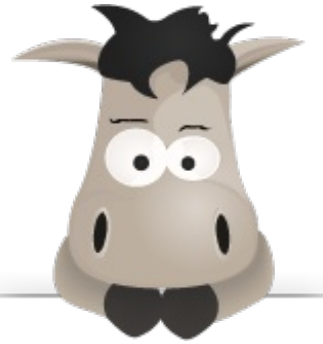


jQuery, dynamisez vos pages plus simplement !

Par Alex-D

www.siteduzero.com

Dernière mise à jour le 12/07/2012

Sommaire

jQuery, dynamisez vos pages plus simplement !

Par  Alex-D

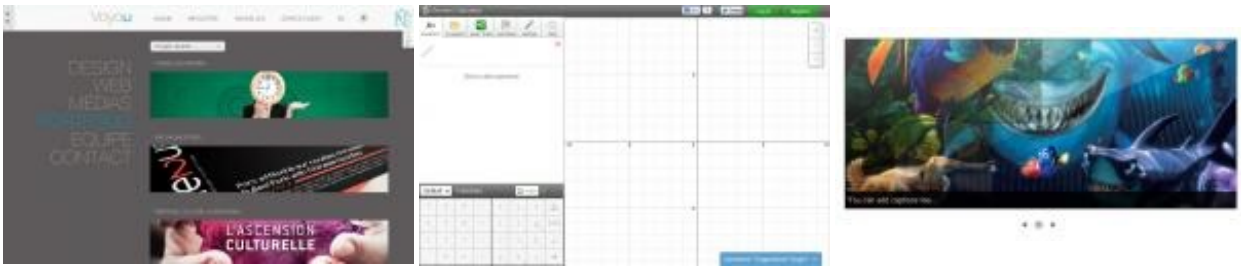
Mise à jour : 12/07/2012
Difficulté : Facile

À l'heure du Web 2.0 et bientôt du Web 3.0, les bibliothèques et frameworks JavaScript jaillissent pour combler la complexité et le manque de souplesse du langage. L'une des plus célèbres et rependues se dénomme jQuery. Vous naviguez chaque jour sur des sites qui utilisent jQuery, à commencer par le Site du Zéro.

jQuery est une bibliothèque destinée à **simplifier l'écriture de vos codes JavaScript**. Grâce à cette librairie vous pourrez dynamiser plus facilement vos pages web. En effet, la bibliothèque est légère, simplifie l'adaptation à tous les navigateurs (vous n'écrivez qu'un code sans vous soucier des différences entre navigateurs), possède une **syntaxe élégante et homogène** qui s'apprend rapidement (beaucoup moins verbeuse que le JavaScript « pur », mais toutefois cette syntaxe ne plait pas à tout le monde), et pour finir, sans limites grâce à la notion de plugin ou la création de bibliothèques additionnelles à jQuery de manière à pouvoir réutiliser ces morceaux de code plus facilement.

Par ailleurs, la bibliothèque jQuery est utilisée par les plus grands comme Google, Twitter, Wordpress ou encore Mozilla. Il est fort compréhensible que ce dernier l'utilise, c'est John Resig, un des développeurs principaux du moteur JavaScript de Firefox, qui a créé jQuery.

Il est préférable d'avoir des connaissances dans les langages HTML5 & CSS ainsi que JavaScript. Je vous donnerai dans ce cours les écritures les plus récentes et correctes à ce jour. Les codes HTML seront donnés dans sa version 5 et seront expliqués s'ils changent du XHTML.



Site à rechargement asynchrone - Application mathématique en ligne - Plugin de carrousel paramétrable

Aperçu de sites, applications et plugins utilisant jQuery

Partie 1 : Premiers pas avec jQuery

Cette bibliothèque va vous permettre de coder plus simplement et plus rapidement car vous n'aurez pas à vous soucier des différentes interprétations du JavaScript selon les navigateurs. jQuery est très simple à comprendre et à utiliser mais nécessite néanmoins quelques petites précisions, notamment sur l'optimisation de son import sur vos pages web.

Introduction à jQuery

jQuery est très simple à comprendre et à utiliser, mais nécessite néanmoins quelques petites précisions. Nous verrons en quoi jQuery simplifie l'écriture ainsi que l'optimisation de son import sur vos pages web. Nous finirons par le classique Hello World pour vérifier le bon fonctionnement de la bibliothèque dans notre page web.

Présentation

Les limites de productivité de JavaScript

Diversité des navigateurs

Il existe aujourd'hui de plus en plus de navigateurs que les visiteurs ne mettent pas forcément à jour. Le tout vous donne une panoplie de navigateurs ayant chacun une déclinaison de versions : Internet Explorer 6 à 10, Firefox 3 à 10, Opera, Chrome, pour ne citer que les plus répandus. Rien qu'avec ceux-là, il vous faudrait faire une série de tests de vos codes pour vérifier leur fonctionnement sur chacun d'eux. En effet il vous serait nécessaire d'écrire plusieurs portions de code pour pallier aux différentes interprétations des multiples moteurs JavaScript. Vous obtenez alors une syntaxe barbante et vous souciez perpétuellement de la compatibilité inter-navigateurs plutôt que d'optimiser votre site. Par exemple, si vous devez récupérer le défilement vertical (offset, en anglais) en pixels vous devez procéder différemment selon le navigateur. On obtient alors le code suivant :

Code : JavaScript - scrollTop sans bibliothèque

```
function scrollTop(){
  if(typeof( window.pageYOffset ) == 'number'){
    // Netscape
    return window.pageYOffset;
  } else if( document.body && (document.body.scrollTop) ||
navigator.appName=="Microsoft Internet Explorer") {
    // DOM
    return document.body.scrollTop;
  } else if( document.documentElement &&
(document.documentElement.scrollTop) ) {
     // Internet Explorer 6
    return document.documentElement.scrollTop;
  }
}
var defil = scrollTop();
```

En utilisant jQuery, eh bien ce code est contenu dans la bibliothèque, c'est-à-dire qu'on nous propose une fonction simple à utiliser qui se débrouille pour faire ce travail dans les coulisses et nous renvoyer le résultat. Ainsi, il nous suffit de faire appel à la fonction comme ceci :

Code : JavaScript - scrollTop avec jQuery

```
var defil = $(document).scrollTop();
```

Une écriture lourde

JavaScript peut être très intéressant et puissant à utiliser dans vos pages web en complément des langages HTML et CSS, cependant lors de développement d'application complexe, votre code va devenir très lourd même pour les choses les plus simples. jQuery vous offre une **simplification de l'écriture** et de lecture de votre code. On comprend mieux le choix de leur

slogan « Write less, do more » qui signifie « Écrivez moins pour en faire plus ». Voici un exemple pour vous convaincre. Vous vous trouvez dans la situation ou vous souhaitez afficher un message aux visiteurs utilisant Internet Explorer, sans bibliothèque il vous faut écrire ce code :

Code : JavaScript - Test du navigateur IE sans bibliothèques

```
var navigateur = "msie";
var place = navigator.userAgent.toLowerCase().indexOf(navigateur) +
1;
if(place){
     alert("C'est Internet Explorer " + detect.charAt(place +
navigateur.length) + " !");
}
```

En utilisant la bibliothèque jQuery vous n'aurez plus qu'à faire :

Code : JavaScript - Test du navigateur IE en utilisant jQuery

```
if($.browser.msie){
    alert("C'est Internet Explorer " + $.browser.version + " !");
}
```

Ceci est quand même **moins casse-tête et plus clair** à lire et à comprendre !

jQuery, une bibliothèque JavaScript

Une bibliothèque JavaScript...

jQuery n'est pas un langage, mais bien une bibliothèque JavaScript ! C'est à dire que vous avez tout un tas de code qui vous mâche le travail pour que vous puissiez développer plus rapidement ce que vous souhaitez. Cela implique, évidemment, que **JavaScript soit activé** au sein du navigateur du visiteur pour que vos scripts fonctionnent. Il est bon de savoir que l'expression « développer en jQuery » est fausse, « développer en JavaScript en utilisant la bibliothèque jQuery » est plus correcte.

... parmi tant d'autres !

Il existe bien d'autres bibliothèques JavaScript dont la plus connue avec jQuery est très certainement Mootools. Vous pourrez les trouver via une simple recherche sur les mots-clés « bibliothèque JavaScript » sur votre moteur de recherche favori. Chacun choisi celle qui lui plait le plus, jQuery étant une des plus réputée et répandue. Il arrive parfois que certains projets utilisent plusieurs bibliothèques pour bénéficier des atouts de chacune d'elles.

Comme toute bibliothèque, **jQuery doit être inclus dans vos pages** tel un script JavaScript écrit par vos soins, c'est ce nous allons voir dans la partie suivante.

Installation

Entrons dans le vif du sujet en commençant par ajouter la bibliothèque à nos pages pour enfin pouvoir l'exploiter par la suite.

