Concepteur Développeur en Informatique

Développer des composants d'interface

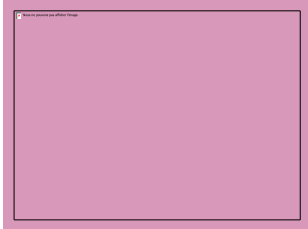
XMLHttpRequest - Ajax

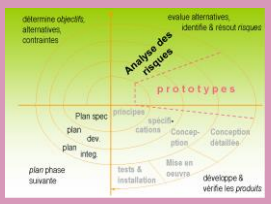
Accueil

Apprentissage

PAE

Evaluation





Localisation : U03-E05-S03

SOMMAIRE

1. Introduction

XMLHttpRequest est un objet ActiveX ou un objet du DOM (Document Object Model) qui permet d'obtenir des données au format XML, JSON Java Script On Notation, ou encore HTML ou text, à l'aide de requêtes http sans pour autant exiger la construction complète de la page côté client.

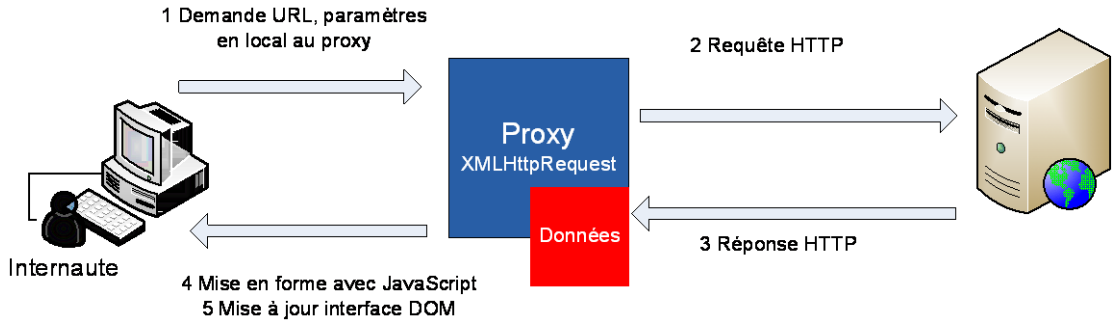
Cet objet, et on en parle bien peu, constitue le noyau de la technologie Ajax dont l'acronyme nous précise bien sa philosophie (Asynchronous Javascript And XML).
Nous verrons que la technologie Ajax propose un ensemble de fonctions dont la richesse est bien supérieure à celles fournies par ce seul objet XMLHttpRequest mais qu'elle repose néanmoins sur ce dernier.

Au départ de cette technologie, Microsoft, qui dès 1998 nous proposa sous la forme d'un composant Activex, MSXML, ces fonctionnalités associées à l'explorateur Internet Explorer 5 ! Cette forme a perduré jusqu'à l'apparition de IE 7.

Des versions sont apparues sous la forme actuelle de XMLHTTPRequest dans Mozilla en 2002, Safari en 2004 puis depuis chez tous les éditeurs de navigateurs dont Microsoft.

Avec XMLHttpRequest nous allons mixer les fonctions côté client et côté server pour proposer un mécanisme qui permet le rafraîchissement d'une partie de la page seulement.
Ce mécanisme autorise ainsi une réduction du trafic sur le Web et une meilleure ergonomie au niveau de l'application cliente (traitement asynchrone, temps de réponse amélioré).

Le schéma de fonctionnement de XMLHttpRequest :



Le composant XMLHttpRequest joue le rôle de proxy. Il envoie une requête et attend de façon synchrone ou asynchrone qu'elle soit entièrement traitée.
Le serveur http traite la demande et transmet la réponse au proxy.
Quand les données liées à la réponse sont prêtes, une méthode de rappel définie au niveau du proxy par le client Web est invoquée pour rafraîchir la portion de page qui doit être actualisée. Pour se faire, nous recourons aux techniques JavaScript couplée au DOM (Document Objet Model) et DHTML.

2. L'objet XMLHttpRequest

Nous allons maintenant nous préoccuper plus précisément de l'objet XMLHttpRequest et de ses méthodes.

