Cours AJAX

Support de cours Version 1.2

Auteur : Grégory Jarrige Document sponsorisé par : Le Défrichoir SAS

Sommaire

Sommaire 2
1. Préambule 3
2. Introduction 4
2.1 Quelques notions 4
2.2 Principe d'utilisation 6
3. Mise en oeuvre 8
3.1 Présentation de l'objet XHR 8
3.2 Premier exemple (texte) 10
3.3 Second exemple (texte/PHP) 14
3.4 Troisième exemple (XML/PHP) 17
3.5 Compléments sur XML 22
3.6 Introduction à JSON 24
3.7 Quatrième exemple (JSON/PHP) 28
3.8 Cinquième exemple (requête de type POST) 31
3.9 Codes retours de XHR 36
4 Annexe 37
4.1 Liens utiles 37
5 Changelog 38

1. Préambule

Cette formation AJAX est destinée à des développeurs d'application souhaitant maîtriser les techniques de communication avancées que les techniques désignées sous l'acronyme AJAX permettent de mettre en oeuvre.

Pour suivre ce cours dans des conditions "correctes", il est souhaitable d'avoir une bonne expérience du langage PHP, une bonne connaissance du langage HTML (HTML 4, XHTML ou HTML5), et quelques notions de Javascript.

Pour suivre ce cours dans des conditions optimales, il est vivement recommandé d'avoir suivi au préalable les cours suivants :
- Cours PHP pour développeurs IBMi
- Cours Javascript et HTML5

2. Introduction

Les définitions des notions étudiées dans ce chapitre sont en partie empruntées aux pages Wikipédia suivantes :

http://fr.wikipedia.org/wiki/Ajax_%28informatique%29
http://fr.wikipedia.org/wiki/XMLHttpRequest

On invitera le lecteur à lire ces pages car elles sont très bien faites.

2.1 Quelques notions

Qu'est ce qu'AJAX ?

AJAX est l'acronyme de "Asynchronous JavaScript and XML".

Le terme « AJAX » fait son apparition en février 2005, dans un article de Jesse James Garrett publié sur le site "Web Adaptive Path". Garrett utilise l'acronyme AJAX pour expliquer son approche du développement d'applications web. Il décrit AJAX selon ces termes :

" Ajax n'est pas une technologie, il s'agit de plusieurs technologies se développant indépendamment. On les combine ensemble pour obtenir des résultats puissants et novateurs. Ajax comporte :

- *une présentation fondée sur les standards XHTML et CSS ;*
- *un affichage dynamique et interactif fondé sur le DOM (Document Object Model) ;*
- *un système d'échange et de manipulation de données utilisant XML et XSLT, mais aussi JSON ;*
- *un mécanisme de récupération de données asynchrones utilisant l'objet XMLHttpRequest*
- *Javascript pour lier le tout. "*

L'objet XMLHttpRequest dont parlait Garrett dans son article n'était pas nouveau. C'est Microsoft qui avait ouvert la voie, en introduisant cet objet dans son navigateur phare (IE) en 1998. Cet objet permettait de mettre en oeuvre des requêtes asynchrones (ou synchrones, au choix) entre le poste client (le navigateur) et le serveur, qui pouvait être un environnement LAMP (pour PHP), ou un environnement ASP. Ce même objet fut implémenté dans les navigateurs concurrents dès le début des années 2000 (2002 pour Mozilla Firefox, 2004 pour Safari, etc...). L'article de Garrett eut un fort retentissement au sein de la communauté des

développeurs webs, et le succès de XMLHttpRequest ne s'est pas démenti par la suite. En effet, on ne compte plus les sites qui utilisent cet objet pour mettre en oeuvre de l'auto-complétion, ou le rafraîchissement partiel de pages.

L'architecture informatique AJAX permet de construire des applications Web et des sites web dynamiques interactifs sur le poste client en se servant de différentes technologies ajoutées aux navigateurs web entre 1995 et 2005.

AJAX combine JavaScript, les CSS, XML, le DOM et l'objet Javascript XMLHttpRequest, afin d'améliorer maniabilité et confort d'utilisation des Applications Internet Riches (abr. RIA). :

- DOM et JavaScript permettent de modifier l'information présentée dans le navigateur en respectant sa structure ;
- L'objet XMLHttpRequest sert au dialogue asynchrone avec le serveur Web ;
- XML structure les informations transmises entre serveur Web et navigateur.

Outre le XML, les échanges de données entre client et serveur peuvent utiliser d'autres formats, tels que JSON. Plus compact et surtout moins verbeux que XML, il faut très peu de manipulation pour convertir du texte au format JSON en un jeu de données Javascript.

Les applications AJAX fonctionnent sur tous les navigateurs Web courants : Mozilla Firefox, Google Chrome, Safari, Opera, Internet Explorer, Konqueror, etc.

Quelques données historiques :

Le langage de programmation JavaScript est apparu pour la première fois en 1996, dans le navigateur Web Netscape Navigator.

L'interface de programmation Document Object Model (abr. DOM) a été normalisée par le W3C en 1998.

XMLHttpRequest est initialement un composant ActiveX créé en 1998 par Microsoft pour leur application web Outlook Web Access, puis il a été ajouté à la norme ECMAScript relative au langage JavaScript et mis en œuvre sur la plupart des navigateurs du marché entre 2002 et 2005.

2.2 Principe d'utilisation

Dans une application Web, la méthode classique de dialogue entre un navigateur et un serveur est la suivante : lors de chaque manipulation faite par l'utilisateur, le navigateur envoie une requête contenant une référence à une page Web, puis le serveur Web effectue des calculs, et envoie le résultat sous forme d'une page Web à destination du navigateur. Celui-ci affichera alors la page qu'il vient de recevoir. Chaque manipulation entraîne la transmission et l'affichage d'une nouvelle page. L'utilisateur doit attendre l'arrivée de la réponse pour effectuer d'autres manipulations.

En utilisant AJAX, le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante : un programme écrit en langage de programmation JavaScript, incorporé dans une page web, est exécuté par le navigateur. Celui-ci envoie en arrière-plan des demandes au serveur Web, puis modifie le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçu du serveur, évitant ainsi la transmission et l'affichage d'une nouvelle page complète.

