Chapitre 7

AJAX

Introduction

AJAX : Asynchronous JavaScript and XML

Ajax permet de modifier partiellement la page affichée par le navigateur pour la mettre à jour sans avoir à recharger la page entière. Par exemple le contenu d'un champ de formulaire peut être changé, sans avoir à recharger la page avec le titre, les images, le menu, etc.

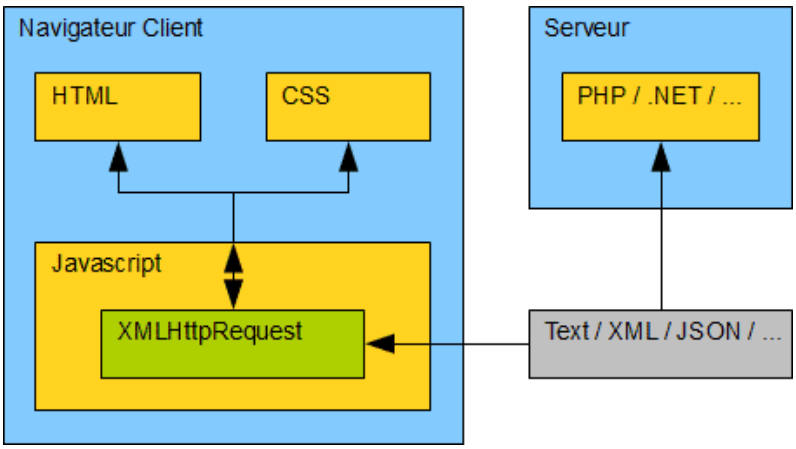
Le terme "Asynchronous", signifie que l'exécution de JavaScript continue sans attendre la réponse du serveur qui sera traitée quand elle arrivera. Tandis qu'en mode synchrone, le navigateur serait gelé en attendant la réponse du serveur.

1. Technologie du "langage" AJAX

Comme son nom l'indique, la mise en place d'AJAX requiert plusieurs langages :

- HTML pour l'interface.
- CSS (Cascading Style-Sheet) pour la présentation de la page
- **Javascript**, pour les traitements côté client ;
- L'objet **XMLHttpRequest** de Javascript lit des données ou fichiers sur le serveur de façon asynchrone.
- **PHP**, ou tout autre langage *"server-side"*, pour les traitements côté serveur
- **XML et JSON**, qui peuvent servir de formats d'échanges standardisés entre le client (Javascript) et le serveur (PHP...)

Pour résumer le processus AJAX



1. Javascript passe une demande à l'objet XMLHttpRequest ;
2. L'objet XMLHttpRequest requête le serveur afin de lui transmettre ou de lui demander des données ;
3. Le serveur retourne un résultat (texte, XML, JSON...) à l'objet XMLHttpRequest ;
4. L'objet XMLHttpRequest retourne le résultat à Javascript ;
5. Javascript traite les données retournées (seconde requête AJAX, mise à jour tout ou partie du contenu de la page via HTML et CSS...)

Fonctionnement

Pour recueillir des informations sur le serveur l'objet XHR dispose de deux méthodes:

- **open**: établit une connexion.
- **send**: envoie une requête au serveur.

Les données fournies par le serveur seront récupérées dans les champs de l'objet XMLHttpRequest:

- **responseXml** pour un fichier XML ou
- **responseText** pour un fichier de texte brute.

Il faut attendre la disponibilité des données, et l'état est donné par l'attribut **readyState** de XMLHttpRequest.

Les états de **readyState** sont les suivants :

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

L'objet XMLHttpRequest

Elle permet d'interagir avec le serveur, grâce à ses méthodes et ses attributs.

- Attributs

readyState	le code d'état passe successivement de 0 à 4 qui signifie "prêt".
status	200 est ok 404 si la page n'est pas trouvée.
responseText	contient les données chargées dans une chaîne de caractères.
responseXml	contient les données chargées sous forme XML
onreadystatechange	propriété activée par un évènement de changement d'état. On lui assigne une fonction.

