APPRENDRE JQUERY

Laurent Sutterlity

Table des matières

Apprendre jQuery

Pré-requis

Ce cours est destiné à des personnes ayant déja de bonnes connaissances HTML/CSS. Nous ne traiterons pas de toutes les méthodes de jQuery, mais des plus utilisées.

Autres références en ligne

- **w3schools** : http://www.w3schools.com/jquery/default.asp
- **Openclassrooms** : http://fr.openclassrooms.com/informatique/cours/jquery-ecrivez-moins-pour-faire-plus

Livres sur le sujet

- jQuery : Simplifiez et enrichissez vos développements JavaScript
  http://www.pearson.fr/livre/?GCOI=27440100609080
- Simplifiez vos développements JavaScript avec jQuery
  http://boutique.fr.openclassrooms.com/boutique-614-1061-simplifiez-vos-developpements-javascript-avec-jquery.html
- jQuery-Ajax avec PHP
  http://www.eyrolles.com/Informatique/Livre/jquery-ajax-avec-php-9782212137200

Autres références en ligne

- Ressource en anglais référence en JS : Eloquent Javascript
- CodeSchool : https://www.codeschool.com/courses/try-jquery

Présentation de jQuery

jQuery est une bibliothèque JavaScript libre et multi-plateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML. La première version est lancée en janvier 2006 par John Resig (un petit génie du JavaScript).

La bibliothèque contient notamment les fonctionnalités suivantes :

- Parcours et modifications du DOM ;
- Gestion des événements ;
- Effets visuels et animations ;
- Manipulation des CSS ;
- Ajax ;
- Plugins ;
- Utilitaires

Pour vous faire un rapide comparatif d'écriture, voici des exemples comparant jQuery à JavaScript.

Évènements

```
// jQuery
$(document).ready(function() {
    // vos scripts
})

// Javascript
document.addEventListener('DOMContentLoaded', function() {
    // vos scripts
})
```

```
// jQuery
$('a').click(function() {
  // vos scripts
})

// Javascript
[].forEach.call(document.querySelectorAll('a'), function(el) {
    el.addEventListener('click', function() {
        // vos scripts
    })
})
```

Sélecteurs

```
// jQuery
var divs = $('div')

// Javascript
var divs = document.querySelectorAll('div')
```

Extrait de code issu de l'article : http://putaindecode.fr/posts/js/de-jquery-a-vanillajs/

Rappels JavaScript

jQuery est un framework JavaScript, il est donc essentiel de comprendre un minimum ce langage pour progresser plus facilement.

Rassurez-vous, rien de bien compliqué à l'horizon. Nous verrons uniquement les bases nécessaires à tout bon intégrateur web désireux de progresser.

Voici un bref aperçu de ce que nous aborderons dans cette partie :

1. Les variables
2. Les types
3. Les opérateurs arithmétiques
4. La concaténation
5. Les conditions
6. Les boucles
7. Les fonctions

Vous trouverez sur le site de "Mozilla developer" une bonne documentation en français sur JavaScript :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide

Les variables

Une variable est un objet repéré par son nom, pouvant contenir tout type de données, qui pourront être modifiées lors de l'exécution du programme.

En JavaSript les noms de variables peuvent être aussi long que l'ont le souhaite, mais doivent répondre à certains critères :

1. un nom de variable doit commencer par une lettre (majuscule ou minuscule) ou un `_` underscore.
2. un nom de variable peut comporter des lettres, des chiffres et les caractères `_` et `&amp;`.
3. **les espaces ne sont pas autorisés**
4. Les noms de variables ne peuvent pas être les noms suivants, qui sont des noms réservés :

   abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, export, extends, false, final, finally, float, for, function, goto, if, implements, import, in, infinity, instanceof, int, interface, label, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, void, volatile, while, with.

Création de notre première variable

Nous déclarons notre variable.

```
var maVariable;
```

Le mot clé **var** indique que vous déclarez une variable, `;` indique que l'instruction est terminée. Une fois déclarée, **var** n'est plus nécessaire, vous pouvez stoker ce que vous souhaitez.

```
var maVariable;
maVariable = "J'aime JavaScript";
```

Nous pouvons modifier la valeur de la variable.

```
var maVariable;
maVariable = "J'aime JavaScript";
maVariable = "Je n'ai pas peur de JavaScript";
// maVariable vaut "Je n'ai pas peur de JavaScript
```

Nous pouvons déclarer notre variable et lui attribuer une valeur sur la même ligne.

```
var maVariable_2 = 7;
```

Nous pouvons déclarer plusieurs variables sur une ligne, mais attribuer une valeur seulement à *maVariable*_2.

Puisque l'ont délare le même type d'élément, le mot clé `var` peut éviter d'être répété en séparant les déclarations par des virgules.

```
var maVariable_1,
    maVariable_2 = 7,
    maVariable_3;
```

Vous pouvez afficher votre 1ère variable dans la console de votre navigateur.

```
var maVariable = "J'aime JavaScript";
console.log(maVariable);
```

Les types

Les chaînes de caractères ou "string"

Ce type représente n'importe quel texte. On peut l'assigner de deux façons différentes.

Avec des guillemets

```
maVariable = "J'aime JavaScript";
```

Avec des apostrophes

```
maVariable = 'J\'aime JavaScript';
```

Attention, si vous voulez des apostrophes pour déclarer votre variable et que vous utilisez des apostrophes dans le texte vous devez échapper les derniers avec le caractère `\` (antislash). Dans le cas contraire, JavaScript pensera que le code s'arrête à la première apostrophe et produira une erreur.

Le type numérique ou "number"

Ce type de variable représente tout nombre que ce soit un entier, un négatif, un nombre scientifique, etc.

```
maVariable = 3;
```

Un chiffre décimal se déclare avec un point comme séparateur et non une virgule.

```
maVariable = 3.5;
```

Attention, si vous écrivez le nombre entre guillemets ou avec apostrophes, il sera reconnu comme une chaîne de caractères.

```
maVariable = '3'; // Chaîne de caractères
```

Les booléens

Désigne un paramètre qui peut avoir comme valeur **true** ou **false**.

Lorsque :

- aucune valeur n'est passée ;
- la valeur est égale à *0* ;
- une chaîne de caractères est vide, *null*, *undefined* ou *NaN*

la valeur de l'objet est initialisée à `False`.

Dans tous les autres cas, l'objet Boolean possédera la valeur `True`.

Ces deux états s'écrivent de la façon suivante :

```
var isTrue = true;
var isFalse = false;
```

Test de type

Nous pouvons vérifier le type de variable avec l'instruction **typeof**.

La fonction **console.log()** affiche le résultat dans la console du navigateur.

```
var myVariable = 2;
console.log(typeof myVariable); // Affiche : « number »

var myVariable = 'Mon texte';
console.log(typeof myVariable); // Affiche : « string »

var myVariable = false;
console.log(typeof myVariable); // Affiche : « boolean »
```

Nous pouvons tester l'existence de variable avec la même instruction.

```
console.log(typeof otherVariable); // Affiche : « undefined »
```

La convertion de type

Convertion type "string" en" number"

Dans certains cas de figure, nous pouvons avoir besoin de convertir une chaîne de caractères en nombre. La méthode `parseInt()` convertie une chaîne de caractères en nombre.

```
var myVariable = '2',
    n = parseInt(myVariable);
console.log(typeof n); // Affiche : « number »
```

Convertion type "number" en "string

Pour convertir une chaîne de charactères en nombre, nous pouvons utiliser la méthode `.toString()`.

```
var myVariable = 2,
    n = myVariable.toString();
console.log(typeof n); // Affiche : « string »
```

Les opérateurs arithmétiques

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable de type **number**.

Ils sont à la base de tout calcul et sont au nombre de 5 :

Opérateurs	Signe	Effet	Exemple	Résultat (avec x valant 7)
Addition	**+**	Ajoute deux valeurs	x + 3	10
Soustraction	**-**	Soustrait deux valeurs	x - 3	4
Multiplication	*****	Multiplie les valeurs	x * 3	21
Division	**/**	Divise deux valeurs	x / 3	2.3333333
Modulo	**%**	Retourne le reste de la division entière de l'opérant de gauche par celle de droite	x % 2	1

Calculs simples

Faire des calculs avec JavaScript s'avère très simple.

```
var result = 7 + 2;
console.log(result); // Affiche : 9
```

Nous pouvons faire des calculs avec des variables contenant des valeurs de type numérique.

```
var number_1 = 7, number_2 = 2, result;
result = number_1 + number_2;
console.log(result); // Affiche : 9
```

Nous pouvons faire des calculs plus complexes comprenant plusieurs opérateurs.

```
var result = (((3 + 4) - 1) / 2) * 3;
console.log(result); // Affiche : 9
```

Simplification des calculs

Il est fréquent d'écrire ce type de calcul.

```
var maVariable = 7;
    maVariable = maVariable + 2
console.log(maVariable); // Affiche : 9
```

Il existe une façon plus concise permettant d'éviter la redondance de variables.

```
var maVariable = 7;
```

```
    maVariable += 2;
console.log(maVariable); // Affiche : 9
```

On procèdera de la même manière pour chaque opérateur.

```
maVariable += 2;
maVariable -= 2;
maVariable *= 2;
maVariable /= 2;
maVariable %= 2;
```

Pour ajouter **+1** nous pouvons utiliser l'opérateur d'incrémentation **++**

```
var maVariable = 7;
    maVariable++;
console.log(maVariable); // Affiche : 8
```

Pour supprimer **-1** nous pouvons utiliser l'opérateur de décrémentation **--**

```
var maVariable = 7;
    maVariable--;
console.log(maVariable); // Affiche : 6
```

La concaténation

L’opérateur arithmétique **+** permet de faire des concaténations entre plusieurs chaînes de caractères.

