

AJAX

Préparé par : Dr.Neila Ben Lakhel | Enicarthage
E-mail : neila.benlakhel@gmail.com

AJAX c'est Quoi déjà?

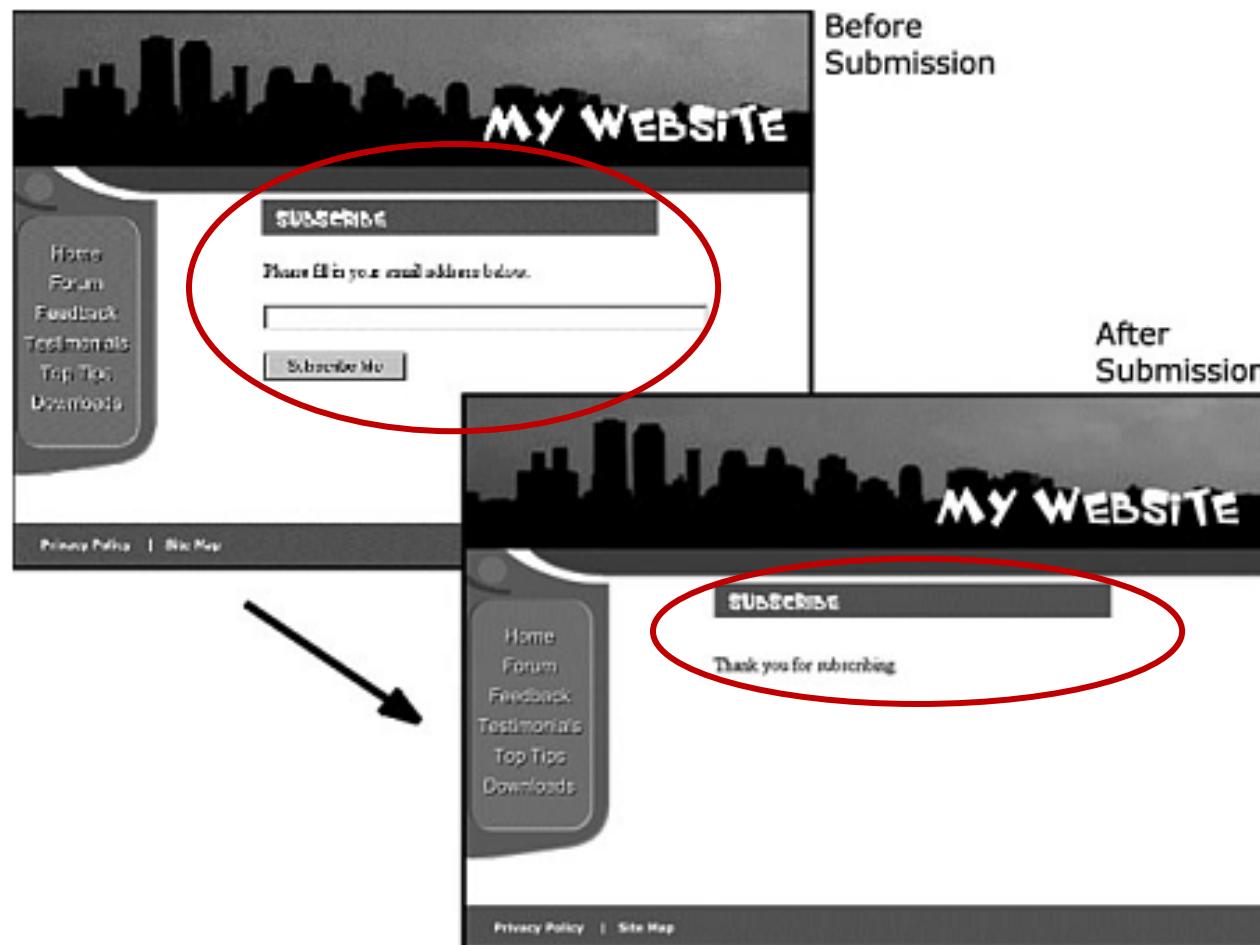
■ Ajax: Asynchronous JavaScript And XML

- N'est pas un nouveau Langage de programmation
- Mais une façon particulière d'utiliser JavaScript
- Permet de télécharger des données en arrière plan du serveur
- Permet la MAJ dynamique du contenu des pages Web sans faire attendre l'utilisateur
- Le mode habituel de fonctionnement du WEB: "click-wait-refresh" n'est plus !!!

