Table des matières

Introduction

Avant de commencer à parler de jQuery, on va commencer par le début : javascript. Quand vous parcourez un site internet, les pages sont envoyées à votre navigateur (Firefox, Chrome ou encore Safari...) par un serveur, sous forme de texte : le code HTML de la page. Le navigateur est appelé le client, et le serveur... le serveur !

Cette différence client/serveur est fondamentale dans le développement web. Certains langages comme PHP sont exécutés côté serveur, et permettent de générer le code HTML qui sera envoyé au client. Mais il existe d'autres langages qui s'exécutent directement chez le client (sur votre navigateur), comme JavaScript.

Le DOM

Vous avez peut-être déjà lu ces 3 lettres quelque part, sans vraiment savoir de quoi il s'agit. Je viens de vous expliquer que votre navigateur reçoit le code HTML envoyé par le serveur. En l'état, ce code est inutile, pour le navigateur c'est juste un gros fichier texte avec plein de balises dedans, comme ça :

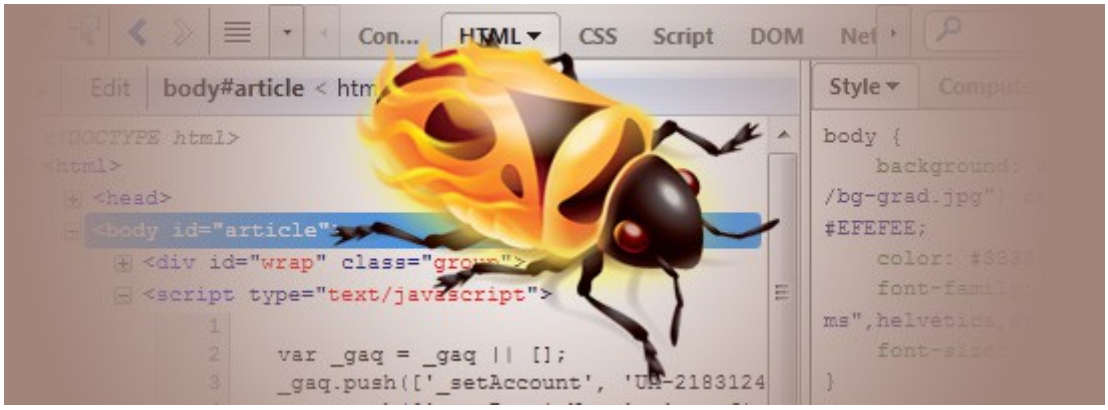
```
<div class="header">
        <a href="/"><img src="/style/images/logo_fc_3.jpg" alt="finalclap"
/></a>
        <div class="descr">
                Tutoriaux gratuits pour<br/>
                <a href="/tutoriaux/photoshop/">Photoshop</a>
                <a href="/tutoriaux/after-effects/">After Effects</a>
                <a href="/tutoriaux/3dsmax/">3D Studio Max</a>
        </div>

        <div class="login">
                <a href="http://forum.finalclap.com/ucp.php?
mode=register">Inscription</a> -
                <a href="#" id="toggle_login">Connexion</a>
        </div>

        <div class="header_liens">
                <a class="lien_j" href="/download/">Ressources</a>
                <a class="lien_b" href="#" id="toggle_tutoriaux">Tutoriaux</a>
                <a class="lien_v" href="/">Blog</a>
                <a class="lien_r" href="http://forum.finalclap.com">Forum</a>
        </div>
</div>
```

Extrait de code HTML Donc votre navigateur va analyser le code, et construire un ensemble d'éléments en les structurant. C'est par ce procédé de rendu qu'un amoncellement de lignes de codes austères se transforme en page web affichée sur votre écran d'ordinateur. Et bien c'est ça le DOM, une structure d'éléments un peu abstraite qui est derrière la page affichée. Le problème du DOM, c'est qu'on ne voit pas concrètement de quoi il s'agit, puisque ça n'est ni le code HTML, ni le rendu graphique de la page. Heureusement, il existe des outils permettant de le voir, mais cela dépend de votre navigateur. On va la faire simple, on va utiliser Firefox pour ce cours. L'extension Firebug permet d'afficher le DOM dans **Firefox**. Allez vite installer Firebug sur votre Firefox avant de continuer...

