Apprendre et comprendre jQuery

par Mathieu ROBIN (http://mathieu-robin.developpez.com) (Blog)

Date de publication : 05 September 2011

Dernière mise à jour :

Je vous propose ici un tutoriel que j'espère aussi complet que possible à propos de jQuery 1.6. Ce tutoriel doit pouvoir vous permettre de démarrer totalement avec jQuery ou vous permettre de juste vous remettre à niveau.

I - Introduction

I-A - Pré-requis

Peu de pré-requis sont nécessaires. Du bons sens et un minimum de connaissances en développement logiciel seront forcément un vrai plus. N'avoir jamais développé avec JavaScript risque sérieusement de vous freiner, mais je vais m'efforcer d'expliquer au mieux les concepts utilisés qui me semblent complexes pour un débutant. La maitrise de HTML et de CSS est, quand à elle, considérée comme acquise. Si ce n'est pas le cas, vous trouverez de nombreux tutos ici :

- HTML : **http://xhtml.developpez.com/cours/**
- CSS : **http://css.developpez.com/cours/**

I-B - Versions de jQuery

Ce tutoriel est voué à évoluer avec le temps et avec les versions de jQuery. Cette partie s'enrichira donc au fur et à mesure.

I-B-1 - Antérieures à la 1.6.3

De façon générale, l'essentiel de la rétro-comptabilité est assurée par le framework. Quand elle ne l'est pas, ce sont pour de bonnes raisons qui seront, si possible, aussi évoquées au fur et à mesure de ce tutoriel.

I-B-2 - jQuery 1.6.3

Je démarre donc ce tutoriel avec la version stable courante de jQuery, la version 1.6.3.

I-C - Philosophie

L'idée originelle de jQuery était "**The less you write, the more you do**". En français, comprendre "Moins vous écrivez, plus vous en faites".
Propos engageants et plaisants mais posant un vrai problème dans le monde de JavaScript qui est habitué à être une machine à copier/coller sans réfléxion.
Ce qu'il faut vraiment retenir de jQuery, c'est qu'il vous permet surtout une compatibilité maximale de votre application entre chaque couple système/navigateur.
Si vous vous basez sur la seule première idée, passez votre chemin, ce tutoriel ne vous apportera rien. Si vous vous concentrez sur la deuxième, vous n'exploiterez pas à fond jQuery mais au moins, vous ne ferez pas non plus n'importe quoi.

I-D - Concurrence

Je pense d'abord à l'historique, à celui avec lequel j'ai démarré, Prototype et son plugin quasi symbiotique : Script.aculo.us. Il y a aussi ExtJS, Dojo, Mootools, YUI ou encore GWT (même si ce n'est pas que du JS). Je ne vous cite que les majeurs, et encore, j'en oublie. Seuls vos goûts et votre façon de penser déterminera lequel est le plus adapté à vos besoins. Ils ont tous une philosophie qui leur est propre. Certains, comme jQuery, sont faciles d'accès pendant que d'autres, plus difficiles, vont aussi beaucoup plus loin.

I-E - Plugins

jQuery a été entièrement pensé pour être hyper extensible. Vous pourrez trouver ou développer vous même des plugins pour faire à peu près tout et n'importe quoi. Même si nous verrons les bases de la création d'un plugin, je ne vous expliquerai pas comment réaliser un plugin spécifique qui à mon avis doit être créé selon votre besoin.

De très nombreuses contributions sont disponibles sur le net, notamment sur **Developpez.com**, qu'elles soient faites par des particuliers ou des entreprises. A l'instar de Microsoft qui a énormément contribué au développement de plugins majeurs pour jQuery.
Il existe une sorte d'annuaire de plugins sur le site officiel de jQuery, vous pouvez le retrouver à cette adresse : **http://plugins.jquery.com/**. Celui-ci ne fournit qu'une liste et un système de notations/discussions, rien ne vous garantit la qualité des plugins ni leur capacité à répondre à votre besoin.