2.1. Instancier un objet XMLHttpRequest

Pour pouvoir renseigner ses propriétés et invoquer ses méthodes, la première démarche est d'obtenir une instance de cet objet et de pouvoir se référer à celle-ci en stockant sa référence dans une variable. Rien de bien nouveau donc…
Nous allons seulement devoir tenir compte de la famille et version du navigateur comme je vous l'ai indiqué précédemment…

Plusieurs approches, la plus belle étant de créer une méthode qui renvoie une instance de l'objet en question de type GetInstanceXMLHR.

Dans un bloc de script JavaScript côté client programmons cette méthode en tenant compte des caractéristiques de notre navigateur.

Nous chercherons à utiliser la dernière version de l'objet XMLHttpRequest.
Je n'ai pas tenu compte d'un navigateur qui ne disposerait pas de cet objet. Il est toutefois peu probable qu'une telle version soit encore utilisée de nos jours …

```
function getXMLHttpRequest() {
    var xMLHttpR;

    if (window.XMLHttpRequest) {
        xMLHttpR = new XMLHttpRequest();
    }
    else if (window.ActiveXObject) {
        try {
            xMLHttpR = new ActiveXObject("Msxml12.XMLHTTP");
        } catch (e) {
            xMLHttpR = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }
    return xMLHttpR;
}
```

2.2. Préparer la requête http

Pas de surprise, nous aurons la possibilité d'envoyer une requête en utilisant les deux méthodes à notre disposition :

- GET : transmission des infos dans l'URL
- POST : transmission des infos dans le corps de la méthode Send

Les modalités de la requête se définissent avec la méthode **open** et la requête s'exécute sur invocation de la méthode **send**.

Nous définirons le type de contenu mime transmis dans la requête. Même si cette option n'est pas toujours obligatoire, nous pourrons ainsi nous assurer que notre requête sera toujours correctement traitée.
Cette définition se fait par le biais de la méthode **SetRequestHeader** après l'ouverture de l'objet XMLHttpRequest.

open s'utilise ainsi : open(sMethod, sUrl, bAsync)

- sMethod : la méthode de transfert : GET ou POST,
- sUrl : la page qui donnera suite à la requête. Page dynamique ou statique,
- bAsync : booléen qui définit si le mode de transfert est asynchrone ou non

Par défaut le mode de traitement de la requête est asynchrone.

SetRequestHeader s'utilise ainsi : SetRequestHeader(sName, sValue)

- sName : nom de l'entête http ici Content-Type ,
- sValue : Valeur donnée à l'entête http ici données de formulaire.

send provoque l'émission de la requête.

Vous pouvez bien entendu passer des paramètres lors de l'émission de votre requête. Dans l'argument sMethod pour la méthode GET ou dans la méthode send en lien avec la méthode POST.

```
open('GET', 'nonPage?param1=valeur1&param2=valeur2',true)
send(null)

open('POST',' nonPage',true)
send('param1=valeur1&param2=valeur2')
```

Les variables transmises doivent être encodées dans un format « URL Compatible ».

Nous invoquerons donc la méthode d'encodage **encodeURI** pour mettre en forme correctement nos paramètres. La méthode decodeURI réalise l'opération inverse.

L'exemple qui suit propose une illustration de la préparation et la transmission d'une requête.
Nous souhaitons récupérer la liste des personnes dont le nom commence par la chaîne saisie dans une boite de texte et insérer le nom de ces personnes dans une liste déroulante.

La requête est transmise :

- Par GET lors du click du bouton RechercherGet.
- Par POST lors du click du bouton RechercherPost

2.2.1. Requête GET

```
var XHTR = getXMLHttpRequest();
XHTR.open("GET", "Ajax01Reponse.aspx" + "?chargerOffresFormation=true", true);
// On envoit un header pour indiquer au serveur que la page est appellée en Ajax
XHTR.setRequestHeader('X-Requested-With', 'xmlhttprequest')

XHTR.onreadystatechange = function () {
    if (XHTR.readyState == 4 && (XHTR.status == 200)) {
        lireReponseCombo(XHTR.responseText)
    }
}
XHTR.onabort = function () {
    alert("Requête abandonnée")
}
XHTR.onerror = function () {
    alert("Une erreur a mis fin au chargement des données")
}
XHTR.send(null);
```

Nous construisons l'URL en concaténant après le nom de page, les noms des contrôle de formulaire = la valeur de ces derniers. Exemple : Page?nom=valeur&nom2=valeur2&… Ici, je n'ai pas recouru à EncodeURI du fait de l'absence de caractères spéciaux dans ma requête.