Firebug




```
console.log(/*<variable, chaine de caractère>*/);
```

Interface de Firebug

Le fonctionnement est relativement simple : quand vous lancez Firebug, ça affiche un panneau dans la partie inférieure de la fenêtre. Ce panneau est composé de plusieurs onglets, avec l'onglet HTML sélectionné par défaut. Cet onglet HTML est en réalité une représentation sous forme de code HTML du DOM. En passant la souris sur les éléments, vous voyez que ceux-ci apparaissent en surbrillance sur la page web. Vous pouvez sélectionner n'importe quel élément en cliquant sur le tag dans l'onglet DOM, ou alors directement en cliquant dessus sur la page web à l'aide de l'inspecteur. Vous pouvez même accéder et modifier certaines propriétés de l'élément sélectionné grâce au panneau Assistant. Je vous laisse regarder un peu partout pour vous familiariser avec votre nouveau jouet. Firebug est sans doute l'outil le plus utile quand on fait du JavaScript (mais il est également d'une efficacité redoutable pour l'intégration HTML/CSS, soit dit en passant). Notamment la console, qui permet d'afficher les éventuelles erreurs javascript, d'exécuter du code JS à la volée, mais qui permet aussi d'afficher des informations de débuggage (équivalent du print_r de PHP...) avec la fonction console.log :

```
console.log(/*<variable, chaine de caractère>*/);
```

Et JavaScript dans tout ça ?

Enfin, nous y voilà, quel est le rapport entre JavaScript et tout ce qu'on vient de dire ??? JavaScript permet ni plus ni moins de manipuler le DOM, d'en modifier la structure, de changer les éléments qui le composent... à la volée, sans recharger la page. On peut par exemple changer la couleur ou la taille de la police de toute la page ou d'un paragraphe en particulier.

Les événements

Dernier point de ce chapitre, la notion d'événement est fondamentale en JavaScript. Un événement, c'est quelque chose qui se produit plus ou moins aléatoirement, comme par exemple le clic sur un élément du DOM (un lien, une image...). JavaScript permet de "capter" ces événements, et d'effectuer des actions de votre choix lorsqu'ils se produisent.

Exemple 1

Affichage d'une boîte de dialogue alert lors du clic sur un élément

Exemple: **Cod'Alais**

```
<a href="http://www.finalclap.com/" onclick="alert('t\'as cliqué sur un lien Cod'Alais !'); return false;">Cod'Alais</a>
```

Ici, le javascript est contenu dans l'attribut onclick de mon lien. Il contient deux instructions : alert, qui affiche une boîte de dialogue, et return false, qui empêche le navigateur de suivre le lien (qui le désactive en quelque sorte).

Reprenons sur Jquery

jQuery est une bibliothèque JS développée principalement par **John Resig**.

Par bibliothèque on entend donc un ensemble cohérent de fonctions permettant de s'affranchir des tâches rébarbatives et répétitives de façon uniforme sur les navigateurs les plus courants. Elle est sous licence GPL et MIT, et donc complètement réutilisable sur des travaux professionnels. Son poids compressé est de 14 Ko, ce qui est tout à fait raisonnable sur la plupart de nos projets. Les navigateurs supportés sont :

- Firefox 1.5+
- Internet Explorer 6
- Safari 2.0.2 +
- Opera 9 +

La bibliothèque est utilisée par exemple sur des sites comme :

- Dell
- Google Code
- Digg
- NBC
- Amazon
- Mozilla
- WordPress
- Drupal
- SPIP
- The Zend Framework
- etc

Par ailleurs cette bibliothèque est compatible (elle n'entre pas en conflit) avec d'autres.