II - Premiers pas

II-A - Architecture de votre projet

Vous allez avoir dans votre site deux ressources JavaScript, l'appel à jquery et un fichier qui contiendra tout votre code.
Au possible, il vaut mieux éviter le code dit "inline", c'est à dire inclus au beau milieu de votre HTML. Le développement en couches s'applique aussi au web et pas uniquement côté serveur. Gardez bien en tête ce découpage, un peu simpliste mais offrant une base de départ pour structurer votre pensée :

- HTML : description de la structure et contenu du site
- JavaScript : implémentation des comportements du site
- CSS : description de l'apparence du site

Nous aurons donc cette architecture de projet : + monProjet - index.html - core.js - style.css Avec pour contenu :

index.html
```
<!DOCTYPE html>
<html>
 <head>
  <title>Tutoriel jQuery</title>
 </head>

 <body>
  <script type="text/javascript" src="http://code.jquery.com/jquery-1.6.3.min.js"></script>
  <script type="text/javascript" src="core.js"></script>
 </body>
</html>
```

core.js
```
/* Pour le moment ce fichier va rester vide */
```

css.js
```
/* Pour le moment ce fichier va rester vide */
```

J'essaierai au possible de rester respectueux des standards du W3C (HTML 5 et CSS 3 en l'occurence).

II-B - Inclure jQuery dans votre site

Je vous conseille fortement l'usage d'un CDN, vous profiterez ainsi du cache du navigateur de vos visiteurs. Ça fait toujours ça de moins à charger pour eux et ça de moins à la charge de votre serveur. Vous avez le choix :

- jQuery : Source **http://code.jquery.com/jquery-1.6.3.min.js**
- Microsoft : Source **http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.6.2.min.js** - La 1.6.3 devrait bientôt être disponible
- Google : Source **http://ajax.googleapis.com/ajax/libs/jquery/1.6.3/jquery.min.js**

Dans le contenu du fichier HTML, vous pouvez voir que j'utilise celui de jQuery.

II-C - Vérifier l'import de jQuery dans votre site

Editons le fichier core.js et mettons ce code-ci :

core.js
```
jQuery(document).ready(function() {
 alert('jQuery prêt à vous aider !');
});
```

Enregistrez et rafraîchissez la fenêtre de votre navigateur. Vous aurez une fenêtre d'alerte vous indiquant le message : "jQuery prêt à vous aider !". Si vous avez ça, c'est bon, c'est que ça marche.

II-D - Evènement document.ready

Il faut comprendre que jQuery a notamment pour rôle de faciliter le développement d'applications web interactives. Et donc pour celà, le gestionnaire d'évènements est un élément clé.
L'évènement le plus important est celui vu précédemment. Il a un rôle simple : signaler et déclencher des actions que vous aurez pu définir dès que le DOM est prêt. C'est à dire que tous les fichiers sont chargés et donc que votre script peut s'exécuter dans les meilleures conditions possibles. Nous verrons dans un autre chapitre les événements que supporte jQuery.

II-D-1 - Différence entre $() et jQuery()

Vous avez certainement croisé de très nombreuses fois l'usage du $ et rarement autre chose.
C'est bête comme bonjour. $(document) est rapide et facile à écrire tout en ne gênant pas la lisibilité, par contre, si vous utilisez aussi Prototype ou Mootools (par exemple), vous allez tout droit sur les problèmes d'incompatibilité. Vu que ceux-ci se servent aussi du $. Ceci dit, je vous déconseille généralement d'utiliser deux frameworks en même temps. Non parce qu'il y en a des moins bons, mais parce que globalement ils font la même chose. L'intérêt d'en utiliser deux est donc nul et dangereux.

Il existe la méthode **.noConflict()** pour ceux qui s'y tenteraient quand même, mais j'insiste sur la dangerosité de telles pratiques. Il y a d'ailleurs un équivalent chez Mootools, voir l'article "**The Dollar Safe Mode**".

