**eBook Gratuit**

# APPRENEZ ajax

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#ajax

# Table des matières

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ajax

It is an unofficial and free ajax ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ajax.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapitre 1: Commencer avec ajax

## Remarques

AJAX **(a** synchrone **J** avascript **un** e **X** ML) permet de demander des données externes **sans** bloquer l'exécution du code. Dans de nombreux cas, ceci est implémenté dans la demande d'éléments d'une page ou d'informations d'un serveur (via XMLhttpRequests), puis en les traitant et les affichant en utilisant javascript.

La nature non bloquante d'AJAX est ce qui en fait un modèle logiciel si répandu. Comme le javascript est bloqué dans le navigateur, un appel externe synchrone ne répond plus pendant la durée de l'appel jusqu'à ce qu'il renvoie des données ou expire. En effet, rendre votre application entièrement dépendante de l'architecture externe et de son efficacité.

Les appels AJAX sont généralement extraits pour fournir des fonctionnalités ou une lisibilité supplémentaires, mais les implémentations sont (généralement) basées sur la spécification XMLHttpRequest .

## Examples

### Installation ou configuration

## Qu'est ce que AJAX?

> AJAX signifie JavaScript et XML asynchrones. En bref, c'est l'utilisation de l'objet XMLHttpRequest pour communiquer avec des scripts côté serveur. Il peut envoyer et recevoir des informations dans divers formats, notamment des fichiers JSON, XML, HTML et même des fichiers texte. -Mozilla Developer Network 2016

La manière la plus simple d'implémenter AJAX, en particulier si vous envisagez de communiquer avec les serveurs, est d'utiliser jQuery.

## Qu'est-ce que jQuery?

> jQuery est une bibliothèque JavaScript rapide, petite et riche en fonctionnalités. Grâce à une API facile à utiliser qui fonctionne sur une multitude de navigateurs, les tâches telles que la traversée et la manipulation de documents HTML, la gestion des événements, l'animation et Ajax sont beaucoup plus simples. -jquery.com

Pour ceux qui n'ont pas beaucoup utilisé jQuery, pensez-y comme des fonctions que nous pouvons utiliser pour nous faciliter la vie. Ceci est parfait pour utiliser avec AJAX car il réduit la quantité de code que nous devons écrire pour accomplir la même chose!

# Comment ajouter jQuery à votre site Web

Si vous devez utiliser Ajax, vous devez ajouter jQuery à votre projet. http://jquery.com/download/ Dans ce lien, vous pouvez voir de nombreuses façons d'ajouter jquery. Vous pouvez utiliser la version téléchargée de jQuery ou utiliser un CDN. http://jquery.com/download/#jquery-39-s-cdn-provided-by-maxcdn . Mais si vous utilisez le CDN, il existe un risque de sécurité. Étant donné que le projet a appelé à utiliser jquery, un pirate pourrait manipuler l'appel. Donc mieux si vous pouviez utiliser la version téléchargée. Voyons comment ajouter jquery au projet HTML. C'est facile. Le premier exemple est d'utiliser la source téléchargée. Utilisez ce lien pour télécharger http://jquery.com/download/#jquery . Si vous voulez juste utiliser jquery, je vous suggère de télécharger **Download the compressed, production jQuery 3.1.1** Lorsque vous le téléchargez, ajoutez jquery-version.min.js à l'endroit approprié (comme le dossier javascript de votre projet). avec src = jquery / location comme ci-dessous.

```
<head>

<script src="path/from/html/page/to/jquery.min.js"></script>

</head>
```

Voyons comment utiliser un CDN. Ce lien http://jquery.com/download/#using-jquery-with-a-cdn vous permet de voir divers CDN (Content Delivery Network).

```
<head>

<script   src="https://code.jquery.com/jquery-3.1.1.min.js"   integrity="sha256-
hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="   crossorigin="anonymous"></script>

</head>
```

Comme vous pouvez le voir, il vous suffit d'ajouter les balises que le fournisseur de CDN fournit au serveur. Ajoutez maintenant des scripts à la page HTML pour vérifier que cela fonctionne.

```
<script>
$(document).ready(function(){
    alert("jQuery Works")
});
</script>
```

Si vous voyez le **jQuery fonctionne** alerte, cela signifie que vous l'avez ajouté correctement.

## Exemple simple de jQuery pour communiquer avec le serveur

Tiré du site Web de l' API `jQuery.ajax` :

```
$.ajax({
    method: "POST",
    url: "some.php",
    data: {
        name: "John",
```

```
        location: "Boston"
    },
    success: function(msg) {
        alert("Data Saved: " + msg);
    },
    error: function(e) {
        alert("Error: " + e);
    }
});
```

Ce morceau de code, dû à jQuery, est facile à lire et à comprendre ce qui se passe.