Méthodes

open(mode, url, boolean)	mode: type de requête, GET ou POST url: l'endroit où trouver les données, un fichier avec son chemin sur le disque. boolean: true (asynchrone) / false (synchrone). en option on peut ajouter un login et un mot de passe.
send("chaine")	null pour une commande GET.

Construire une requête, pas à pas

Première étape: créer une instance

C'est juste une instance de classe classique mais deux options à essayer pour compatibilité avec les navigateurs.

```
 if (window.XMLHttpRequest)    //  Objet standard
  {
   xhr = new XMLHttpRequest();     //  Firefox, Safari, ...
   }
   else  if (window.ActiveXObject)      //  Internet Explorer
   {
   xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

ou plus simplement, on peut utiliser les exceptions:

```
   try
   {
     xhr = new ActiveXObject("Microsoft.XMLHTTP");      // Essayer IE
   }
     catch(e)                // Echec, utiliser l'objet standard
{
 xhr = new XMLHttpRequest();
}
```

Seconde étape: attendre la réponse

Le traitement de la réponse et les traitements qui suivent sont inclus dans une fonction, et la valeur de retour de cette fonction sera assignée à l'attribut **onreadystatechange** de l'objet précédemment créé.

```
xhr.onreadystatechange = function()

{

 // instructions de traitement de la réponse

}

if (xhr.readyState == 4)

{

// Reçu, OK

}

else

{

// Attendre...

}
```

Troisième étape: faire la requête elle-même

Deux méthodes de XMLHttpRequest sont utilisées:
- **open**: commande GET ou POST, URL du document, true pour asynchrone.
- **send**: avec POST seulement, données à envoyer au serveur.

La requête ci-dessous lit un document sur le serveur.

```
xhr.open('GET', 'https://www.xul.fr/fichier.xml', true);

xhr.send(null);
```

Exemple de Programme Ajax : Exemple1.html

```
<html>
<head>
<script>
function submitForm()
{
   var xhr;
   try {  xhr = new ActiveXObject('Msxml2.XMLHTTP');   }
   catch (e)
   {
      try {   xhr = new ActiveXObject('Microsoft.XMLHTTP'); }
      catch (e2)
      {
        try {  xhr = new XMLHttpRequest();  }
        catch (e3) {  xhr = false;   }
       }
   }
   xhr.onreadystatechange  = function()
   {
     if(xhr.readyState  == 4)
     {
      if(xhr.status  == 200)
          document.ajax.dyn="Received:"  + xhr.responseText;
      else
```

```
            document.ajax.dyn="Error code " + xhr.status;
        }
    };

    xhr.open("GET", "data.xml",  true);
    xhr.send(null);
}
</script>
</head>
<body>
    <FORM method="POST" name="ajax" action="">
        <INPUT type="BUTTON" value="Submit"  ONCLICK="submitForm()">
        <INPUT type="text" name="dyn"  value="">
    </FORM>
</body>
</html>
```

Commentaires sur le code:

Commentaires sur le code:

new ActiveXObject(Microsoft.XMLHTTP)
Ce constructeur est pour Internet Explorer.

new XMLHttpRequest()
Ce constructeur est pour tout autre navigateur incluant Firefox.

http.onreadystatechange
On associe un traitement (une fonction anonyme en l'occurrence) à cet indicateur d'évènement.

http.readyState == 4
L'état 4 signifie que la réponse est envoyée par le serveur et disponible.

http.status == 200
Ce status signifie ok, sinon un code d'erreur quelconque est envoyé, 404 par exemple.

h**ttp.open("POST", "data.xml", true);**
- POST ou GET
-url du fichier.
- true pour asynchrone (false pour synchrone).

http.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Cette méthode s'utilise seulement avec POST.

http.send(document.getElementById("TYPEDTEXT").value);
Envoi des données au serveur. Les données sont prises dans le champ dont l'id est "TYPEDTEXT" et qui est rempli par l'utilisateur grâce à un formulaire.

Démonstration

Reçu:

Lire dans un fichier XML

Pour lire des données dans un fichier XML il suffit de remplacer la ligne:

1document.ajax.dyn="Reçu: " + xhr.responseText;

par ce code:

```
//  Assigner le fichier XML à une variable
var doc = xhr.responseXML;
//  Lire le premier élément avec DOM
var element = doc.getElementsByTagName('root').item(0);
//  Copier le contenu dans le formulaire
document.ajax.dyn.value= element.firstChild.data;
```

Démonstration

Reçu:

Ecrire dans la page

Le texte récupéré est inscrit dans le corps de la page et non dans un champ de texte. Le code ci-dessous remplace l'objet de formulaire textfield et la seconde partie remplace l'assignement dans la fonction JavaScript.

... un texte à remplacer...

document.getElementById("zone").innerHTML = "Received:" + xhr.responseText;

Démonstration

Attendre...

Envoyer un texte

Un texte est envoyé au serveur et sera écrit dans un fichier. L'appel de la méthode "open" change, le premier argument est POST, le second est le nom du fichier ou du script qui recevra les données et doit les traiter. Et la méthode "send" aussi a pour argument une valeur qui est une chaîne de paramètres.

xhr.open("POST", "ajax-post-text.php", true);

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send(data);

L'argument de send est au format des paramètres de la méthode POST. S'il y a plusieurs données, on les sépare par le symbole "et" commercial:

var data = "file=" + url + "&content=" + content;

Le paramètre "file" est un fichier qui contiendra le contenu "content". Le nom du fichier doit être vérifié par le serveur pour éviter qu'un autre fichier ne puisse être modifié.

Une démonstration de POST est incluse dans l'archive.

Utiliser un fichier externe

Il est plus simple d'inclure un fichier JavaScript. Cette ligne sera incluse dans la section head de la page HTML:

1<script src="ajax.js" type="text/javascript"></script>

Et la fonction est appelée avec cette instruction:

1var xhr = createXHR();

Le script du fichier externe:

```
function createXHR() {
   var request = false;
   try    {
     request = new ActiveXObject('Msxml2.XMLHTTP');
   }
   catch (err2)    {
   try {
      request = new ActiveXObject('Microsoft.XMLHTTP');
   }
   catch (err3) {
      try {  request = new XMLHttpRequest();}
      catch (err1) { request = false;}
    }
   }
  return request;
}
```

Exemple d'un script AJAX

Voici un petit exemple concret :

Au chargement de la page, **on demande au serveur de nous retourner le résultat d'un script PHP, puis on l'affiche dans un élément HTML**. Si la requête AJAX échoue, on affiche le texte "Error..." dans l'élément.

Fichier PHP

```
/*** Fichier PHP (ajax1.php)
* Notre script va prendre un élément aléatoirement
* dans le tableau et l'afficher.*/

<?php
$somebody = array('World', 'Foobar', 'John Doe');
shuffle($somebody);
printf('Hello %s!', $somebody[0]);
?>
```

Fichier HTML

```
/*** HTML ***/
<div id="ajaxBox"></div>
```

Fichier HTML

```
/*** Javascript ***/

// Au chargement de la page, déclencher la fonction ajaxBox_update()

window.onload = ajaxBox_update;

// Fonction effectuant la requête AJAX

function ajaxBox_update()
{
// Récupération de l'objet XHR
```

```
var xhr = getXhr();

// On assigne une fonction qui, lorsque l'état de la requête change, va traiter le résultat


xhr.onreadystatechange = function()
{


// readyState 4 = requête terminée
if (xhr.readyState == 4)
{


// status 200 = page requêtée trouvée


if (xhr.status == 200)
ajaxBox_setText(xhr.responseText);


// Page non trouvée
else
ajaxBox_setText('Error...');
}
};

// Préparation et éxecution de la requête
var url = 'http://site.com/script.php';
xhr.open('GET', url, true);
xhr.send(null);
}