II-E - Chaînage des appels

Une des spécificités de jQuery est d'implémenter un chaînage quasi-systématique. Rares sont les méthodes qui ne retournent pas l'objet appellant. L'avantage de cette pratique, c'est de pouvoir faire ce genre de choses :

Exemple
```
$(document)
 .premiereFonction()
 .secondeFonction()
 .encoreUne().puisEncoreUne()
 .etAinsiDeSuite()
 .autantQuOnVeutEnFait();
```

III - Les sélecteurs CSS

Comme nombreux de ses concurrents, jQuery vous aide notamment à manipuler le DOM et quoi de mieux que le système des sélecteurs CSS pour ça ?

III-A - Qu'est-ce qu'un sélecteur CSS

Un sélecteur est une chaîne de caractère, également appellée "**motif**", qui permet de définir un ou plusieurs éléments dans votre DOM. Il existe des sélecteurs pour à peu près tout. Et si quand bien même il vous manque quelques expressions pour faire ce que vous voulez, jQuery complète suffisamment bien le panel pour que vous ayez tous les outils sous le coude.

Attention, tous les sélecteurs ont certes la même fiabilité mais pas forcément les capacités côté performances. *Il s'agit donc de les utiliser avec parcimonie mais j'essaierai au possible de vous guider sur ce point.*

III-B - Liste des sélecteurs

En conséquence de mes propos précédents, je divise donc en deux la liste des sélecteurs. Les intéressants d'abord puis nous verront les moins performants et leur alternative plus intéressante.
Si vous voulez vous intéresser de près à la question des performances avec jQuery, je vous suggère de parcourir cet excellent diaporama d'Addy Osmani : **jQuery Proven Performance Tips & Tricks**.

III-B-1 - Sélecteurs intéressants

Sélecteur	Cible
$("#monId")	... ou toute autre type de balise possédant l'attribut *id="monId"*
$("input")	<input />
$("p")	
$("b")	

Et ainsi de suite pour toutes les balises HTML. Ceci dit, pour certaines, il n'y aura sûrement aucun intérêt à les sélectionner.
Ces sélecteurs sont particulièrement rapides du fait qu'ils sont nativement gérés par les navigateurs, par exemple la fonction getElementById(). Vous pouvez donc démarrer sur ces bases en toute tranquilité, ce ne sont pas elles qui vous ralentiront.

III-B-2 - Sélecteurs à éviter

III-B-2-a - Sélection de classe

Exemple
```
$(".uneClasse")
```

Permet de capturer tous les éléments possédant une classe CSS précise.
Pratique, mais getElementByClassName() :

- N'est pas supporté de IE 5 à 8 ;
- Supporté sur Firefox 3, Safari 4, Opera 10.10 et Chrome 4 ;
- Sur toutes les versions supérieures aux versions suggérées.

Un bricolage a été mis en place pour palier au manque sur les anciens navigateurs mais globalement, ça ralentit forcément la recherche. Mais c'est loin d'être le pire.

III-B-2-b - Pseudo-sélecteurs

Exemple
```
$(":hidden"); // Retourne tous les éléments <input> de type "hidden"
```

Ici l'explication de la lenteur est très simple. Il n'existe pas de méthode native pour ce genre de recherches. Il a donc fallu en implémenter une, compatible avec tous les navigateurs au passage.
Même si il existe des pseudo-sélecteurs CSS, ils sont très peu nombreux par rapport à ce que propose jQuery.

III-B-2-c - Sélection sur attributs

Exemple
```
$("[attribute=value]"); // Retourne tous les éléments ayant un attribut nommé "attribute" et qui a pour valeur "v
```

Encore une fois, il n'existe pas de méthode native pour ce genre de recherches.
Les navigateurs les plus récents implémentent des méthodes querySelector() et querySelectorAll() qui facilitent le travail de recherche. Mais ça ne change rien pour les anciens navigateurs.