- `$.ajax` - Ce bit appelle la fonctionnalité `ajax` de jQuery.
- `method: "POST"` - cette ligne déclare ici que nous allons utiliser une méthode POST pour communiquer avec le serveur. Lisez sur les types de demandes!
- `url` - cette variable déclare où la requête va être **envoyée** . Vous envoyez une demande quelque part. C'est l'idée.
- `data` - assez simple. Ce sont les données que vous envoyez avec votre demande.
- `success` - cette fonction ici vous écrivez pour décider quoi faire avec les données que vous recevez en retour `msg` ! Comme le suggère l'exemple, il ne fait que créer une alerte avec le `msg` renvoyé.
- `error` - cette fonction vous écrivez ici pour afficher des messages d'erreur, ou pour fournir des actions à travailler lorsque la requête `ajax` est passée par des erreurs.
- une alternative à `.done` est

```
success: function(result) {
    // do something
});
```

## Sync - Requêtes asynchrones ajax

### Appel ajax asynchrone

Avec ce type d'appel ajax, le code n'attend pas la fin de l'appel.

```
$('form.ajaxSubmit').on('submit',function(){
    // initilization...
    var form   = $(this);
    var formUrl    = form.attr('action');
    var formType   = form.attr('method');
    var formData   = form.serialize();

    $.ajax({
        url: formUrl,
        type: formType,
        data: formData,
        async: true,
```

```
        success: function(data) {
            // .....
        }
    });
    //// code flows through without waiting for the call to complete
    return false;
});
```

**Appel ajax synchrone**

Avec ce type d'appel ajax, le code attend la fin de l'appel.

```
$('form.ajaxSubmit').on('submit',function(){
    // initilization...
    var form   = $(this);
    var formUrl    = form.attr('action');
    var formType   = form.attr('method');
    var formData   = form.serialize();
    var data = $.ajax({
            url: formUrl,
            type: formType,
            data: formData,
            async: false
        }).responseText;
    //// waits for call to complete
    return false;
});
```

## Requête Ajax simple envoyée à l'aide de l'objet XMLHttpRequest

```
var httpRequest = new XMLHttpRequest();

httpRequest.onreadystatechange = getData;

httpRequest.open('GET', 'https://url/to/some.file', true);
httpRequest.send();

function getData(){
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        alert(httpRequest.responseText);
    }
}
```

`new XMLHttpRequest()` crée un nouvel objet *XMLHttpRequest* - c'est ce que nous allons envoyer notre demande avec

Le bit `onreadystatechange` indique à notre demande d'appeler `getData()` chaque changement de statut

`.open()` crée notre demande - cela prend une *méthode de demande* ( « **GET** », « **POST** », etc.), une URL de la page que vous interrogez, et le cas échéant, si la demande doit être ou non **asynchrynous**

`.send()` envoie notre requête - accepte éventuellement les données à envoyer au serveur comme `.send(data)`

Enfin, le `getData()` est la fonction que nous avons dit devoir appeler chaque fois que le **statut de** notre requête **change** . Si le readyState est égal à **DONE,** il alerte le `responseText` qui n'est que les données reçues du serveur.

Vous trouverez plus d'informations dans le guide de mise en route sur MDN.

## Utiliser ajax en javascript avec un simple rappel

Voici notre fonction pour créer un simple appel ajax écrit en javascript vanilla (pas es2015):

```
function ajax(url, callback) {
    var xhr;

    if(typeof XMLHttpRequest !== 'undefined') xhr = new XMLHttpRequest();
    else {
        var versions = ["MSXML2.XmlHttp.5.0",
                        "MSXML2.XmlHttp.4.0",
                        "MSXML2.XmlHttp.3.0",
                        "MSXML2.XmlHttp.2.0",
                        "Microsoft.XmlHttp"]

         for(var i = 0, len = versions.length; i < len; i++) {
            try {
                xhr = new ActiveXObject(versions[i]);
                break;
            }
            catch(e){}
         } // end for
    }

    xhr.onreadystatechange = ensureReadiness;

    function ensureReadiness() {
        if(xhr.readyState < 4) {
            return;
        }

        if(xhr.status !== 200) {
            return;
        }

        // all is well
        if(xhr.readyState === 4) {
            callback(xhr);
        }
    }

    xhr.open('GET', url, true);
    xhr.send('');
}
```

et il pourrait être utilisé comme:

```
ajax('myFile.html', function(response) {
    document.getElementById('container').innerHTML = response.responseText;
});
```

Si vous souhaitez utiliser Ecmascript 6 (également appelé es2015), vous pouvez utiliser la méthode **fetch** , qui renvoie une promesse:

```
fetch('myFile.json').then(function(res){
    return res.json();
 });
```