La concaténation consiste à ajouter une chaîne de caractères à la fin d'une autre, comme dans cet exemple.

```
var maVariable = "J'aime JavaScript";
var autreVariable = "je n'en ai pas peur.";
var concatenation = maVariable + ", " + autreVariable;
console.log(concatenation); // affiche : J'aime JavaScript, je n'en ai pas peur.
```

Les conditions

On appelle structure conditionnelle les instructions qui permettent de tester si une condition est vraie (true) ou non (false).

En JavaScript les conditions sont constituées de :

- valeurs à tester ;
- un opérateur logique ;
- un opérateur de comparaison (optionnel).

Les opérateurs de comparaison

Opérateurs	Désignation	Effet	Exemple	Résultat
==	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x == 3	Retourne True si X est égal à 3, sinon False
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x != 3	Retourne 1 si X est différent de 3, sinon 0
===	opérateur d'identité	Vérifie l'identité de valeur et de type de deux valeurs	a === b	Retourne True si a est égal à b et est de même type, sinon False
!==	opérateur de non identité	Vérifie la non identité de valeur et de type de deux valeurs, c'est-à-dire si les deux valeurs n'ont pas la même valeur ou bien sont de types différents.	a !== b	Retourne True si a est différent de b ou bien est de type différent, sinon False
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x > 3	Retourne True si X est supérieur à 3, sinon False
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x >= 3	Retourne True si X est supérieur ou égal à 3, sinon False
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x < 3	Retourne True si X est inférieur à 3, sinon False
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x <= 3	Retourne True si X est inférieur ou égale à 3, sinon False

Les opérateurs logiques (booléens)

Opérateurs	Désignation	Effet	Syntaxe

||	OU logique	Vérifie qu'une des conditions est réalisée	((expression1) || (expression2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((expression1) && (expression2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

Instruction if

Elle permet d'exécuter une série d'instructions lorsqu'une condition est réalisée.

La syntaxe de cette expression est la suivante.

```
if ( si condition réalisée ) {
    liste d'instructions
}
```

Il est possible de définir plusieurs conditions à remplir avec les opérateurs `&&` et `||`.

Par exemple, l'instruction ci-dessous teste si les deux conditions sont réalisées.

```
if ( (condition1) && (condition2) ) { … }
```

L'instruction suivante exécutera les instructions si l'une ou l'autre des deux conditions est réalisée.

```
if ( (condition1) || (condition2) ) { … }
```

Exemple

```
// Déclaration de variable et assignation de valeur
var monArgent = 1;
var prixCafe  = 1.2;

// Condition
// Si
if ( monArgent >= prixCafe) {
    console.log("J'ai assez d'argent pour boire un café.");
}
```

Instruction if … else

L'instruction **if** dans sa forme basique permet de tester qu'une condition. Or, la plupart du temps, on aimerait pouvoir choisir les instructions à exécuter lorsque la condition n'est pas remplie.

L'expression **if … else** permet dès lors d'exécuter une autre série d'instructions en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante :

```
if ( si condition réalisée ) {
    // liste d'instructions
}
```

```
else {
    // sinon autre série d'instructions
}
```

Remarque :
Les structures conditionnelles pouvant être imbriquées, il peut être utile d'indenter le code pour plus de lisibilité. En d'autres termes, il s'agit de décaler à l'aide d'une tabulation chaque bloc d'instructions pour pouvoir rapidement visualiser l'imbrication des structures.

Exemple

```
// Déclaration de variable et assignation de valeur
var monArgent = 1;
var prixCafe  = 1.2;

// Condition
// Si
if ( monArgent >= prixCafe) {
    console.log("J'ai assez d'argent pour boire un café.");
}
// Sinon
else {
    console.log("Je n'ai pas assez d'argent pour boire un café.");
}
```

Instruction else if

Cette instruction permet d'exécuter une série d'instructions lorsque plusieurs conditions sont réalisées.

La syntaxe de cette expression est la suivante :

```
// Une première condition est testée
if ( si condition réalisée ) {
    // liste d'instructions
}
// Une deuxième condition est présente et sera testée si la première échoue
else if ( sinon si condition réalisée ) {
    // liste d'instructions
}
// Et si aucune condition ne se vérifie, la structure else fait alors son travail.
else {
    // sinon autre série d'instructions
}
```

Exemple

```
// Déclaration de variable et assignation de valeur
var monArgent = 1;
var prixCafe  = 1.2;

// Condition
// Si
if ( monArgent >= (prixCafe*250) ) {
    console.log("J'ai assez pour pouvoir inviter toute l'école.");
}
// Sinon si
else if ( monArgent >= (prixCafe*30) ) {
    console.log(" J'ai assez pour pouvoir inviter toute le groupe.");
}
else if ( monArgent >= (prixCafe*2) ) {
    console.log(" J'ai assez pour pouvoir inviter mon professeur.");
}
```

```
// Sinon
else {
    console.log("Je n'ai pas assez d'argent pour boire un café. ");
}
```

Instruction switch...case

L'instruction switch permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Ce branchement conditionnel simplifie beaucoup le test de plusieurs valeurs d'une variable. Cette opération aurait ainsi été plus compliquée (mais possible) avec des if imbriqués.

La syntaxe de cette expression est la suivante :

```
switch (Variable) {
    case Valeur1:
        Liste d'instructions;
        break;
    case Valeur2:
        Liste d'instructions;
        break;
     default:
        Liste d'instructions;
        break;
}
```

Les parenthèses qui suivent le mot clé **switch** indiquent une expression dont la valeur est testée successivement par chacun des **case**. Lorsque l'expression testée est égale à une des valeurs suivant un **case**, la liste d'instructions qui suit celui-ci est exécutée.

Le mot clé **break** indique la sortie de la structure conditionnelle. Le mot clé **default** précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs.

Attention, il est essentiel de terminer chaque bloc d'instructions par l'instruction **break**.
N'oubliez pas d'insérer des instructions **break** entre chaque test. Ce genre d'oubli est difficile à détecter, aucune erreur n'étant signalée.

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée.

La façon la plus commune de faire une boucle est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

L'instruction for

L'instruction **for** permet d'exécuter plusieurs fois la même série d'instructions : c'est une boucle !

Dans cette syntaxe, il suffit de préciser le nom de la variable qui sert de compteur et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui *incrémente* ou *décrémente* le compteur.

La syntaxe de cette expression est la suivante :

```
for (compteur; condition; modification du compteur) {
    // liste d'instructions
}
```

Exemple

```
for (var i=1; i<6; i++) {
    console.log(i);
}
```

Cette boucle affiche 5 fois la valeur de i, c'est-à-dire 1,2,3,4,5.

Elle commence à `i=1`, vérifie que i est bien inférieur à 6, etc... jusqu'à atteindre la valeur `i=6`, pour laquelle la condition ne sera plus réalisée. La boucle s'interrompra et le programme continuera son cours.

Attention

1. Il faudra toujours vérifier que la boucle a bien une condition de sortie.
2. Une instruction `Alert(i)` ; dans votre boucle est un bon moyen pour vérifier la valeur du compteur pas à pas.

Bien entendu, si nous pouvons incrémenter dans une boucle, nous pouvons aussi décrémenter.

```
for (var i=10; i>0; i--) {
    console.log(i);
}
```

L'instruction while

L'instruction `while` représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.

La syntaxe de cette expression est la suivante :

```
while (condition réalisée) {
    // liste d'instructions
}
```

Cette instruction exécute la liste d'instructions **tant que** la condition est réalisée.

Attention aux boucles infinies (boucle dont la condition est toujours vraie) elle provoque un plantage du navigateur.

Exemple

```
var x = 0;
while(x < 10) {
    console.log("I'm looping!");
    x++;
}
```

Les fonctions

En programmation, une fonction est un sous-programme qui permet d'exécuter un ensemble d'instructions dans le corps du programme principal.

Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions. Ce procédé permet une simplicité du code et donc une taille de programme minimale.

Par ailleurs, une fonction peut faire appel à elle-même. On parle alors de fonction récursive.

Création de notre première fonction

Avant d'être utilisée, une fonction doit être déclarée afin que le navigateur l'interprète.

Une déclaration de fonction est constituée du mot-clé `function`, suivi :

1. du nom de la fonction (suit les mêmes règles que les noms de variables) ;
2. des parenthèses **obligatoires** pouvant contenir des paramètres facultatifs séparés par des virgules ;
3. des instructions JavaScript définissant la fonction, entourés d'accolades `{ }`.
   Les instructions d'une fonction peuvent comprendre des appels à d'autres fonctions définies dans l'application courante.

La syntaxe de cette expression est la suivante.

```
function Nom_De_La_Fonction(argument1, argument2) {
    // liste d'instructions
}
```

Appel de notre fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivi d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée.

```
Nom_De_La_Fonction();
```

Attention, veillez toujours à ce qu'une fonction soit déclarée avant d'être appelée.

Exemple

```
// Déclaration de la fonction
function hello () {
    console.log('Hello world');
}

// Appel de la fonction
hello();
```

La portée des variables

Afin d'aborder ce concept assez simple mais pouvant induire en erreur, nous allons étudier deux exemples.

Exemple 1

```
var hello = "Hello world !";
function sayHello() {
    console.log(hello);
}
sayHello();
```

Ici, pas de problème. On déclare une variable dans laquelle on stocke du texte puis on crée une fonction qui se charge de l'afficher à l'écran et enfin on l'exécute.

Les variables déclarées en dehors des fonctions sont nommées variables globales.

Exemple 2

Modifions maintenant l'ordre des déclarations. Dans cette hypothèse, la variable sera déclarée dans la fonction.

```
function sayHello() {
    var hello = 'Hello world !';
}
sayHello();
console.log(hello);
```

Nous pouvons nous rendre compte que cet exemple ne fonctionne pas. Le script est arrêté car il produit une erreur (visible dans l'inspecteur web).

Voilà tout le concept de portée de variables :
Toute variable déclarée dans une fonction n'est utilisable que dans cette même fonction ! Ces variables spécifiques à une seule fonction ont un nom : les variables locales.

Variable globale VS variable locale

Maintenant que nous savons faire la différence entre les variables globales et les variables locales, essayons de comprendre leur comportement.

Que se passe t'il si nous déclarons une variable globale et une locale avec le même nom, mais avec une autre valeur ?

```
var hello = "Hello world !";
function sayHello() {
    var hello = "Bonjour la terre !";
    console.log("Dans la fonction : " + hello);
}
sayHello();
console.log("En dehors de la fonction : " + hello);
```

Nous obtenons le résultat suivant :

```
"Dans la fonction : Bonjour la terre !"
"En dehors de la fonction : Hello world !"
```

Bonne pratique

De manière générale, il est préférable d'utiliser les variables locales pour les fonctions, afin de ne pas interférer avec d'autres fonctions qui peuvent utiliser le même nom de variables.

Les paramètres d'une fonction

Il est possible de passer des paramètres à une fonction, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer des opérations sur ces paramètres.

Lorsque vous passez plusieurs paramètres à une fonction il faut les séparer par des virgules aussi bien dans la déclaration que dans l'appel. Il faudra également veiller à passer le bon nombre de paramètres lors de l'appel. Dans le cas contraire, une erreur se produira dans votre script.

Exemple avec un argument

```
// Déclaration de la fonction
function carre(nombre) {
    console.log( nombre * nombre );
}

// Appel de la fonction
carre(3);
```

Exemple avec deux arguments

```
Exemple avec deux arguments
// Déclaration de la fonction
function remise(nombre, pourcentage) {
    console.log( (pourcentage / 100) * nombre );
}

// Appel de la fonction
remise(150, 20);
```

La valeur return

Comme son nom l'indique, la valeur **return** est la valeur retournée par la fonction. Les fonctions ne peuvent renvoyer qu'une valeur de retour.

```
function Nom_De_La_Fonction(argument1, argument2) {
    return argument1 * argument2
}
```

Attention
L'instruction **return** met fin à la fonction, puis retourne la valeur. Tout ce qui est renseigné en dessous est ignoré.

Exemple

```
// Déclaration de la fonction
function carre(nombre) {
    return nombre * nombre;
}

// Appel de la fonction
console.log(carre(20));
```

Installation

jQuery est une bibliothèque JavaScript qui n'est pas installée de base dans votre navigateur. Il faut l'ajouter par le biais d'un fichier ayant pour extension `.js`.

Pour l'utiliser dans une page HTML, il faut lier le fichier jQuery par le biais de la balise `script`.

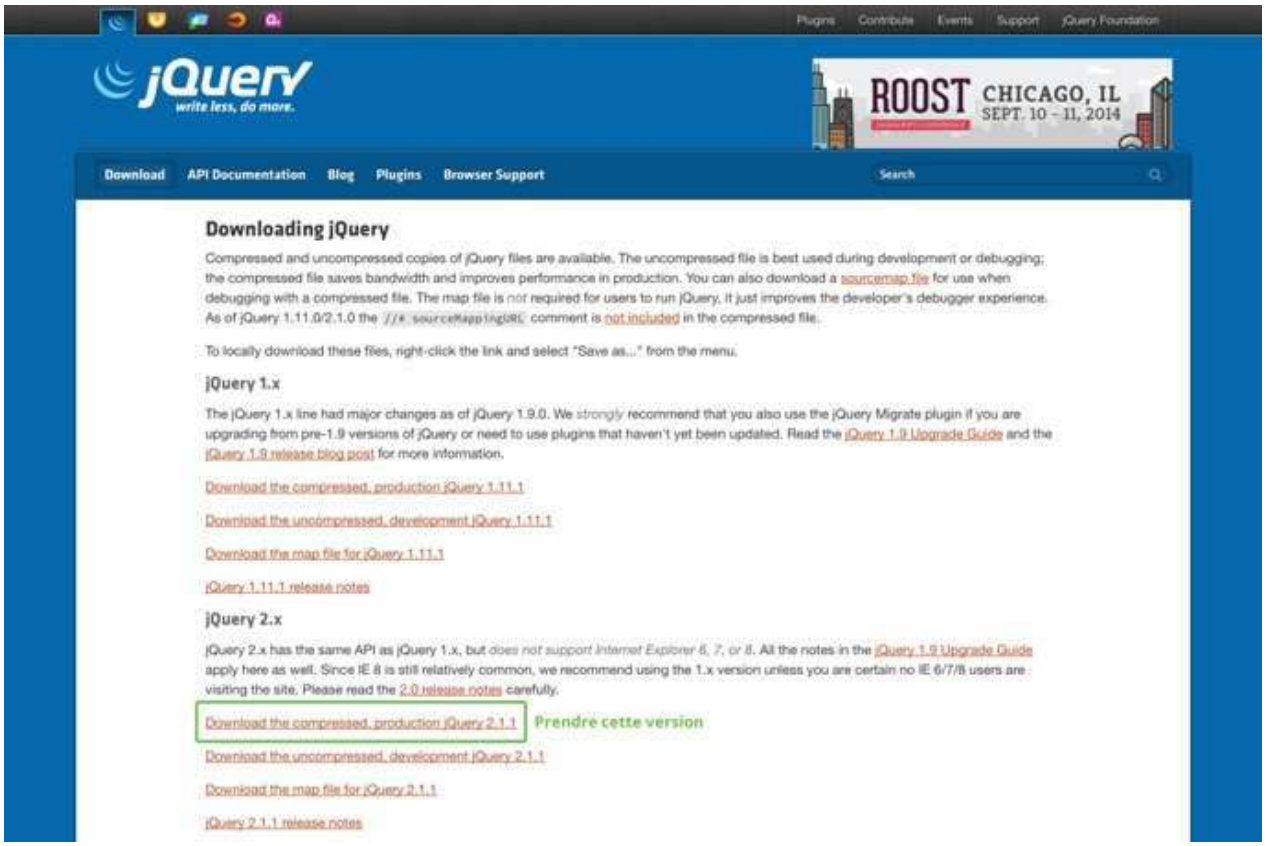
```
<head>
    <script src="jquery-2.1.1.min.js"></script>
</head>
```

Installation locale

L'installation en local est idéale si vous n'avez pas toujours accès à internet ou si votre connexion n'est pas très rapide.

Nous verrons par la suite que ce n'est pas la solution optimale pour une utilisation en production.

Vous avez besoin de télécharger le fichier sur le site officiel de jQuery : Télécharger jQuery



Vous pouvez vous rendre compte qu'il existe plusieurs versions de jQuery. Mi 2013 jQuery est passé en version 2.XX laissant de côté la compatibilité avec les versions antérieures d'Internet Explorer 8, allégeant aussi son poids.