- Exemples d'applications utilisant AJAX: Google Maps, Google Suggests, Gmail, Youtube, Facebook tabs etc...

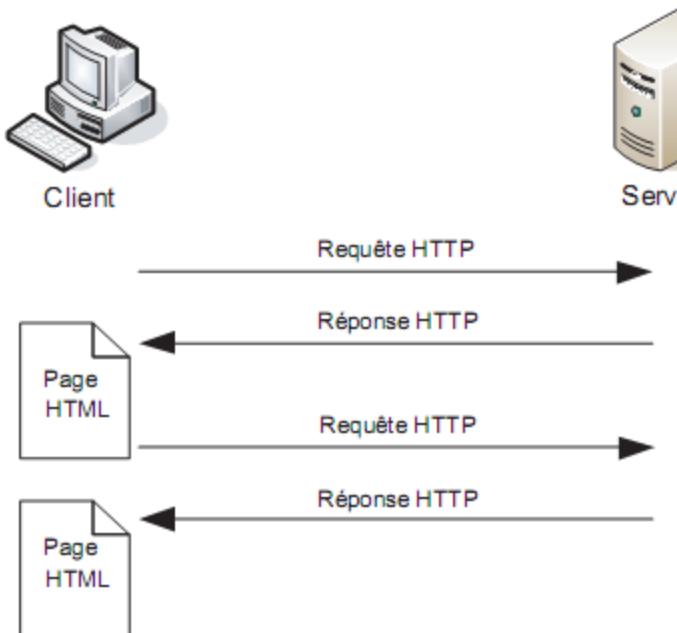
Pourquoi AJAX ?

- Une grande partie de la page est rechargée alors que le contenu est le même !!

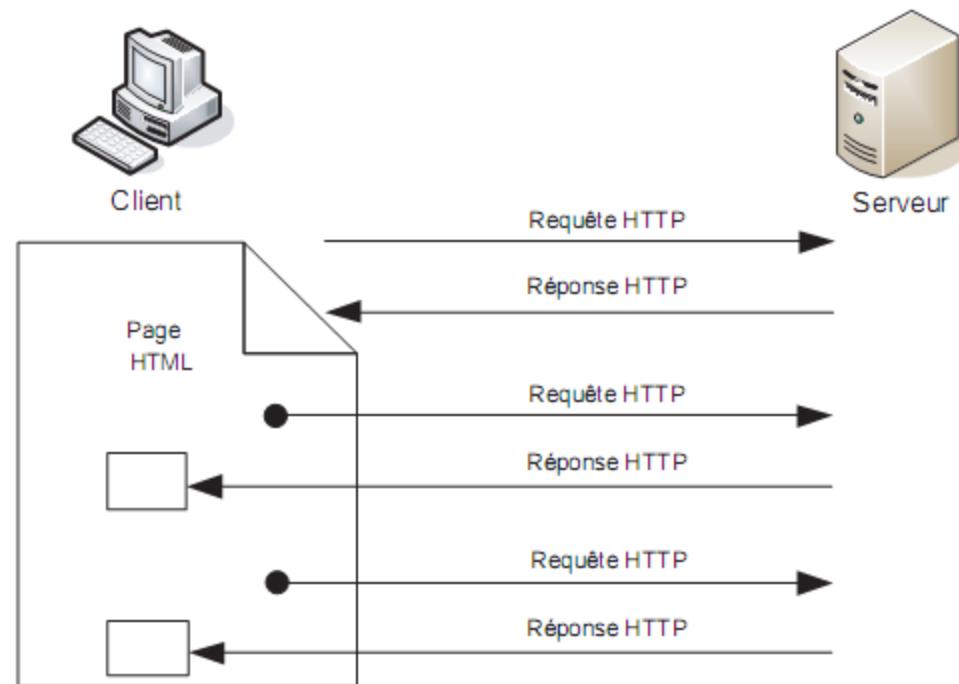


Que permet AJAX?