La méthode classique de dialogue utilise des mécanismes propres au World Wide Web, qui sont incorporés dans tous les navigateurs ainsi que les robots d'indexation, et ne nécessite pas de programmation. Au contraire, le fonctionnement d'AJAX nécessite de programmer en JavaScript (côté navigateur) les échanges entre le navigateur et le serveur Web. Il nécessite également de programmer les modifications à effectuer dans la page Web à la réception des réponses, sans quoi les dialogues se font à l'insu de l'utilisateur.

En AJAX, comme le nom l'indique, les demandes sont effectuées de manière asynchrone : le navigateur Web continue d'exécuter le programme JavaScript alors que la demande est partie, il n'attend pas la réponse envoyée par le serveur Web et l'utilisateur peut continuer à effectuer des manipulations pendant ce temps.

Pour créer un nouvel objet XMLHttpRequest en Javascript, on va déclarer une variable (en fait un objet) fondé sur l'objet constructeur XMLHttpRequest :

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

Sous Internet Explorer 6, cela donne :

```
var xhttpobj = new ActiveXObject("Microsoft.XMLHTTP");
```

On a simplement créé ici un canal potentiel de communication entre le navigateur et un (ou plusieurs) serveur(s).

Il convient maintenant de préciser l'URL à atteindre et à ouvrir effectivement ce canal :

```
xhttpobj.open('GET', 'http://www.monsite/data.php', false);
```

On va également définir le nom de la fonction (callback) qui sera appelée par Javascript, dès que la réponse du serveur aura été réceptionnée par le poste client (dont on rappelle qu'il est l'appelant).

```
xmlHttp.onreadystatechange = handleServerResponse;
```

Et enfin, on envoie la requête HTTP vers le serveur :

```
xhttpobj.send(null);
```

Nous allons revoir ces instructions dans les chapitres suivants, au travers de différents exemples.

3. Mise en oeuvre

3.1 Présentation de l'objet XHR

XMLHttpRequest est un objet ActiveX (pour les navigateurs de la famille IE antérieurs à la version 9) ou JavaScript (pour les autres navigateurs). Il permet d'obtenir des données au format XML, JSON, mais aussi HTML, ou encore "texte simple" à l'aide de requêtes HTTP.

L'objet XMLHttpRequest est souvent désigné sous le terme abrégé XHR.
Il est facile de créer un objet XHR en Javascript, cela se fait de la façon suivante :

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

On rappelle ici que c'est Microsoft qui est l'initiateur de cette technique. Or Microsoft n'utilisait pas à l'origine la syntaxe ci-dessus, mais une autre syntaxe basée sur l'objet window.ActiveXObject. Les éditeurs des navigateurs concurrents (Mozilla en tête) ont choisi une autre terminologie, en créant l'objet window.XMLHttpRequest. C'est cette notation qui a été adoptée par le W3C pour la normalisation du HTML, et Microsoft s'est aligné sur la norme du W3C à partir de IE9.

Si vos applications sont destinées à des navigateurs récents, vous pouvez vous contenter de travailler avec l'objet window.XMLHttpRequest.

Si en revanche, vous devez assurer une compatibilité avec les navigateurs de la famille IE (Internet Explorer) antérieurs à la version 9, alors vous devez utiliser une fonction Javascript telle que celle qui est proposée sur la page Wikipédia introduisant l'objet XMLHttpRequest.

La fonction ci-dessous est une version un peu modifiée de celle proposée sur la page Wikipédia :

```
function createXhrObject() {
    "use strict" ;
    var xhr = false;
    // pour la plupart des navigateurs sauf IE < 9
    if (window.XMLHttpRequest) {
        try {
            xhr = new XMLHttpRequest();
            return xhr;
        }
        catch (e) {
        }
    }
    // pour les navigateurs IE < 9
    if (window.ActiveXObject) {
        var names = [
            "Msxml2.XMLHTTP.6.0",
            "Msxml2.XMLHTTP.3.0",
            "Msxml2.XMLHTTP",
            "Microsoft.XMLHTTP"
        ];
        for (var i = 0, i_len = names.length; i < i_len; i += 1) {
            try {
                xhr = new ActiveXObject(names[i]);
                break;
            }
            catch (e) {
            }
        }
    }
    if (xhr === false) {
        console.log('Votre navigateur ne prend pas en charge l'objet
XMLHTTPRequest.');
    }
    return xhr;
}
```

On trouve dans les forums et dans de nombreux livres des variantes de la fonction ci-dessus. La version ci-dessus a le mérite de la simplicité et elle gère correctement les différentes versions d'objet ActiveX de Microsoft, renvoyant l'objet adéquat en fonction des capacités du navigateur utilisé. Le message d'erreur piloté ici par la fonction Javascript console.log() pourrait être géré de manière plus élégante, mais on s'en contentera pour l'instant.

Pourquoi la détermination de l'objet XHR est-elle si compliquée sur les vieilles versions d'IE ? Des éléments de réponse se trouvent sur la page suivante :

http://blogs.msdn.com/b/xmlteam/archive/2006/10/23/using-the-right-version-of-msxml-in-internet-explorer.aspx

3.2 Premier exemple (texte)

Pour un premier exemple de mise en oeuvre d'AJAX, commencez par créer une première page HTML (très simple) que vous appellerez "page01.html" :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>Ma première page avec AJAX</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <h1>Premier exercice AJAX</h1>
        <div id="bloc01">Franchement, je ne vois pas où il y a de l'AJAX dans
cette page</div>
    </body>
</html>
```

Dans le même répertoire que la page HTML, créez un fichier texte que vous appellerez "loremipsum.txt", et collez-y un texte aléatoire généré par exemple via le site suivant (ou tout autre texte à votre convenance) :

http://fr.lipsum.com

Ouvrez votre page "page01.html" dans votre navigateur préféré (Firefox, Chrome, IE, Safari, Opera...) pour vérifier qu'elle fonctionne bien.

Créez dans le même répertoire que la page le fichier "ajax_quickstart01.js". Collez-y le code de la fonction `createXhrObject()` vue au chapitre précédent, et ajoutez-y le code suivant :