Téléchargement

La première étape est de télécharger jQuery sur le site officiel, pour être certain d'avoir la dernière version. Pour se faire, rendez-vous à l'adresse : http://jquery.com/. Vous y trouverez sur la droite un bloc où sont disponibles deux compressions possibles :

- **Production** : le code est compressé au maximum et optimisé pour l'utilisation en production (Lien direct de téléchargement : http://code.jquery.com/jquery.min.js)
- **Développement** : le code n'est pas compressé et est prévu pour être lu et/ou pour développer avec pour contribuer au

projet notamment (Lien direct de téléchargement : http://code.jquery.com/jquery.js)

Choisissez la version « Production », car nous n'allons pas toucher au fichier, simplement l'utiliser. Aussi, j'ose espérer que vous êtes un bon développeur ayant toujours à l'esprit le souci d'alléger vos pages au maximum pour un temps de chargement optimal. Notez que le temps de chargement d'une page influe maintenant sur le référencement naturel de celle-ci.

Copiez le fichier vers votre projet. Je vous suggère l'adresse `/js/libs/jquery.min.js` pour une organisation optimale de vos scripts. En effet, si vous utilisez plusieurs bibliothèques (*library* en anglais, d'où « libs ») vous pourrez les séparer de vos propres scripts en les rangeant dans un dossier distinct. Il n'est en aucun cas obligatoire de ranger ce fichier à cet endroit, vous pouvez le mettre où bon vous semble, ceci n'étant qu'une proposition pour plus de clarté dans votre projet.

Dans ce tuto, je considèrerais l'arborescence suivante :

Code : Autre

```
/
|- index.html
|- js/
    |- libs/
        |- jquery.min.js
    |- Nos fichiers JS ici
```

Pensez à vérifier l'encodage du fichier pour qu'il soit le même que celui de vos pages. Généralement, on choisira d'**encoder nos fichiers et pages en UTF-8**.

Pour changer l'encodage d'un fichier, si vous êtes sous UNIX et que vous êtes fan de la console, il existe la commande

Code : Bash

```
iconv -f iso-8859-1 -t utf-8 votre_fichier
```

.

Autrement, voici une liste des manipulations pour les éditeurs les plus répandus :

- **Notepad++** : allez dans Paramétrage, Préférences, sélectionnez l'onglet Nouveau document / Répertoire puis cochez UTF-8 sans BOM ;
- **Eclipse** : cliquez sur Window > Préférences > dans le menu de gauche sur General > en bas sur Workspace > dans la boite Text file encoding cochez Other et sélectionnez UTF-8 ;
- **Dreamweaver** : cliquez sur Edition > Préférences > Nouveau document > Codage par défaut ;
- **Zend Studio** : allez sur Tools > Desktop > Apparence ;
- **gedit** : cela se fait lors de l'enregistrement du fichier. Dans la boite de dialogue ouverte après avoir cliqué sur Fichier > Enregistrer sous..., choisissez Locale actuelle (UTF-8) dans la liste déroulante située en bas.

Ajout de jQuery à nos pages HTML

Le fichier est maintenant prêt à être exploité dans vos pages HTML, il ne reste plus qu'à l'importer, tentons de le faire **proprement**.

La balise script

La manière la plus classique et sûre pour que l'import soit effectué sans souci est certainement l'utilisation de la balise script vers le fichier téléchargé préalablement. Pour rappel la balise script se présente ainsi :

Code : HTML

```
<script src="<URL_DU_SCRIPT>"></script>
```

Vous pourrez noter que je ne précise pas `type="text/javascript"` car je considère que l'**on développe en HTML5**. Celui-ci considérant la **balise script** comme contenant du **JavaScript par défaut**.
Si vous développez votre site dans une **version antérieure à HTML5, la bonne syntaxe sera** `<script type="text/javascript" src="<URL_DU_SCRIPT>"></script>`

Il est à noter que les fichiers JS se chargent de manière synchrone : tant que le premier fichier JS n'est pas chargé, le navigateur ne continue pas l'affichage de la page, il attends d'avoir la totalité du fichier pour continuer à avancer. Ainsi, pour fluidifier le chargement, on préfèrera placer les fichiers JS après tout le contenu pour que celui-ci ne soit chargé qu'une fois que notre visiteur a de quoi se mettre sous la dent. La balise `script` est donc à placer **juste avant la fermeture du body**, après tout votre contenu. Ainsi, vous ne bloquerez pas le chargement de votre page en attente de votre script. Si toutefois votre fichier est primordial dans le fonctionnement de votre page, il faudra le mettre dans le head, auquel cas il sera chargé avant le contenu et s'exécutera immédiatement.

Attention, jQuery sera utilisé par vos fichiers *.js chargés via cette balise script, veillez donc à placer l'import de jQuery avant les différents imports de vos scripts. En effet, les scripts sont chargés un à un dans l'ordre où ils sont placés dans le code HTML. Si vous placez un de vos *.js avant jQuery, celui-ci tentera de chercher à exécuter des choses qui ne sont pas encore chargées.

Nous obtenons donc le code suivant :

Code : HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Titre</title>
    </head>

    <body>
        <p>
            Contenu de la page
        </p>
        <script src="/js/libs/jquery.min.js"></script>
    </body>
</html>
```

Utilisons l'API Google

Une grande partie de **vos visiteurs** font chaque jour des **recherches sur Google**, qui utilise jQuery. Le **cache de Google** étant très bien configuré, le fichier jquery.min.js est déjà en cache chez le visiteur. Quoi de mieux que d'**utiliser le cache navigateur** de votre visiteur ? Cela présente plusieurs avantages :

- **alléger** la bande passante de votre site ;
- **accélérer** le chargement des pages.

Tout cela est basé sur le fait que lorsqu'un visiteur se rend sur un site< et que la mise en cache est bien configurée, les fichiers JavaScript, images, etc. (dont jQuery) sont mis en cache par le navigateur chez le client. jQuery n'est donc pas re-téléchargé par le visiteur à chaque site qu'il visite et qui utilise jQuery. Cela arrange tout le monde, profitons-en !

Voyons comment faire appel à l'API de Google prévue à cet effet. Nous utiliserons également la balise script, nous allons simplement aller chercher le fichier sur les serveurs de Google à l'adresse `//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js`, le double slash au début permettant

d'avoir une adresse en fonction du protocole que vous utilisez : http ou https.

Maintenant, imaginons qu'un jour, les serveurs de Google soient indisponibles ou qu'ils changent leur API : il faut prévoir une solution de secours. Nous alors donc nous resservir de notre fichier en local avec une petite touche de JavaScript. Nous obtenons alors quelque chose comme cela :

Code : HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Titre</title>
    </head>

    <body>
        <p>
            Un paragraphe ici
        </p>
        <script
src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
        <script>window.jQuery || document.write('<script
src="/js/libs/jquery.min.js"><\/script>')</script>
    </body>
</html>
```

La ligne de JavaScript signifie que s'il n'y a pas jQuery, c'est que l'import juste au dessus n'a pas fonctionné. Nous ajoutons donc à la page, via JavaScript, un import de jQuery depuis notre fichier local. Ainsi, nous profitons du cache de Google tout en étant certains de la disponibilité de jQuery.

Hello World, le grand classique

Pour savoir si vous avez réussi à importer jQuery convenablement, voici un « Hello World » utilisant jQuery.

Code : HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Titre</title>
    </head>

    <body>
        <p>
            Un paragraphe
        </p>
        <script
src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
        <script>window.jQuery || document.write('<script
src="/js/libs/jquery.min.js"><\/script>')</script>
        <script>$('body p').html('Hello World');</script>
    </body>
</html>
```

Vous comprendrez mieux la petite ligne de code dès la prochaine partie.

Si vous voyez encore « Un paragraphe » c'est que jQuery **n'est pas importé**, vérifiez donc les étapes précédentes. Si votre page affiche « Hello World » vos pages sont prêtes à être dynamisée en utilisant jQuery !

jQuery() ou $() ?

Vous avez vu plus haut dans le « Hello World » apparaître la fonction `$()` et oui, c'est court comme nom et il vaut mieux ! En effet, elle va nous servir un peu partout.

jQuery() ou $() ?

L'heure est au choix, si vous êtes prudents vous préférerez prévenir les conflits, si vous êtes certains que vous n'utiliserez que jQuery, alors ne vous souciez pas d'éventuels conflits : c'est à vos risques et périls.

Protéger l'alias $()

$() est un alias (un synonyme) de jQuery() qui n'est autre que la fonction principale de jQuery, dès que nous aurons besoin de jQuery, il faudra y faire appel. Le souci c'est que jQuery n'est pas la seule librairie JavaScript a avoir choisi cet alias, si vous utilisez plusieurs librairies, vous allez avoir des conflits si vous ne protégez pas les différentes parties de code, pour ce faire c'est très simple :

Code : JavaScript

```
jQuery(document).ready(function($){
    // Là on peut utiliser $() en ayant la certitude que c'est
jQuery, sans conflit possible
});