Pré-requis

A présent ces notions devraient vous être familières :

- JavaScript (évènements, etc)
- DOM (nœuds, arbre, etc)
- CSS (sélecteurs, etc)
- AJAX

Installer

Bases

Appel

Typiquement, l'appel à la bibliothèque se fera de la manière suivante :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
    <head>
        <meta http-equiv="Content-Type"
content="application/xhtml+xml;charset=iso-8859-15" />
        <title>Exemple</title>
        <script type="text/javascript" src="./js/lib/jquery.js"></script>
        <script type="text/javascript" src="./js/exemple.js"></script>
    </head>
    <body>

    </body>
</html>
```

On charge dans un premier temps la bibliothèque, puis notre fichier

`exemple.js`

qui contiendra nos scripts.

Compression

jQuery est disponible sous trois formats :

- Sans compression (93 Ko)
- Compressée à l'aide de **packer** (28 Ko)
- Compressée à l'aide de **JSMin** puis de **gzip** (15 Ko)

Il conviendra d'appeler le format le plus léger dans un environnement de production, mais il reste possible de parcourir la bibliothèque dans sa version non compressée.

La fonction `jquery()`

jQuery repose sur une seule fonction : `jQuery()` ou `$()`. Comme toutes les fonctions JavaScript elle accepte des paramètres, et retourne un objet, que nous appellerons par la suite “objet jQuery”.

Concernant les paramètres acceptés, nous pouvons passer :

Une Fonction

Dans ce cas, jQuery va exécuter cette fonction lorsque la page sera chargée par le navigateur. Plus précisément, lorsque le DOM sera chargé.

C’est pourquoi on en-capsule souvent l’ensemble du code écrit en jQuery dans :

```
$(function(){
        // ...
});
```

On s’assure ainsi que le code sera exécute une fois la page chargée, et on est sûr de pouvoir manipuler le DOM sans erreur. Contrairement à l’évènement `window.onload`, le chargement complet des images de la page n’est pas nécessaire.

Une chaine de caractères (query)

Les sélecteurs CSS

Une des grandes forces de jQuery est d’intégrer la syntaxe des sélecteurs CSS. Il est alors facile de cibler facilement les nœuds DOM qui nous intéressent.

Par exemple :

```
var mon_objet_jQuery = $("#mon_image");
var mon_objet_jQuery = $("#menu a");
```

Ou encore :

```
var mon_objet_jQuery = $("#id > .classe, #id td:last-child");
```

Notez au passage que contrairement au support CSS des navigateurs, la librairie comprends tout à fait les pseudos-classes et les pseudos-éléments.

Voir la documentation des sélecteurs CSS

Les sélecteurs jQuery

Il existe des sélecteurs spécifiques à jQuery qui ne trouvent pas d'équivalences en CSS. Ils sont toutefois très pratiques. Par exemple :

```
var mon_objet_jQuery = $('tr:odd td');
// les `td` dans les `tr` impairs var mon_objet_jQuery = $("p:eq(4)");
// le quatrième paragraphe var mon_objet_jQuery = $("p:lt(8)");
// les sept premiers paragraphes
```

etc...

Voir la documentation des sélecteurs jQuery

Méthodes

Maintenant que nous avons un objet jQuery, il va falloir l'utiliser. Et pour cela toute une palette de méthodes sont disponibles dans la bibliothèque : manipulation du DOM, des CSS, des évènements et autres effets visuels.

Si on désire par exemple masquer progressivement les paragraphes d'une page, on utilise :

```
$("p").fadeOut();
```

Soit sans passer de paramètre comme cela, soit en précisant une durée en mili-secondes ou encore à l'aide des chaines `slow`, `fast` et `normal`.

Deux autres exemples :

```
$("li").html("lorem <strong>ipsum</strong> dolor");
$("#menu a").click(function(){
     window.open($(this).attr("href"));
     return false;
});
```

Dans le dernier, vous remarquerez qu'on combine des fonctions JavaScript "classiques" au code jQuery, mais aussi qu'on transforme l'objet courant `this` en objet jQuery avec `$(this)`.

L'ensemble des méthodes et leurs documentations sont disponibles sur **Visual jQuery**.

Callback

Certaines fonctions (comme la fonction `fadeOut()`) acceptent une autre fonction en paramètre. Cette dernière sera exécutée à la fin de l'exécution de la première. C'est ce qu'on appelle un callback.

Cela permet par exemple de créer des animations :