```
// création d'une requête HTTP asynchrone
function processAjax() {
    // Traitement si l'objet XHR n'est pas occupé
    if (xmlHttp.readyState === 4 || xmlHttp.readyState === 0) {
        // chargement du fichier TXT
        xmlHttp.open("GET", "loremipsum.txt", true);
        // Définition de la fonction JS à appeler une fois la requête aboutie
        xmlHttp.onreadystatechange = handleServerResponse;
        // Envoi de la requête au serveur
        xmlHttp.send(null);
    } else {
        // Si la connexion avec le serveur est occupée,
        //   on retente le coup au bout d'une seconde
        setTimeout('processAjax()', 1000);
    }
}

// Fonction exécutée automatiquement lorsque la réponse du serveur arrive
function handleServerResponse() {
    // Si readystate = 4 c'est que la requête a complètement abouti
    if (xmlHttp.readyState === 4) {
        // Le statut 200 indique que la requête HTTP s'est bien passée
        if (xmlHttp.status === 200) {
            // Mise à jour de la div bloc01 dans la page
            //  en fonction du texte récupéré
            document.getElementById("bloc01").innerHTML =
                    '<i> ' + xmlHttp.responseText + ' </i>';
        } else {
            // Récupération du message d'erreur
            document.getElementById("bloc01").innerHTML +=
                    '<br><i>Décidément AJAX, ça ne marche pas : ' +
                     xmlHttp.statusText + ' </i>';

        }
    }
}

var xmlHttp = createXhrObject();
```

Ajoutez enfin une ligne de code (en gras) à votre page "page01.html" :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>Ma première page avec AJAX</title>
        <meta charset="UTF-8">
        <script type="text/javascript" src="ajax_quickstart01.js"></script>
    </head>
    <body>
        <h1>Premier exercice AJAX</h1>
        <div id="bloc01">Franchement, je ne vois pas où il y a de l'AJAX dans cette
page</div>
    </body>
</html>
```

Rafraîchissez la page dans votre navigateur, normalement il ne doit rien se passer. C'est normal.

Activez les outils de développement de votre navigateur (généralement via la touche de fonction F12) et passez en mode Console.
Dans la ligne de commande de la console, saisissez le code suivant :

```
processAjax() ;
```

Si tout s'est bien passé, le contenu de la div "bloc01" a dû changer dans la page affichée par le navigateur. Dans le cas contraire, vous avez certainement un message d'erreur dans la console.

Dans ce premier exemple, il y a peu d'interactivité, ou plutôt nous l'avons déclenchée (cette interactivité) en appelant manuellement la fonction processAjax() que nous avions omis dans le code source du fichier JS "ajax_quickstart01.js". D'ailleurs vous pouvez ajouter cette instruction à la fin du fichier JS pour ne pas avoir à la relancer à chaque fois.

C'est pauvre, mais c'est déjà de l'AJAX, même si nous nous sommes contentés de récupérer le contenu d'un fichier texte pour l'insérer dans une div de la page HTML.

C'était donc notre "hello world" de l'AJAX. Ca restera encore assez simple dans le chapitre suivant, ensuite ça va se corser progressivement.

Point important :

En réponse de requêtes AJAX, on utilise très souvent le format HTML, qui est de fait un type Texte.
Dans ce cas de figure, il est facile de "coller" la réponse du serveur dans une zone de l'écran - par exemple une DIV, via la méthode innerHTML, comme dans l'exemple suivant :

```
document.getElementById('data-container').innerHTML = xhttpobj.responseText;
```

Dans l'exemple ci-dessus, le contenu de l'élément ayant pour identifiant "data-container" est remplacé par le code HTML contenu dans la propriété "responseText" de l'objet xhttpobj.

Le problème avec cette technique est que HTML est un format de données encore plus verbeux que XML. Néanmoins, cette technique est intéressante car le code renvoyé par le serveur peut être immédiatement "collé" dans un élément du DOM, sans nécessité de développer des traitements intermédiaires pour mettre en forme les données au format HTML. Mais l'inclusion de code HTML implique un rafraîchissement du DOM, ce qui peut se révéler gourmand en ressources (pour le poste client), si le code HTML est de taille importante. On s'efforcera de limiter l'utilisation de cette technique à de petits volumes de données.

3.3 Second exemple (texte/PHP)

Pour notre second exemple, nous allons créer une page "page02.html" en copiant le fichier "page01.html", et nous allons légèrement le modifier :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>Second exercice avec AJAX</title>
        <meta charset="UTF-8">
        <script type="text/javascript" src="ajax.js"></script>
        <script type="text/javascript" src="ajax_quickstart02.js"></script>
    </head>
    <body>
        <h1>Second exercice AJAX</h1>
        <div id="bloc01">Franchement, je ne vois pas où il y a de l'AJAX dans cette
page</div>
    </body>
</html>
```

On crée le fichier "ajax.js" en dupliquant le fichier "ajax_quickstart01.js" et on ne conserve dans ce nouveau fichier que la fonction createXhrObject().
On crée le fichier "ajax_quickstart02.js" en dupliquant le fichier "ajax_quickstart01.js" et on retire de ce nouveau fichier la fonction createXhrObject().
On a donc "externalisé" la fonction createXhrObject(), fonction hautement réutilisable, car nous la retrouverons sur la plupart des projets qui suivent.

Dans le fichier "ajax_quickstart02.js", à l'intérieur de la fonction processAjax() on va modifier la ligne l'appel de la fonction xmlHttp.open() comme suit :

```
function processAjax() {
    // Traitement si l'objet XHR n'est pas occupé
    if (xmlHttp.readyState === 4 || xmlHttp.readyState === 0) {
        // Appel d'un script PHP
        xmlHttp.open("GET", "ajax_quickstart02.php?param1=test", true);
        // Définition de la fonction JS à appeler une fois la requête aboutie
        xmlHttp.onreadystatechange = handleServerResponse;
        // Envoi de la requête au serveur
        xmlHttp.send(null);
    } else {
        // Si la connexion avec le serveur est occupée,
        //   on retente le coup au bout d'une seconde
        setTimeout('process()', 1000);
    }
}
```

```
// Fonction exécutée automatiquement lorsque la réponse du serveur arrive
function handleServerResponse() {
    // Si readystate = 4 c'est que la requête a complètement abouti
    if (xmlHttp.readyState === 4) {
        // Le statut 200 indique que la requête HTTP s'est bien passée
        if (xmlHttp.status === 200) {
            // Mise à jour de la div bloc01 en fonction du texte récupéré
            document.getElementById("bloc01").innerHTML =
                    '<i> ' + xmlHttp.responseText + ' </i>';
        } else {
            // Récupération du message d'erreur
            document.getElementById("bloc01").innerHTML +=
                    '<br><i>Décidément AJAX, ça ne marche pas : ' +
                         xmlHttp.statusText + ' </i>';