- L'idée même d'AJAX est de faire communiquer une page Web avec un serveur Web sans occasionner le rechargement de la page.
- Permettre la MAJ **partielle** d'une page :



Communication client-serveur



Communication client-serveur en Ajax

Mode asynchrone possible avec AJAX

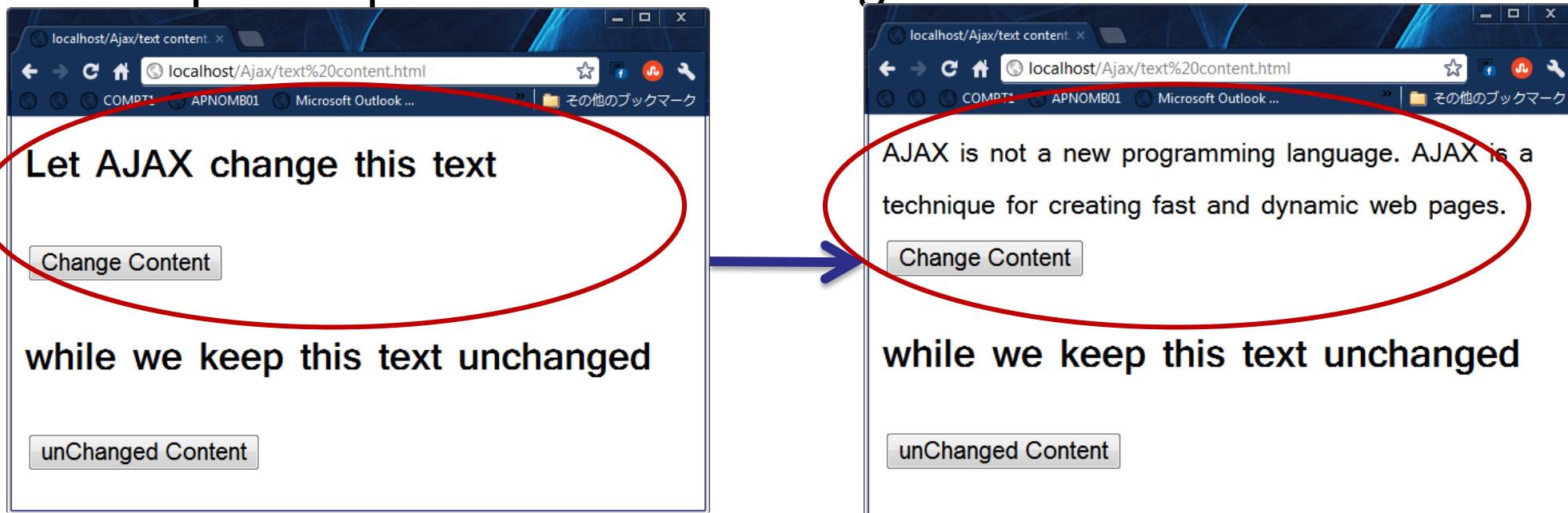
- Que veut-on dire par communication **asynchrone** avec un serveur ?

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

- On dit que le script est exécuté de façon **synchrone** : 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 une valeur
- Quand un appel est **asynchrone**, le script principal n'attend pas d'avoir reçu les données pour continuer.
- 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.

Un premier exemple

- Lorsqu'on clique sur le bouton "change content" :



- Le contenu qui se trouve côté serveur :

The image shows a screenshot of the Notepad++ text editor. The file 'ajax_info.txt' is open, containing the following text:

```
1 AJAX is not a new programming language.  
2  
3 AJAX is a technique for creating fast and dynamic web pages.
```

File menu: Fichier Edition Recherche Affichage Encodage Langage Paramétrage Macro Exécution TextFX Compléments