// Version courte :
jQuery(function($){
    // $() est ici disponible comme alias de jQuery()
});
```

Le `jQuery(document).ready()` va attendre que la page (HTML + CSS + JS) soit chargée pour exécuter le code qui est dans la fonction. Le mieux reste de coder proprement et d'importer jQuery tout en bas du body puis de faire/importer nos scripts qui l'utilisent. Le chargement d'une page HTML est synchrone, c'est-à-dire qu'ils sont chargés dans l'ordre qu'ils apparaissent dans le code, de fait, si `jquery.js` est placé avant notre `script.js`, rien ne sert de faire cette vérification puisque c'est le dernier élément de la page qui est chargé.

Code : JavaScript

```
// Version encore plus courte qui suppose que la page soit chargée
(function($){
    // On peut ici utiliser $() en tant que jQuery()
})(jQuery);
```

Utiliser $() directement

Si vous êtes certains que vous ne coderez qu'avec jQuery sur votre site, et jamais rien d'autre, vous pouvez tout à fait utiliser directement `$()` en veillant toujours à avoir importé jQuery plus haut dans la page (en général, juste avant notre script). Si vous procédez ainsi, vous pourrez directement taper votre code comme nous allons le voir dans le prochain chapitre !

Code : JavaScript

```
// Import de jQuery
$(...)... // Utilisation de jQuery
```

Maintenant que votre page est prête à utiliser jQuery, et ce, de manière optimisée si vous passez par l'API Google, vous pouvez apprendre la première brique fondamentale de jQuery que sont **les sélecteurs**.

Les Sélecteurs

Pour agir sur la page, il va falloir sélectionner les divers éléments la composant. L'une des grandes forces de jQuery réside tout justement en sa simplicité de sélection des éléments des pages. La bibliothèque réutilise le potentiel des sélecteurs CSS en les complétant de sélecteurs additionnels.

Les sélecteurs CSS

Les sélecteurs que vous connaissez certainement et utilisez en CSS sont pratiques et permettent de cibler précisément n'importe quel élément d'une page bien construite. Si nous pouvions les utiliser en JavaScript, cela serait quand même bien plus pratique que les `getElementById()` et autres fonction au nom à rallonge, non ? Eh bien, avant de développer jQuery, son créateur avait créé Sizzle : une bibliothèque JavaScript qui permet la sélection des éléments avec les sélecteurs CSS. Celle-ci a été intégrée à jQuery, qui profite ainsi de la souplesse d'utilisation de Sizzle. Voyons dès maintenant comment sélectionner un élément.

Comment utiliser les sélecteurs ?

Les sélecteurs que nous allons voir par la suite sont à utiliser très simplement de la façon suivante :

Code : JavaScript

```
jQuery("<selecteur>");
// Si on utilise uniquement jQuery ou que l'on est
// dans un contexte bien défini, $ est un alias de jQuery
$("<selecteur>");
```

En sachant que `<selecteur>` sera votre sélecteur constitué des éléments détaillés ci-après.

Attention toutefois, si vous utilisez des caractères spéciaux dans vos sélecteurs `!"#$%&'()*+,./:;<=>?@[\]^`{|}~` il faudra les protéger en les précédant d'un double backslash `\\`. Par exemple pour sélectionner cet élément HTML : `5` il faudra faire ainsi : `$("head\\.nav\\.link\\.lbl");`. Le mieux étant de ne pas utiliser ce genre de caractères pour éviter tout problème dans le CSS comme dans le JavaScript.

Un sélecteur `$("<selecteur>");` va retourner un objet jQuery qui contient tous les éléments qui ont satisfait le sélecteur, et s'il n'y en a aucun cela vaudra null.

Les sélecteurs CSS compatibles avec jQuery

La grande majorité des sélecteurs CSS ont été directement implémentés à jQuery, certains autres ne sont pas compatibles. jQuery est très à jour et est déjà prêt à supporter CSS3 et ses nouveaux sélecteurs.

Le joker

Expression : `*`
Il permet la sélection de tous les éléments. Celui-ci est à utiliser avec précaution, si on l'exécute seul, on récupère également les éléments html, body, ... ce qui n'est généralement pas désiré. On l'utilisera plutôt pour récupérer tous les éléments à l'intérieur d'un autre, par exemple : `header nav *` pour récupérer tous les éléments dans les nav des header.

Le nom de la balise HTML

Expression : **`elem`**
Exemples : **`span`** ; **`div`** ; **`p`** ; **`section`**
On récupèrera alors tous les éléments qui sont ce type d'élément. Il est également possible de chaîner les éléments : **`elem1 elem2`** qui ira chercher tous les éléments **`elem2`** dans **`elem1`**.

Tout comme en CSS, on peut tester la parenté et la position par rapport à ses éléments frères :

- **`elem1`** `>` **`elem2`** : ira chercher tous les **`elem2`** parmi les enfants directs de **`elem1`**, sans aller plus loin dans les sous-enfants ;
- **`elem1`** `+` **`elem2`** : sélectionnera tous les **`elem2`** situés tout de suite après un **`elem1`** ;
- **`elem1`** `~` **`elem2`** : pour chaque **`elem1`**, on va chercher tous les éléments **`elem2`** suivant **`elem1`** dans le même

parent.

La chaînabilitité des sélecteurs elemX est également applicable aux sélecteurs qui vont suivre.

L'id

Expression : `#id`
Exemple : `#part_2`
Sélectionne l'élément ayant l'id **`id`**. Normalement, ce sélecteur ne renvoie qu'un élément puisqu'un id est censé être unique. Les id sont également chaînables, comme indiqué plus haut, bien qu'en soit cela soit inutile puisqu'un id est théoriquement dédié à un seul élément, pas besoin de filtrer à travers un élément parent.

La classe

Expression : **`.class`**
Exemple : **`.souligne`**
On a alors, à la manière d'une sélection par le nom de la balise, tous les éléments qui ont la classe indiquée. Ici la chainabilité des sélecteurs de classe ont bien un sens. En effet, une classe est faite pour être utilisé à plusieurs endroits, on pourra donc trouver des choses comme **`.titre_1 .ico`** ou encore `#corps` **`h3 .ico`**. On peut également filtrer avec plusieurs classes sur le même élément, typiquement : **`html.js.cssrotation`**.

La présence d'attribut sur un élément

Expression : **`elem`**`[`**`attr`**`]`
Exemple : **`img`**`[`**`alt`**`]`
Retourne tous les éléments `elem` ayant l'attribut `attr`, quelque soit la valeur de `attr`, pourvu que l'attribut soit présent.

On trouvera également le test de présence de plusieurs attributs :
Expression : **`elem`**`[`**`attr1`**`][`**`attr2`**`]`
Exemple : **`img`**`[`**`alt`**`][`**`src`**`]`
Le sélecteur vérifiera la présence de chacun des attributs **`attrX`** sur l'élément **`elem`**

Une valeur particulière pour un attribut

Expression : **`elem`**`[`**`attr`**`="val"]`
Exemple : **`a`**`[`**`href`**`^="#"]` qui permettra de sélectionner tous les liens vers des ancres.
De cette manière, on va récupérer tous les éléments `elem` qui ont un attribut `attr` ayant la valeur `val`.

Il existe toute une série de variantes au niveau du test de la valeur de l'attribut. En voici une rapide description :

- **`elem`**`[`**`attr`**`!="val"]` : la valeur de **`attr`** doit être différente de **`val`** ;
- **`elem`**`[`**`attr`**`^="val"]` : la valeur de **`attr`** doit exactement commencer par **`val`** ;
- **`elem`**`[`**`attr`**`$="val"]` : à l'inverse ici **`attr`** doit exactement terminer par **`val`**.
- **`elem`**`[`**`attr`**`*="val"]` : **`attr`** doit simplement contenir **`val`**.

Des sélecteurs du même genre, mais un peu plus particuliers, ils sont basés sur des séparateurs.

- **`elem`**`[`**`attr`**`~="val"]` : **`attr`** contient un ou plusieurs mots séparés par des espaces dont l'un est exactement **`val`** ;
- **`elem`**`[`**`attr`**`|="val"]` : va sélectionner les **`elem`** dont l'attribut `attr` sera strictement égal à **`val`** ou bien, que la valeur de l'attribut soit une suite de mots séparés de tirets (**`-`**) et que le premier est égal à **`val`**.

On peut bien sûr tester la valeur de plusieurs attributs, tout en utilisant des tests différents sur les attributs, ce qui dans des cas extrêmes pourrait donner des choses comme ceci : **`a`**`[`**`href`**`^="#"][`**`data-bulle`**`="true"][`**`data-truc`**`|="une-valeur"][`**`data-class`**`="js"]`

Sélecteur multiple

Parfois nous voulons avoir deux sélecteurs différents et leur faire subir le même traitement. Pour cela, comme en CSS, on peut séparer les sélecteurs d'une virgule. Ainsi, on obtient quelque chose comme :

Code : JavaScript