        }
    }
}

var xmlHttp = createXhrObject();

processAjax() ;
```

On crée maintenant dans notre projet un script PHP que nous appellerons "ajax_quickstart02.php" et qui contiendra le code suivant :

```
<?php
/*
 * ça c'est juste pour vérifier que le paramètre transmis dans la requête est
 * bien réceptionné via la variable globale $_GET
 */
error_log($_GET['param1']) ;

/*
 * préparation de la variable contenant le lorem ipsum
 */
$texte = <<<BLOC_TXT
 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce sed egestas libero.
Praesent nec elit sit amet nibh egestas sagittis ut quis magna. Nam vulputate
imperdiet purus non consectetur. Quisque imperdiet ex lorem, vel mollis risus gravida
sit amet. Suspendisse venenatis elementum arcu, nec feugiat felis euismod sed.
Pellentesque urna erat, aliquam eget augue et, ultricies luctus turpis. Nulla
facilisi. Sed dictum congue ullamcorper. Curabitur nisi massa, mattis vel viverra sit
amet, luctus at metus. Nam facilisis tempus finibus. Sed maximus, tortor vel lobortis
laoreet, augue mauris egestas turpis, vitae accumsan magna ex ac odio. Proin mattis,
mi nec accumsan porta, massa elit faucibus augue, ut volutpat ipsum arcu nec ante. Ut
dictum sed erat id volutpat. Nullam placerat placerat pretium. Mauris sit amet tempus
mi, vel euismod justo.
BLOC_TXT;
```

```
/*
 * Avant de renvoyer le lorem ipsum au poste client, on doit l'informer que
 * le format des données renvoyées est du pur texte, ce qui se fait avec la
 * ligne de code suivante :
 */
header('Content-Type: text/text');

/*
 * On balance les infos et c'est fini
 */
echo $texte ;
```

Dans ce second exemple, la requête AJAX ne va plus chercher le contenu d'un banal fichier texte, elle fait appel à un script PHP qui lui renvoie la réponse « prémâchée ».

3.4 Troisième exemple (XML/PHP)

Pour ce troisième exemple, nous allons créer une page "page03.html" en copiant le fichier "page01.html", et nous allons légèrement la modifier :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>3ème exercice AJAX (avec XML)</title>
        <meta charset="UTF-8">
        <script type="text/javascript" src="ajax.js"></script>
        <script type="text/javascript" src="ajax_quickstart03.js"></script>
    </head>
    <body>
        <label for="yourID">Saisissez un identifiant</label>
        <input type="text" id="yourID" />
        <input type="button" id="startAjax" value="Recherche" />
        <div id="divMessage" ></div>
    </body>
</html>
```

Le fichier "ajax.js" existe déjà, il contient la fonction createXhrObject(), on n'y revient pas pour l'instant.
On crée le fichier "ajax_quickstart03.js" en y plaçant le code ci-dessous :

```
// création d'une requête HTTP asynchrone
function processAjax() {
    "use strict";
    // Traitement si l'objet XHR n'est pas occupé
    if (xmlHttp.readyState === 4 || xmlHttp.readyState === 0) {
        // On récupère l'identifiant saisi dans le champ "yourID"
        var name = encodeURIComponent(
                    document.getElementById("yourID").value);
        // Exécute le script quickstart.php sur le serveur
        xmlHttp.open("GET", "ajax_quickstart03.php?name=" + name, true);
        // Définition de la fonction JS à appeler une fois la requête aboutie
        xmlHttp.onreadystatechange = handleServerResponse;
        // Envoi de la requête au serveur
        xmlHttp.send(null);
    } else {
        // Si la connexion avec le serveur est occupée,
        //   on retente le coup au bout d'une seconde
        setTimeout('process()', 1000);
    }
}
```

```
// Fonction exécutée automatiquement lorsque la réponse du serveur arrive
function handleServerResponse() {
    "use strict";
    // Si readystate = 4 c'est que la requête a complètement abouti
    if (xmlHttp.readyState === 4) {
        // Le statut 200 indique que la requête HTTP s'est bien passée
        if (xmlHttp.status === 200) {
            // Récupération du flux XML renvoyé par le serveur
            var xmlResponse = xmlHttp.responseXML;

            // Obtention du "doc element" (l'élément "racine") du flux XML
            var xmlDocumentElement = xmlResponse.documentElement;

            // Récupération du msg qui se trouve être le premier noeud enfant
            // dans le flux XML
            var helloMessage = xmlDocumentElement.firstChild.data;

            var reponse = '' ;
            if (xmlDocumentElement.getAttribute('status') == 'KO') {
                reponse = 'ERREUR : ' + helloMessage ;
            } else {
                reponse = '<i>Vous vous appelez '+helloMessage+', Bienvenue !</i>' ;
            }

            // update the client display using the data received from the server
            document.getElementById("divMessage").innerHTML = reponse ;

        } else {
            // Récupération du message d'erreur
            document.getElementById("divMessage").innerHTML +=
                    '<br><i>ANOMALIE AJAX : ' + xmlHttp.statusText + ' </i>';
        }
    }
}

var xmlHttp = createXhrObject();

document.addEventListener('DOMContentLoaded', function ( ) {
    "use strict";
    if (xmlHttp !== false) {

        var ajaxSubmit = document.getElementById('startAjax');

        ajaxSubmit.addEventListener('click', function (evt) {
            processAjax();
        }, false);

    }
}, false);
```

