

# AJAX cross domain

par Pierre Schwartz ([retour aux articles](#))


Date de publication : 21 octobre 2008


Dernière mise à jour :

Cet article va présenter une méthode permettant de réaliser dynamiquement des appels JavaScript vers des sites sur des domaines différents.

I - Le problème.....	3
II - AJAX sans XMLHttpRequest.....	3
III - Implémentation.....	4
IV - Risques.....	5
V - Conclusion.....	5

## I - Le problème

 **AJAX** se base sur l'objet JavaScript XMLHttpRequest qui permet de requêter dynamiquement une url via le protocole HTTP. L'un des avantages est de pouvoir échanger des données entre la page et un serveur sans avoir à recharger entièrement la page.

Outre l'immense champ de possibilités qui s'offre au développeur, quelques problèmes de sécurité apparaissent. Avec AJAX, la page web peut elle-même requêter le serveur, sans même que l'utilisateur s'en rende compte. C'est vrai qu'AJAX permet une bien meilleure ergonomie, mais ce n'est pas tout, AJAX permet aussi de lier une page web à un grand nombre d'autres pages web, c'est la porte ouverte aux attaques  **XSS** et aux vols de cookies : un appel AJAX peut rapatrier du code JavaScript qui va récupérer des cookies clients et les envoyer à un site tiers. C'est notamment l'une des raisons qui ont poussé les éditeurs de navigateurs à encadrer davantage l'usage de l'objet XMLHttpRequest, en particulier en restreignant les url AJAXables à celles du même domaine que la page web de départ. Tout appel à une url en dehors du domaine provoquera une erreur **uncaught exception: Access to restricted URI denied (NS\_ERROR\_DOM\_BAD\_URI)**


Bien qu'étant une mesure de sécurité, l'usage de requêtes d'un domaine à l'autre manque cruellement. Comment faire dans ce cas ?

La première solution consiste à utiliser un proxy situé sur le même domaine que la page qui l'utilisera. Ce proxy devra appeler toutes les url externes et il ne sera pas bloqué par le navigateur. Cependant cette solution n'est pas toujours utilisable.

JavaScript est un langage suffisamment puissant et varié pour nous permettre de contourner cette limitation tout en retombant sur nos pieds. Attention la solution que je vous propose est sujette à des attaques XSS si vous n'y prenez pas garde !

## II - AJAX sans XMLHttpRequest

La méthode que je propose n'utilise pas l'objet XMLHttpRequest. Il est bridé, nous devons donc passer outre. Quel serait un autre moyen JavaScript de récupérer du contenu dynamiquement ? L'idée est très simple : il suffit de rajouter dynamiquement des balises <script>. Ces balises permettent à l'origine d'inclure des fichiers javascript, quelques domaine qu'ils proviennent. Traditionnellement ces balises étaient fixées une bonne fois pour toute et n'évoluaient plus dès lors que le contexte JavaScript était chargé. Mais qu'en est-il si l'on rajoute des balises <script> ? Et bien le moteur JavaScript va les interpréter à la volée et inclura le résultat de l'appel dans le contexte JavaScript. Et bien évidemment rien ne vous oblige de faire pointer les balises script vers des fichiers .js statiques ... un fichier dynamique peut tout aussi bien faire l'affaire.

 *Le contexte JavaScript est l'état d'interprétation d'une page web, avec toutes les variables définies, leurs valeurs ainsi que les fonctions existantes.*

Il y a cependant une différence entre l'ajout dynamique de balises scripts et l'utilisation de l'objet XMLHttpRequest : l'ajout de balise ne renvoie rien, il se contente d'enrichir le contexte JavaScript. L'idée est alors d'appeler une fonction déjà connue avec le flux XML/JSON/autre qu'aurait renvoyé un appel AJAX classique. C'est notamment le choix fait par google pour l'implémentation de google suggest. Voici **un petit exemple** qui vous en convaincra. Il est évident que je n'ai pas copié la totalité des résultats google sur le domaine khayyam.developpez.com, l'appel sort donc du domaine pour aller interroger directement Google.

developpez -&gt; 2 370 000 résultats

developpement photo -&gt; 2 600 000 résultats

developpement photos -&gt; 883 000 résultats

développement durable -&gt; 9 140 000 résultats

developper photo -&gt; 4 060 000 résultats

developpement personnel -&gt; 3 090 000 résultats

developpement photo numerique -&gt; 922 000 résultats

developper des photos -&gt; 4 140 000 résultats

developpez.net -&gt; 1 290 000 résultats

developpement foetus -&gt; 409 000 résultats

Autant l'utilisation de l'objet XMLHttpRequest nous permet de manipuler des urls presque comme si elles étaient des webservices, fournissant simplement un contenu JSON (ou autre), autant l'approche est légèrement différente avec les balises script. En effet, le webservice se fiche complètement de la manière dont seront traitées les données qu'il renvoie, il ne renvoie que des données déclaratives ; alors qu'un contenu accédé par une balise script doit pouvoir être directement interprétable par le moteur javascript. Le résultat doit donc contenir par lui-même son moyen de traitement. C'est pour cette raison qu'il ne faut plus simplement renvoyer un flux de données, mais qu'il faut au moins les encapsuler dans un appel de fonction déjà connu du contexte courant (l'appel à une fonction non définie provoquera une erreur).