```
$("body.accueil #bandeau_pub, aside .pub");
```

Les Pseudo-classes

Expression : `elem:pseudo-classe`
Exemple : `a:nth-child(2n)`
Sizzle permet d'utiliser toutes les pseudos-classes que propose CSS3 à l'exception de celles qui suivent :

- `::selection`
- `:first-letter`, `:first-line`
- `:before`, `:after`
- `:link`, `:visited`, `:active`, `:hover`, `:target`
- `:nth-last-child`
- `:nth-of-type`, `:nth-last-of-type`, `:first-of-type`, `:last-of-type`, `:only-of-type`

Nous verrons plus en détail les pseudos-classes dans la prochaine sous-partie.

Résumé

Pour résumer, voici un exemple de sélecteur tout à fait plausible en pratique, mais tordu je dois vous l'avouer :

Code : JavaScript

```
$("html.js.cssgradient #corps > section p[id*="part"].contenu
a[href^="#"] + .infobulle");
```

Si on regarde ce code de près, on déduit alors que le sélecteur ira chercher les éléments dans **`html`** s'il a les classes **`js`** et **`cssgradient`**, puis il y trouvera un élément ayant l'id **`corps`** qui a un ou des enfant(s) direct(s) **`section`**. Nous avons fait une partie du chemin, mais ça n'est pas fini. Dans les **`section`** trouvées, on cherche alors les paragraphes **`p`** ayant un id qui contient la suite de caractères "part" et qui a également la classe **`contenu`**. Dans ces paragraphes on ira chercher les éléments ayant la classe **`infobulle`** qui sont directement précédés d'un lien dont l'attribut **`href`** commence strictement par le caractère #, c'est à dire un lien vers une ancre. Le chemin a été long, mais nous y somme parvenus !

Code : HTML

```
<html class="js cssgradient">
    <... id="corps">
        <section>
            <p id="uhqsfouhg_part123" class="contenu">
                <a href="#...">...</a>
                <... class="infobulle"></...>
```

Voici une arborescence du code qui satisfera notre sélecteur. Lorsqu'il y a "..." c'est que ce qui y est nous importe peu.

Tout bon développeur le sait, rien ne vaut la documentation officielle lorsque l'on cherche quelque chose de précis ; voici donc la documentation officielle des sélecteurs CSS3 : http://www.w3.org/TR/css3-selectors/.

Les pseudos-classes gérées par jQuery

Comme je vous l'avais promis, nous voilà au moment ou nous allons découvrir les pseudo-classes CSS que jQuery gère, mais aussi celles que jQuery ajoute. Pas de souci à se faire au niveau des pseudo-classes CSS, elles sont compatibles tout navigateurs puisque c'est jQuery qui gère tout cela. Les pseudo-classes vont nous permettre de sélectionner certains éléments encore plus facilement ! D'autre part, ils sont très utilisés, ce qui leur vaut au moins une partie dédiée que voici.

À noter que toutes les pseudos-classes que nous allons voir sont à utiliser comme en CSS, c'est à dire comme ceci : `$("elem:pseudo-classe");`
ou encore `$(":pseudo-classe");` qui équivaut en fait à `$("*:pseudo-classe");`

La négation

Il est parfois très important d'avoir connaissance de la négation pour éviter une série de sélecteurs à la place d'un. Il s'agit de la pseudo-classe :not('selecteur') dont il est ici question. Il est possible d'être dans la situation où nous avons 3 tailles pour nos liens : `.small`, `.normal`, `.big` et que l'on veuille ne sélectionner que les `.small` et les `.normal`. Pour l'utiliser c'est très simple :

Code : JavaScript

```
$('a:not(.big)');
// Équivalent (dans notre cas) à :
$('a.small, a.normal');
```

On aurait également pu faire comme sur la deuxième ligne, mais vous voyez que c'est plus long : imaginez avec des sélecteurs plus complexes !

Les Formulaires

jQuery va nous être utile lorsque l'on voudra traiter les formulaires côté client pour les valider avant de les envoyer sur le serveur, par exemple. Ou encore, envoyer les données en asynchrone (AJAX). Pour pouvoir envoyer ces informations, ces valeurs, il va nous falloir les récupérer dans la page. Une première solution, qui vous vient naturellement à l'esprit avec ce que vous avez appris jusqu'ici, consiste à sélectionner une checkbox (par exemple) ainsi : `$("input[type='checkbox']");`. C'est là que jQuery va encore nous simplifier la tâche !

Les boutons

En HTML5, tout comme dans les précédentes versions, il y a 4 sortes de boutons :

- bouton simple : `<input type="button">` ;
- bouton de remise à zéro : `<input type="reset">` ;
- bouton de soumission : `<input type="submit">` ;
- bouton de soumission en image : `<input type="image">`.

En HTML5, il est possible et recommandé de ne pas fermer la balise input.

jQuery propose une pseudo-classe associée respectivement à chacun de ces boutons :

- `:button`
- `:reset`
- `:submit`
- `:image`

Les noms étant directement repris des types, quoi de plus simple à retenir ?

Les champs

Le plus imposant des champs est certainement textarea, mais ayant une balise rien qu'à lui, pas besoin de pseudo-classe. En revanche pour les champs de texte et mot de passe, cela devient intéressant :

- champ texte : `<input>` plus connus ainsi : `<input type="text">` ;
- champ mot de passe : `<input type="password">` ;

Auxquels on trouvera les pseudo-classes associées suivantes :

- `:text`
- `:password`

Encore une fois, même nom pour la pseudo-classe que le type du champ.

Les options

Seront regroupés ici les les champs radio, select, checkbox ; autrement dit, tout ce qui permet un choix parmi une liste définie. Sauf que, les select n'ont pas droit à leur pseudo-classe, en effet ils sont privilégiés tout comme les textarea avec une balise dédiée et donc simplement sélectionnable.
Toujours selon le même modèle :

- bouton radio : `<input type="radio">` ;
- checkbox : `<input type="checkbox">` ;

Pour sélectionner ces éléments nous auront donc, vous l'aurez deviné :

- `:radio`
- `:checkbox`

Mais, ce qui est très intéressant c'est que l'on peut également filtrer les champs radio/checkbox qui sont cochés grâce à la pseudo-classe `:checked`. Il est donc possible, par exemple, de récupérer toutes les checkbox cochées en faisant $(":checkbox:checked").

Le champ d'envoi de fichier

Dernier type de champs sélectionnable avec une pseudo-classe implémentée par jQuery : les champs de type file `<input type="file">` avec la pseudo-classe `:file`, comme vous auriez pu vous en douter.

Les états des champs

Les champs input peuvent être désactivés comme ceci : `<input type="text" disabled>`. C'est également valable pour les autres balises qui composent un formulaire (textarea, select, ...). Ainsi, ils sont affichés mais l'utilisateur ne peut pas modifier leur valeur. jQuery nous a une fois de plus concocté une pseudo-classe pour sélectionner les champs activés et une autre pour les champs désactivés.

- `:enabled` pour les champs « normaux »
- `:disabled` pour les champs ayant l'attribut `disabled`

Il existe également la pseudo-classe `:focus` qui permet de récupérer l'élément sur lequel l'utilisateur a le focus. Celle-ci fonctionne sur tous les éléments, même en dehors des formulaires, mais elle est particulièrement utile dans ce contexte.

Les Enfants

Nous avions vu dans la partie précédente des sélecteurs permettant de sélectionner certains enfant, mais voici quelques pseudos-classes supplémentaire bien pratiques, qui combinées à d'autres permettront de contourner le problème du non-support de certains sélecteurs CSS3.

Par rapport à leur position dans leur parent

Un enfant peut être le premier, le n^{ième} ou encore le dernier, en sachant que le premier et le dernier sont tout simplement des cas particuliers parmi les n^{ième} positions. Attention toutefois, ici on ne va pas aller chercher l'élément étant à la position n mais on va ne renvoyer l'élément que s'il est à la position demandée, un exemple plus loin vous expliquera mieux tout cela. Pour répondre à cette problématique nous avons trois pseudos-classes, dont deux ne sont que des raccourcis :

- `:nth-child()` à utiliser ainsi `:nth-child(5)` avec 5, l'index de l'élément dans son parent ;
- `:first-child` équivaut à `:nth-child(1)` ;
- `:last-child`.

Ces sélecteurs étant assez complexes, voici un exemple :

Code : HTML

```
<body>
 <div id="contenu">
  <ul>
   <li><a href="#">Lien 1</a></li>
   <li><a href="#">Lien 2</a></li>
   <li><a href="#">Lien 3</a></li>
   <li><a href="#">Lien 4</a><a href="#">Lien 4.1</a></li>
  </ul>
  <ul>
   <li><a href="#">Lien 5</a></li>
   <li><a href="#">Lien 6</a></li>
   <li><span>Là c'est un span</span><a href="#">Lien 7</a><a
href="#">Lien 7.2</a></li>
   <li><a href="#">Lien 8</a></li>
  </ul>
 </div>