Code source de ajax_quickstart03.php :

```
/*
 * on commence par récupérer la valeur de $_GET qui nous permettra de
 * faire une recherche dans notre liste d'identifiants
 */
$id = strtoupper(trim($_GET['name']));

/*
 * liste d'identifiants fictifs, pour info, les 4 larrons en dessous de GJA
 * sont des membres du meilleur groupe de musique électro français : SMOOTH
 * et les 2 derniers ont sorti un album incroyable en 2009 : UDOLPHO
 */
$userNames = array(
    'GJA' => 'Gregory Jarrige',
    'NBE' => 'Nicolas Berrivin',
    'DDA' => 'David Darricarrère',
    'CDE' => 'Christophe Declercq',
    'ACH' => 'Alain Chauvet',
    'MMO' => 'Marc Morvan',
    'BJA' => 'Ben Jarry'
);

if ($id == '') {
    $reponse = 'Avec un identifiant ce serait beaucoup mieux';
    $statut = 'KO';
} else {
    if (array_key_exists($id, $userNames)) {
        $reponse = $userNames[$id];
        $statut = 'OK';
    } else {
        $reponse = "Vade Retro espèce d'ananas";
        $statut = 'KO';
    }
}

/*
 * Il est très important de renvoyer un "header" indiquant au navigateur
 * la nature du flux qu'il va recevoir, en l'occurrence du XML :
 */
header('Content-Type: text/xml');
// Génération de l'entête du XML
echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
// Création de l'élément XML "response"
echo '<response status="' . $statut . '">';
// Utiliser htmlentities pour l'échappement de certains caractères
echo htmlspecialchars($reponse, ENT_QUOTES, "UTF-8");
// Fermeture de l'élément "response"
echo '</response>';
```

Le dernier bloc de code est important et il est souhaitable que l'on s'y attarde quelques instants. La fonction addEventListener() appliquée à l'objet "document", met en place un écouteur d'évènement qui va consister à attendre que le DOM soit complètement "chargé" (cf. premier paramètre "DOMContentLoaded"), avant de passer à l'exécution du code fourni par la fonction anonyme qui est transmise en second paramètre.

ATTENTION :
Le troisième paramètre de la fonction addEventListener() est fixé à "false". Son utilisation est hors du périmètre de ce cours dédié à l'étude d'AJAX, et en règle générale on ne le fixera à "true" que dans de très rares occasions. Certains navigateurs considèrent que ce paramètre est optionnel et que s'il n'est pas transmis il est à "false" par défaut. Mais d'autres navigateurs (dont Firefox) se plantent si ce paramètre est omis, il est donc important de le transmettre systématiquement, même s'il est toujours fixé à "false".

POINT IMPORTANT :
La méthode addEventListener() n'est pas connue des navigateurs de la famille IE antérieurs à la version 9. Il existe une solution de repli avec la méthode attachEvent() qui fonctionne de manière sensiblement équivalente, pour les navigateurs de la famille IE. Pour obtenir un code compatible avec plusieurs générations de navigateurs, on pourra préférer le code ci-dessous au code précédent:

```
document.body.onload = function() {
    "use strict";
    if (xmlHttp !== false) {

        var ajaxSubmit = document.getElementById('startAjax');

        if (ajaxSubmit.addEventListener) {
           ajaxSubmit.addEventListener('click', function (evt) {
               processAjax();
           }, false);
        } else {
           // Pour les navigateur IE < à la version 9
           ajaxSubmit.attachEvent('click', function (evt) {
               processAjax();
           });
        }

    }
}, false);
```

DERNIER POINT IMPORTANT :

La structuration du code XML, au niveau du script PHP, est laissée à votre entière discrétion. Vous pouvez en effet le structurer de différentes manières, selon que vous souhaitez renvoyer une seule ligne d'information (comme c'était le cas dans cet exemple), ou une liste de données (par exemple pour le remplissage d'un tableau HTML). Au niveau du code PHP, on a utilisé la méthode la plus simple de création de flux XML, sachant que l'on peut aussi s'appuyer sur des fonctions plus sophistiquées qui sont étudiées en détail dans le cours dédié à PHP5.

En ce qui concerne le flux XML lui-même, on a choisi de créer un flux contenant un seul élément XML, avec un statut (OK ou KO) transmis en tant qu'attribut, et un message transmis en tant que "data". Ce choix arbitraire nous donnait l'occasion de manipuler plusieurs éléments du flux XML. On aurait tout aussi bien pu créer 2 éléments XM distincts, un pour transmettre le statut, et l'autre pour transmettre le message (contenant un nom ou un message d'erreur).

3.5 Compléments sur XML

Le format XML a été beaucoup utilisé pour le développement de requêtes de type AJAX. Mais il est aujourd'hui détrôné par JSON, format d'échange plus compact, tout en restant lisible, et surtout reconnu nativement par Javascript, alors que XML ne l'est pas.

Néanmoins, XML conserve un avantage car il peut être manipulé avec les méthodes getElementByid(), getElementByTagName(), ou encore par les méthodes querySelector() et querySelectorAll(), que tout développeur Javascript connaît bien.

Exemple de manipulation de réponse au format XML :

```
    // quand readyState est à 4, nous pouvons lire la réponse du serveur
    if(xmlHttp.readyState == 4) {
        // on continue seulement si HTTP status est "OK"
        if(xmlHttp.status == 200) {
            // lecture de la réponse
            response = xmlHttp.responseXML;
      ...
```

Soit la réponse XML suivante :

```
<books>
         <book id="89">
           <filename>JavaScript_Cookbook.pdf</filename>
           <title>JavaScript Cookbook</title>
           <author>Shelley Powers</author>
           <subject>JavaScript Cookbook</subject>
         </book>
         <book id="91">
           <filename>JavaScript_Patterns.pdf</filename>
           <title>JavaScript Patterns</title>
           <author>Stoyan Stefanov</author>
           <subject>JavaScript Patterns</subject>
         </book>
</books>
```

Il va être possible de "parser" le code XML renvoyé par l'objet XMLHttpRequest, et de le convertir en objet, au moyen de la fonction suivante :

```
function parseXML(responseXML) {
	var books = [];
	var bookNodes = responseXML.getElementsByTagName('books');
	for (var i = 0, len = userNodes.length; i < len; i++) {
		books[i] = {
			id: bookNodes[i].getAttribute('id'),
			title: bookNodes[i].getAttribute('title'),
			author: bookNodes[i].getAttribute('author'),
		};
	}
	return books;
}
```

3.6 Introduction à JSON

Quelques remarques préalables par rapport au format JSON.
La définition Wikipédia indique ceci en introduction :

JSON (JavaScript Object Notation) est un format de données textuelles, générique, dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. Créé par Douglas Crockford entre 2002 et 2005, il est décrit par la RFC 4627 de l'IETF.

Moins verbeux que le XML, donc plus compact, le JSON a supplanté peu à peu le XML dans les échanges de données spécifiques à AJAX.