```
$(".test").fadeOut("slow",function(){
$(this).fadeIn("slow");
});
```

Chainabilité

On l'a vu les méthodes jQuery retournent un objet jQuery. Cela permet de **"chainer"** les fonctions.

Ainsi au lieu d'écrire :

```
$(".toto").append(" un texte");
$(".toto").css("border","1px solid red");
$(".toto").addClass("titi");
$(".toto").removeClass("toto");
```

On peut simplifier par l'exécution du code par :

```
$(".toto").append(" un texte").css("border","1px solid
red").addClass("titi").removeClass("toto");
```

Parcourir les éléments du DOM

Le parcours du DOM permet de se déplacer aisément sur la page, afin de récupérer certains éléments HTML. Un peu comme pour les sélecteurs, il faudra bien comprendre le principe de la descendance, car c'est un concept très puissant et très utilisé dans les langages orienté objet comme JavaScript, rappelez-vous-en !

Naviguer dans le DOM

Rappel sur le Document Object Model

Le Document Object Model, beaucoup plus couramment appelé DOM, permet littéralement de naviguer entre les éléments HTML. Nous rappelons qu'il s'agit d'une interface de programmation, utilisée exclusivement pour les documents XML et HTML, qui se base sur la structure. C'est grâce à ce concept qu'il est possible d'agir sur une page web avec JavaScript, et plus précisément ses fonctions ; sans cela, nous ne pourrions pas *désigner* les éléments !

Vous vous en doutez, nous faisons constamment appel au DOM, même si en jQuery, on s'en rend beaucoup moins compte. En effet, même si nous n'avons pas encore appris à utiliser les fonctions de parcours, sachez que la sélection d'éléments, par exemple, use de certaines fonctions JavaScript qui ont besoin du DOM (c.f. `getElementById()`).

Positionnement des éléments

La position des éléments dans la structure d'une page HTML est très importante. C'est pourquoi on vous répète souvent : ayez un code propre, une structure claire et bien rangée, et vous aurez beaucoup plus de facilité à manipuler vos éléments en JavaScript ! Une sémantique bien faite, c'est un gain de temps et d'énergie. Prenons pour exemple un code HTML très mauvais, que l'on peut malheureusement rencontrer sur le web :

```
<div id="bloc">
<a href="#" id="lien">Un lien</a><a href="#" id="lien">Un autre lien</a>

<div>

<span class="paragraphe">Contenu</span>
<div></div>
</div>
```

Nous espérons sincèrement que vous n'écrirez JAMAIS ce genre d'aberration...

Dans un cas comme celui-là, la sélection simple ne sera pas altérée ; en revanche, si vous devez parcourir le DOM pour ne pas devoir refaire une requête d'élément inutile, alors bonne chance ! Une bonne sémantique est primordiale, nous ne vous le répéterons jamais assez.

La descendance

Si on vous rabâche depuis quelques temps que la descendance est un concept très utilisé en programmation orientée objet, ce n'est pas pour rien : souvent apparentée à l'*héritage*, cette technique se base sur le principe de parents et d'enfants. Dans un contexte comme le nôtre, cela peut paraître un peu tiré par les cheveux comme méthode de travail, et pourtant, sans cela, il serait beaucoup plus compliqué de réaliser de véritables applications en jQuery !

Quelques chapitres auparavant, nous avons abordé rapidement la chose avec les sélecteurs, et plus précisément ceux-ci :

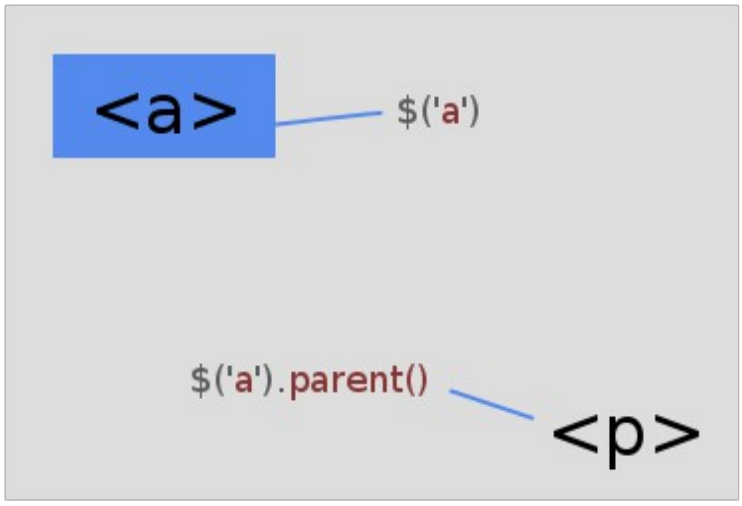
- `$('.parent .enfant');`
- `$('.parent > .enfant').`

En effet, nous vous avions expliqué qu'une structure HTML permettait d'imbriquer des blocs les uns dans les autres : ceux étant à l'intérieur sont appelés enfants, ceux qui entourent un élément sont des parents. Le parcours du DOM va énormément se baser sur ce concept, à retenir et maîtriser absolument. Nous allons étudier quelques fonctions qui permettent d'accéder aux éléments, qu'ils soient parents, enfants, ou même frères !

Parents, enfants et ancêtres

Commençons par une des fonctions de parcours les plus utilisées : `parent()`. Plus facile à retenir, tu meurs ! Comme vous vous en doutez, cette méthode va nous permettre d'accéder au bloc parent de l'élément actuellement ciblé. Lorsque nous agirons sur notre objet jQuery, ce ne sera donc plus ce dernier qui sera influencé, mais bien le bloc qui l'entoure.