Pour en savoir plus sur es2015 Promesses suivez le lien ci-dessous: Promesses

### Exécution d'un appel AJAX asynchrone à l'aide de TypeScript

# Ajouter un produit à un panier

L'exemple suivant montre comment ajouter un produit (ou tout élément) à une table de base de données de manière asynchrone, à l'aide d'AJAX et de TypeScript.

```
declare var document;
declare var xhr: XMLHttpRequest;

window.onLoad = () =>
{
    Start();
};

function Start()
{
    // Setup XMLHttpRequest (xhr).
    if(XMLHttpRequest)
    {
        xhr = new XMLHttpRequest();
    }
    else
    {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }

    AttachEventListener(document.body, "click", HandleCheckBoxStateChange);
}

function AttachEventListener(element: any, e, f)
{
    // W3C Event Model.
    if(element.addEventListener)
    {
        element.addEventListener(e, f, false);
    }
    else if(element.attachEvent)
    {
        element.attachEvent("on" + e, (function(element, f)
        {
            return function()
            {
                f.call(element, window.event);
            };
        })
```

```
        (element, f));
    }

    element = null;
}

function HandleCheckBoxStateChange(e)
{
    var element = e.target || e.srcElement;

    if(element && element.type == "checkbox")
    {
        if(element.checked)
        {
            AddProductToCart(element);
        }
        else
        {
            // It is un-checked.
            // Remove item from cart.
        }
    }
    else
    {
        break;
    }
}

AddProductToCart(e)
{
    var element = <HTMLInputElement>document.getElementById(e.id);

    // Add the product to the Cart (Database table)
    xhr.onreadystatechange = function()
    {
        if(xhr.readyState == 4)
        {
            if(xhr.status == 200)
            {
                if(element != null)
                {
                    console.log("200: OK");
                }
                else
                {
                    console.log(":-(");
                }
            }
            else
            {
                // The server responded with a different response code; handle accordingly.
                // Probably not the most informative output.
                console.log(":-(");
            }
        }
    }

    var parameters = "ProductID=" + encodeURIComponent(e.id) + "&" + "Action=Add&Quantity=" +
element.value;

    xhr.open("POST", "../Cart.cshtml");
```

```
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.setRequestHeader("Content-length", parameters.length.toString());
    xhr.setRequestHeader("Connection", "close");

    xhr.send(parameters);

    return e.id;
}
```

Pour que cet exemple soit complet, mettez à jour votre code côté serveur pour insérer ces données dans la base de données. Le code ci-dessus suppose que vous utilisez C # et que vous avez un fichier `Cart.cshtml` . Cependant, remplacez simplement `cshtml` par `php` et écrivez votre propre logique côté serveur en utilisant la langue de votre choix.

Lire Commencer avec ajax en ligne: https://riptutorial.com/fr/ajax/topic/1082/commencer-avec-ajax

# Chapitre 2: Rappels

## Examples

### Interprétation des erreurs avec un rappel "erreur"

Les erreurs, correctement gérées par le serveur, seront renvoyées à votre client avec un code d'état HTTP spécifique différent de 2xx (voir la section 10 de la RFC 2616 ).

Il est conseillé d'attraper globalement vos erreurs de votre `$.ajaxSetup()` comme illustré dans l'exemple ci-dessous. Par conséquent, toutes les erreurs provenant de vos appels ajax seront automatiquement interprétées à partir de la configuration ajax.

```
$.ajaxSetup({
    error: function (jqXHR, exception, errorThrown) {
        var message;
        var statusErrorMap = {
            '400': "Server understood the request, but request content was invalid.",
            '401': "Unauthorized access.",
            '403': "Forbidden resource can't be accessed.",
            '500': "Internal server error.",
            '503': "Service unavailable."
        };
        if (jqXHR.status) {
            message = statusErrorMap[jqXHR.status];
            if (!message) {
                message = "Unknown Error.";
            }
        } else if (exception == 'parsererror') {
            message = "Error.\nParsing JSON Request failed.";
        } else if (exception == 'timeout') {
            message = "Request Time out.";
        } else if (exception == 'abort') {
            message = "Request was aborted by the server";
        } else {
            message = "Unknown Error.";
        }

        // How you will display your error message...
        console.log(message);
        console.log(errorThrown);
    }
});
```

Vous pouvez également vouloir "surcharger" le rappel d' `error` dans un `$.ajax()` spécifique lorsque vous attendez un message d'erreur spécifique.

```
$.ajax({
    url: './api',
    data: { parametersObject },
    type:'post',
    dataType: 'json',
    success:function(output){
```

```
        // Interpret success
    },
    error: function(xhr,textStatus,ErrorThrown){
        // Specific error will not be interpreted by $.ajaxSetup
    }
});
```

Lire Rappels en ligne: https://riptutorial.com/fr/ajax/topic/7175/rappels

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Commencer avec ajax | acupajoe, adielhercules, Alessio Cantarella, Community, delete me, JhWebDevGuy, jignesh prajapati, MAZux, Menuka Ishan, Nicholas Qiao, TimTheEnchanter, Tolga Evcimen |
| 2 | Rappels | maxime_039 |