### III - Implémentation

Rien de vraiment compliqué, il suffit de pouvoir créer des balises script à la volée. Une manipulation classique du DOM le permet :

```
function callExternalScript(url){
    var n = document.createElement("script");
    n.setAttribute("type", "text/javascript");
    n.setAttribute("src", url);
    document.getElementsByTagName("head")[0].appendChild(n);
}
```

Il suffira d'appeler une url externe avec des paramètres pour avoir un contenu dynamique rajouté à la volée. L'implémentation va aussi contenir la déclaration d'une fonction à appeler depuis l'url externe, par exemple f(contenu dynamique). Chaque script externe devra passer son résultat à la fonction f. Rien de plus compliqué. Bien sûr, vous pouvez mettre en place une logique de vidage des balises script. En effet, chaque appel externe va enrichir le DOM avec un noeud script.

Google suggest répond sur l'url suivante

```
http://clients1.google.com/complete/search?hl=fr&q=developpez
```

des messages du type

```
window.google.ac.h(["developpez", [
  ["developpez.net", "1 290 000 résultats", "2"],
  ["developpez .com", "1 320 000 résultats", "3"],
  ["developpez forum", "4 040 000 résultats", "4"],
```

```
[ "developpez java", "1 430 000 résultats", "5"],  
[ "developpez php", "4 790 000 résultats", "6"],  
[ "developpez.com forum", "426 000 résultats", "7"],  
[ "developpez sql", "1 300 000 résultats", "8"],  
[ "developpez.fr", "71 résultats", "9"],  
[ "developpez javascript", "1 140 000 résultats", "10"],  
[ "développez photo", "1 980 000 résultats", "11"]  
])
```

Il suffit alors d'implémenter de votre côté une fonction `window.google.ac.h` qui traitera les données JSON. Rien de plus compliqué. Le code est même très concis :

```
window.google = {};  
window.google.ac = {};  
  
window.google.ac.h = function(content){  
    s.innerHTML = '';  
    var c = content[1];  
    for(i=0;i<c.length; i++){  
        var cc = c[i];  
        s.innerHTML += cc[0] + ' -> ' + cc[1] + '<br/>';  
    }  
}
```

où `s` représente l'objet DOM que vous souhaitez remplir.

## IV - Risques

Si vous ne contrôlez pas l'url externe (ce qui est le cas de l'exemple sur Google suggest), il suffira que l'url externe change sa manière de renvoyer les données pour que vous deviez la réimplémenter entièrement.

Rien n'oblige une url externe (non contrôlée) à appeler effectivement une fonction `f` avec un contenu bien formaté. Elle peut tout simplement aller modifier le DOM de la page appelante. Un `document.getElementsByTagName("body")[0].innerHTML` est si vite modifié ;). Et il n'y a aucune limite, la page appelante va inclure toute la page externe, même si elle contient des modifications massives de son DOM ou de son contexte JavaScript ... C'est pourquoi mon exemple de réimplémentation de Google suggest peut s'avérer dangereux dans le cas où Google ajouterait du code malveillant au milieu de ses suggestions.

Une fois un morceau de code malveillant ajouté dans un contexte JavaScript, il n'y a aucune barrière l'empêchant de récupérer tous les cookies du client et de les renvoyer de la même manière, en cross domain à une url tierce. Car ne l'oublions pas, si vous pouvez appeler une url externe, celle-ci peut aussi en appeler une.

Chaque appel externe est conservé dans le DOM sous la forme d'un noeud script, mais encore une fois, rien n'empêche une url externe malveillante d'effacer ces balises script en modifiant le DOM. Une fois qu'elles sont interprétées, le contexte JavaScript ne sera pas impacté par leur suppression. Et bien sûr, faire disparaître une balise script du DOM est le meilleur moyen pour effacer toute trace d'un appel externe.

## V - Conclusion

Bien que cette astuce permette de contourner efficacement une limitation des XMLHttpRequests, il est nécessaire de maîtriser les urls cross domain que vous allez appeler. Sans quoi, c'est la porte ouverte à tous les abus.

Je remercie toute l'équipe de la rubrique Web pour leur conseils avisés et particulièrement **Bovino** pour sa relecture affûtée.