Ne prenez pas peur, vous n'aurez pas à modifier ces fichiers, vous devez juste y faire référence dans votre HTML pour l'installation.

Vous trouverez deux versions du même fichier :

1. un nommé "jquery-2.1.1.js"
2. un autre nommé "jquery-2.1.1.min.js"

La version **'2'** avec le suffixe **min** est une version minifiée du fichier de base. C'est-à-dire une version où les commentaires, les espaces et les tabulations ont été supprimées pour réduire le poids du fichier (chargement plus rapide).

Voici l'installation avec le script dans le header :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Installation de jQuery</title>
    <link rel="stylesheet" href="css/styles.css">
    <script src="js/jquery-2.1.1.min.js"></script>
</head>
<body>
    <!-- Contenu de la page -->
</body>
</html>
```

Vous pouvez faire l'import de jQuery avant la balise de fermeture `</body>` :

Cette technique améliore la vitesse de chargement du site internet.

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Installation de jQuery</title>
    <link rel="stylesheet" href="css/styles.css">
</head>
<body>
    <!-- Contenu de la page -->
    <script src="js/jquery-2.1.1.min.js"></script>
</body>
</html>
```

Bien évidement, il faudra placer nos fichiers utilisant la librairie jQuery en dessous de son appel.

Installation de jQuery distante

La manière la plus courante et aussi la plus utilisée est de passer par un CDN.

Le fichier est mis à disposition sur de multiples serveurs à travers le monde. Ces serveurs sont souvent plus rapides et économisent aussi votre bande passante. L'autre avantage est que de très nombreux sites utilisent cette méthode, donc le fichier jQuery est surement déjà en cache chez votre utilisateur.

Il existe plusieurs CDN pouvant vous fournir le fichier :

1. CDN jQuery : http://code.jquery.com/jquery-2.1.1.js
2. CDN Google : http://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js

Pour faire l'installation, il suffit donc de remplacer le chemin du fichier en local par l'URL du CDN.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
```

Conclusion

1. Pour le développement de votre site, utilisez jQuery en local pour la rapidité d'exécution.
2. Pour la production, passez par un CDN afin d'utiliser le cache de vos utilisateurs.
3. Placez vos scripts en fin de page, pour optimiser le chargement de la page.

Éviter les conflits

Dans cette partie nous allons voir comment gérer les conflits entre plusieures bibliothèques jQuery.

Les pluparts des bibliothèques jQuery utilisent le signe `$` comme préfix de sélecteurs d'éléments. Bien entendu, cela a pour effet de produire des conflits.

Jquery à prévu une alternative afin de palier à ce problème avec la méthode `noConflict()`. Il suffira de remplacer `$` par `jQuery`.

```
jQuery.noConflict();
```

Une fois ce code déclaré, vous pouvez continuer à utiliser `$` pour l'autre bibliothèque.

```
<script src="js/jquery.js"></script>
<script src="js/mootools.js"></script>
<script>
    jQuery.noConflict();

    // Ici vos codes jQuery avec comme préfixe `jQuery`
    jQuery('.box_inner').addClass('rounded');

    // Pour l'autre bibliothèque `$`
    $('.box').set('opacity',0);
</script>
```

Premiers pas

Nous avons vu à l'étape précédente comment installer jQuery dans notre projet. Maintenant, rentrons dans le vif du sujet, l'utilisation de la librairie.

Nous allons créer un nouveau fichier **app.js** contenant un script modifiant une page web. Par convention nous utiliserons un dossier **js** contenant l'ensemble des scripts du projet.

Voici le contenu de notre page "index.html" :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Permiers pas avec jQuery</title>
</head>
<body>
    <div id="text-html">Texte en HTML</div>
    <div id="text-js"></div>
    <!-- en dessous les scripts JS -->
    <script src="js/jquery-2.1.1.min.js"></script>
    <script src="js/app.js"></script>
</body>
</html>
```

Voici le contenu de notre script "app.js" :

```
$(function() {
    $('#text-js').html('Texte renseigné par le bais de jQuery');
});
```

Voici le résultat :
Nous nous apercevons que le DOM a bien été modifié par jQuery.

See the Pen Premier pas jQuery by Sutterlity (@sutterlity) on CodePen.

Le DOM

Le DOM ou **Document Object Model** est un standard du W3C qui définit l'arborescence du document HTML.

jQuery est utilisé pour manipuler le DOM en écriture et en lecture.

Prenons le cas où le DOM n'est pas encore totalement construit lors du chargement de la page et que le script jQuery est exécuté ; les sripts produiront des erreurs.

Pour attendre que le DOM soit entièrement chargé et ainsi éviter des erreurs.

Version JavaScript

```
document.addEventListener('DOMContentLoaded', function() {
  // Ici le DOM est prêt
})
```

Version jQuery

Avec jQuery, nous utiliserons la méthode `.ready()` prévue à cet effet.

```
jQuery(document).ready(function() {
    // Ici le DOM est prêt
});
```

Cette écriture jQuery peut être simplifiée par l'alias JavaScript `$` :

```
$(document).ready(function() {
    // Ici le DOM est prêt
});
```

Afin d'être encore plus concis le `$(document).ready` peut être omis :

```
$(function() {
    // Ici le DOM est prêt
});
```

Ces 3 instructions sont équivalentes, mais ma préférence va à la troisième solution beaucoup plus facile à retenir.

Sélectionner des éléments

De JavaScript vers jQuery

jQuery nous permet de sélectionner des éléments de façon plus simple qu'en JavaScript.

Exemple : Sélectionner une balise `span`

```
// Version Javascript
document.getElementsByTagName('span');

// Version jQuery
$('span');
```

Exemple : Sélectionner un identifiant `#sub-footer`

```
// Version Javascript
document.getElementById('sub-footer');

// Version jQuery
$('#sub-footer');
```

Exemple : Sélectionner une classe `box`

```
// Version Javascript
document.getElementsByClassName('box');

// Version jQuery
$('.box');
```

Exemple : Sélectionner `#nav a:nth-child(odd)`

```
// Version Javascript
// Ne fonctionne pas sous IE8
document.querySelectorAll('#nav a:nth-child(odd)');

// Version jQuery
// Fonctionne sous IE8
$('#nav a:nth-child(odd)');
```

Article d'Alsacréation sur `querySelector` et `querySelectorAll` : http://www.alsacreations.com/article/lire/1445-dom-queryselector-queryselectorall-selectors-api.html

Sélectionner avec jQuery

Un des grands avantages de jQuery est d'utiliser la syntaxe des sélecteurs CSS et permet d'utiliser les sélecteurs CSS3 avec de vieux navigateurs.

Pour sélectionner les *noeuds du DOM* (éléments) nous utiliserons la syntaxe `$(sélecteur)`.

Bien entendu, la liste ci-dessous n'est pas exhaustive et ne reprend que les sélecteurs les plus utilisés.

Vous pouvez en trouver davantage dans l'API jQuery.

Certains **sélecteurs jQuery** sont redondants avec les **sélecteurs CSS3**.

En effet, les sélecteurs jQuery ont été implémentés bien avant la sortie de CSS3. Je vous conseille donc de préviligier les sélecteurs CSS3 dont l'utilisation est plus pérenne.

Combiner des sélecteurs

Les combinateurs standards

Sélecteurs	Exemple	Éléments selectionnés
.class, .class	`$(&#39;.box, .pod&#39;)`	Les éléments **class="box"** ou **class="pod"**
#id .class	`$(&#39;#content .box&#39;)`	Les éléments **class="box"** à l'intérieur **id="content"**
#id.class	`$(&#39;#content.wrap&#39;)`	Les éléments **class="wrap"** ayant pour identifiant **id="content"**

Le sélecteur de contexte

Le sélecteur de contexte permet d'éviter la redondance de caractères au sein du sélecteur.

Sélecteurs	Exemple	Éléments selectionnés
'.class', '#id'	`$(&#39;.box, h1&#39;, &#39;#content&#39;)`	Les éléments **class="box"** et **h1** contenu dans **id="content"**

Dans l'exemple suivant, nous pouvons être plus concis.

```
$('#product .box, #product .thumbs').fadeIn(1000);
```

Utilisons maintenant la même déclaration, mais avec un sélecteur de contexte.

```
$('.box, .thumbs','#product').fadeIn(1000);
```

Nous avons ainsi optimisé notre sélecteur.