 <!-- Importer jQuery -->
 <script>
  $premier = $('#contenu :nth-child(2)');
  $second = $('#contenu a:first-child');
 </script>
</body>
```

Le premier sélecteur va aller chercher dans le #content tous les éléments qui ont la position 2 par rapport à leur parent direct. Ici on aura donc le second `<ul>`, le deuxième `<a>` (celui qui contient « Lien 4.1 ») et le deuxième `<li>` de chaque `<ul>`.
Le second sélecteur ajoute une contrainte en plus : il faut que les éléments potentiels soient des `<a>`. Ce qui aura ici pour effet de sélectionner tous les `<a>` qui sont les premiers éléments de leur parent. Ici c'est le cas pour presque tous les liens sauf « Lien 4.1 » car il n'est pas le premier de son parent, « Lien 7 » c'est le premier `<a>` mais pas le premier de son parent, de même pour « Lien 7.2 ».

Par rapport à leur position en fonction du sélecteur

Les pseudo-classes qui vont suivre dans ce paragraphe sont ajoutées par jQuery, elles n'existent pas en CSS.

Contrairement aux sélecteurs ci-dessus, qui sont du CSS, jQuery rajoute des sélecteurs plus simples à utiliser et qui vont nous satisfaire dans la plupart des cas. Que diriez-vous de sélectionner le n^{ième} paragraphe dans une div particulière ? Beaucoup plus simple à comprendre, n'est-ce pas ? Voici donc trois pseudo-classes qui vont vous rappeler les précédentes, mais dont le fonctionnement est différent.

- `:eq(n)` qui permet de récupérer l'élément en position n (n doit être positif ou nul) du type précisé (si précisé) sinon l'élément n, tout simplement ;
- `:first` équivaut à `:eq(0)` car les indices avec `:eq` commencent à 0 contrairement à :nth-child() qui commence à 1, ceci est du aux conventions différentes du CSS et du JS ;
- `:last` va récupérer le dernier élément du type (si précisé) sinon le dernier élément, tout simplement.

Enfant unique

Un parent peut aussi n'avoir qu'un seul enfant. Pour les trouver, nous avons la pseudo-classe `:only-child`, tout simplement.

Exemple

Ces sélecteurs étant un peu pointus, voici un exemple

Code : HTML

```
<header>
 <nav>
  <a href="#">Un lien</a>
  <a href="#">Un autre lien</a>
  <a href="#">Vers une ancre</a>
  <a href="#">Retour en haut</a>
 </nav>
</header>
```

Ici, le sélecteur `$("header:only-child");` va nous renvoyer l'élément nav.
Si l'on souhaite récupérer le lien « Vers une ancre » on utilisera ce sélecteur `$("header nav a:nth-child(3)");` tandis que pour avoir « Un lien » on va plutôt utiliser `$("header nav a:first-child");`.

Pair ? Impair ?

Il existe deux sélecteurs très pratiques que sont les suivants :
:even qui sélectionne les éléments dont l'index est pair
:odd qui prends lui les impairs

Il est souvent question de colorer une ligne sur deux d'une couleur différente dans les tableaux, pour une meilleure lisibilité. Cela est également faisable en CSS avec la pseudo-classe :nth-child(2n) pour les pairs et :nth-child(2n+1) pour les impairs. jQuery nous offre une nouvelle fois une notation très simplifiée et, qui plus est, fonctionne sur tous les navigateurs.
Vous savez sélectionner des éléments dans vos pages, et ce, de différentes façons. Je vous invite à consulter la documentation (en anglais) pour découvrir la liste complète et détaillée des sélecteurs qui sont gérés par jQuery à cette adresse :
http://api.jquery.com/category/selectors/
Que faire une fois que nous avons réussi à sélectionner ceux-ci ? C'est ce qui vous attend dans le chapitre suivant avec **les méthodes**.

Les Méthodes

Sélectionner les éléments c'est bien, les manipuler c'est mieux ! Les méthodes sont là pour ça, et jQuery en intègre beaucoup. Voyons ce que sont les méthodes, leurs propriétés pour finir sur les différentes formes de paramètres que nous pourrons leur donner.

Qu'est-ce qu'une méthode ?

Souvenez-vous, nous voyions dans le chapitre précédant les sélecteurs que nous utilisions ainsi : `$('#selecteur')` par exemple. Ceci permet de récupérer un objet jQuery sur lequel nous allons pouvoir appliquer des actions, modifications, ou tout simplement de simples lectures d'attributs ou de propriétés ; c'est cela que permettent les méthodes. Pour résumer, une méthode permet une action sur un objet jQuery récupéré préalablement.

À partir d'ici il va m'arriver de stocker des objets jQuery dans des variables pour pouvoir leur appliquer des méthodes différentes sans devoir sélectionner l'élément à chaque fois. La convention sera celle-ci : une variable contenant un objet jQuery sera précédée d'un dollar, exemple :

Code : JavaScript

```
var $body = $('body');
```

Vous pouvez tout à fait respecter cette convention ou non (d'où son nom), mais celle-ci permet une meilleure compréhension de ce que contiennent les variables, et de fait, de mieux comprendre votre code en général.

Utilisation

Pour appliquer ces modifications, nous utiliserons bien souvent des méthodes de jQuery. Celles-ci se présentent et s'applique à l'objet de cette façon : `$('.class').methode();` La méthode pouvant prendre un ou plusieurs paramètres : `$('#selecteur').methode(250, "Texte");` que nous détaillerons plus tard dans ce chapitre, car ces paramètres peuvent prendre de multiples formes.

Les méthodes simples proposées par jQuery

Comme toute bonne bibliothèque, jQuery nous propose tout une liste de méthodes qui permettent de gérer de plusieurs façons le contenu de nos pages. Voyons ensemble les méthodes les plus simples qui existent, c'est-à-dire celles qui ne prennent pas de paramètres.

Afficher et masquer

Il est très courant de devoir afficher ou masquer des éléments de nos pages, et pour cela nous n'avons que l'embarras du choix puisque nous pouvons faire cela avec ou sans animation. En ce qui concerne les méthodes sans animées elles sont au nombre de deux, j'ai l'honneur de vous les présenter :

- `.hide()` qui permet de masquer l'objet concerné ;
- `.show()` qui affiche l'objet.

Elles vous seront utiles assez couramment notamment lors de vos tests rapides pendant le développement. Celles-ci n'ont pour simple effet que de « jouer » avec la propriété **`display`** en la passant à block pour `.show()` et à none pour `.hide()`. Pour vous prouver que je ne vous mens pas, voici le code tout prêt à tester :

Code : HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Titre</title>
    </head>

    <body>
```

```
            <p class="a-masquer">
                Un paragraphe
            </p>
            <script
src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
            <script>window.jQuery || document.write('<script
src="/js/libs/jquery.min.js"><\/script>')</script>
            <script>$('.a-masquer').hide();</script>
        </body>
    </html>
```

Qui, si vous allez dans votre explorateur de code via F12, devrait produire ceci :

Code : HTML

```
<p class="a-masquer" style="display: none;">
    Un paragraphe
</p>
```

Vous voyez que cette bibliothèque n'est pas magique, elle vous simplifie juste le travail en allégeant le code de vos applications web utilisant le JavaScript.

Supprimer un élément

Il est parfois utile d'alléger le code source des éléments qui ne sont plus utiles, il existe pour cela une méthode toute simple qui s'applique ainsi :

Code : JavaScript

```
$('.delete').remove();
```

Celle-ci retire purement et simplement tous les éléments ciblés du DOM.

Paramètres et enchaînements de méthodes

Jusqu'ici les méthodes étaient simples, gentilles et pas très évoluées ni paramétrables. Voilà ce sur quoi nous allons désormais nous attarder : les paramètres !

Des paramètres ? Où ça ?

Souvenez-vous de la méthode `.hide()` je vous ai menti quand je vous disais qu'elle était simple et qu'elle ne prenait pas de paramètres.

Où mettre les paramètres ?

La méthode `.hide()`, comme nombre d'autres, peut parfois prendre des paramètres, il suffit de consulter la documentation pour le voir. Vous voyez depuis tout à l'heure des parenthèses vides, et oui, c'est dans celles-ci que nous mettrons les paramètres. Ces paramètres peuvent être de différents types que nous verrons un peu plus loin ici, mais il est toujours indiqué quels sont les types des paramètres dans la documentation, avec des exemples vous permettant de mieux visualiser si ça n'était pas le cas avant.

Lire et comprendre la documentation

Continuons plus en détail, voyons la signature de la méthode `.hide()` que l'on nous donne :

Citation : Documentation jQuery

```
.hide( [duration] [, easing] [, callback] )
```

Que signifie donc tout ce charabia ? Certains sont déjà familiers à ce genre de notation, mais un rappel ne fait pas de mal !

Les paramètres sont désignés par des « mots » entre les parenthèses, ceux qui sont facultatifs étant entre crochets. Ainsi, ici nous avons les paramètres *duration*, *easing* et *callback*. Bien qu'ils soient tous facultatifs, si vous voulez renseigner le paramètre easing **vous devrez donner une valeur pour *duration***, pour conserver l'ordre et la correspondance. En dessous de la signature nous sont donnés les détails sur chacun des paramètres que nous allons voir sans plus tarder.

