pour charger/modifier de petites parties d'une page. Un exemple peut être Facebook ; sur Facebook il y a la possibilité d'afficher les amis d'une personne sans recharger la page. Quand vous cliquez sur le lien *Afficher les amis*, une requête est envoyée au serveur, lequel renvoie la liste des amis de la personne dont vous souhaitez connaitre les amis. Cette liste est alors affichée dynamiquement via JavaScript. Pas de rechargement de page, juste une requête rapide – quasi instantanée – et un confort de navigation accru.

Dialoguer avec le serveur

Si le pendule est bien au centre, l'application est en équilibre et le dialogue entre le navigateur et le serveur se passe bien. Mais comment dialoguer ? Ou plutôt, en quel format parler ?

Le navigateur et le serveur ne peuvent se parler que via un format de type *texte brut*. Le navigateur peut donc demander au serveur "Liste amis Alyx" et le serveur renverra la liste des amis d'Alyx. C'est assez nul comme format : si la question est claire, la liste des amis que renverra le serveur le sera nettement moins et c'est ici qu'il va falloir opter pour le format adéquat. Plusieurs formats sont possibles :

- Texte simple ;
- HTML ;
- XML ;

- JSON.

Le texte simple

Comme on vient de le voir, ce format n'est pas du tout adapté s'il s'agit de recevoir des données devant êtres "formatées" ou classées comme une liste d'amis.

Le HTML

Le HTML est intéressant car il suffira de l'insérer directement dans la page avec la propriété `innerHTML`. C'est rapide. Cela dit le HTML est verbeux et peut se révéler assez lourd quand il s'agit d'un grand volume de données.

```
<ul>
        <li><span title="a4242">Gordon</span></li>
        <li><span title="j3781">Barney</span></li>
        <li><span title="j7638">Eli</span></li>
        <li><span title="o7836">Chell</span></li>
        <li><span title="e5831">Odessa</span></li>
</ul>
```

Le XML

Avec certains objets AJAX, comme `XMLHttpRequest` il est possible de récupérer du texte sous forme de XML et de l'interpréter comme tel, ce qui permet de manipuler les données avec les fonctions DOM. Ca peut être intéressant mais XML souffre du même problème que le HTML : il est verbeux. De plus le traitement via DOM peut se révéler assez lent s'il s'agit d'un grand volume de données et suivant le navigateur utilisé par le client.

```
<friends>
        <f name="Gordon" id="a4242" />
        <f name="Barney" id="j3781" />
        <f name="Eli" id="j7638" />
        <f name="Chell" id="o7836" />
        <f name="Odessa" id="e5831" />
</friends>
```

Le JSON

Reste le JSON. Le JSON est une manière de structurer l'information en utilisant la syntaxe objet de JavaScript – des objets et des tableaux. JSON est très léger, car non-verbeux mais nécessite d'être évalué par le compilateur JavaScript pour pouvoir être utilisé comme un objet. L'évaluation se fait via `eval` pour les navigateurs obsolètes ou via la méthode `parse` de l'objet natif `JSON`. L'évaluation est souvent décriée car peut se révéler dangereuse, mais dans la mesure où vous connaissez la source des données à évaluer il n'y a pas de danger.

```
[
        { "name":"Gordon", "id":"a4242" },
        { "name":"Barney", "id":"j3781" },
        { "name":"Eli", "id":"j7638" },
        { "name":"Chell", "id":"o7836" },
        { "name":"Odessa", "id":"e5831" }
]
```

Le JSON est donc le format travaillant de paire avec AJAX quand il s'agit de recevoir des données classées et structurées. Les autres formats peuvent bien évidemment servir et se révéler intéressants dans certains cas, mais d'une façon générale les grandes pointures du JavaScript, comme Douglas Crockford incitent à utiliser JSON.

La librairie json2, écrite par Doug Crockford, permet d'émuler le comportement de l'objet natif `JSON` s'il n'est pas pris en charge par le navigateur. La librairie crée un objet global `JSON` pourvu des méthodes `parse` et `stringify`. La méthode `parse` est en fait un `eval` sécurisé, c'est-à-dire qu'un traitement est fait sur la chaine à évaluer pour s'assurer qu'elle ne présente aucun danger.

Une démonstration de cette petite application peut être découverte ici. Il ne s'agit que de HTML statique bien évidemment, cela sert juste à illustrer la théorie.

Principes synchrones et asynchrones

La principale particularité d'AJAX est l'asynchronisme : la fonction qui envoie une requête au serveur n'est pas la même que celle qui en recevra la réponse. Avant d'aborder la pratique d'AJAX, il est bon de bien cerner cette notion d'asynchronisme qui est très importante.

Quand un programme ou un script s'exécute, il appelle les différentes instructions dans l'ordre dans lequel elles sont placées :

```
var plop = 0; // première instruction
plop += 2;    // deuxième
alert(plop);  // et troisième
```

Maintenant, imaginons qu'il y ait un appel de fonction :

```
var plop = 0;                 // première instruction
plop = additionner(plop, 2); // deuxième
alert(plop);                  // et troisième
```

Quand la fonction `additionner()` est appelée, le script principal se met en pause, et attend que la fonction soit exécutée, et qu'elle ait renvoyé ou non une valeur .

On dit que le script est exécuté de façon **synchrone** : quand un appel externe au script principal est réalisé, le script en attend la réponse ou la fin de l'exécution.

Le contraire de synchrone est **asynchrone**. Quand un appel est asynchrone, le script principal n'attend pas d'avoir reçu les données pour continuer. Evidemment, si mon exemple synchrone marche bien avec des fonctions, il ne marche pas si le script est asynchrone ; imaginons donc une requête de type AJAX !

Le script s'exécute et rencontre une requête AJAX, laquelle est envoyée en mode asynchrone. Dans ce cas, la requête est envoyée, mais le script n'attend pas que la requête ait abouti, il continue quoiqu'il arrive. L'intérêt est que si la requête met quelques secondes à être traitée par le serveur, le script n'est pas ralenti.

Mais si la requête est envoyée et que le script n'attend pas sa réponse, comment savoir quand cette requête renvoie quelque chose ?

Bonne question. Et c'est ici qu'interviennent les fonctions dites de *callback*. Une fonction callback est exécutée quand la requête aboutit à quelque chose, que son traitement est fini. Et c'est cette fonction de callback qui va se charger de récupérer les données renvoyées par la requête.

Ainsi, quand la requête est envoyée, le script continue son exécution, et quand la requête renvoie quelque chose, c'est la fonction de callback qui est appelée, et c'est elle qui va faire "suite" au script principal, en traitant les informations renvoyées.

On peut résumer l'asynchronisme en AJAX par ce schéma :