2.2.2. Requête POST

```
var XHTR = getXMLHttpRequest();
XHTR.open("POST", "Ajax01Reponse.aspx", true);
```

Ici, dans la méthode open, seul le nom de la page est indiqué dans l'URL.
Les paramètres sont passés dans la méthode send.

Pour l'alimentation des paramètres fournis au serveur vous pouvez recourir à la concaténation des paires clés valeurs comme dans cet exemple :

```
var parametres = "IDOffreFormation=" + encodeURIComponent(recupOffreSelectionnee())
    + "&ChoixSerialisation="
    + encodeURIComponent(recupBoutonChecked('ChoixSerialisation'))
    + "&ChoixDAC="
    + encodeURIComponent(recupBoutonChecked('ChoixDAC'));
```

Ou utiliser l'objet formData. Vous pouvez alors l'utiliser sous deux formes. En sélection les éléments de formulaire à communiquer :

```
var data = new FormData();
data.append('IDOffreFormation', recupOffreSelectionnee());
data.append('paramètre2', 'valeur');
```

En soumettant l'ensemble des valeurs des contrôles présents sur le formulaire.

```
var form = document.querySelector('#form')
var data = new FormData(form)
```

Les données sont fournies à la méthode Send

```
XHTR.send(data);
```

2.3. Traiter la requête côté serveur

Nous devons dans la ressource indiquée dans l'URL comme paramètre de la méthode open récupérer les paramètres transmis, préparer et envoyer la réponse.

Je ne vais pas ici entrer dans les détails de ce traitement que nous verrons par la suite mais uniquement préciser les points liés à Ajax et aux requêtes hors bande.

Ici, le résultat sera fourni sous la forme texte brut. J'indique donc au client qu'il va recevoir un document dans cette forme. C'est plus prudent. Néanmoins, le client serait capable de le déterminer et de le traiter si ce type de contenu est accepté.

```
Response.ContentType = "text/plain";
```

Conclusion : Nous avons soumis notre requête en asynchrone mais nous ne nous sommes par préoccuper du traitement de la réponse …
C'est ce que nous allons réaliser maintenant.

2.4. Traiter les résultats

Nous allons maintenant nous occuper de la réception des données après traitement par le serveur.
Nous devons détecter les changements d'états de la requête associée à l'objet XMLHttpRequest.
Cet objet possède un événement **onreadystagechange** auquel on assigne une fonction qui sera exécutée lors du changement d'état de la requête (gestionnaire d'événement).
Le tableau suivant vous présente les différents états et le statut final.

Propriété	Description
onreadystatechange	Stocke une fonction anonyme ou le nom d'une fonction qui sera invoquée automatiquement à chaque changement de la propriété readystate
readyState	Représente l'état de la requête. Varie de 0 à 4: 0: Requête non initialisée 1: Connexion établie avec le serveur 2: Requête reçue par le serveur 3: Requête en cours d'exécution 4: Requête finie et réponse prête
status	200: "OK" 404: non trouvé

Nous allons donc associer une fonction de callback qui sera exécutée à chaque changement d'état et, en fonction de la valeur de **readyState**.

Nous traiterons le résultat du serveur lorsque :

- l'état de la requête indique qu'elle est finie et que les données sont disponibles,
- que le statut précise que la requête s'est terminée normalement

Les données se récupèrent, en fonction du type de contenu, dans la propriété :

- responseText si les données sont transmises au format texte (quel que soit le format txt, XML ou Json)
- responseXML, si les données sont dans un fichier XML

Il est nécessaire aussi de s'assurer que les données suivent et résultent du bon traitement de la page.