Status bar: length : 107 lines : 5 Ln : 1 Col : 1 Sel : 0 Dos\Windows ANSI INS

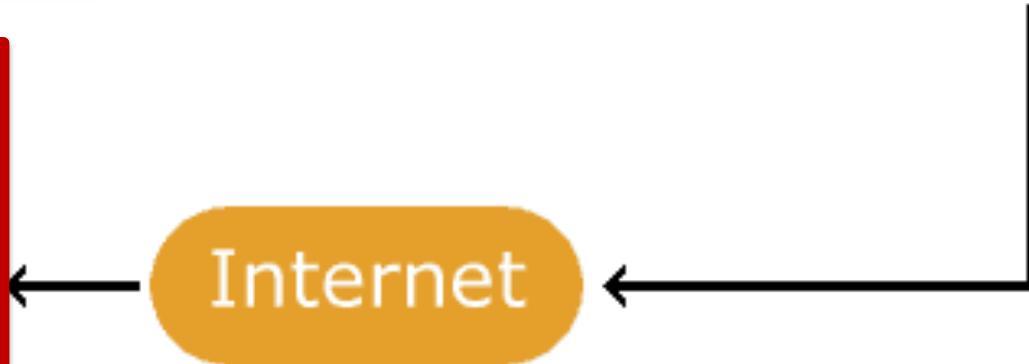
Ce qui se passe derrière

1. Un événement se produit
(onclick sur le bouton)
2. On crée un objet
XMLHttpRequest
3. On envoie la requête
HttpRequest



4. Traiter la requête
HttpRequest
5. Créer une réponse et
l'envoyer au
navigateur

6. Exécuter la page
retournée(JS , HTML et
CSS)
7. MAJ le contenu de la
page



Navigateur

Serveur

The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar with icons for file operations, search, and other development tools. The main area displays a file named "text content.html" with the following content:

```
text content.html — first-app
JS app.js ● text content.html ● JS global.js JS mymod.js
1 <!DOCTYPE HTML>
2 <html>
3
4 <head>
5   <script type="text/javascript">
6     function getXMLHttpRequest() { ...
23    function loadXMLDoc() { ...
33   </script>
34 </head>
35
36 <body>
37   <div id="myDiv">
38     <h2>Let AJAX change this text</h2>
39   </div>
40   <button type="button" onclick="loadXMLDoc()">Change Content</button>
41   <br/>
42   <div>
43     <h2>while we keep this text unchanged</h2>
44   </div>
45   <button type="button">unChanged Content</button>
46 </body>
47 </html>
```

Constructeur d'un objet XMLHttpRequest pour tout type de navigateur

```
function getXMLHttpRequest() {  
    var xhr = null;  
    if (window.XMLHttpRequest || window.ActiveXObject) {  
        if (window.ActiveXObject) { // un navigateur IE  
            try {  
                xhr = new ActiveXObject("Msxml2.XMLHTTP"); // IE, le parseur MSXML2  
            } catch (e) {  
                xhr = new ActiveXObject("Microsoft.XMLHTTP");  
            }  
        } else {  
            xhr = new XMLHttpRequest(); // navigateur qui supporte XMLHttpRequest  
        }  
    } else {  
        alert("Votre navigateur ne supporte pas l'objet XMLHttpRequest...");  
        return null;  
    }  
    return xhr;  
}
```

CallBack Function

```
function loadXMLDoc() {  
var xmlhttp = getXMLHttpRequest(); // initialise an XMLHttpRequest  
xmlhttp.open("GET", "ajax_info.txt", true);  
// ask our XMLHttpRequest object to open a server connection  
xmlhttp.send(); // and finally send the request  
xmlhttp.onreadystatechange = function () {  
//monitor the request state  
if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {  
document.getElementById("myDiv").innerHTML = xmlhttp.responseText;  
//response received from the server  
}  
}  
}
```

Étape 1: Créer l'objet XMLHttpRequest,

- La communication avec le serveur repose sur l'objet JavaScript XMLHttpRequest, qui permet d'émettre une requête en arrière plan et de spécifier le traitement à effectuer à la réception de sa réponse.
- C'est grâce à cette objet que la MAJ partielle de la page devient possible sans avoir à actualiser la totalité de la page.
- Syntaxe pour créer un objet XMLHttpRequest :

```
var xmlhttp= new XMLHttpRequest();
```

- Pour instancier un objet XMLHttpRequest , il suffira de faire :

```
var xmlhttp= getXMLHttpRequest();
```

getXMLHttpRequest() est un constructeur qui prend en considération tous les différents types de navigateurs vu que XMLHttpRequest n'est pas supporté par tous les navigateurs !!!

Instanciation d'une requête XMLHttpRequest

■ La grande différence entre IE et les autres navigateurs apparaît dans l'instanciation des objets de type XMLHttpRequest :

- Dans IE 5 et 6 sous Windows, le constructeur XMLHttpRequest est obtenu à travers un composant ActiveX

window.ActiveXObject

- Alors que c'est un objet natif de IE 7 et des autres navigateurs (autrement dit, il est une propriété de window)

window.XMLHttpRequest

L'instruction : *xmlhttp=new XMLHttpRequest();*

est équivalente à : *xmlhttp=new window.XMLHttpRequest();*

puisque les fonctions globales sont en fait des méthodes de l'objet window.