```
$('a').css('color', 'blue'); // rend le lien ciblé seulement de couleur bleue\n\n$('a').parent().css('color', 'blue');\n// ici, c'est le parent de l'enfant (un paragraphe, si l'on respecte la sémantique) qui verra son texte devenir bleu
$('a').css('color', 'blue'); // rend le lien ciblé seulement de couleur bleue
$('a').parent().css('color', 'blue'); // ici, c'est le parent de l'enfant (un paragraphe, si l'on respecte la sémantique) qui verra son texte devenir bleu
```



Il est possible de préciser la requête, en sélectionnant la classe, l'identifiant et le type de l'élément à récupérer en tant qu'argument :

```
$('a').parent('.texte'); // retourne seulement l'ensemble des blocs parents ayant la classe .texte('a').parent('.texte'); // retourne seulement l'ensemble des blocs parents ayant la classe .texte
```

Inversement, la fonction `children()` permet de cibler l'élément enfant descendant directement de l'élément sélectionné. Cette méthode équivaut donc au sélecteur >, mais permet, comme la fonction `parent()`, de préciser la recherche avec un argument.

```
$('div').children(); // cible l'élément enfant direct du bloc div
$('div').children('p'); // cible seulement l'ensemble des paragraphes enfants du bloc div
$('div').children(); // cible l'élément enfant direct du bloc div
$('div').children('p'); // cible seulement l'ensemble des paragraphes enfants du bloc div
```

Cette méthode est néanmoins quelque peu restrictive : elle ne va cibler que les enfants directs d'un élément. Que se passerait-il si l'on voulait récupérer tous les fils et petits-fils d'un bloc ? On ne pourrait tout simplement pas, ou tout du moins avec `children()`. Car oui, bien heureusement, jQuery possède la fonction `find()`, qui va se charger de trouver tous les enfants d'un élément, quelle que soit leur position par rapport à ce dernier ! Il suffit alors de donner le type d'élément à trouver, et `find()` se charge du reste :

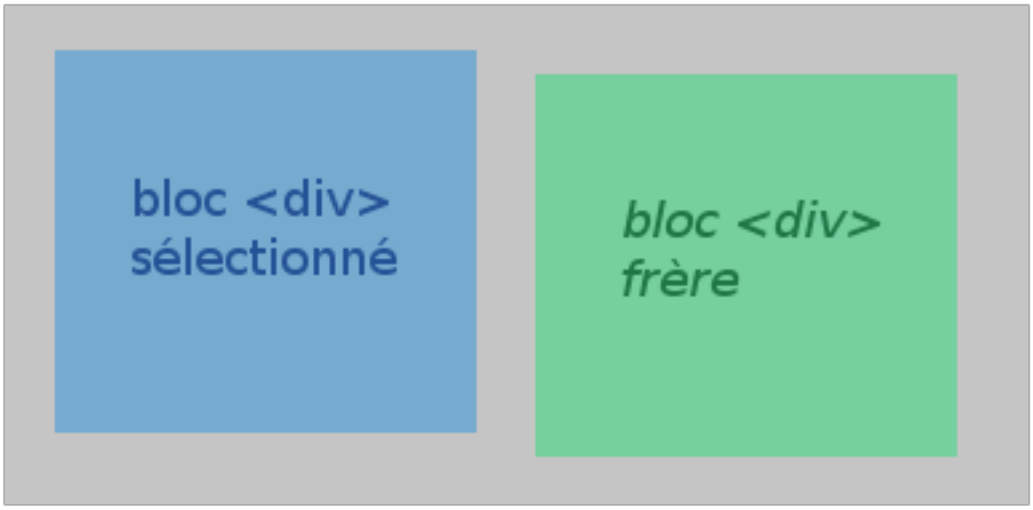
```
$('body').find('p'); // cible l'ensemble des paragraphes contenus dans le corps du document, quelle que soit leur position !
$('body').find('p'); // cible l'ensemble des paragraphes contenus dans le corps du document, quelle que soit leur position !
```

