

## JQuery, petite introduction

Le 25 novembre 2007 à 21:55.

JQuery, pour faire simple, est une boîte à outils javascript ultra-compacte permettant de faire abstraction du type de navigateur pour manipuler la structure dynamique d'une page WEB, le [DOM](#)<sup>[1]</sup>, et ceci avec une aisance déconcertante. Il existe plusieurs autres plates-formes du même genre (Scriptaculous, Prototype, etc.) mais JQuery a l'avantage de faire aujourd'hui parti du cœur de Drupal. Il serait donc dommage de ne pas en connaître les possibilités.

### Installation

JQuery se télécharge sur le site <http://docs.jquery.com><sup>[2]</sup> et il existe 2 parfums différents pour chaque version : Packed et Uncompressed. Téléchargez plutôt la première qui prend moins de place. Renommez le fichier téléchargé en `jquery.js` et posez-le dans un dossier accessible par apache. Dans la suite de ce tutorial, tous les exemples seront considérés comme étant des fichiers php stockés au même endroit que ce `jquery.js` et lisible par apache.

Une fois que vous avez posé `jquery.js`, commençons par écrire un exemple simple pour vérifier que tout fonctionne.

```
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <style>
      body {
        background-color:black;
        color:white;
      }
      A {
        color:white;
      }
    </style>
  </head>
  <body>
    <a href="#" onclick='alert("Hello World. La version de votre moteur HTML est
"+jQuery.browser.version); return false'>Cliquez ici !</a>
  </body>
</html>
```

Si tout est bien installé, vous pourrez afficher cette page via une url du genre `http://mon.serveur.com/jquery/exemple.php`. Et en cliquant sur le lien proposé, devrait s'afficher un message de bienvenue suivi de la version du moteur de rendu de votre navigateur. Si tel est le cas, nous pouvons poursuivre.

### Mise en œuvre

Pour mieux comprendre ce que permet JQuery, prenons directement un exemple, et imaginons que nous voulions désactiver pour tous les liens d'une page, le rectangle en pointillés grisâtres et disgracieux qui apparaît lorsque l'utilisateur vient de cliquer dessus : le "focus". Ce rectangle n'est pas gênant sur un fond blanc, mais devient très laid sur un fond sombre. Vous pouvez vous en rendre compte sur l'exemple précédent.

En programmation "standard", la solution consistait à modifier chaque balise `<A href=...>` en ajoutant un attribut `onfocus="this.blur();" . Le rôle de cette modification est de retirer le focus (fonction blur()) à tout élément DOM de type "hyperlien", et ce au moment même où celui-ci le gagne (événement onfocus). Ce type de "hack" est le genre de magouille peu élégante (car rendant le code difficile à lire) et fastidieuse au possible que JQuery traite avec simplicité. Pour s'en rendre compte, après le bloc style de notre exemple, rajoutez le script suivant :`

```
<script>
  function initializePage() {
    $("A").focus(function() { this.blur(); });
  }
  $(initializePage);
</script>
```

Au rechargement de la page, cliquez sur le lien, validez le dialogue, et vous constaterez que le focus a disparu.

Pour comprendre ce premier exemple, il faut commencer par se diriger vers la dernière ligne. Elle indique à JQuery d'exécuter la fonction `initializePage` lorsque le document sera complètement chargée par le navigateur. Généralement tous les scripts manipulant le DOM utilisent cette technique pour être bien sûr que tout est en place avant de jouer avec la structure

document.

Donc une fois le document correctement chargé, est lancée la fonction `initializePage`. La seule ligne de cette fonction est du pur jQuery. La fonction `$("#A")` permet de rechercher tous les éléments du DOM ayant pour type balise `A`. Son résultat est un **objet jQuery** qui contient la collection d'éléments DOM trouvés. Nous demandons ensuite à cet objet d'ajouter, à chacun de ses éléments, l'attribut `onfocus`, qui va bien. Cela semble simple, non ? En tout cas l'important ici est de comprendre l'esprit... les détails vont venir.

jQuery n'est pas le seul à utiliser la notation `$(...)`. Scriptaculous par exemple le fait aussi, et si l'on est amené à utiliser ensemble les deux bibliothèques, il est possible d'éviter les conflits en revenant à la notation longue, c'est à dire en remplaçant `$(...)` par `jQuery(...)`.