// Fonction de mise à jour du contenu de la div #ajaxBox
// Cette fonction ajoute un element <p> contenant le message, dans le div #ajaxBox

function ajaxBox_setText(pText)
{
var elt = document.getElementById('ajaxBox');
var p = elt.appendChild(document.createElement('p'));
p.appendChild(document.createTextNode(pText));
}
```

Libre à vous donc de lier, sur différents évènements, des appels AJAX et leurs traitements spécifiques.

AJAX (Asynchronous JavaScript and XML) est un moyen d'envoyer des données au serveur et recevoir une réponse sans recharger la page. C'est ce qu'on appelle. AJAX n'est pas un langage de programmation, mais un concept de communication avec un serveur.

À un certain stade du script côté client (JS), une fonction est appelée qui envoie une requête au serveur et spécifie le script à exécuter côté serveur (PHP). Sur le serveur, un script est exécuté, selon les résultats du travail, le client reçoit une réponse et le script du côté client travaille avec les données reçues. AJAX simplifie grandement le fonctionnement de l'application Web. Par exemple, vous devez envoyer des données de formulaire. Si cela est fait de la manière habituelle, après avoir soumis le formulaire, il sera nécessaire d'accepter la réponse du serveur et de télécharger un nouveau document dans la fenêtre. **Lorsque vous utilisez AJAX, le nouveau document ne se charge pas, ce qui signifie que la quantité de données peut être considérablement réduite**. Ceci est particulièrement important pour les appareils mobiles qui utilisent des communications mobiles et qui sont limités dans les débits de transfert de données.

L'élément span dans la page HTML permet la mies à jour sans recharger la page

```
<span id="ajax_test">Click pour voir !</span>
```

La principale chose à comprendre est qu'après avoir envoyé la requête, nous attendrons la réponse du serveur. Et cela signifie que le script exécutable doit générer ces données

Créer un fichier **script.php** et écrivez une instruction simple:

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

Il est absolument évident que si vous accédez explicitement à ce script, vous obtiendrez du texte à l'écran **Hello**. C'est le corps de la réponse du serveur.

Essayons maintenant ce script avec AJAX.

Tout d'abord, créez un script JS qui gère un clic sur un élément #ajax_test

```
<script>
        $('#ajax_test').click(function(){

// something

        });
</script>
```

Il reste maintenant à appliquer une des fonctions de jQuery pour travailler avec AJAX. Dans cet exemple, nous utilisons la fonction $.get() - il enverra une requête GET à la ressource spécifiée. Dans notre cas, il s'agit du fichier **script.php** .

```
<script>
   $('#ajax_test').click(function(){

      $.get('script.php', function(data) {
         $('body').append(data);
      })

   });
</script>
```

Ici, le premier argument est notre script, et le second est une fonction anonyme qui dépend de la variable qui contiendra le corps de la réponse du serveur. Dans le corps même de la fonction, nous ajoutons simplement le contenu résultant au modèle objet (méthode append()).

En conséquence - après chaque clic, un nouveau mot apparaîtra sur la page **Hello**.

1. Exemple1 d'AJAX, avec JS pur et avec jQuery.

//Index.html

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <script src='jquery-V-3.3.1.js'></script>
    <title>AJAX</title>
</head>
<body>
<span id="ajax_test">Click pour voir!</span> <br/>
<script>
    $('#ajax_test').click(function(){
        $.get('script.php', function(data) {
            $('body').append(data);
        })
    });
</script>
</body>
</html>
```

//Script.php

```
<?php
echo ' vous qui a fait un click Bonjour mon ami';
?>
```

2. Implémentation des requêtes JavaScript asynchrones.

XMLHttpRequest est une classe par laquelle des requêtes HTTP synchrones et asynchrones peuvent être implémentées.

AJAX nous permet les demandes asynchrones.