La définition Wikipédia fournit quelques informations intéressantes :

Un document JSON a pour fonction de représenter de l'information accompagnée d'étiquettes permettant d'en interpréter les divers éléments, sans aucune restriction sur le nombre de celles-ci.

Un document JSON ne comprend que deux types d'éléments structurels :

- *des ensembles de paires nom / valeur ;*
- *des listes ordonnées de valeurs.*

Ces mêmes éléments représentent trois types de données :

- *des objets ;*
- *des tableaux ;*
- *des valeurs génériques de type tableau, objet, booléen, nombre, chaîne ou null.*

Exemple de jeu de données au format XML :

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Le même jeu de données au format JSON :

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateNewDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Close", "onclick": "CloseDoc()" }
            ]
        }
    }
}
```

Le langage PHP fournit 2 fonctions intéressantes pour manipuler le JSON : json_encode() et json_decode().

Exemple de tableau associatif tel que l'on peut écrire en PHP :

```
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
```

La fonction PHP json_encode() nous permet de convertir très simplement le tableau $arr en JSON :

```
echo json_encode($arr);  // renvoie : {"a":1,"b":2,"c":3,"d":4,"e":5}
```

PHP possède également une fonction permettant d'effectuer l'opération inverse :

```
$json = ('{"a":1,"b":2,"c":3,"d":4,"e":5}') ;
$arr = json_decode ($json) ;
         // renvoie : array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
```

Côté Javascript, on dispose de la fonction JSON.parse() qui nous permet de transformer en objet Javascript des données au format JSON :

```
var donnees = JSON.parse(donnees_json);
```

La plupart des navigateurs récents intègrent cette foncion en standard, mais certains navigateurs plus anciens - notamment les versions de IE antérieures à la 8 - ne disposent pas de cette fonction. On peut néanmoins contourner (non sans risque !) le problème en utilisant la solution de repli suivante :

```
 var donnees = eval('('+donnees_json+')');
```

Pourquoi "non sans risque" ? Parce que le contenu de la variable "donnees_json" peut très bien contenir autre chose que des données au format JSON, comme par exemple du code Javascript malveillant (ce qui constitue une sérieuse faille de sécurité).

Si vous êtes contraint de prendre en compte ce problème de compatibilité, vous pouvez écrire le code suivant :

```
var donnees = '' ;
if (JSON.parse) {
   // utilisation de la fonction JSON.parse() quand elle est disponible
   donnees = JSON.parse(donnees_json);
} else {
   // solution de repli pour les vieux navigateurs (pas sécurisée)
   donnees = eval('('+donnees_json+')');
}
```

La méthode JSON.stringify() fait le travail inverse de JSON.parse(), à savoir qu'elle transforme n'importe quel objet ou tableau en chaîne au format JSON (on dit que l'objet ou le tableau est "sérialisé").

Exemple :

```
var dog = {
       name: "Fido",
       dob: new Date(),
       legs: [1, 2, 3, 4]
};
var jsonstr = JSON.stringify(dog);
console.log(jsonstr) ; // {"name":"Fido","dob":"2014-02-
26T00:54:20.476Z","legs":[1,2,3,4]}
```

En résumé, voici un exemple de code manipulant des données au format JSON :

```
// une donnée au format JSON
var jstr = '{"mykey": "my value"}';

// antipattern (à éviter)
var data = eval('(' + jstr + ')');
console.log(data.mykey); // "my value"

// solution à privilégier
var data = JSON.parse(jstr);
console.log(data.mykey); // "my value"
```

Le framework jQuery embarque sa propre méthode de parsing du format JSON :

```
// an input JSON string
var jstr = '{"mykey": "my value"}';
var data = jQuery.parseJSON(jstr);
console.log(data.mykey); // "my value"
```

Le framework YahooUI embarque lui aussi sa propre méthode :

```
//an input JSON string
var jstr = '{"mykey": "my value"}';
// parse the string and turn it into an object
// using a YUI instance
YUI().use('json-parse', function (Y) {
	var data = Y.JSON.parse(jstr);
	console.log(data.mykey); // "my value"
});
```

3.7 Quatrième exemple (JSON/PHP)

Dupliquons le 3ème exemple en créant les fichiers suivants :
- page04.html
- ajax_quickstart04.js
- ajax_quickstart04.php

Et adaptons légèrement le code pour travailler cette fois-ci avec le format JSON.

Code du fichier page04.html :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>4ème exercice AJAX (avec JSON)</title>
        <meta charset="UTF-8">
        <script type="text/javascript" src="ajax.js"></script>
        <script type="text/javascript" src="ajax_quickstart04.js"></script>
    </head>
    <body>
        <label for="year">Saisissez une année</label>
        <input type="text" id="year" />
        <input type="button" id="startAjax" value="Recherche" />
        <div id="divMessage" ></div>
    </body>
</html>
```

Code du fichier ajax_quickstart04.js :

```
// création d'une requête HTTP asynchrone
function processAjax() {
    "use strict";
    // Traitement si l'objet XHR n'est pas occupé
    if (xmlHttp.readyState === 4 || xmlHttp.readyState === 0) {
        // On récupère l'identifiant saisi dans le champ "yourID"
        var year = encodeURIComponent(document.getElementById("year").value);
        // Exécute le script quickstart.php sur le serveur
        xmlHttp.open("GET", "ajax_quickstart04.php?year=" + year, true);
        // Définition de la fonction JS à appeler une fois la requête aboutie
        xmlHttp.onreadystatechange = handleServerResponse;
        // Envoi de la requête au serveur
        xmlHttp.send(null);
    } else {
        // Si la connexion avec le serveur est occupée,
        //   on retente le coup au bout d'une seconde
        setTimeout('process()', 1000);
    }
}
```

```
// Fonction exécutée automatiquement lorsque la réponse du serveur arrive
function handleServerResponse() {
    // Si readystate = 4 c'est que la requête a complètement abouti
    if (xmlHttp.readyState === 4) {
        // Le statut 200 indique que la requête HTTP s'est bien passée
        if (xmlHttp.status === 200) {
            var json_data = JSON.parse(xmlHttp.responseText) ;
            console.log(json_data);

            // Mise à jour de la div bloc01 en fonction du texte récupéré
            document.getElementById("divMessage").innerHTML =
                    '<i> ' + xmlHttp.responseText + ' </i>';
        } else {
            // Récupération du message d'erreur
            document.getElementById("divMessage").innerHTML +=
                    '<br><i>Décidément AJAX, ça ne marche pas : ' +
                        xmlHttp.statusText + ' </i>';

        }
    }
}

var xmlHttp = createXhrObject();

document.addEventListener('DOMContentLoaded', function ( ) {
    "use strict";
    if (xmlHttp !== false) {

        var ajaxSubmit = document.getElementById('startAjax');

        ajaxSubmit.addEventListener('click', function (evt) {
            processAjax();
        }, false);

    }
}, false);
```