III-B-3 - Du bon usage des sélecteurs : mise en cache

Il est important de noter ceci tout de même : un sélecteur reste une opération de recherche à réaliser à chaque appel. Et celà coûte en ressources et en temps. Il est donc important de mettre en cache les sélecteurs qui vous servent plusieurs fois. La méthode est très simple :

Exemple
```
var monId = $("#monId");
```

Oui, c'est aussi basique que ça. L'avantage réside dans le fait que le navigateur n'aura plus à chercher dans le DOM, il sait déjà ce que vous cherchez et l'a mis de côté. Le gain est donc particluièrement intéressant. Et ça augmente la lisibilité de votre code si vous vous y prenez bien.

III-B-4 - Principe de contexte

Evidemment, viendra le moment où vous aurez besoin des éléments que j'ai cité comme peu performants. Vous ne devez pas vous les interdire mais les utiliser avec dicernement. Cependant, pour aider un peu le navigateur dans ces recherches, il y a souvent moyen de préciser un peu la zone de recherche plutôt que de lui faire fouiller tout le DOM. Ainsi :

Exemple
```
var sideBar = $("#sideBar");
sideBar.find("input");
```

Retourne tous les éléments <input> compris dans l'élément ayant pour id "sideBar". Ce qui réduit certainement considérablement le champ de recherches et donc la vitesse d'exécution. Il existe de nombreuses façons d'utiliser le contexte, toutes moins performantes, les voici :

Exemple
```
$("#sideBar").find("input"); // 16% moins efficace
$("input", $("#sideBar")); // 23% moins efficace
$("#sideBar").children("input"); // 50% moins efficace
$("#sideBar > input"); // 70% moins efficace
$("#sideBar input"); // 77% moins efficace

// Utilisation du cache

var sideBar = $("#sideBar");
$("input", sideBar); // 5 à 10% moins efficace
```

Les chiffres avancés sont ceux que fournit Addy Osmani dans le lien proposé plus tôt dans ce tutoriel.

Je vous conseille de parcourir la page suivante de la documentation qui vous permettra d'en apprendre plus sur l'utilisation du contexte : ***Traverser le DOM***

III-B-5 - N'utilisez jQuery que si nécessaire

Normalement, vous ne devriez pas avoir ce cas en production mais il illustrera très bien le propos :

Exemple
```
jQuery(document).ready(function() {
 $("a").click(function () {
  console.log("Id de l'élément cliqué : " + $(this).attr("id"));
 });
});
```

Ici vous cherchez à afficher en console l'id de l'élément sur lequel on viendrait de cliquer. En soit, ce code n'est pas erroné. Mais vous n'avez pas besoin d'un objet jQuery pour récupérer l'id de this :

Exemple
```
jQuery(document).ready(function() {
 $("a").click(function () {
  console.log("Id de l'élément cliqué : " + this.id);
 });
});
```

Beaucoup plus lisible, plus performant. Et forcément compatible avec tous les navigateurs, c'est du JavaScript tout ce qu'il y a de plus banal pour faire exactement la même chose.

IV - Evènements

IV-A - Document.ready

Nous avons déjà brièvement étudié ce cas durant le paragraphe "TODOLIEN II-C Vérifier l'import de jQuery dans votre site".
Il faut savoir que votre navigateur n'attend pas naturellement que toute la page soit chargée pour exécuter le script JavaScript qu'il a déjà chargé. Cela peut-être pratique pour les scripts de tracking mais peut aussi être particulièrement dangereux si vos scripts sont inter-dépendants. Attendre que tout soit chargé pour commencer l'exécution de vos scripts est donc une protection évidente et simple à mettre en oeuvre.