## Rechercher dans le DOM

le DOM est, de manière simplifiée, une arborescence d'objets dont chaque noeud correspond à une balise HTML. Le navigateur fabrique le DOM au fur et à mesure de l'arrivée des données du site web. Et c'est ce DOM qui est ensuite représenté visuellement pour former ce que voit l'utilisateur.

La première force de jQuery est donc de proposer un moyen simple et direct de sélectionner des éléments du DOM pour y appliquer des modifications. Et quoi de plus simple que d'utiliser comme syntaxe de recherche, quelque chose de vraiment conçu pour cela, la notation des descripteurs CSS<sup>[3]</sup>.

Dans l'exemple précédent nous avons sélectionné toutes les balises `A` et y avons attachées un événement `focus`. Même si c'est une recherche très basique, en réalité, ce simple `"A"` est le même type de descripteur CSS que dans une notation que tout le monde connaît bien :

```
A {
  color : blue;
}
```

Ainsi, une chaîne de recherche pour sélectionner l'ensemble des éléments `A` de classe `ma_classe` cette fois, peut donc s'écrire :

```
// Sélection des éléments A de classe ma_classe
$("#A.ma_classe"). ...
```

De la même manière, pour sélectionner le div d'id `mon_id` :

```
// Sélection des éléments A d'id mon_id
$("#DIV#mon_id"). ...
```

Et comme quasiment tous les descripteurs CSS sont supportés, pour sélectionner l'ensemble des balises `LI` de classe `item` contenues directement dans une balise `UL` de classe `menu`, cela donne très logiquement :

```
// Sélection des éléments LI.item contenus dans des UL.menu
$("#UL.menu>IL.item"). ...
```

## L'objet jQuery

La bonne question est maintenant de savoir ce que renvoie exactement la fonction `$("#...")`. Et bien la réponse est.. la bibliothèque jQuery elle-même. En effet, toutes les fonctions de la bibliothèque sont en réalité les méthodes d'un **objet jQuery**. Et cet objet contient une liste d'éléments du DOM sur laquelle il va travailler. Ainsi lorsque l'on regarde le code suivant :

```
$("#.ma_classe").css("background", "black");
```

L'appel `$("#.ma_classe")` renvoie en réalité un objet jQuery contenant la liste de tous les éléments DOM de la classe `ma_classe`. L'appel `css(...)` modifie donc la couleur de fond de tous les éléments connus de l'objet jQuery.

De la même manière un appel de la forme `$(document.body)` va renvoyer un nouvel objet jQuery contenant cette fois une seule référence, celle de l'élément DOM `body`. Il est donc non seulement possible de créer un objet jQuery à partir d'un sélecteur CSS (liste d'éléments DOM) mais aussi d'injecter n'importe quel élément DOM dans un objet jQuery.

A l'inverse, il peut être intéressant de récupérer les éléments DOM sélectionnés dans l'objet jQuery, et cela se fait de manière très simple, comme si l'objet en question était un tableau :

```
// Création d'un objet jQuery contenant une sélection de lien de class "ma_classe"
var mesLiens = $("#A.ma_classe");

// obtention du nombre d'éléments sélectionnés
alert ("Nous avons "+mesLiens.length+" éléments sélectionnés");
```

```
// obtention du premier élément DOM de la sélection
alert ("Le premier élément DOM est "+mesLiens[0]);

// ajout d'un attribut CSS au premier élément DOM en repassant par jQuery
$(mesLiens[0]).css("color", "red");
```

Plus "fort" encore, nombre des fonctions de l'objet jQuery renvoie l'objet jQuery qui les a appelé... Du coup nous pouvons partir du code suivant :

```
// Objet jQuery représentant les éléments DOM de classe ma_classe
var objet_jquery=$(".ma_classe");

// Changement du fond et de la couleur de tous les éléments DOM sélectionnés dans
l'objet jQuery
objet_jquery.css("background", "red");
objet_jquery.css("color", "green");
```