Les sélecteurs de base

API : http://api.jquery.com/category/selectors/basic-css-selectors/

Sélecteurs	Exemple	Éléments selectionnés
*	`$(&#39;*&#39;)`	Tous les éléments
Balise html	`$(&#39;p&#39;)`	Tous les paragraphes
#id	`$(&#39;#header&#39;)`	Les éléments avec **id="header"**
.class	`$(&#39;.box&#39;)`	Les éléments avec **class="box"**
A > B	`$(&#39;.box > p&#39;)`	Les élément **B** enfants directs de **A**
A + B	`$(&#39;li + li&#39;)`	L'élément **B** frère adjacent de **A**

Les sélecteurs d'enfant

API : http://api.jquery.com/category/selectors/child-filter-selectors/

Sélecteurs	Exemple	Éléments selectionnés
:first-child	`$(&#39;p:first-child&#39;)`	1er élément **p**
:last-child	`$(&#39;p:last-child&#39;)`	Dernier élément **p**
:nth-child(x)	`$(&#39;li:nth-child(2)&#39;)`	2ème élément **li**
:nth-child(xn + x)	`$(&#39;li:nth-child(3n + 2)&#39;)`	2ème élément **li** tous les 3 **li**
:only-child	`$(&#39;.box p:only-child&#39;)`	Le fils unique **p** de **class="box"**

Les sélecteurs de contenu

API : http://api.jquery.com/category/selectors/content-filter-selector/

Sélecteurs	Exemple	Éléments selectionnés
:contains(texte)	`$(&#39;p:contains(&quot;Markdown&quot;)&#39;)`	Les éléments contenant la chaîne de caractères **Markdown**. *Attention ce sélecteur est sensible à la casse*
:has(sélecteur)	`$(&#39;.box:has(strong)&#39;)`	Les éléments **class="box"** ayant une balise **strong**
:parent	`$(&#39;.box:parent&#39;)`	Les éléments **class="box"** non vides
:empty	`$(&#39;.box:empty&#39;)`	Les éléments **class="box"** vides

Les sélecteurs d'attribut

API : http://api.jquery.com/category/selectors/attribute-selectors/

Sélecteurs	Exemple	Éléments selectionnés
[attr]	`$(&#39;[href]&#39;)`	Les éléments contenant un attribut **href**
[attr=valeur]	`$(&#39;[href=&quot;index.html&quot;]&#39;)`	Les éléments contenant un attribut **href** ayant pour valeur **index.html**
[attr!=valeur]	`$(&#39;[href!=&quot;index.html&quot;]&#39;)`	Les éléments contenant un attribut **href** ayant pas pour valeur **index.html**
[attr^=valeur]	`$(&#39;[class^=&quot;box-&quot;]&#39;)`	Les éléments ayant un classe préfixée de **box-** comme **.box-inner**
[attr$=valeur]	`$(&#39;[class$=&quot;-active&quot;]&#39;)`	Les éléments ayant un classe sufixée de **-active** comme **.pod-active**
[attr*=valeur]	`$(&#39;[class*=&quot;box&quot;]&#39;)`	Les éléments ayant un classe contenant **box** comme **.pod-boxed**
[attr~=valeur]	`$(&#39;[class~=&quot;box&quot;]&#39;)`	Les éléments ayant un classe dont la valeur est une liste séparée par des espaces dont l'un d'eux est exactement **box** comme **class="first box"**

Les sélecteurs de filtre

API : http://api.jquery.com/category/selectors/basic-filter-selectors/

Sélecteurs	Exemple	Éléments selectionnés
:animated	`$(&#39;.box:animated&#39;)`	Les éléments **class=".box"** animés en jQuery
:eq	`$(&#39;.box:eq(2)&#39;)`	Le 3ème élément **class=".box"**
:even	`$(&#39;tr:even&#39;)`	Les éléments **tr** pairs
:odd	`$(&#39;tr:odd&#39;)`	Les éléments **tr** impairs
:first	`$(&#39;tr:first&#39;)`	Le premier élément **tr** trouvé
:last	`$(&#39;tr:last&#39;)`	Le dernier élément **tr** trouvé
:focus	`$(&#39;.box:focus&#39;)`	Les éléments qui ont le **focus**
:gt	`$(&#39;.box:gt(3)&#39;)`	Les éléments **class="box** à partir du 4ème
:lt	`$(&#39;.box:lt(3)&#39;)`	Les éléments **class="box** jusqu'au 3ème
:header	`$(&#39;.title:header&#39;)`	Toutes les balises de **h1** à **h6** ayant **class="title"**

Les sélecteurs de visibilité

API : http://api.jquery.com/category/selectors/visibility-filter-selectors/

Sélecteurs	Exemple	Éléments selectionnés
:hidden	`$(&#39;.box:hidden&#39;)`	Les éléments **class="box"** cachés
:visible	`$(&#39;.box:visible&#39;)`	Les éléments **class=".box"** ayant une largeur ou une hauteur plus grande que **0**.

Les sélecteurs de formulaires

API : http://api.jquery.com/category/selectors/form-selectors/

Sélecteurs	Exemple	Éléments selectionnés
:input	`$(:input)`	Tous les éléments de type **input**
:text	`$(&#39;:text&#39;)`	Les éléments **input type="text"**
:password	`$(&#39;:password&#39;)`	Les éléments **input type="password"**
:radio	`$(&#39;:radio&#39;)`	Les éléments **input type="radio"**
:checkbox	`$(&#39;:checkbox&#39;)`	Les éléments **input type="checkbox"**
:submit	`$(&#39;:submit&#39;)`	Les éléments **input type="submit"**
:reset	`$(&#39;:reset&#39;)`	Les éléments **input type="reset"**
:button	`$(&#39;:button&#39;)`	Les éléments **button**
:image	`$(&#39;:image&#39;)`	Les éléments **input type="image"**
:file	`$(&#39;:file&#39;)`	Les éléments **input type="file"**
:enable	`$(&#39;input:enabled&#39;)`	Les éléments **input** actifs
:disabled	`$(&#39;input:disabled&#39;)`	Les éléments **input** désactivés
:selected	`$(&#39;select option:selected&#39;)`	Les éléments **option** de **select** sélectionnés

:checked	`$(&#39;input:checked&#39;)`	Les éléments **input** qui sont vérifiés ou sélectionnés

Vous pouvez aussi sélectionner un élément de formulaire avec un sélecteur CSS : `$(&#39;input[type=&quot;text&quot;]&#39;)`

Notions indispensables

Les fonctions de rappel

Les fonctions de rappel ou **callback** sont des fonctions qui s'exécutent lorsque l'autre fonction se termine.

Prenons l'exemple ci-dessous, au clic sur un bouton nous faisons disparaître un texte, puis nous affichons un texte dans une boite d'alerte.

```
$('button').click(function(){
    $('.bg-muted').hide('slow',function(){
      alert('Le paragraphe est maintenant caché');
    });
});
```

See the Pen JFxlm by Sutterlity (@sutterlity) on CodePen.

Chainage de méthodes

Il est possible d'appliquer plusieurs méthodes les unes à la suite des autres.

Plutôt que d'écrire les instructions comme ceci :

```
$('.box').addClass('is-collapsed');
$('.box').slideUp();
```

Optimisez votre code comme cela :

```
$('.box').addClass('is-collapsed').slideUp();
```

Vous pouvez même faire des choses plus complexes.
Par exemple, ajouter une classe `.active` sur un élément au clic, puis faire une recherche dans ce *pattern* pour rechercher un élément qui a la classe `.inner` pour lui supprimer sa classe.

```
$('.btn').click(function(){
    $(this).addClass('active').find('.inner').removeClass();
    return false;
});
```

See the Pen mjqnx by Sutterlity (@sutterlity) on CodePen.

Parcourir le DOM

API : http://api.jquery.com/category/traversing/

Même si les sélecteurs jQuery nous permettent de sélectionner des éléments, nos possibilités sont relativement limitées. jQuery nous propose plusieurs méthodes afin de parcourir le DOM et de faire des tris.

Dans l'exemple ci-dessous, nous ajouterons une classe `.first-level` à l'enfant `ul` de `#menu`.

```
$('#menu').children('ul').addClass('first-level');
```

Bien sûr nous ne verrons pas toutes les méthodes en détail, mais voici un tableau récapitulatif.