Donc, données traitées si **readyState=4** et **status = 200**.

```
XHTR.onreadystatechange = function () {
    if (XHTR.readyState === 4 && (XHTR.status === 200)) {
        Traiter(XHTR.responseText); }
```

Ensuite, le flux de données présent dans la propriété responseText sera retraité en fonction de sa nature (plain text, xml ou json)

```
function deserialJson(donneesJSON) {
    var jSonParse = "";
    if (donneesJSON != "") {
        jSonParse = JSON.parse(donneesJSON);
    }
```

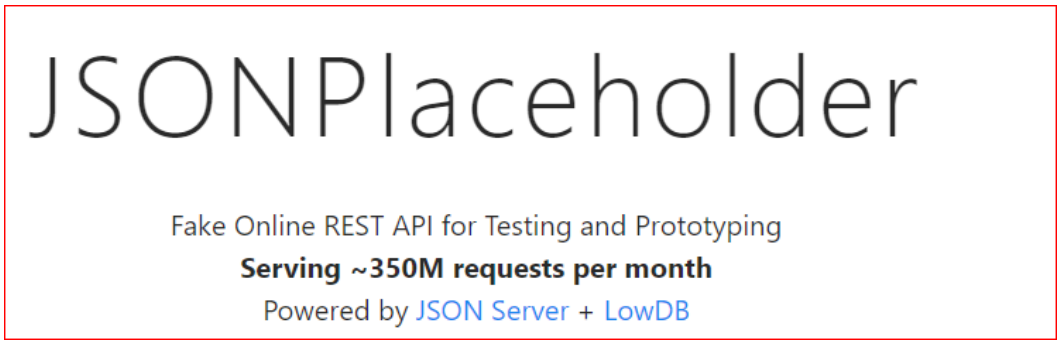
```
function deserialisationXML(donnees) {
    var xmlDoc = null;

    if (window.ActiveXObject) {
        xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
        xmlDoc.async = false;
        xmlDoc.loadXML(donnees);
    } else {
        var parser = new DOMParser();
        xmlDoc = parser.parseFromString(donnees, "text/xml");
    }
```

Pour le flux XML, il faudra retraiter les nœuds pour identifier les éléments pertinents.

3. Exemple avec Json

Souvent les requêtes Ajax s'adressent à des services Web de type SOAP ou REST (Api Web). Le format d'échange le plus simple reste, notamment avec un client JavaScript, Json.
Il sera en général privilégié dans ce contexte (client navigateur) même si les échanges peuvent se faire aussi en texte brut ou XML comme vu précédemment.

Je vais ici utiliser un site qui propose des Fake pour faciliter nos tests.



La préparation et la soumission de la requête : fonction auto-exécutable (voir IIEF)

```
(function () {
    var XHTR = getXMLHttpRequest();
    XHTR.open("GET", "https://jsonplaceholder.typicode.com/users", true);
    XHTR.onreadystatechange = function () {
        if (XHTR.readyState == 4 && (XHTR.status == 200)) {
            chargerDonnees(XHTR.responseText)
        }
    }
    XHTR.onabort = function () {
        alert("Requête abandonnée")
    }
    XHTR.onerror = function () {
        alert("Une erreur a mis fin au chargement des données")
    }
    XHTR.send(null);
})();
```

Fonction call back de récupération des données. Création d'un tableau d'objets à partir du parsing de la chaîne retournée.

```
function chargerDonnees(donneesJSON) {
    var jSonParse = "";
    if (donneesJSON != "") {
        jSonParse = JSON.parse(donneesJSON);
        deserialJson(jSonParse);
    }
}
```

Extrait du tableau des objets présents dans le résultat de la requête.


```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
      "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
      }
    },
    "phone": "010-692-6593 x09125",
    "website": "anastasia.net",
    "company": {
      "name": "Deckow-Crist",
      "catchPhrase": "Proactive didactic contingency",
      "bs": "synergize scalable supply-chains"
    }
```


```
function deserialJson(jSonParse) {

    var affUsers = document.getElementById("res");
    affUsers.value = "";
    for (var i = 0; i < jSonParse.length; i++) {
        affUsers.value += jSonParse[i].name + " "
        + jSonParse[i].username + "\n";
}
}
```

Résultat page

```
Leanne Graham Bret
Ervin Howell Antonette
Clementine Bauch Samantha
Patricia Lebsack Karianne
Chelsey Dietrich Kamren
Mrs. Dennis Schulist Leopoldo_Corkery
Kurtis Weissnat Elwyn.Skiles
Nicholas Runolfsdottir V Maxime_Nienow
Glenna Reichert Delphine
Clementina DuBuque Moriah.Stanton
```