Code du fichier ajax_quickstart04.php :

```
<?php

/*
 * on commence par récupérer la valeur de $_GET qui nous permettra de
 * faire une recherche dans notre liste d'albums sur le critère de l'année
 */
$year = intval($_GET['year']);
```

```
/*
 * Tableau associatif présentant une série d'albums
 */
$theBestCDs = array(
    0 => array('title'=>'The Parade', 'artist'=>'Smooth', 'year'=>2010),
    1 => array('title'=>'An Electro soul experience', 'artist'=>'Smooth', 'year'=>2005),
    2 => array('title'=>'The Endless rise of the sun', 'artist'=>'Smooth', 'year'=>2006),
    3 => array('title'=>'Beyond Man and Time', 'artist'=>'RPWL', 'year'=>2012),
    4 => array('title'=>'We were dead before the ship even sank', 'artist'=>'Modest Mouse',
        'year'=>2007),
    5 => array('title'=>'Udolpho', 'artist'=>'Marc Morvan et Ben Jarry', 'year'=>2009),
    6 => array('title'=>'Maudits Français', 'artist'=>'JAVA', 'year'=>2009),
    7 => array('title'=>'Nightbook', 'artist'=>'Ludovico Einaudi', 'year'=>2009),
    8 => array('title'=>'Grace', 'artist'=>'Jeff Buckley', 'year'=>1994),
    9 => array('title'=>'Just a Poke', 'artist'=>'Sweet Smoke', 'year'=>1974),
    10 => array('title'=>'The Big Picture', 'artist'=>'Michael Shrieve & David Beal',
        'year'=>1989),
    11 => array('title'=>'Inaxycvgtgb', 'artist'=>'16 BIT', 'year'=>1987),
    12 => array('title'=>'When the guns come out', 'artist'=>'Hanni El Khatib', 'year'=>2011),
    13 => array('title'=>'The seldom seen kid', 'artist'=>'Elbow', 'year'=>2008),
    14 => array('title'=>'The something rain', 'artist'=>'Tindersticks', 'year'=>2012),
    15 => array('title'=>'In this light and on this evening', 'artist'=>'Editors',
       'year'=>2009),
    16 => array('title'=>'Head in the dirt', 'artist'=>'Hanni El Khatib', 'year'=>2013)
);

$liste = array();
if ($year == 0) {
    $reponse = 'Critère de recherche non transmis';
    $statut = 'KO';
    $liste [] = array('error'=>'année inexistante');
} else {
    $statut = 'OK';
    foreach ($theBestCDs as $album) {
        if ($album['year'] == $year) {
            $liste [] = $album ;
        }
     }
}

/*
 * Il est très important de renvoyer un "header" indiquant au navigateur
 * la nature du flux qu'il va recevoir, en l'occurrence du JSON, donc du texte :
 */
header('Content-Type: text/text');
echo json_encode ($liste) ;
```

3.8 Cinquième exemple (requête de type POST)

Jusqu'ici nous avons exclusivement utilisé des requêtes HTTP de type GET.
Nous allons maintenant étudier un exemple de requête AJAX utilisant les requêtes HTTP de type POST.

Pour ce faire, dupliquons le 4ème exemple en créant les fichiers suivants :
- page05.html
- ajax_quickstart05.js
- ajax_quickstart05.php

Et adaptons légèrement le code.

Code du fichier page05.html :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>4ème exercice AJAX (avec JSON)</title>
        <meta charset="UTF-8">
        <script type="text/javascript" src="ajax.js"></script>
        <script type="text/javascript" src="ajax_quickstart05.js"></script>
    </head>
    <body>
        <form>
        <fieldset><legend>Formulaire de recherche</legend>
        <label for="year">Saisissez une année</label>
        <input type="text" id="year" ><br>
        <label for="titre">Saisissez un titre</label>
        <input type="text" id="titre" ><br>
        <label for="groupe">Saisissez un groupe</label>
        <input type="text" id="groupe" ><br>
        <input type="submit" id="startAjax" value="Recherche" >
        </fieldset>
        </form>
        <div id="divMessage" ></div>
    </body>
</html>
```

Code du fichier ajax_quickstart05.js :

```
// création d'une requête HTTP asynchrone
function processAjax() {
    "use strict";
    // Traitement si l'objet XHR n'est pas occupé
    if (xmlHttp.readyState === 4 || xmlHttp.readyState === 0) {
        formatOutput = '' ;
        // récupération des champs du formulaire de recherche
        var params = [] ;
        var parent = document.getElementById('startAjax').parentNode ;
        var inputs = parent.querySelectorAll("input[type='text']");
        [].forEach.call(inputs, function(element, index){
            params.push(encodeURIComponent(element.id) + '=' +
                        encodeURIComponent(element.value)) ;
        }) ;
        var data = params.join("&");
        // Exécute le script quickstart.php sur le serveur
        xmlHttp.open("POST", "ajax_quickstart05.php" , true);
        // Définition de la fonction JS à appeler une fois la requête aboutie
        xmlHttp.onreadystatechange = handleServerResponse;
        xmlHttp.setRequestHeader('Content-Type',
                 'application/x-www-form-urlencoded');
        xmlHttp.setRequestHeader('Content-Length', data.length);
        // Envoi de la requête au serveur
        xmlHttp.send(data);

    } else {
        // Si la connexion avec le serveur est occupée,
        //    on retente le coup au bout d'une seconde
        setTimeout('process()', 1000);
    }
}

// Fonction exécutée automatiquement lorsque la réponse du serveur arrive
function handleServerResponse() {
    // Si readystate = 4 c'est que la requête a complètement abouti
    if (xmlHttp.readyState === 4) {
        // Le statut 200 indique que la requête HTTP s'est bien passée
        if (xmlHttp.status === 200) {

            var reponse = xmlHttp.responseText ;