Méthode	Description
`.add()`	Ajoute des éléments au sélecteur actuel
`.addBack()`	Ajoute les précédents éléments sélectionnés au sélecteur actuel
`.children()`	Retourne tous les enfants directs de l'élément sélectionné
`.closest()`	Retourne le plus proche élément ancêtre de l'élément sélectionné
`.contents()`	Les noeuds enfants (y compris les noeuds de texte)
`.each()`	Exécute une fonction à chaque élément identifié
`.end()`	Termine le filtrage en cours et retourne aux éléments sélectionnés précédemment
`.eq()`	Retourne un élément avec un numéro d'index spécifique des éléments sélectionnés (commence à 0)
`.filter()`	Filtre les éléments qui correspondent au sélecteur indiqué
`.find()`	Recherche les éléments enfants qui correspondent au sélecteur
`.first()`	Retourne le premier élément des éléments sélectionnés
`.has()`	Retourne les éléments ayant à l'intérieur le sélecteur indiqué.
`.is()`	Vérifie l'ensemble des éléments sélectionnés contre un sélecteur / élément / objet jQuery, et retourne vrai si au moins un de ces éléments correspond arguments donnés
`.last()`	Retourne le dernier élément des éléments sélectionnés
`.map()`	Retourne un tableau de valeurs, de l'élément sélectionné
`.next()`	Retourne le frère suivant de chaque élément sélectionné, avec filtre (facultatif)
`.nextAll()`	Retourne tous les frères suivants de chaque élément sélectionné, avec filtre (facultatif)
`.nextUntil()`	Retourne tous les frères suivants de chaque élément sélectionné entre deux arguments donnés, avec filtre (facultatif)
`.not()`	Supprime les éléments qui ne correspondent pas au sélecteur
`.offsetParent()`	Retourne le parent positionné (par exemple de manière relative ou absolue) du 1er élément sélectionné.
`.parent()`	Retourne le parent direct, avec filtre (facultatif)
`.parents()`	Retourne tous les éléments ancêtres de l'élément sélectionné
`.parentsUntil()`	Retourne tous les éléments parents entre deux arguments donnés

`.prev()`	Retourne le frère précédent de chaque élément sélectionné, avec filtre (facultatif)
`.prevAll()`	Retourne tous les frères précédents de chaque élément sélectionné, avec filtre (facultatif)
`.prevUntil()`	Retourne tous les frères précédents de chaque élément sélectionné entre deux arguments donnés, avec filtre (facultatif)
`.siblings()`	Retourne tous les frères, avec filtre (facultatif)
`.slice()`	Retourne les éléments se trouvant dans la plage d'indices indiquée (commence à 0)

Parcourir les ancêtres

Dans cette partie nous verrons les méthodes `parent()`, `.parents()`, `.parentUntill()` et `.closest()`.

Méthode .parent()

API : http://api.jquery.com/parent/

La méthode `.parent()` retourne l'élément parent direct de l'élément sélectionné. Cette méthode ne traverse qu'un seul niveau dans l'arborescence DOM.

L'exemple suivant retourne l'élément parent direct de chaque `class=&quot;box-inner&quot;` et lui ajoute `class=&quot;box&quot;`.

```
<main>
    <section>
        <div>
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

```
$('.box-inner').parent().addClass('box');
```

Résultat

```
<main>
    <section>
        <div class="box">
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

Méthode .parents()

API : http://api.jquery.com/parents/

La méthode `.parents()` retourne tous les éléments ancêtres de l'élément sélectionné.

L'exemple suivant retourne les éléments parents jusqu'à `main` et leur ajoute `class=&quot;clearfix&quot;`.

```
<main>
    <section>
        <div>
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

```
$('.box-inner').parents('main').addClass('clearfix');
```

Résultat

```
<main class="clearfix">
    <section class="clearfix">
        <div class="clearfix">
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

Méthode .parentsUntil()

API : http://api.jquery.com/parentsUntil/

La méthode `.parentsUntil()` retourne tous les éléments ancêtres entre deux arguments donnés.

L'exemple suivant retourne les éléments parents au niveau en dessous de `main` et leur ajoute `class=&quot;clearfix&quot;`.

```
<main>
    <section>
        <div>
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

```
$('.box-inner').parentsUntil('main').addClass('clearfix');
```

Résultat

```
<main>
    <section class="clearfix">
        <div class="clearfix">
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

Méthode .closest()

API : http://api.jquery.com/closest/

La méthode `.closest()` retourne le plus proche élément ancêtre de l'élément sélectionné.

L'exemple suivant retourne le plus proche élément ancêtre `section` et lui ajoute `class=&quot;clearfix&quot;`.

```
<main>
    <section>
        <div>
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

```
$('.box-inner').closest('section').addClass('clearfix');
```

Résultat

```
<main>
    <section class="clearfix">
        <div>
            <div class="box-inner">…</div>
        </div>
    </section>
</main>
```

Parcourir les descendants

Dans cette partie, nous verrons les méthodes `children()` et `.find()`.

Méthode .children()

API : http://api.jquery.com/children/

La méthode `.children()` retourne tous les enfants directs de l'élément sélectionné. Cette méthode ne traverse qu'un seul niveau dans l'arbre DOM.

L'exemple suivant ajoute `class=&quot;box-inner&quot;` aux enfants directs de `class=&quot;box&quot;`.

```
<main>
    <section>
        <div class="box">
            <div>
                <div>…</div>
            </div>
        </div>
    </section>
</main>
```

```
$('.box').children().addClass('box-inner');
```

Résultat

```
<main>
    <section>
        <div class="box">
            <div class="box-inner">
                <div>…</div>
            </div>
        </div>
    </section>
</main>
```

Méthode .find()

API : http://api.jquery.com/find/

La méthode `.find()` retourne tous les enfants passés en paramètre.

L'exemple suivant ajoute `class=&quot;is-fixed&quot;` à `id=&quot;header&quot;` puis cherche l'enfant `nav` et lui ajoute `class=&quot;parent-is-fixed&quot;`.

```
<div id="header">
    <header>
        <h1><a href="index.html" class="logo">Logotype</a></h1>
    </header>
    <nav>
        <ul>
            <li>…</li>
```

```
        </ul>
    </nav>
</div>
```

```
$('#header').addClass('is-fixed').find('nav').addClass('parent-is-fixed');
```

Résultat

```
<div id="header" class="is-fixed">
    <header>
        <h1><a href="index.html" class="logo">Logotype</a></h1>
    </header>
    <nav class="parent-is-fixed">
        <ul>
            <li>…</li>
        </ul>
    </nav>
</div>
```

Parcourir les frères

Dans cette partie, nous verrons les méthodes `.siblings()`, `.next()`, `.nextAll()`, `.nextUntil()`, `.prev()`, `.prevAll()` et `.prevUntil()`

Méthode .siblings()

API : http://api.jquery.com/siblings/

La méthode `.siblings()` retourne tous les éléments frères de l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` aux frères de `class=&quot;item&quot;`.

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

```
$('.item').siblings().addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="item">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
</ul>
```

Méthode .next()

API : http://api.jquery.com/next/

La méthode `.next()` retourne le frère suivant de l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` au frère suivant de `class=&quot;item&quot;`.

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

```
$('.item').next().addClass('selected');
```

Résultat

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li class="selected">…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

Méthode .nextAll()

API : http://api.jquery.com/nextAll/

La méthode `.nextAll()` retourne tous les frères suivants de l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` aux frères suivants de `class=&quot;item&quot;`.

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

```
$('.item').nextAll().addClass('selected');
```

Résultat

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
</ul>
```

Méthode .nextUntil()

API : http://api.jquery.com/nextUntil/

La méthode `.nextUntil()` retourne tous les frères suivants entre les éléments sélectionnés.

L'exemple suivant ajoute `class=&quot;selected&quot;` aux frères suivants entre `class=&quot;item&quot;` et le `class=&quot;other-class&quot;`.

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li class="other-class">…</li>
</ul>
```

```
$('.item').nextUntil('.other-class').addClass('selected');
```

Résultat

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="other-class">…</li>
</ul>
```

Méthode .prev()

API : http://api.jquery.com/prev/

La méthode `.prev()` retourne le frère prédédent l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` au frère prédédent de `class=&quot;item&quot;`.

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

```
$('.item').prev().addClass('selected');
```

Résultat

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li class="selected">…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

Méthode .prevAll()

API : http://api.jquery.com/prevAll/

La méthode `.prevAll()` retourne tous les frères prédédents de l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` aux frères prédédents de `class=&quot;item&quot;`.

```
<ul class="list">
    <li>…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

```
$('.item').prevAll().addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

Méthode .prevUntil()

API : http://api.jquery.com/prevUntil/

La méthode `.prevUntil()` retourne tous les frères prédédents entre les éléments sélectionnés.

L'exemple suivant ajoute `class=&quot;selected&quot;` aux frères prédédents entre `class=&quot;item&quot;` et `class=&quot;other-class&quot;`.

```
<ul class="list">
    <li>…</li>
    <li class="other-class">…</li>
    <li>…</li>
    <li>…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

```
$('.item').prevUntil('.other-class').addClass('selected');
```

Résultat

```
<ul class="list">
    <li>…</li>
    <li class="other-class">…</li>
    <li class="selected">…</li>
    <li class="selected">…</li>
    <li class="item">…</li>
    <li>…</li>
    <li>…</li>
</ul>
```

Les filtres

Dans cette partie, nous verrons les méthodes `.first()`, `.last()`, `.eq()`, `.filter()` et `.not()`.

Méthode .first()

API : http://api.jquery.com/first/

La méthode `.first()` retourne le premier élément de l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` au premier élément de `class=&quot;item&quot;`.

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

```
$('.item').first().addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="item selected">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

Méthode .last()

API : http://api.jquery.com/last/

La méthode `.last()` retourne le dernier élément de l'élément sélectionné.