Pour arriver à une version beaucoup plus compacte :

```
objet_jquery=$(".ma_classe")
  .css("background", "red")
  .css("color", "green");
```

## Les évènements

Un autre grand domaine où jQuery nous simplifie beaucoup la tâche, est la gestion des évènements. Il est en effet possible de connecter tous les types d'évènements connus du DOM à une fonction JavaScript, et ceci de manière dynamique, c'est à dire à l'exécution du script sur le client.

Pour gérer les évènements, jQuery dispose de deux fonctions `bind` et `unbind`. L'une permet de lier, sur un élément donné du DOM, un type d'évènement (`click`, `mouseover`, `keydown`, etc..) à une fonction de votre choix. La seconde permet quant à elle de rompre le lien qui existait entre un type d'évènement et une fonction. Cela nous donne un code ressemblant à cela :

```
$("#DIV#mon_id").bind('click', maFonction);
...
$("#DIV#mon_id").unbind('click', maFonction);
```

Mais pour simplifier la syntaxe du lien, jQuery propose une batterie de fonction toute faite dédiées à un type d'évènement donnée :

```
$("#DIV#mon_id").click(maFonction);
$("#DIV#mon_id").mouseover(maFonction);
$("#DIV#mon_id").keydown(maFonction);
etc..
```

Comme vous le voyez la mise en oeuvre des évènements est on ne peut plus simple. Pour un exemple plus concret, imaginons une zone (div) qui affiche un message lorsque la souris passe au dessus. Cela nous donnerait :

```
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script>
      function afficherMessage() {
        alert("Vous me roulez dessus !!");
      }

      function initializePage() {
        $("#mon_div").mouseover(afficherMessage);
      }
      $(initializePage);
    </script>
  </head>
  <body>
    <div id="mon_div">Survolez-moi !</div>
  </body>
</html>
```

Au premier regard, à la syntaxe près, tout ceci ressemble beaucoup à ce qu'il était déjà possible de faire avec l'attribut `onmouseover`. Il y a en revanche une grosse différence. Dans le mode "classique", on ne peut ajouter un type d'évènement donné qu'une seule et unique fois sur un élément. En jQuery, il est possible de lier plusieurs fonctions différentes à un élément donné, et pour le même type d'évènement. La raison de ce comportement est que jQuery ajoute la fonction liée à une liste par type d'évènement. Ainsi lorsque l'évènement est déclenché, la liste est parcourue, et toutes les fonctions de la liste sont exécutées.

Cet aspect est important à assimiler car si par erreur vous liez plusieurs fois de suite une même fonction sur un même type d'évènement à un même objet, la fonction sera exécutée autant de fois de suite... En revanche, cet aspect est une réelle force car cela nous permet de séparer proprement les différents travaux à effectuer dans différentes fonctions, et ceci même si elle sont déclenchées sur le même type d'évènement.

Certains doutent de l'intérêt de câbler les évènements avec jQuery, du côté client, plutôt que directement sur la page. Le premier intérêt de l'approche dynamique est la lisibilité. En effet, avec cette technique, le code HTML n'est plus "pollué" par des attributs (`onmouseover`, `onclick`, etc...) qui n'ont rien à voir avec le contenu. Il s'agit du même type de séparation déjà effectué avec le formatage des pages HTML déplacé dans des feuilles de style. Ici, c'est le comportement qui est ainsi séparé du HTML et placé dans un script aux frontières clairement définies. Ce code devient dès lors facile à faire évoluer et à maintenir. Autre avantage de cet approche, construire un site à l'ancienne, fonctionnel et respectant l'accessibilité. Et le pimenter dynamiquement, au besoin, de fonctions plus "sexy" mais néanmoins optionnelles.

Voilà, c'est à peu près tout ce qu'il y a à savoir sur les évènements en jQuery, la liste complète des fonctions correspondant à cet aspect est disponible [ici](#) [4].