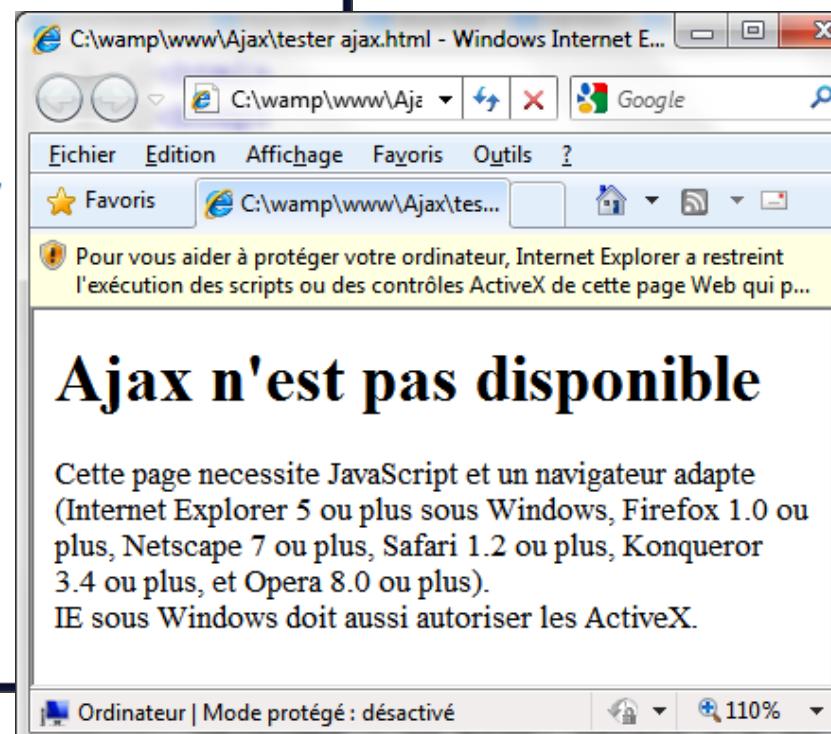
```

tester ajax.html
view-source:file:///C:/wamp/www/Ajax/tester%20ajax.htm
Microsoft Outlook ... Téléchargement de ... Mise en forme de ...
1 <html>
2 <body>
3 <script type="text/javascript">
4 function bodyOnLoad() {
5   if (window.XMLHttpRequest) {
6     document.getElementById("prerequis").style.display = "none";
7     document.getElementById("page").style.display = "block";
8   }
9 }
10 </script>
11 <body onload="bodyOnLoad()">
12 <div id="prerequis">
13 <h1>Ajax n'est pas disponible</h1>
14 Cette page nécessite JavaScript et un navigateur adapté
15 (Internet Explorer 5 ou plus sous Windows, Firefox 1.0 ou plus,
16 Netscape 7 ou plus, Safari 1.2 ou plus, Konqueror 3.4 ou plus,
17 et Opera 8.0 ou plus).<br/>
18 IE sous Windows doit aussi autoriser les ActiveX.
19 </div>
20 <div id="page" style="display: none">
21 <h1>Ajax est disponible</h1>
22 ... reste de la page qui utilise XMLHttpRequest ...
23 </div>
24 </body>
25 </html>

```



*display: propriété CSS
display: none (ne pas afficher)
display: block(afficher en tant que block)*



Étape 2: Envoie de la requête

HttpRequest

- Il faut d'abord définir les modalités d'envoi de la requête avec la méthode **open()**, et on l'enverra ensuite avec la méthode **send()** :
- Syntaxe de la méthode open : **open(Method, Url, Async)**
 - **Method** : la méthode de transfert : **GET** ou **POST**;
 - **Url** : la page destination de la requête. Ça peut être une page dynamique (PHP, ASP, etc.) ou une page statique (TXT, XML...); cette URL va comporter des paramètres et leurs valeurs si la méthode d'envoi est GET
 - **Async** : définit si le mode de transfert est asynchrone ou non (synchrone). Dans ce cas, mettez **true**. Ce paramètre est optionnel et vaut **true** par défaut.
- Exemple :

```
var xmlhttp = getXMLHttpRequest(); //instance d'une requête  
xmlhttp.open("GET","ajax_info.txt",true); // obtenir le contenu du fichier .txt  
xmlhttp.send(); // ne prends pas de paramètre quand le mode est GET
```