L'exemple suivant ajoute `class=&quot;selected&quot;` au dernier élément de `class=&quot;item&quot;`.

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

```
$('.item').last().addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item selected">…</li>
</ul>
```

Méthode .eq()

API : http://api.jquery.com/eq/

La méthode `.eq()` retourne un élément avec un numéro d'index spécifique. Les numéros d'index commencent à 0, de sorte que le premier élément aura le numéro d'index 0 et non 1.

L'exemple suivant ajoute `class=&quot;selected&quot;` au 2ème élément de `class=&quot;item&quot;`.

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

```
$('.item').eq(1).addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="item">…</li>
    <li class="item selected">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

Méthode .filter()

API : http://api.jquery.com/filter/

La méthode `.filter()` permet de spécifier un critère. Les éléments qui ne correspondent pas aux critères sont retirés de la sélection, et ceux qui correspondent seront retournés.

L'exemple suivant ajoute `class=&quot;selected&quot;` aux éléments `class=&quot;item&quot;` ayant une autre `class=&quot;other-class&quot;`.

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item other-class">…</li>
    <li class="item">…</li>
    <li class="item other-class">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

```
$('.item').filter('.other-class').addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="item other-class selected">…</li>
    <li class="item">…</li>
    <li class="item other-class selected">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
</ul>
```

Méthode .not()

API : http://api.jquery.com/not/

La méthode `.not()` retourne tous les éléments qui ne correspondent pas aux critères. C'est le contraire de la méthode `.filter()`

L'exemple suivant ajoute `class=&quot;selected&quot;` aux éléments `li` n'ayant pas la classe `.item`.

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li>…</li>
    <li class="item">…</li>
    <li class="item">…</li>
    <li>…</li>
    <li class="item">…</li>
</ul>
```

```
$('li').not('.item').addClass('selected');
```

Résultat

```
<ul class="list">
    <li class="item">…</li>
    <li class="item">…</li>
    <li class="selected">…</li>
    <li class="item">…</li>
    <li class="item">…</li>
```

```
    <li class="selected">…</li>
    <li class="item">…</li>
</ul>
```

Manipuler le DOM

jQuery nous propose plusieurs méthodes pour manipuler le DOM. Avec ces méthodes, nous pouvons supprimer, créer, cloner, ajouter… Bref nous pouvons remodeler le DOM à notre convenance.

Méthode	Description
`.addClass()`	Ajoute un ou plusieurs noms de classe à des éléments sélectionnés
`.after()`	Insére le contenu après les éléments sélectionnés
`.append()`	Insére le contenu à la fin les éléments sélectionnés
`.appendTo()`	Insére le contenu sélectionné à la fin les éléments sélectionnés
`.attr()`	Définit ou retourne les attributs / valeurs des éléments sélectionnés
`.before()`	Insére le contenu avant les éléments sélectionnés
`.clone()`	Effectue une copie des éléments sélectionnés
`.css()`	Définit ou retourne une ou plusieurs propriétés de style pour les éléments sélectionnés
`.detach()`	Supprime les éléments sélectionnés (conserve les données et les événements)
`.empty()`	Supprime tous les nœuds enfants et le contenu de l'élément sélectionné
`.hasClass()`	Vérifie si l'un des éléments sélectionnés ont un nom de classe spécifié
`.height()`	Définit ou retourne la hauteur des éléments sélectionnés
`.html()`	Définit ou retourne le contenu des éléments sélectionnés
`.innerHeight()`	Renvoie la hauteur d'un élément (y compris le padding, mais pas les border)
`.innerWidth()`	Renvoie la largeur d'un élément (y compris le padding, mais pas les border)
`.insertAfter()`	Insère des éléments HTML après les éléments sélectionnés
`.insertBefore()`	Insère des éléments HTML avant les éléments sélectionnés
`.offset()`	Définit ou retourne les coordonnées de l'offset (left et top) pour les éléments sélectionnés (en position relative dans le document)
`.offsetParent()`	Retourne l'élément le premier parent positionné
`.outerHeight()`	Renvoie la hauteur d'un élément (y compris le padding et les border)
`.outerWidth()`	Renvoie la largeur d'un élément (y compris le padding et les border)
`.position()`	Renvoie la position (par rapport à l'élément parent) d'un élément
`.prepend()`	Introduire le contenu au début des éléments sélectionnés
`.prependTo()`	Insére des éléments de HTML au début des éléments sélectionnés au format html
`.prop()`	Définit ou retourne propriétés / valeurs des éléments sélectionnés
`.remove()`	Supprime les éléments sélectionnés (y compris des données et des événements)
`.removeAttr()`	Supprime un ou plusieurs attributs à partir d'éléments sélectionnés
`.removeClass()`	Supprime une ou plusieurs classes à partir d'éléments sélectionnés
`.removeProp()`	Supprime une propriété définie par la méthode prop()
`.replaceAll()`	Remplace les éléments sélectionnés avec de nouveaux éléments HTML
`.replaceWith()`	Remplace les éléments sélectionnés par un nouveau contenu

`.scrollLeft()`	Définit ou retourne la position de la barre de défilement horizontale des éléments sélectionnés
`.scrollTop()`	Définit ou retourne la position de la barre de défilement vertical des éléments sélectionnés
`.text()`	Insére des éléments de HTML au début des éléments sélectionnés au format texte
`.toggleClass()`	Bascule entre l'ajout / suppression d'une ou plusieurs classes à partir d'éléments sélectionnés
`.unwrap()`	Supprime l'élément parent des éléments sélectionnés
`.val()`	Définit ou retourne la value des éléments sélectionnés (pour les éléments de formulaire)
`.width()`	Définit ou retourne la largeur des éléments sélectionnés
`.wrap()`	Entoure d'un balisage HTML l'élément sélectionné
`.wrapAll()`	Entoure d'un balisage HTML tous les éléments sélectionnés
`.wrapInner()`	Entoure d'un balisage HTML le contenu de l'élément sélectionné

Manipuler le texte

Dans cette partie, nous verrons les méthodes `.text()`, `.html()` et `.val()`.

Méthode .text()

API : http://api.jquery.com/text/

La méthode `.text()` récupère ou remplace le contenu en texte brut sans prendre en compte les balises HTML.

L'exemple suivant remplace le contenu texte de la balise `class=&quot;box&quot;`.

```
<div class="box">Je suis magicien</div>
```

```
$('.box').text('Hello world');
```

Résultat

```
<div class="box">Hello world</div>
```

Méthode .html()

API : http://api.jquery.com/html/

La méthode `.html()` récupère ou remplace le contenu HTML de l'élément.

L'exemple suivant remplace le contenu HTML de la balise `class=&quot;box&quot;`.

```
<div class="box">Je suis magicien</div>
```

```
$('.box').html('<h1>Hello world</h1>');
```

Résultat

```
<div class="box">
    <h1>Hello world</h1>
</div>
```

Méthode .val()

API : http://api.jquery.com/val/

La méthode `.val()` récupère ou remplace la value d'un élément de formulaire *input text*, *textarea* etc. Dans les champs de

type *radio* et *checked* elle renvoie **:checked** ou pas.

Dans l'exemple suivant, nous affichons la valeur du champ `#name` dans une boite d'alerte lors du click sur `#submit`.

```
<form role="form" action="#" class="box">
    <div class="form-group">
        <label for="#name">Nom :</label>
        <input type="text" name="name" id="name">
    </div>
    <input id="submit" class="btn btn-primary" type="submit" value="Envoyer">
</form>
```

```
$('#submit').click(function(){
    alert($('#name').val());
});
```

See the Pen .val() by Sutterlity (@sutterlity) on CodePen.

Supprimer du contenu

Dans cette partie, nous verrons les méthodes `.remove()` et `.empty()`.

Méthode .remove()

API : http://api.jquery.com/remove/

La méthode `.remove()` supprime du DOM l'élément sélectionné.

L'exemple suivant supprime du DOM `class=&quot;box&quot;`.

```
<div id="main">
    <div class="other">Je suis ingénieur informaticien</div>
    <div class="box">Je suis magicien</div>
</div>
```

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

Résultat

```
<div id="main">
    <div class="other">Je suis ingénieur informaticien</div>
</div>
```

Méthode .empty()

API : http://api.jquery.com/empty/

La méthode `.empty()` vide le contenu de l'élément sélectionné.

L'exemple suivant vide le contenu de l'élément `class=&quot;box&quot;`.

```
<div id="main">
    <div class="other">Je suis ingénieur informaticien</div>
    <div class="box">Je suis magicien</div>
</div>
```

```
$('.box').empty();
```

Résultat

```
<div id="main">
    <div class="other">Je suis ingénieur informaticien</div>
    <div class="box"></div>
</div>
```

Manipuler le CSS

Dans cette partie, nous verrons les méthodes `.addClass()`, `.removeClass()`, `.toogleClass()` et `.css()`.