IV-B - Ajout d'un gestionnaire d'événements sur les éléments du DOM déjà existants

jQuery propose une fonction puissante pour ajouter facilement un gestionnaire d'évènements sur des éléments déjà existants. Admettons qu'on veuille afficher une alerte quand on clique sur un paragraphe. Pour faire simple, nous allons afficher le texte du paragraphe dans l'alerte.

core.js
```
jQuery(document).ready(function() {
 $("p").click(function(event) {
  alert(this.innerHTML);
 });
});
```

index.html
```
<html>
 <head>
  <title>Tutoriel jQuery</title>
 </head>

 <body>
  <p>Hello</p>
  <span>world</span>
  <br />
```

index.html
```
   <script type="text/javascript" src="http://code.jquery.com/jquery-1.6.3.min.js"></script>
   <script type="text/javascript" src="core.js"></script>
  </body>
 </html>
```

Si vous cherchez à cliquer sur "Hello", vous aurez une alerte avec le texte "Hello", si vous cliquez sur "world", vous n'aurez rien.
Certains vont alors me demander pourquoi je n'ai pas plutôt écrit ça :

core.js
```
jQuery(document).ready(function() {
 $("p").click(function(event) {
  alert($(this).html()); // La différence est ici
 });
});
```

La raison est simple, c'est beaucoup plus lent pour faire la même chose.
D'autres auront remarqués la présence d'une variable "event" dans la signature de la fonction anonyme. Elle permet d'accéder à l'objet correspondant à l'événement lui-même. Elle n'est pas obligatoire et peut tout simplement ne pas figurer. J'aurais donc pu écrire ceci sans problème :

core.js
```
jQuery(document).ready(function() {
 $("p").click(function() {
  alert(this.innerHTML);
 });
});
```

Mais au moins maintenant, vous savez que les informations sur l'évènement sont là.

Attention! Comme l'indique le titre de ce paragraphe ainsi que sa première phrase, cette méthode n'est vraie qu'avec les éléments du DOM déjà existants. Si vous ajoutez des éléments au DOM correspondant au sélecteur, ils ne seront pas concernés.

IV-C - Ajout d'un gestionnaire d'événements sur les futurs éléments du DOM

Le souci de la méthode précédente est qu'à chaque fois que vous allez modifier le DOM, vous allez devoir rappeller tous vos ajouts de gestionnaire d'événements. Un peu lourd, même si vous pouvez aussi créer une fonction qui les appelle toutes, au risque de demander des transformations inutiles.

Depuis jQuery 1.3, nous disposons de la fonction **.live()**. Cette fonction apporte une idée simple : gérer de façon dynamique les différents événements. Que les éléments existent ou non, ils sont/seront affectés. Très pratique, mais aussi très lent. Je ne vous conseille de l'utiliser que si vous n'êtes pas totalement maître d'un événement très précis. Faisons un exemple. Reprennons le précédent, et à la place d'afficher le texte sur lequel on a cliqué, nous allons ajouter un nouveau pargraphe à la suite du document. D'abord avec l'événement click comme nous l'avons vu précédemment, puis avec la fonction live().

index.html
```
<html>
 <head>
  <title>Tutoriel jQuery</title>
 </head>

 <body>
  <p>Hello</p>
  <span>world</span>
  <br />
  <script type="text/javascript" src="http://code.jquery.com/jquery-1.6.3.min.js"></script>
  <script type="text/javascript" src="core.js"></script>
 </body>
```

index.html
```
</html>
```

core.js
```
jQuery(document).ready(function() {
 $("p").click(function(event) {
  $("body").append('<p>Hello world!</p>');
 });
});
```

Premier test, ça ne marche que si vous cliquez sur le premier paragraphe. Pas avec les nouveaux paragraphes ajoutés. Imaginez donc que vous chargiez dynamiquement une liste d'éléments et que vous vouliez afficher des informations au clic ou au survol de ces éléments (grâce à la fonction **hover()**). Vous seriez alors obligés de faire ça :

core.js
```
var ajouterP = function () {
 $("body").append("<p>Hello world!</p>");
 $("p").unbind("click").click(ajouterP);
}

jQuery(document).ready(function(){
 $("p").click(ajouterP);
});
```