Étape 3 : traitement de la requête côté serveur

- On veut faire un certain traitement, en fonction de la réponse du serveur à la requête envoyée,
- Mais comment savoir où l'on est côté serveur dans le traitement de la requête HttpRequest envoyée?
- Solution : À cette fin, il y a :
 - La propriété **readyState** : permet de suivre de prêt l'état de la requête
 - D'un autre coté, il y a un événement, **onreadystatechange** qui se déclenche à chaque changement d'état de la requête : changement dans la valeur de la propriété **readyState**
- Lorsque la propriété **readyState** change de valeur, l'événement **onreadystatechange** est détecté et prend la valeur **true**

Le changement d'état

■ Les valeurs de la propriété **readyState** :

- **0** : L'objet XMLHttpRequest a été créé, mais pas encore initialisé (la méthode **open** n'a pas encore été appelée)
- **1** : L'objet XMLHttpRequest a été créé, mais pas encore envoyé (avec la méthode **send**)
- **2** : La méthode **send** vient d'être appelée
- **3** : Le serveur traite les informations et a commencé à envoyer des données
- **4** : Le serveur a fini son travail, et toutes les données sont réceptionnées

■ On associe à l' événement **onreadystatechange**, une fonction qui sera appelée à chaque fois que l'événement sera déclenché,

```
xmlhttp.onreadystatechange=reponseAjax;
```

■ donc, en tout cette événement sera déclenchée 4 fois, et la fonction **reponseAjax** sera appelée 4 fois aussi !!

Le changement d'état

- En plus de la propriété **readyState**, il y a aussi la propriété **status** : permet de connaître le code renvoyé par le serveur HTTP
 - code 404 pour les pages non trouvées
 - code 500 pour l'erreur de serveur de la requête,
 - code **200** tout est OK.
 - Etc...

```
xmlhttp.onreadystatechange=function() { //monitor the request state  
if (xmlhttp.readyState==4 && xmlhttp.status==200){  
//traitement à faire coté client à la réception de la réponse du serveur  
}  
}
```

```

function getXMLHttpRequest() {...}
function loadXMLDoc() {
var xmlhttp = getXMLHttpRequest();
// initialise an XMLHttpRequest
xmlhttp.open("GET", "ajax_info.txt", true);
// ask our XMLHttpRequest object to open a server connection
xmlhttp.send(); // and finally send the request
xmlhttp.onreadystatechange = function () { //monitor the request state
if (xmlhttp.readyState = 4 && xmlhttp.status = 200) {
    document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
//response received from the server
//changer le contenu de l'élément html ayant pour identifiant myDiv par le contenu
//reçu du serveur: le contenu du fichier ajax_info.txt
}}}</script></head><body>
<div id="myDiv">
    <h2>Let AJAX change this text</h2>
</div>
<button type="button" onclick="loadXMLDoc()">Change Content</button><br/>
<div><h2>while we keep this text unchanged</h2></div>
<button type="button">unChanged Content</button>
</body></html>

```

The screenshot shows a code editor window with two tabs: 'text content.html' and 'ajax_info.txt'. The 'text content.html' tab contains the main HTML code with color-coded syntax highlighting. The 'ajax_info.txt' tab contains the content of the file 'ajax_info.txt', which is a plain text file with two lines of text: 'AJAX is not a new programming language.' and 'AJAX is a technique for creating fast and dynamic web pages.'.

```

1 | AJAX is not a new programming language.
2 |
3 | AJAX is a technique for creating fast and dynamic web pages.
4 |
5 |
length :107  lines :5      Ln:1 Col:1 Sel:0      Dos\Windows  ANSI  INS

```

Récupération des données

- On a envoyé la requête, à la fin de l'envoie de la requête XMLHttpRequest au serveur (détecté par la condition `myRequest.readyState == 4`), La requête a trouvé des données à récupérer (les données fournies par le fichier PHP, TXT ou XML).
- La récupération des données se fait avec les propriétés de l'objet XMLHttpRequest:
 - **responseText** : pour récupérer les données sous forme de texte brut
 - **responseXML** : pour récupérer les données sous forme d'arbre XML