La dernière méthode usant de ce principe de descendance est `parents()`. N'oubliez pas le "s", car elle est légèrement différente de la fonction `parent()`, et ne s'utilise pas tout à fait de la même façon ! Concrètement, cette dernière retournait seulement la première occurrence trouvée : dès qu'elle tombait sur un parent, elle s'arrêtait sans aller plus loin. Il se peut que vous vouliez agir sur tous les parents d'un élément en même temps. Par soucis de précision, on ne parlera ici plus de parents, mais d'ancêtres.

```
$('a').parents(); // cible tous les éléments ancêtres du lien : paragraphe, bloc(s), balise <body>...
$('a').parents(); // cible tous les éléments ancêtres du lien : paragraphe, bloc(s), balise <body>...
```

La fraternité d'éléments

Après le père et le grand-père, nous demandons le frère ! Eh oui, comme pour les méthodes développées précédemment, nous sommes dans l'obligation d'étudier les fonctions ciblant les élément frères de la sélection. Le frère d'un élément, c'est tout simplement la balise présente directement à côté de celui-ci.



Il y a quatre façons de faire appel à ce concept :

- `prev()`, qui sélectionne l'élément frère précédant directement l'objet ciblé ;
- `next()`, qui sélectionne l'élément frère suivant directement l'objet ciblé ;
- `prevAll()`, qui sélectionne tous les éléments frères précédant l'objet ciblé ;
- `nextAll()`, qui sélectionne tous les éléments frères suivant l'objet ciblé.

Filtrer et boucler les éléments

Moins utilisés mais tout de même d'une praticité flagrante, les filtres et les conditions permettent de cibler des éléments de manière plus précise et concise que les méthodes apparentées à la descendance. En effet, jusqu'ici, vous ne savez que sélectionner les éléments enfants, parents, ancêtres ou frères d'un objet. En revanche, le ciblage par l'index, par exemple, vous est totalement inconnu. Afin de combler ces lacunes, nous vous invitons à lire la suite.

Filtrer les éléments

Filtre par sélecteur

La méthode principale permettant de filtrer des éléments se nomme `filter()`. Elle va permettre de supprimer tous les objets ne correspondant pas à la recherche de la sélection, ce qui est très pratique dans le cas où vous souhaitez ne récupérer que certains éléments et pas d'autres, dans une recherche plus globale. La requête est à spécifier en tant qu'argument, et peut être vraiment très précise dans le cas où il est possible de donner plusieurs sélecteurs d'un seul coup en les séparant de virgules !