J'ai dû définir une fonction *ajouterP* qui ajoute le texte. Puis retirer tous les gestionnaires d'évènements de clic sur les paragraphes, via la fonction **unbind()**. Et pour finir redonner à tous les pragraphes la fonction *ajouterP* comme réponse à l'évènement clic. Pas terrible mais obligatoire. Essayez de vous passer de la méthode unbind() sur ce coup là, vous allez rire, votre chef de projet moins.
Avec la fonction .live() maintenant :

core.js
```
jQuery(document).ready(function(){
 $("p").live('click', function () {
  $("body").append("<p>Hello world!</p>");
 });
});
```

Tout de suite, c'est beaucoup plus simple à lire et à mettre en place.

Mais attention aux performances quand même. Nombreux sont les cas où la fonction live() est bien plus lente qu'une affectation classique comme celle vue ci-dessus.

Notez qu'on donne d'abord à .live() le nom de l'évènement à gérer avant de lui donner la fonction à appliquer.

IV-D - Évènements disponibles

IV-D-1 - Évènements du document

Évènement	Correspondance	Documentation officielle
load / .load()	Permet de déterminer lorsque tout est chargé mais que le navigateur effectue encore des traitements liés au chargement, par exemple	Lien

	une décompression à la volée d'images.	
unload / .unload()	Se déclenche dès que la navigation cherche à sortir du document actuel. Cela peut-être dû à plusieurs raisons. A un clic sur un lien, à l'usage des boutons précédent/suivant ou simplement à un changement d'adresse dans la barre d'URL. L'actualisation de la page courante déclenchera forcément en premier lieu cet évènement.	Lien
ready / .ready()	Comme vu auparavant, permet de déterminer lorsque tout est chargé et que le navigateur n'a plus rien à faire concernant le chargement de la page.	Lien

V - CSS et effets graphiques

jQuery n'est pas qu'une librairie pour faciliter Ajax et c'est trop souvent oublié. Elle fournit aussi des assistants pour gérer non seulement le DOM mais aussi vos CSS et diverses effets graphiques. Sur ces points que nous allons nous pencher maintenant.

V-A - CSS

Partons du principe que vous voulez changer la couleur (disons en rose) d'un paragraphe lorsque vous cliquez dessus. Voyons comment vous feriez ça habituellement en JavaScript pur après avoir défini une page HTML simple pour illustrer l'exercice.

index.html
```
<html>
 <head>
  <title>Tutoriel jQuery</title>
 </head>

 <body>
  <p onclick="javascript:enRose(this);">Hello</p>
  <script type="text/javascript" src="core.js"></script>
 </body>
</html>
```

core.js
```
var liste = document.getElementsByTagName('p');
for(i = 0; i < liste.length; i++) {
   var paragraphe = liste.item(i);
   paragraphe.setAttribute('onclick','enRose(this);');
}
function enRose(ele) {
 ele.style.backgroundColor = '#FFC0CB';
}
```

Et maintenant, la même chose avec jQuery :

index.html
```
<html>
 <head>
  <title>Tutoriel jQuery</title>
 </head>

 <body>
  <p>Hello</p>
  <script type="text/javascript" src="http://code.jquery.com/jquery-1.6.3.min.js"></script>
  <script type="text/javascript" src="core.js"></script>
 </body>
</html>
```

core.js
```
jQuery(document).ready(function() {
 $("p").click(function(event) {
  $(this).css('background-color', '#FFC0CB');
 });
});
```

On gagne quand même pas mal en lisibilité non ? Et encore, ici j'ai seulement utilisé click() pour faire simple, imaginez si j'avais préféré live().