## Modification du DOM

Une fois que nous savons rechercher des éléments, accrocher des évènements, il devient intéressant de savoir modifier dynamiquement le DOM pour créer des affichages dynamiques. jQuery fournit dans ce sens une batterie de fonctions dédiées à cet usage qu'il serait difficile de couvrir dans ce tutorial. Nous allons donc limiter notre découverte à la modification dynamique du code HTML contenu dans un div.

```
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script>
      function initializePage()
      {
        $("#mon_div")
          .mouseover(
            function() {
              $(this).html("Je suis dedans"); })
          .mouseout(
            function() {
              $(this).html("Je suis dehors"); });
      }
      $(initializePage);
    </script>
  </head>
  <body>
    <div id="mon_div">Survolez-moi !</div>
  </body>
</html>
```

D'un point de vue utilisateur, une fois l'exemple chargé dans le navigateur, il suffit que l'utilisateur survole le texte affiché pour que son contenu change, et qu'il change encore lorsque l'utilisateur sort de la zone.

Je ne reviens pas sur l'ajout des deux évènements à l'élément `#mon_div`. La seule chose nouvelle la fonction `html(...)`. Cette fonction permet de remplacer le contenu HTML de l'élément DOM par un autre contenu. Attention, il s'agit du *contenu* HTML, pas du code HTML de l'élément lui-même. En d'autre terme, cette fonction change ce qu'il y a à l'intérieur de la balise, mais pas la balise lui-même.

Une autre manière de modifier le DOM est de manipuler les attributs des éléments. Que ce soit les attributs standards ou les attributs de style CSS.

```
$("#A").attr("title", "Nouveau titre pour tous les liens");
// Changement de la couleur de tous les liens
$("#A").css("color", #FF00FF);
```

Un aspect souvent oublié du DOM est que l'on peut mettre sur une balise HTML des attributs "non-standards". Il est ainsi

possible de charger, sans impacte visuel, une balise en informations nouvelles qui seront stockés dans les éléments du DOM correspondant. Par exemple, vous pouvez déclarer un balise lien comme suit :

```
<a id="mon_lien" href="http://quelquepart.com/" monMessage="Un message personnel">Un lien</a>
```

Comme vous l'avez remarqué, monMessage est tout sauf un attribut de la balise A . Et pourtant, cet attribut personnalisé est parfaitement lisible dans le DOM, du côté client :

```
$("#mon_lien").mouseover(function(){  
    alert ($(this).attr("monMessage"));  
});
```

## JQuery et AJAX

Le dernier aspect de jQuery que nous allons aborder, est sa capacité à gérer les requêtes asynchrones, plus connues sous le nom de requêtes AJAX. AJAX, en deux mots, c'est la possibilité offerte en javascript de faire une requête sur un serveur distant et de récupérer dans le script appelant les données que le serveur à renvoyées. Et tout ceci sans qu'une action utilisateur ne soit nécessaire, d'où la dénomination *d'asynchrone*.

Pour rapidement comprendre ce qu'est une requête AJAX et sa mise en oeuvre dans jQuery, un petit exemple simple. Tout d'abord créons une page PHP, nommée `getTime.php` , renvoyant simplement l'heure courante :

```
<? [5]  
print [6] date [7]('h:i:s');  
?> [5]
```

Rien de bien sorcier dans ce code, mis à part qu'il ne contient aucune balise, et c'est normal car notre requête AJAX va s'attendre à recevoir des données, pas une page WEB. Il est possible de tester cette fonction avec votre navigateur pour vérifier qu'il affiche bien l'heure.