Créer une instance de cette classe :

```
XMLHttp = new XMLHttpRequest();
```

Puis dans la variable XMLHttp nous aurons un objet qui contient des méthodes pour gérer les requêtes asynchrones.
Il convient de noter que dans les versions antérieures d'Internet Explorer, cette classe doit être créée en utilisant **ActiveXObject** :

```
XMLHttp = new ActiveXObject('Microsoft.XMLHTTP');
```

Cela signifie que dans un script qui instancie une classe **XMLHttpRequest**, vous devez vérifier s'il existe une propriété dans le modèle objet du navigateur **XMLHttpRequest**. Si c'est le cas (dans la plupart des cas), nous utilisons la première option. Sinon, nous utilisons la deuxième option. Vous pouvez obtenir une instance de cette classe en tant que fonction:

```
function getXMLHttpRequest()
{
  if (window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }

  return new ActiveXObject('Microsoft.XMLHTTP');
}
```

Envisagez d'utiliser une classe XMLHttpRequest. Pour déterminer les paramètres de la requête HTTP, vous devez utiliser open(method, url, if Async). Cette méthode utilise trois paramètres:

- (chaîne) method - Méthode de protocole HTTP;
- (chaîne) url - URL de la ressource
- (bool) if Async - paramètre qui détermine si la requête sera asynchrone (true si oui).

Cette méthode n'envoie pas encore la requête elle-même, mais prépare tout pour l'envoyer. Pour envoyer la demande elle-même, vous devez utiliser le send(content), qui dépend d'un paramètre de chaîne - le corps de la requête. Pour les requêtes GET, le corps est vide, dans ce cas, vous pouvez le remplacer null.

Considérez l'application de ces méthodes. Nous utilisons la fonction getXMLHttpRequest()

```
request = getXMLHttpRequest();
request.open('GET', 'http://google.com', true);
request.send(null);
```

Ensuite, vous devez suivre le changement dans le statut de la demande. En d'autres termes - si la demande a été envoyée, si les données ont été reçues. L'état actuel correspond à la propriété readyState

- 0 - demande n'a pas été initialisée
- 1 - Demande ouvert
- 2 - envoi de données
- 3 - réception de données
- 4 - Les données obtenues
  Gardez une trace de l'état actuel de la propriété onreadystatechange, qui doit être assignée à une fonction anonyme qui sera appelée avec chaque changement d'état:

```
request.onreadystatechange = function() {
    // some actions
}
```

Essayons de suivre les changements dans le statut de la requête. Pour ce faire, créez un autre fichier file.txt. Dans celui-ci, vous pouvez écrire une chaîne arbitraire, par exemple Bonjour tout le monde. Écrivons le script dans le fichier index.html

```
request = getXMLHttpRequest();

request.onreadystatechange = function() {
    console.log(request.readyState);
}

request.open('GET', 'file.txt', true);
request.send(null);
```

Dans ce cas, nous avons envoyé une requête asynchrone pour obtenir le contenu du fichier **file.txt** .

Exemple 3

- **readyState** - contient l'état actuel de la requête
- **responseText** - contient le corps de la réponse

```
<!DOCTYPE html>
<html>
<head lang="en">
   <meta charset="UTF-8">
   <title></title>
   <script src="scripts.js"></script>
</head>
<body>

Testing ...

<script>

   request = getXMLHttpRequest();

   request.onreadystatechange = function() {
```

```
        if (request.readyState == 4) {
            alert(request.responseText);
        }
    };

    //request.open('GET', 'some_file.txt', true);
    request.open('GET', 'test.php', true);

    request.send(null);

</script>

</body>
</html>
```

Créer un fichier **test.php** , dans lequel nous écrivons le code suivant:

```
<?php
$a = 2 + 2 * 2;
echo $a;
?>
```

```
//Script.js

function getXMLHttpRequest()

{

    if (window.XMLHttpRequest) {

        return new XMLHttpRequest();

    }

    return new ActiveXObject('Microsoft.XMLHTTP');

}
```

Exemple 4 :

Maintenant, essayez de passer la chaîne en tant que corps de réponse du serveur JSON. Pour cela, nous avons besoin de la fonction PHP json_encode($data) qui prend comme argument tout type de données. Par exemple, un tableau. Définir un tableau dans le fichier test.php $data et affiche la chaîne JSON de ce tableau.