Pour ceux qui développent souvent hors-ligne, vous pouvez télécharger la documentation de jQuery sur jQAPI.com

Les différentes formes de paramètres

Comme vous venez de le voir, il y a parfois plusieurs paramètres à une méthode, obligatoires ou facultatifs, mais quoi qu'il en soit ils ont un type. Dans l'exemple ci-dessus cité, la documentation nous donne les informations nécessaires :

Citation : Documentation jQuery

duration A string or number determining how long the animation will run.
easing A string indicating which easing function to use for the transition.
callback A function to call once the animation is complete.

Alors, bien sûr c'est en anglais, mais cela reste très compréhensible. Nous avons ici la possibilité d'ajuster la durée de l'animation en donnant un entier/chaîne, de donner le nom d'une fonction de *easing* et une fonction de *callback*. Autant la durée est quelque chose de concret, autant les deux autres termes sont moins clair pour vous.

- Une fonction de *easing* est une courbe qui va définir si l'animation va être linéaire, avec rebond(s), accélérée, ralentie, accélérée puis ralentie avec rebonds, ...
- Une fonction de *callback*, c'est la fonction que la méthode va lancer une fois qu'elle aura fini son travail. Typiquement ici si on défini une *duration* égale à 500ms, alors le *callback* sera lancé après les 500ms d'animation, et même si cette valeur est nulle, cela sera appelé après que l'élément ait été masqué.

La fonction de callback, entre autres, sera une fonction anonyme qui pourra, si nécessaire, appeler à son tour des fonctions nommées. Pour rappel une fonction anonyme va être ainsi :

Code : JavaScript

```
$('#selecteur').hide(500, "linear", function(){
    // Votre code, une fois l'élément masqué
});
```

jQuery c'est avant tout du JavaScript, et de ce fait les durées sont toujours en millisecondes ! Par ailleurs, il est également possible d'utiliser les constantes de jQuery que sont "fast" et "slow" qui correspondent respectivement à des durées de 200ms et 600ms.

Nous verrons des méthodes qui demanderont des objets, vous savez ces espèces de tableaux associatifs avec des accolades ? Voici un petit exemple :

Code : JavaScript

```
$('#selecteur').css({
    'height': '15px',
    'width': '3px'
});
```

Ce qu'il faut retenir de ce code c'est qu'une méthode peut prendre un objet en paramètre. Dans l'exemple ici, c'est la méthode .css() qui est utilisée, nous la verrons dans un chapitre ultérieur.

Plusieurs méthodes à la suite

Sachant ce qu'est une méthode, et que nous savons bien nous en servir, nous allons pouvoir enchaîner avec les enchaînements ! Que sont-ils ? À quoi servent-t-ils ? Toutes les réponses dans ces quelques lignes (paragraphes) ! L'idée qu'il faut retenir de l'enchaînement des méthodes, c'est que ce sont plusieurs méthodes à la suite. En effet, cela consiste tout simplement à appliquer plusieurs méthodes à un même objet (susceptible d'évoluer suite à l'application de certaines méthodes) de manière séquentielle. Mais alors, comment faire ? Si l'on suit ce que l'on a vu jusqu'ici, nous aurions cela :

Code : JavaScript

```
$('#anim').hide(0, "", function(){
    $('#anim').show();
});
```

Ce code est en soit inutile, mais il explique ce qui est faisable. L'écriture est lourde et jQuery est fait pour que l'écriture soit légère ! C'est pour cela que le point sera notre liant préféré dans notre cuisine jQuery.

Code : JavaScript

```
$('#anim').hide().show();
```

Ce code beaucoup plus clair et concis aura le même effet que le précédent code. Pas besoin de mettre des valeurs pour les paramètres dont on ne se sert pas, tout est géré !
Le fonctionnement des méthodes n'a plus aucun secret pour vous ! Pourquoi ne pas en découvrir quelques-unes ? C'est l'objet du prochain chapitre, allez-y !

Méthodes principales

Les principes des méthodes en poche, nous pouvons voir et comprendre les principales méthodes fournies par jQuery.

Méthodes traversantes

Souvenez-vous du chapitre sur les sélecteurs (oui c'est loin) et bien je ne vous ai pas encore tout dit ! En effet, il existe des méthodes de sélection, dites traversantes, qui complètent les sélecteurs CSS.

Parcourir un objet jQuery

Chercher un élément

Cela vous arrivera plus d'une fois d'avoir un objet jQuery dans lequel vous voudrez chercher un élément particulier, et pour cela tout est prévu. En effet, il y a même deux manières de procéder. La première est d'utiliser la méthode `.find()` :

Code : JavaScript

```
var $conteneur = $('.conteneur:eq(0)');
$conteneur.find('.a-trouver');
```

Cette méthode est strictement identique à celle-ci :

Code : JavaScript

```
var $conteneur = $('.conteneur:eq(0)');
$('.a-trouver', $conteneur);
```

Ici la méthode `.find()` va aller chercher dans l'objet jQuery tous les éléments ayant la classe `.a-trouver`. Les deux méthodes sont identiques puisque que dans le second cas, jQuery va appeler la méthode `.find()`. C'est plus court à écrire, ça revient au même, à vous de choisir ce que vous préférez !

Le parcours

S'il y a bien une chose qui est pratique avec jQuery, c'est de pouvoir parcourir un objet jQuery très simplement. Lorsque vous utilisez les sélecteurs, il est possible que celui-ci sélectionne plusieurs éléments, dans ce cas vous aurez parfois besoin de la méthode `.each()` qui prend en paramètre une fonction de *callback* comme ceci :

Code : JavaScript

```
$('.selecteur').each(function(){
    // Votre code
});
```

Maintenant, les plus malicieux se sont certainement posés la question suivante : comment récupérer l'élément courant dans cette boucle ? Et bien c'est l'ultime sélecteur que je ne vous avais pas donné, j'ai nommé `$(this)`. Nous aurions alors par exemple :

Code : JavaScript

```
$('.to-hide').each(function(){
    $(this).hide();
});
```

Le `$(this)` est également disponible dans les fonctions de *callback* pour récupérer l'élément sur lequel avait été appliqué la méthode « mère ». Si vous avez bien suivi, le code ci-dessus masquera tous les éléments ayant la classe *to-hide*. Ce code est équivalent à `$('.to-hide').hide();` dans le cas présent, mais il sera utile lorsque nous devrons parcourir une liste d'élément pour ensuite faire des manipulations à l'intérieur.

Filtrons notre résultat

Une fonction moins courante, mais très utile : `.filter()` ! Elle va nous permettre de ne sélectionner qu'une partie des éléments sélectionnés dans un objet jQuery. Dans les cas les plus simples ça n'est spécialement utile, car très facilement simplifiable :

Code : JavaScript

```
$lignesPaires = $('table tbody tr').filter(':even');
// Équivalent, mais moins performant :
$lignesPaires = $('table tbody tr:even');
```

Mais il faut savoir que l'utilisation de `.filter()` est plus performante. La documentation nous recommande de sélectionner tout d'abord nos éléments avec `$('selecteur')` en utilisant un sélecteur en CSS pur pour profiter de la rapidité de la fonction JavaScript `querySelectorAll()`. Il est également possible d'utiliser `.filter()` en lui donnant en paramètre un objet jQuery ou un élément : voir la documentation pour plus de détails, rien ne vaut la source !

Le plus intéressant dans cette méthode est la possibilité de lui passer une fonction anonyme qui devra renvoyer un booléen disant si l'élément doit être sélectionné ou pas. En clair, `.filter()` va faire comme le `.each()` : parcourir les éléments un à un et faire appel pour chacun d'eux à la méthode anonyme. Ainsi, plutôt que de faire des choses complexes avec `.each()` et des tests dedans, `.filter()` nous fait ça très simplement. En voici une démonstration :

Code : HTML

```
<body>
    <div id="contenu">
        <ul>
            <li><a href="#">Lien 1</a></li>
            <li><a href="#">Lien 2</a></li>
            <li><a href="#">Lien 3</a></li>
            <li><a href="#">Lien 4</a><a href="#">Lien 4.1</a></li>
        </ul>
        <ul>
            <li><a class=".intrus" href="#">Lien 5</a></li>
            <li><a href="#">Lien 6</a></li>
            <li><span>Là c'est un span</span><a href="#">Lien
7</a><a href="#">Lien 7.2</a></li>
            <li><a href="#">Lien 8</a></li>
        </ul>
    </div>

    <!-- Importer jQuery -->
    <script>
        $tousLesTrois = $('a').filter(function(index){
            return index != 3 && index % 3 == 0 || $('.intrus',
this);
        });
    </script>
</body>
```

On obtient alors dans notre variable uniquement les éléments dont l'index est divisible par 3. Dans notre cas les liens : « Lien 1 » (index 0), « Lien 7 » (index 6). On aurait pu avoir « Lien 4 » mais son index étant de 3, la première condition est à faux. Vous pouvez très largement complexifier le contenu de la fonction sachant qu'il est possible d'avoir l'élément courant avec $(this).