            // Mise à jour de la div bloc01 en fonction du texte récupéré
            document.getElementById("divMessage").innerHTML =
                    '<i> ' + reponse + ' </i>';
        } else {
            // Récupération du message d'erreur
            document.getElementById("divMessage").innerHTML +=
                    '<br><i>Décidément AJAX, ça ne marche pas : ' +
                         xmlHttp.statusText + ' </i>';
```

```
        }
    }
}


var xmlHttp = createXhrObject();
var formatOutput = 'html';

document.addEventListener('DOMContentLoaded', function ( ) {
    "use strict";
    if (xmlHttp !== false) {

        var ajaxSubmit = document.getElementById('startAjax');

        ajaxSubmit.addEventListener('click', function (evt) {
            evt.preventDefault() ;
            evt.stopPropagation() ;
            processAjax();
        }, false);

    }
}, false);
```

Code du fichier ajax_quickstart05.php :

```
<?php

$params = array();
$params['year'] = isset($_POST['year']) ? intval($_POST['year']) : 0 ;
$params['titre'] = isset($_POST['titre']) ?
strtolower(htmlentities(trim($_POST['titre']))) : '' ;
$params['groupe'] = isset($_POST['groupe']) ?
strtolower(htmlentities(trim($_POST['groupe']))) : '' ;

/*
 * Tableau associatif présentant une série d'albums
 */
$theBestCDs = array(
    0 => array('title'=>'The Parade', 'artist'=>'Smooth', 'year'=>2010),
    1 => array('title'=>'An Electro soul experience', 'artist'=>'Smooth', 'year'=>2005),
    2 => array('title'=>'The Endless rise of the sun', 'artist'=>'Smooth', 'year'=>2006),
    3 => array('title'=>'Beyond Man and Time', 'artist'=>'RPWL', 'year'=>2012),
    4 => array('title'=>'We were dead before the ship even sank', 'artist'=>'Modest Mouse',
          'year'=>2007),
    5 => array('title'=>'Udolpho', 'artist'=>'Marc Morvan et Ben Jarry', 'year'=>2009),
    6 => array('title'=>'Maudits Français', 'artist'=>'JAVA', 'year'=>2009),
    7 => array('title'=>'Nightbook', 'artist'=>'Ludovico Einaudi', 'year'=>2009),
    8 => array('title'=>'Grace', 'artist'=>'Jeff Buckley', 'year'=>1994),
    9 => array('title'=>'Just a Poke', 'artist'=>'Sweet Smoke', 'year'=>1974),
    10 => array('title'=>'The Big Picture', 'artist'=>'Michael Shrieve & David Beal',
         'year'=>1989),
    11 => array('title'=>'Inaxycvgtgb', 'artist'=>'16 BIT', 'year'=>1987),
    12 => array('title'=>'When the guns come out', 'artist'=>'Hanni El Khatib', 'year'=>2011),
    13 => array('title'=>'The seldom seen kid', 'artist'=>'Elbow', 'year'=>2008),
    14 => array('title'=>'The something rain', 'artist'=>'Tindersticks', 'year'=>2012),
    15 => array('title'=>'In this light and on this evening', 'artist'=>'Editors',
        'year'=>2009),
    16 => array('title'=>'Head in the dirt', 'artist'=>'Hanni El Khatib', 'year'=>2013)
);

$liste = array();
if ($params['year'] == 0 && $params['titre'] == '' && $params['groupe'] == '') {
    $reponse = 'Critères de recherche non transmis';
    $statut = 'KO';
} else {
    $statut = 'OK';
    foreach ($theBestCDs as $album) {
        $comptage = 0 ;
        if ($params['year'] == 0 || $album['year'] == $params['year']) {
            if ($params['groupe'] == '' ||
               strtolower($album['artist']) == $params['groupe']) {
                if ($params['titre'] == '' ||
                   strtolower($album['title']) == $params['titre'] ) {
                    $liste [] = $album ;
                }
            }
        }
```

```
     }
}

/*
 * Il est très important de renvoyer un "header" indiquant au navigateur
 * la nature du flux qu'il va recevoir, en l'occurrence du JSON, donc du texte :
 */
header('Content-Type: text/text');

echo '<table border="1" width="100%" cellspacing="0" cellpadding="5"
><tr><td>Titre</td><td>Groupe</td><td>Année</td></tr>'.PHP_EOL ;
foreach ($liste as $key_liste=>$key_value) {
    echo '<tr>' ;
    echo '<td>' . $key_value['title'] . '</td>' ;
    echo '<td>' . $key_value['artist'] . '</td>' ;
    echo '<td>' . $key_value['year'] . '</td>' ;
    echo '</tr>'. PHP_EOL ;
}
echo '</table>'.PHP_EOL ;
```

Exemple de sélection :

Formulaire de recherche

Saisissez une année 2009

Saisissez un titre

Saisissez un groupe

Recherche

Titre	*Groupe*	*Année*
Udolpho	*Marc Morvan et Ben Jarry*	*2009*
Maudits Français	*JAVA*	*2009*
Nightbook	*Ludovico Einaudi*	*2009*
In this light and on this evening	*Editors*	*2009*

3.9 Codes retours de XHR

La liste des codes retour les plus utilisés sur le protocole HTTP est la suivante :

- 200 (OK, Requête traitée avec succès)
- 401 (non autorisé)
- 403 (droits d'accès insuffisants)
- 404 (ressource introuvable : l'URL demandée ne correspond à aucun fichier côté serveur)
- 500 (erreur interne : le code exécuté côté serveur comporte une erreur empêchant son exécution).

La liste complète des codes http est disponible sur le site Wikipedia :
http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

4 Annexe

4.1 Liens utiles

http://www.w3schools.com/ajax/

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

http://www.html5rocks.com/en/tutorials/file/xhr2/

https://dev.opera.com/articles/xhr2/

https://www.w3.org/TR/XMLHttpRequest/

https://www.w3.org/TR/XMLHttpRequest2/

5 Changelog

Version 1.1

- Corrections sur 3ème exemple (code PHP manquant)

Version 1.2

- Ajout du chapitre 4 (Annexe) et du chapitre 4.1 (Liens utiles)