```
$('p').filter('.texte'); // supprime de la sélection tous les paragraphes n'ayant pas la classe .texte
$('p').filter('.texte, #description'); // supprime de la sélection tous les paragraphes n'ayant pas la classe .texte ou l'identifiant #description
$('p').filter('.texte'); // supprime de la sélection tous les paragraphes n'ayant pas la classe .texte
$('p').filter('.texte, #description'); // supprime de la sélection tous les paragraphes n'ayant pas la classe .texte ou l'identifiant #description
```

La méthode `not()` permet de faire la même chose, mais prend en argument les objets à ne pas prendre en compte.

Filtre par index

Vous avez déjà rencontré le sélecteur `:eq()` plus tôt dans le cours. Il suffisait de l'utiliser avec un index pour sélectionner un élément grâce à celui-ci. Il existe également la méthode `eq()` qui elle s'exécute sur un objet. Elle va donc parcourir le tableau d'occurrences trouvées et sélectionner seulement celui qui possède l'index indiqué en argument.

```
$('p').eq(2); // cible le troisième paragraphe trouvé (l'index commence à 0)
$('p').eq(2); // cible le troisième paragraphe trouvé (l'index commence à 0)
```

Petite astuce, vous pouvez spécifier un nombre négatif : jQuery commencera alors à compter à partir du dernier index. Si vous possédez quatre paragraphes et que vous donnez la valeur -1 à la méthode, alors votre objet sera le quatrième paragraphe.

Moins utilisée, la méthode slice() permet de prendre une portion d'un tableau d'objets, grâce à leur index. Elle prend un argument obligatoire, et un second facultatif :

1. *start*, qui est la position (ou l'index) du premier élément à filtrer ;
2. *end*, qui est la position (ou l'index) du dernier élément à filtrer, non pris en compte par la sélection.

Il s'agit donc littéralement de *couper* un tableau contenant l'ensemble de nos éléments ciblés. Seuls ceux étant compris entre les index spécifiés seront gardés, les autres étant alors supprimés de la sélection :

```
$('div').slice(1, 3); // garde seulement les blocs div ayant l'index 1 ou 2
$('div').slice(1,3); // garde seulement les blocs div ayant l'index 1 ou 2
```

Vérifier le type d'un élément

La fonction `is()` est peu utilisée, mais il est bon de la connaître dans le cas où vous rencontrez un code qui la contient pour une quelconque raison. Elle renvoie simplement un booléen, `true` si elle déduit que l'argument donné correspond au type de l'élément analysé, `false` si ce n'est pas le cas :

```
var vrai = $('div').is('div');
var faux = $('div').is('p');

console.log(vrai); // affiche true
console.log(faux); // affiche false
var vrai = $('div').is('div');
var faux = $('div').is('p');
console.log(vrai); // affiche true
console.log(faux); // affiche false
```

Boucler les éléments

Non content de sa puissance, jQuery révolutionne le concept des boucles en permettant une nouvelle forme : le bouclage d'éléments. Assez particulier, ce système est toutefois bien utile : il va traiter chaque occurrence trouvée et exécuter une fonction définie dessus. Il s'agit de la méthode`each()`. Elle va agir sur chaque objet trouvé, en effectuant littéralement une boucle. Cela va vous permettre par exemple d'exécuter des fonctions qui, normalement, s'arrête au premier élément rencontré, les autres étant laissé à l'abandon.

```
$('p').each( function(){
    alert( $(this).text() ); // $(this) représente l'objet courant
} );
$('p').each( function(){
    alert( $(this).text()); // $(this) représente l'objet courant
} );
```

Il est possible de récupérer l'index de l'élément courant grâce à un argument, à placer dans la fonction de retour.

Si la fonction retourne `false`, alors la boucle s'arrête brutalement. Si au contraire elle retourne `true`, alors la boucle passe directement à l'élément suivant. Il existe d'autres utilisations de cette méthodes, notamment avec sa sœur `$.each()`, mais nous ne les aborderons pas ici car elles ne sont pas vraiment courantes et utilisées.

Maintenant que vous savez parcourir le DOM, que vous maîtrisez les notions de descendance, de fraternité et de filtrage d'éléments, vous pouvez enfin passer à la manipulation du code HTML lui-même!