Exemple de récupération de donnée texte

```
//hello.php
```

```
<?php echo "Hello Ajax caller!"; ?>
```

```
<html><head><script type="text/javascript">  
function getXMLHttpRequest{//voir page précédente }  
var myRequest = getXMLHttpRequest();  
function callAjax() {  
myRequest.open("GET","hello.php", true);  
myRequest.onreadystatechange = reponseAjax;  
myRequest.send(null);  
}  
function reponseAjax()  
if(myRequest.readyState == 4) {  
if(myRequest.status == 200) {  
alert("The server said: "+ myRequest.responseText);}  
else {  
alert("An error has occurred: "+ myRequest.statusText);}}}  
</script></head><body onload="callAjax()";></body></html>
```

La fonction **reponseAjax()** s'appelle une fonction de callback, appelé à chaque fois que l'événement **onreadystatechange** est détecté c.à.d. 4 fois



Récupération des données xml

- Récupération avec la propriété **responseXML** :
- On va considérer le fichier XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
  
//hello-xml.xml  
  
<greeting>  
Hello Ajax caller!  
</greeting>
```

Récupération des données xml: exemple

```
<html><head><script type="text/javascript">  
function getXMLHttpRequest(){}/>voir les pages précédentes  
var myRequest = getXMLHttpRequest();  
function callAjax() {  
myRequest.open("GET","hello-xml.xml", true);  
myRequest.onreadystatechange = response;  
myRequest.send(null); }  
function response() {  
if(myRequest.readyState == 4) {  
if(myRequest.status == 200) {  
var greetNode= myRequest.responseXML.getElementsByTagName("greeting")[0];  
var greetText = greetNode.childNodes[0].nodeValue;  
alert("The server said: "+ greetText);}  
document.getElementById("myDiv").innerHTML=greetText;  
else {  
alert("An error has occurred: "+ myRequest.statusText);} } }  
</script></head><body onload="callAjax()";>  
<div id="myDiv"></div></body></html>
```

La fonction **reponse()** s'appelle une fonction de callback, appelé à chaque fois que l'événement **onreadystatechange** est détecté c.à.d. 4 fois

Passer des variables au serveur

- Quand vous utilisez la méthode **GET** , vous pouvez passer des variables au serveur : les variables sont transmises directement dans l'URL :

```
xmlhttp.open("GET","attendre.php?variable1=nom&variable2=tel", true);  
xmlhttp.send(null);
```

- Pour **POST**, il faut spécifier les variables dans l'argument de **send()** :

```
xhr.open("POST", "attendre.php", true);  
xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  
xhr.send("variable1=nom&variable2=tel");
```

La méthode open(): mode GET ou POST?

- Si vous utilisez la méthode **POST**, vous devez absolument changer le type MIME de la requête avec la méthode **setRequestHeader**, sinon le serveur ignorera la requête :

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

- GET ou POST?
 - GET est plus rapide et plus simple que POST
- Utilisez POST quand :
 - Il faut MAJ une base de donnée ou un fichier sur le serveur
 - Envoyer une quantité de données importante

Protéger les caractères

- Avant de passer des variables, il est important de les protéger pour conserver les caractères spéciaux et les espaces.
- Pour cela, utilisez la fonction globale **encodeURIComponent** :

```
var sVar1 = encodeURIComponent("contenu avec des espaces");
var sVar2 = encodeURIComponent("contenu avec des caractères accentués!");
xhr.open("GET","attendre.php?var1=" + sVar1 + "&var2=" + sVar2, true);
xhr.send(null);
```

Transmission de variable avec GET:

```
var myRequest = getXMLHttpRequest();
function callAjax() {
    var lastname = 'Smith';// l'information à passer au serveur
    var url = "myserverscript.php?surname=" + lastname;
// ouvrir une connexion au serveur avec l'objet XMLHttpRequest
    myRequest.open("GET", url, true);
// la fonction reponseAjax va être exécuter à l'arrivée d'une réponse
    myRequest.onreadystatechange = reponseAjax();
    myRequest.send(null); // envoyer la requête avec PARAMÈTRE vide
}
```

Transmission de variable avec POST:

```
var myRequest = getXMLHttpRequest();
function callAjax() {
    var lastname = 'Smith';// l'information à passer au serveur
// l'URL du script à appeler sur le serveur
    var url = "myserverscript.php";
// ouvrir une connexion au serveur avec l'objet XMLHttpRequest
    myRequest.open("POST", url, true);
    myRequest.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    myRequest.onreadystatechange = reponseAjax();
    myRequest.send("var=" + lastname); // envoyer la requête avec paramètre
}
```

Exemple avec callback fonction et transmission de variables au serveur

- Considérons le simple formulaire suivant :

```
<form name='form1'>  
Name: <input type='text' name='myname' onChange='callAjax()'><br/>  
Tel: <input type='text' name='telno' /><br/>  
<input type='submit' />  
</form>
```

- Lorsque l'événement **onchange** se produit , une requête XMLHttpRequest est lancée à travers l'appel de la fonction **callAjax()**
- On souhaite que le contenu de la zone **myname** soit transmit au serveur, à un script php, **hellovariable.php**,
- Si la valeur saisie est "user" alors on affichera son contenu dans une boite de dialogue ajax avec hello ajax caller, sinon on l'affiche concaténé à wrong ajax user.

Exemple d'utilisation de reponseAjax()

- Lorsque l'utilisateur quitte le champs myname du formulaire, la valeur saisie (**document.form1.myname.value**) sera envoyée au serveur via la requête ajax asynchrone
- On pourrait à titre d'exemple vérifier si l'utilisateur existe dans la base de données et retourner les autres valeurs pour qu'il soient rempli de façon automatique avant même que l'utilisateur finisse de remplir les champs

envoie-variable.html — first-app



```
pp.js ●  text content.html ●  helloworld.php ●  hello.php  envoie-simple.html ●  envoie-variable.html ●  ...  
1  <!DOCTYPE HTML>  
2  <html>  
3  <head>  
4      <meta charset="iso-8859-1">  
5      <title>Ajax Send Variable</title>  
6      <script type="text/javascript">  
7          function getXMLHttpRequest() { //voir cours ...  
29         var myRequest = getXMLHttpRequest();  
30         function callAjax() { ...  
43         function responseAjax() {  
44             // we are only interested in readyState of 4,  
45             // i.e. completed  
46             if (myRequest.readyState == 4) {  
47                 // if server HTTP response is OK  
48                 if (myRequest.status == 200) {  
49                     alert("The server said: " + myRequest.responseText);  
50                 } else {  
51                     // issue an error message for  
52                     // any other HTTP response  
53                     alert("An error has occurred: " + myRequest.statusText);  
54                 }  
55             }  
56         }</script>  
57     </head>  
58     <body>  
59         <form name='form1'>  
60             Name:<input type='text' name='myname' onChange='callAjax()' />  
61             <br/> Tel:<input type='text' name='telno' />  
62             <br/><input type='submit' />  
63         </form>  
64     </body>
```

envoie-variable.html — first-app

```
pp.js • < text content.html • helloworld.php • hello.php • envoie-simple.html • envoie-variable.html • 🔍 ⚡ ...  
5  
30     function callAjax() {  
31         var myname = document.form1.myname.value;  
32         // ask our XMLHttpRequest object to open a  
33         // server connection  
34         myRequest.open("POST", "helloworld.php", true);  
35         // prepare a function responseAjax() to run when  
36         // the response has arrived  
37         myRequest.onreadystatechange = responseAjax;  
38         myRequest.setRequestHeader('Content-Type',  
39             'application/x-www-form-urlencoded');  
40         // and finally send the request  
41         myRequest.send("var1=" + myname);  
42     }  
43     function responseAjax() {  
44         // we are only interested in readyState of 4,  
45         // i.e. completed  
46         if (myRequest.readyState == 4) {  
47             // if server HTTP response is OK  
48             if (myRequest.status == 200) {  
49                 alert("The server said: " + myRequest.responseText);  
50             } else {  
51                 // issue an error message for  
52                 // any other HTTP response  
53                 alert("An error has occurred: " + myRequest.statusText);  
54             }  
55         }  
56     }</script>  
57 </head>  
58 <body>  
59     <form name='form1'>  
60         Name:<input type='text' name='myname' onChange='callAjax()' />
```

To do

- TP Ajax.pdf