Récupérer le n^{ième} élément d'un objet jQuery

Comme je vous l'ai déjà dit, un objet jQuery peut contenir plusieurs éléments, notamment lorsque le sélecteur est large (sélection par une classe ou un nom de balise). Souvenez-vous, il existe une pseudo-classe introduite par jQuery nommée `:eq(n)`, et bien elle est reprise avec le même nom sous forme de méthode, et se présente comme ceci :

Code : JavaScript

```
$('p').eq(5).hide();
```

Le paragraphe masqué est alors le 6^{ième} de la page.

Il est à noter que .eq(n) prend un entier n qui vaut 0 pour le premier élément, ceci est du au JavaScript qui a pour convention de commencer à 0 contrairement au CSS (et à `:nth-child(n)` qui commence à 1).

Jusqu'ici rien de nouveau. En revanche, il est possible, contrairement au `:eq(n)`, de compter en partant de la fin avec des entiers négatifs. Ainsi, pour sélectionner l'avant dernier paragraphe, nous aurions ceci :

Code : JavaScript

```
var $avantDernierP = $('p').eq(-2);
```

Ce qui n'est pas possible avec `:eq(n)` car pour cette pseudo-classe, *n* doit être positif ou nul.

Il existe des méthodes dédiées à la sélection du premier et du dernier élément de l'objet jQuery, je vous présente `.first()` et `.last()` qui s'utilisent de la même façon, mais du coup sans paramètre :

Code : JavaScript

```
var $premier = $('p').first();
var $dernier = $('p').last();
```

Ce code aura pour effet de récupérer d'une part le premier paragraphe trouvé dans la page, et d'autre part le dernier paragraphe trouvé sur la page. Il se peut que ces deux variables contiennent le même objet jQuery s'il n'y a qu'un seul paragraphe.

Les balises, une grande famille : parents et enfants

Pour se déplacer dans l'arbre généalogique de notre page (plus couramment appelé « arbre DOM ») il existe quelques méthodes bien pratiques. Comme vous le savez, dans une famille il y a pour un membre des parents et des enfants. Il y a également les frères et sœurs, c'est pareil dans le DOM et jQuery nous propose, des méthodes dont les noms sont très inspirés pour naviguer dans l'arbre DOM.

Pour un objet jQuery `$elem` il est possible de récupérer son parent direct ou tous ses parents avec les méthodes :

Code : JavaScript

```
$elem.parent();
$elem.parents();
```

Chacune de ces méthodes peut prendre un sélecteur en paramètre pour filtrer le résultat. Par exemple, vous êtes dans un formulaire avec le code suivant :

Code : HTML

```
<body>
<form action="#" method="post">
    <fieldset>
        <legend>Connexion</legend>
        <div>
            <label for="input-pseudo">Pseudo : </label>
            <input type="text" name="pseudo" id="input-pseudo">

            <label for="pass-input">Mot de passe : </label>
            <input type="password" name="pass" id="pass-input">
        </div>
    </fieldset>
</form>

<!-- Import de jQuery -->
<script>
    var $inputPseudo = $('#input-pseudo');
    var $form = $inputPseudo.parents('form');
    var $parent = $inputPseudo.parent();
</script>
<body>
```

Ici nous avons donc la méthode `.parents()` qui va aller chercher tous les parents et filtrer pour ne prendre que les **<form>**, sachant qu'un input ne peut être que dans un **<form>** à la fois, nous aurons forcément le seul et unique **<form>** parent.
Par ailleurs $parent va contenir la ** ** qui entoure tous les **<label>** et **<input>** dans notre cas.

Pour les enfants c'est à peu près la même chose, mais nous n'avons que `.children()` pour récupérer tous les enfants directs. De la même manière que pour les parents il est possible de mettre un sélecteur en paramètre pour filtrer le résultat.

En ce qui concerne les frères et sœurs, nous sommes pas mal fournis :

- `.prev()` récupère l'élément juste avant l'élément courant
- `.prevAll()` récupère tous les éléments avant l'élément courant
- `.next()` l'élément juste après
- `.nextAll()` tous les éléments après

Ces méthodes s'applique uniquement aux éléments du même niveau, c'est à dire à ceux qui ont le même parent que l'élément courant.

Contraintes et comparaisons

Doit contenir un certain élément

Quelques fois nous voudrions uniquement sélectionner un type d'élément qui en contient un autre. Certains diront que pour cela c'est simple : on spécifie un sélecteur qui va sélectionner l'élément contenu et remonter au parent pour avoir le conteneur souhaité, le tout dans un `.each()`.

Si vous ne m'avez pas suivi ça n'est pas bien grave, ce qu'il faut retenir c'est qu'il existe la méthode `.has()` qui fait ça pour nous ! Elle s'utilise de la façon suivante :

Code : JavaScript

```
var $ulNonVide = $('ul').has('li');
```

Simple n'est-ce pas ? Ceci va récupérer tous les **<ul>** qui ont au moins un **<li>** à l'intérieur.

Comparer deux éléments

Pour tester si un objet jQuery est d'un certain type ou correspond à un certain critère il existe la méthode `.is()` qui s'utilise comme ceci :

Code : JavaScript

```
var isLi = $('ul li:first').is('li');
```

`isLi` contiendra un booléen selon qui dans le cas ci-dessus vaudra vrai s'il y a présence d'un **<ul>** avec un **<li>** dedans.

À noter que `.is()` peut tester plusieurs choses en même temps sur le même élément, il suffit pour ça de séparer les tests par une virgule comme ceci :

Code : JavaScript

```
var test = $('ul li:first').is('li, .red, :odd');
```

Résumé

Pour résumé tout cela, voici un exemple plus concret, nous allons coupler tout ce que nous avons vu jusqu'ici pour former un exemple passablement vicieux :

Code : JavaScript

```
$('.to-hide').each(function(){
    $('button a',
$(this).find('span').has('b')).parent().nextAll().hide(500,
"linear", function(){
        if($(this).is('p')){
            $(this).show();
        } else {
            $(this).prevAll('p:hidden').show();
        }
    });
});
```

C'est complexe, mais tentons d'expliquer ce code. Premièrement, on va chercher tous les éléments ayant la classe `.to-hide`. Ensuite, on va prendre tous les **** qui contiennent eux-même des **<b>** et on va chercher dans ces **** tous les **<a>** qui sont dans des **<button>**. Une fois que nous sommes ici, on remonte au parent et on masque tous les éléments frères qui suivent celui-ci. Une fois que l'animation est terminée, on fait un test sur chaque élément masqué pour savoir si c'est un paragraphe (ce qui est possible puisqu'on masque tous les éléments suivants un **<button>** si vous avez suivi) : s'il c'est le cas on le masque sinon on affiche tous ses frères précédents qui sont masqués et qui sont des paragraphes.

Je dois vous avouer que je préfère lire le code que le pavé que je viens d'écrire ! Si vous avez compris le code sans l'explication : BRAVO !

Gestion du contenu

Tout d'abord, qu'est-ce que le contenu ? Ça n'est pas uniquement le texte, c'est le code HTML en général. Nous allons voir une foultitude de méthodes qui permette de manipuler le contenu : ajout, modification, suppression...

Récupération et remplacement du contenu

Pour récupérer le contenu d'un élément il suffit de le sélectionner, et d'utiliser l'une des deux méthodes suivantes selon les besoins :

- `.text()` qui va récupérer uniquement le texte en faisant abstraction des balises HTML ;
- `.html()` qui au contraire va récupérer le code source de l'élément concerné.

Ces méthodes sont à la fois des *getters* et des *setters*. Si elles sont utilisées sans paramètre elles vont retourner le contenu, sinon elles vont remplacer le contenu par ce qui va être passé en paramètre. Ainsi, nous aurions ceci :

Code : JavaScript

```
var $htmlDuP = $('p').text("Salut à tous !").html();
// $htmlDuP vaudra "<p>Salut à tous !</p>" s'il existe un p dans la page, bien sûr

// On peut également utiliser les getters et setters en même temps pour ajouter du contenu par exemple
$elem = $('p');
$elem.html("<span>Salut à tous !</span>" + $elem.html())
```

Mais dans ce second cas, nous allons voir que des méthodes plus simples existent : les méthodes d'ajout.

Ajout de contenu

Ajout à l'intérieur de l'élément

À la différence de la modification, l'ajout est le fait de conserver le contenu actuel et d'y ajouter du texte ou du HTML à la suite. Pour cela jQuery nous propose deux distinctions de cas : l'ajout avant le contenu déjà présent, et l'ajout après celui-ci. Cependant, la bibliothèque permet « deux sens de lecture » que voici :

- `$elem.preprend(content)` et `content.preprendTo($elem)` ajoutent le contenu au début de `$elem`;
- `$elem.append(content)` et `content.appendTo($elem)` ajoute le contenu à la fin de `$elem` ;

Dans les deux cas content est le contenu à ajouter, il peut s'agir de HTML ou texte brut, tout comme il peut s'agir d'un objet jQuery. Attention toutefois s'il s'agit d'un élément de la page, celui-ci sera supprimé après l'opération si vous l'utilisez comme ceci :

Code : HTML

```
<p>Mon ami ! </p>
<span>Hey ! </span>

<!-- Import jQuery -->
<script>
    $('p').append('span');
</script>
```

Alors, on aura le code HTML suivant après exécution du code :

Code : HTML

```
<p>Mon ami ! <span>Hey ! </span></p>
```

Ajout avant/après

On peut vouloir ajouter du contenu avant ou après un élément existant, notamment lorsque l'on veut créer un nouveau nœud dans l'arbre DOM de la page. Pour cela deux méthodes simples qui fonctionnent de la même manière que les précédantes :

- `.before()` et `.insertBefore()` sont équivalentes à `.prepend()` et `.prependTo()` mais AVANT l'élément courant ;
- `.after()` et `.insertAfter()` sont équivalentes à `.append()` et `.appendTo()` mais APRÈS l'élément courant.