Maintenant nous allons fabriquer une page `horloge.php` dont le but est, toutes les secondes, de faire une requête AJAX et de mettre à jour le contenu d'un `div` avec le résultat de la requête.

```
<html>  
  <head>  
    <style>  
      body {  
        background-color:black;  
        color:white;  
      }  
      A {  
        color:white;  
      }  
      span#heure {  
        background-color:white;  
        color:black;  
      }  
    </style>  
    <script type="text/javascript" src="jquery.js"></script>  
    <script>  
      function [8] changeHeure()  
      {  
        jQuery.get("getTime.php", {}, reponseServeur);  
      }  
  
      function [8] reponseServeur(texte, erreur, requête)  
      {  
        $('#heure').html(texte);  
        setTimeout(changeHeure, 1000);  
      }  
      function [8] initializePage()  
      {  
        changeHeure();  
      }  
      $(initializePage);  
    </script>  
  </head>  
  <body>  
    Bienvenue sur l'Horloge AJAX/PHP. Il est actuellement
```

```
<span id="heure"/>
</body>
</html>
```

Tout commence, comme toujours, par la commande `jQuery $(initializePage)` et l'appel à la fonction `initializePage` lorsque la page a fini d'être chargée par le navigateur. Cette fonction fait un appel à `changeHeure`.

Cette fonction contient notre appel AJAX, avec la commande `jQuery jQuery.get(...)`. Son but est de faire une requête sur l'url passée en paramètre ("`getTime.php`") et lorsque le serveur a répondu, appeler la fonction passée en dernier paramètre (`reponseServeur`). Cette fonction est asynchrone, c'est à dire qu'elle ne bloque pas le script.

Lorsque `jQuery` reçoit la réponse du serveur (ici l'heure), il fait donc appel à notre fonction `reponseServeur`, en lui passant en premier paramètre le texte renvoyé (ex. "`12:11:13`"). Nous allons alors utiliser les techniques vues plus haut pour rechercher le div ayant pour id `heure` et changer le contenu HTML de cet élément avec l'heure reçue de notre serveur. Voilà, fin de la magie AJAX, ce n'est rien de plus que cela même si cela révolutionne beaucoup la manière dont nous concevons les sites web aujourd'hui.

La dernière instruction de la fonction est un classique "`setTimeout`" qui va indiquer à javascript de refaire un appel à la fonction `changeHeure` dans une seconde. Et ainsi de suite.

Comme vous le voyez, `jQuery` cache ici beaucoup de la complexité d'AJAX, complexité qui réside surtout dans le fait de devoir prendre en compte les manières diverses dont chaque navigateur gère les appels asynchrone. Vous voyez aussi qu'AJAX peut être utilisé dans plein de situation différentes. Pour récupérer des données à traiter, pour récupérer des valeurs à insérer dans une liste, pour lancer une commande sur le serveur, et aussi pour changer, comme ici, une portion du DOM. Et comme ce dernier usage est très commun, pour simplifier encore le code, `jQuery` propose une fonction permettant de réaliser le `get` et l'injection de code HTML dans le DOM, sans fonction `reponseServeur` et en un seule passe. Le script devient alors :

Comme vous le voyez, la fonction `load` va effectuer un `get`, mais va aussi directement renvoyer la réponse du serveur dans l'élément d'id `#heure`. Le dernier paramètre n'est plus là que pour réarmer le timer pour le prochain appel.

## Conclusion

Cette introduction ne couvre pas, loin de là, tous les possibles de cette petite mais puissante librairie. `jQuery` dispose de fonctions d'effets visuels, peut aussi utiliser XPath pour sélectionner les éléments du DOM, peut être étendu par l'ajout de plugins. Bref, la bonne source d'information est [ici](#) <sup>[9]</sup> et il vous reste encore beaucoup à découvrir 😊

<http://artisan.karma-lab.net/node/1275>  
(C) artisan numerique - CC BY-SA

### Liens:

- [1] <http://fr.wikipedia.org/wiki/DOM>
- [2] [http://docs.jquery.com/Downloading\\_jQuery](http://docs.jquery.com/Downloading_jQuery)
- [3] <http://fr.wikipedia.org/wiki/CSS>
- [4] <http://docs.jquery.com/Events>
- [5] <http://www.php.net/>
- [6] <http://www.php.net/print>
- [7] <http://www.php.net/date>
- [8] <http://www.php.net/function>
- [9] <http://docs.jquery.com>