test.php

```
<?php
$data = array(
   'name' => 'Moussa',
   'email' => 'moussa@gmail.com',
   'phone' => '123-45-67',
   'message' => 'Bonjour je veux acheter un ballon.'
);

echo json_encode($data);
```

Pour vérification, vous pouvez directement accéder à ce fichier dans votre navigateur. Ensuite, nous obtenons une ligne sur l'écran

{"nom": "Steven", "email": "moussa@gmail.com", "téléphone": "123-45-67", "message": "Je veux acheter de l'herbe. "}

Il reste à corriger le script JS. Nous utilisons l'objet JSON pour extraire des données de la chaîne passée.

```
request = getXMLHttpRequest();


request.onreadystatechange = function() {
if (request.readyState == 4) {
responseBody = request.responseText;
 data = JSON.parse(responseBody);
console.log(data);
alert(data['message']);
}
};
```

request.open('GET', 'test.php', true);
request.send(null);

Après avoir obtenu le corps de la réponse, nous avons converti la chaîne JSON en un objet JS. Ensuite, les informations de sortie de la console à ce sujet, et dans la fenêtre modale, le texte de la propriété **message.**
Il est nécessaire de comprendre que dans le script test.php nous avons le droit d'écrire n'importe quel code PHP, si seulement cela a fonctionné et de produire des données, par exemple, en utilisant echo.

index.html

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
    <script src="scripts.js"></script>
</head>
<body>

Testing ...

<script>

    request = getXMLHttpRequest();

    request.onreadystatechange = function() {
        if (request.readyState == 4) {
            responseBody = request.responseText;
            data = JSON.parse(responseBody);
            console.log(data);
```

```
        alert(data['message']);
    }   };
  request.open('GET', 'test.php', true);
  request.send(null);

</script>
</body>
</html>
```

Test.php

```
<?php
$data = array(
  'name' => 'Moussa',
  'email' => 'moussa@gmail.com',
  'phone' => '0600-45-67-22',
  'message' => 'Bonjour  je veux acheter votre maison.'
);
echo json_encode($data);
?>
```

Même script.js

Exemple 5 : Bases de données

La base de données en utilisant une requête asynchrone.

Pour obtenir des données de la base de données, créez une nouvelle table **tabMessage** et écrivez une ligne dedans, par exemple. En tant que base de données, nous utiliserons mabase.

la table (id, name, email, phone, message) avec Id autincrement

```
CREATE TABLE IF NOT EXISTS tabMessage (
 id smallint(5) unsigned NOT NULL,
name varchar(50) NOT NULL,
 email varchar(50) NOT NULL,
```

```
    phone char(15) DEFAULT NULL,
    message text NOT NULL
    ) ENGINE=InnoDB;

INSERT INTO tbl_message (name, email, phone, message) VALUES
('Bobby', 'prof@gmail.com', '0666-45-64-34', 'Hello. Je veux apprendre.');
```

connecter à la base de données et récupérer les données de la table.

Nous écrivons les instructions correspondantes dans test.php

```
//test.php
<?php
$msg_info = array();
$status = 'Success';
try {
    $dbh = new PDO('mysql: host=localhost; dbname=mabase', 'root', '');
    $dbh->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $sth = $dbh->query('SELECT * FROM tabMessage');
    $msg_info = $sth->fetch(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    $status = 'Fail: ' . $e->getMessage();
}
$data = array(
    'msg_info' => $msg_info,
    'status' => $status
);
echo json_encode($data);
?>
//index.html
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
```

```
    <script src="scripts.js"></script>
</head>
<body>
Testing ...
<script>
    request = getXMLHttpRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4) {
            responseBody = request.responseText;
            data = JSON.parse(responseBody);
            if (data['status'] == 'Success') {
                for (key in data['msg_info']) {
                    document.write(key + ': ');
                    document.write(data['msg_info'][key]);
                    document.write('<br>');
                }
            } else {
                document.write(data['status']);
            }

            window.stop();
        }
    };
    request.open('GET', 'test.php', true);
    request.send(null);
</script>
</body></html>
```