Comme toujours, petite illustration :

Code : HTML

```
<p>Salut !</p>

<!-- Import jQuery -->
<script>
    $('p').after("<span>Comment vas-tu ?</span>");
</script>
```

Donnera ceci après exécution du code :

Code : HTML

```
<p>Salut !</p>
<span>Comment vas-tu ?</span>
```

Supprimons !

Après avoir ajouté du contenu, quoi de plus normal que de vouloir en retirer ? Il est possible de supprimer uniquement le contenu texte ou bien de supprimer le nœud HTML directement. Voici les spécifications de ces méthodes :

- `.empty()` qui retire le texte de l'élément courant ;
- `.remove()` va quant à lui supprimer littéralement l'élément du code HTML.

Créer de nouveaux nœuds dans le DOM

Il est nécessaire d'avoir des notions relativement avancées en JavaScript pour comprendre le paragraphe qui va suivre.

Pour finir, voici la bonne manière de faire pour que votre script reste conforme au WC3. Lorsque vous voulez créer un nouveau

paragraphe avec du contenu il est préférable de faire comme ceci :

Code : JavaScript

```
var contenu = "Salut ! Je suis du texte à mettre dans un
paragraphe";
$("<p/>").append(contenu);
```

En effet, `$("<p/>")` ou `$("<p></p>")` va réellement créer un nouveau nœud dans le DOM tandis que `$("<p>"+contenu+"</p>");` va faire appel à `innerHtml` qui est plus rapide dans le cas de grosses portions de code, mais moins efficaces dans les cas simples.

Voilà en ce qui concerne la manipulation du contenu, c'est grâce à ces quelques méthodes qu'il nous sera possible de faire tout ce que l'on voudra.

Manipuler les attributs

Nos pages sont constituées de balises qui comportent de nombreux attributs, ceux-ci sont primordiaux dans le fonctionnement d'une page web. Il est possible en JavaScript de les manipuler, et comme toujours jQuery nous propose des méthodes assez simples que je vous découvrir ensemble.

Les attributs en général

Récupérer et soumettre des valeurs

Tout comme `.text()` il existe `.attr()` qui sert à la fois de *getter* et de *setter*. Voici les détails de cette méthode pas des plus simples que l'on ait vu :

- `.attr("attribut")` est le *getter* qui prend en paramètre le nom de l'attribut, il retournera alors sa valeur ;
- `.attr("attribut", "valeur")` est la version la plus simple du *setter* qui va mettre à l'attribut la valeur souhaitée ;
- `.attr({ "attribut1" : "valeur1", "attribut2" : "valeur2" })` permet de mettre une valeur pour chaque attribut spécifier, le tout en une fois.

Dans tous les cas, les attributs et valeurs peuvent être le résultat d'une fonction (souvent anonyme) qui renvoie une chaine. Voici sans plus tarder un petit exemple de la création d'un paragraphe en lui ajoutant un id, des classes et une *data-admin-url* :

Code : JavaScript

```
$('<p/>').attr({
    'id' : 'main-content',
    'class' : 'overlay full-width bg-black',
    'data-admin-url' : '/admin/main-content/edit'
}).appendTo('body');
```

Supprimer un attribut totalement

Pour retirer totalement un attribut (pas le vider, réellement le retirer) il faut utiliser la méthode `.removeAttr("attribut")`. jQuery est tellement simple qu'une seule phrase suffit à expliquer cette méthode.

Les classes

Les classes sont des attributs très manipulés en JavaScript pour alléger l'écriture jQuery nous offre des raccourcis. Comme

toujours, nous avons une méthode par actions, celles-ci sont : ajouter, supprimer, tester, ... classique ! Quoi de plus simple que d'utiliser ces méthodes ? Voici un résumé de la situation :

- `.addClass("la-classe autre-classe")` ou encore `.addClass("class1 class2 classX")` ajoutent à l'élément courant la ou les classe(s) indiquées, pas de doublon si déjà présente(s) ;
- `.removeClass("une-classe autre-classe")` fonctionne de la même façon, mais va supprimer la ou les classe(s) pas d'erreur si ces classes ne sont pas présentes ;
- `.hasClass("une-classe")` retourne un booléen valant vrai si l'élément a la classe demandée.

Pour faire ce test sur de multiples classes il est préférable d'utiliser la méthode `.is()` vu dans la première partie de ce chapitre, pour rappel : `.is("class1, .class2, .classX")`.

Difficile de développer plus sur d'aussi simples méthodes, enchaînons tout de suite avec un autre type d'attribut tout récent : les data-types.

Les data-* du HTML5

Rappels : que sont les data- ?*

Rafraîchissons-nous la mémoire, ou découvrez pour ceux qui ne connaissent pas les `data-*`. Il arrivait fréquemment auparavant de devoir utiliser l'attribut rel ou des éléments en `display: none;` pour stocker des informations relatives à la page à utiliser en JavaScript. Par exemple, dans un formulaire, il arrive qu'il soit nécessaire de multiplier un champ X fois, autant de fois que l'utilisateur le voudra (ajouter des membres à une conversation privée, tags, catégories...). Dans ce cas précis en général on utilisait la technique consistant à cacher l'élément original et à le dupliquer autant de fois que nécessaire. Avec le HTML5, nous avons la possibilité de rajouter des attributs au format `data-*` où * vaut ce que l'on souhaite. Dans notre situation nous pourrions avoir dans la div qui contient tous les membres de la conversation privée, un data-membre qui contiendrait le modèle d'un nouveau champ pour ajouter un membre.

Code : HTML

```
<div data-membre="&lt;input type=&quot;text&quot;
name=&quot;membres[]&quot; placeholder=&quot;Pseudo&quot;&gt;">
    <input type="text" name="membres[]" placeholder="Pseudo">
</div>
```

Il n'y aura plus qu'à récupérer le HTML dans l'attribut data-membre et à l'afficher en dessous avec `.append()` par exemple. Certains malins, dirons qu'il n'y a qu'à dupliquer la ligne existante. Oui, mais ici c'est un cas simple, imaginez avec un label ayant un `id="membre{{id}}"`, il faudra générer l'id en JavaScript, on voit alors l'intérêt des `data-*`.

De multiples utilisations sont possibles, certaines fois on va tout simplement y stocker des informations pour les réutiliser plus tard, d'autres fois elles seront déjà présentes dans le HTML à sa génération, il faut juste retenir que c'est un espace de stockage. Il est bien sûr possible d'avoir plusieurs `data-*` sur un même élément.

data- et jQuery*

Ces attributs sont un peu spéciaux, jQuery a prévu deux méthodes pour faciliter leur récupération, bien que faisable avec `.attr()` qui servira par ailleurs s'il y a besoin de modifier la valeur d'une `data-*`.

- `.data()` récupère un objet JavaScript contenant tous les `data-*` et accessible comme dans l'exemple qui suit ;
- `.data("nom")` récupère la valeur qui est dans l'attribut *data-nom*.

Gardons la mémoire fraiche et enchaînons avec un exemple, celui de tout à l'heure, mais complété :

Code : JavaScript

```
$('<p/>').attr({
    'id' : 'main-content',
    'class' : 'overlay full-width bg-black',
    'data-edit-url' : '/admin/main-content/edit',
    'data-delete-url' : '/admin/main-content/delete'
}).appendTo('body');

// Récupérer tous les data-*
var datas = $('p').data();
var editURL = datas.editUrl; // Attention ici, les tirets sont
retirés et la lettre suivante est en majuscule
var editUrl2 = $('p').data("edit-url"); // Revient à la même chose
qu'au dessus sans passer par une variable
```

Pour supprimer un `data-*` il faut passer par `.removeAttr('data-*')` pour que ça ait effet sur le code HTML directement. Il ne faut pas confondre avec `.removeData()` qui ne supprimera que la valeur en mémoire gérée par le JavaScript.
Nous sommes plein d'outils plus pratiques les uns que les autres, mais vous n'êtes pas au bout de vos découvertes, il y a encore tant de choses à voir !

Le fonctionnement des méthodes n'a plus aucun secret pour vous !
N'hésitez pas à vous référer à la documentation officielle de jQuery lorsque vous voulez des détails sur une fonction ou tout autre point concernant jQuery. Si vous le souhaitez, vous pouvez partager votre plugin sur le site de jQuery dans la partie qui est consacrée à ceux-ci.