Méthode .addClass()

API : http://api.jquery.com/addClass/

La méthode `.addClass()` permet d'ajouter une classe sur l'élément selectionné.

L'exemple suivant ajoute la classe `pod` à `class=&quot;box&quot;`.

```
<div class="box">…</div>
```

```
$('.box').addClass('pod');
```

Résultat

```
<div class="box pod">…</div>
```

Nous pouvons ajouter plusieurs classes en les séparant d'un espace.

```
$('.box').addClass('pod product');
```

Résultat

```
<div class="box pod product">…</div>
```

Exemple d'ajout de classe au click.

See the Pen .addClass() by Sutterlity (@sutterlity) on CodePen.

Méthode .removeClass()

API : http://api.jquery.com/removeClass/

La méthode `.removeClass()` permet de supprimer une classe sur l'élément selectionné.

L'exemple suivant supprime `class=&quot;box pod&quot;`.

```
<div class="box pod">…</div>
```

```
$('.box').removeClass();
```

Résultat

```
<div>…</div>
```

Nous pouvons aussi passer le nom d'une classe en paramètres. Par exemple, ici nous supprimons la classe `.pod` de l'élément `class=&quot;box pod&quot;`.

```
$('.box').removeClass('pod');
```

Résultat

```
<div class="box">…</div>
```

Exemple de supression de class au click.

See the Pen .removeClass() by Sutterlity (@sutterlity) on CodePen.

Méthode .toggleClass()

API : http://api.jquery.com/toggleClass/

La méthode `.toggleClass()` permet d'ajouter une classe sur l'élément selectionné si elle n'est pas présente et inversement.

L'exemple suivant ajoute la classe `.is-active` à `class=&quot;btn&quot;` au click, ou la supprime si elle est présente.

```
$('#a').click(function(){
  $('#b').toggleClass('bg-success');
});
```

See the Pen .toggleClass() by Sutterlity (@sutterlity) on CodePen.

Méthode .css()

API : http://api.jquery.com/css/

La méthode `.css()` affecte les styles CSS à l'élément sélectionné.

Elle agit aussi comme un *getter* et permet donc d'obtenir la chaîne de caractères de la valeur CSS demandée.

L'exemple suivant ajoute un fond rouge à `class=&quot;box&quot;`.

```
<div class="box">…</div>
```

```
$('.box').css('background-color','red');
```

Résultat

```
<div class="box" style="background-color: red;">…</div>
```

Nous pouvons bien sûr ajouter plusieurs styles CSS en même temps sous forme d'un objet avec des paramètres.

```
$('.box').css({
    'background-color': 'red',
    'font-size': '20px',
    'color': 'yellow'
});
```

Résultat

```
<div class="box" style="font-size: 20px; color: yellow; background-color: red;">…</div>
```

Getter

Comme je l'ai expliqué plus haut, nous pouvons obtenir la valeur d'une propriété CSS comme dans l'exemple ci-dessous.

```
<div class="box">…</div>
<div class="pod">…</div>
```

```
.pod { background: red }
```

```
$('.box').css('background-color', $('.pod').css('background-color') );
```

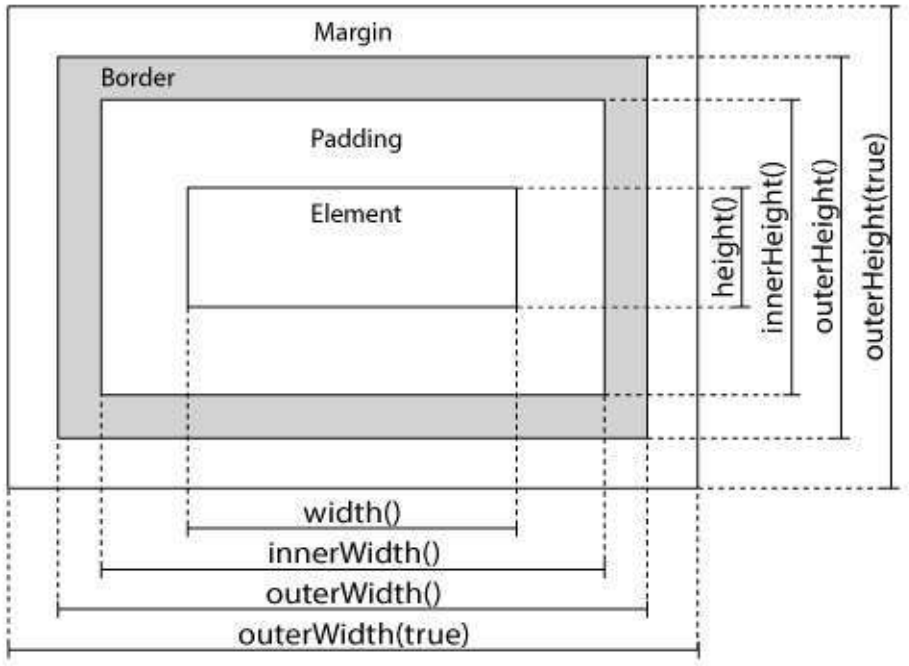
Résultat

```
<div class="box" style="background-color: red;">…</div>
<div class="pod">…</div>
```

Les dimensions

Dans cette partie, nous verrons les méthodes de calcul de dimensions `.width()`, `.innerWidth()`, `.outerWidth()`, `.height()`, `.innerHeight()`, `.outerHeight()`.

Voici le modèle de boite suivant lequel se base les méthodes.



Méthode	Description
`.width()`	Définit ou retourne la largeur d'un élément. Les padding, border et les margin ne sont pas compris dans le calcul.
`.innerWidth()`	Définit ou retourne la largeur d'un élément. Les padding sont compris dans le calcul mais pas les border et les margin.
`.outerdWidth()`	Définit ou retourne la largeur d'un élément. Les padding et les border sont compris dans le calcul, mais pas les margin.
`.outerdWidth(true)`	Définit ou retourne la largeur d'un élément. Les padding, border et les margin sont compris dans le calcul.
`.height()`	Définit ou retourne la hauteur d'un élément. Les padding, border et les margin ne sont pas compris dans le calcul.
`.innerHeight()`	Définit ou retourne la hauteur d'un élément. Les padding sont compris dans le calcul mais pas les border et les margin.
`.outerdHeight()`	Définit ou retourne la hauteur d'un élément. Les padding et les border sont compris dans le calcul, mais pas les margin.
`.outerdHeight(true)`	Définit ou retourne la hauteur d'un élément. Les padding, border et les margin sont compris dans le calcul.

Voyons maintenant à quoi peuvent maintenant nous servir ces méthodes.

Exemple 1

Nous pouvons donner la hauteur de la fenêtre aux éléments `<section>` , redimensionnable au resize du navigateur (astuce pour le responsive).

```
// Création de la fonction
function halfHeight() {
    // Attribuons à 'hWindow' la hauteur du navigateur
    var hWindow = $(window).height();
    // Ajoutons la hauteur de 'hWindow' aux éléments section
    $('section').css('height', hWindow);
}

$(function() {
    // Lancer la fonction quand le DOM est ready
    halfHeight();
});

$(window).resize(function() {
    // Lancer la fonction au redimensionnement de la fenêtre
    halfHeight();
});
```

See the Pen .height() by Sutterlity (@sutterlity) on CodePen.

Exemple 2

Nous pouvons aussi calculer la hauteur d'un élément pour l'assigner à un autre.

```
// Création de la fonction
function sameHeight(elem) {
    // initialisation de la hauteur de base à 0
    var tallest = 0;
    // Parcours de tous les éléments
    elem.each(function() {
        // Mise en variable de la hauteur de l'élément
        var thisHeight = $(this).height();
        // Condition pour savoir si la hauteur de l'élément est plus grande que 'tallest'
        if(thisHeight > tallest) {
            // Si l'élément est plus grand, nous assignons la nouvelle valeur à 'tallest'
            tallest = thisHeight;
        }
    });
    // Nous appliquons le style CSS sur l'élément
    elem.height(tallest);
}

$(function(){
    // Lancer la fonction quand le DOM est ready
    sameHeight( $('.box-list-item') );
});
```

See the Pen .height()-2 by Sutterlity (@sutterlity) on CodePen.

La gestion des événements

En JavaScript, nous travaillons en permanence avec des événements comme le clic, une touche du clavier pressée, le chargement d'un élément…

jQuery nous propose une gestion des événements au travers de différentes méthodes. Elles sont utilisées pour enregistrer les comportements lorsque l'utilisateur interagit avec le navigateur. Elles permettent également de manipuler les comportements enregistrés.

Liste des méthodes d'événements : http://api.jquery.com/category/events/

Effectuer une tâche au chargement d'une page

Prenons l'exemple d'un événement que vous connaissez déjà :

```
// Quand le DOM est prêt, nous appelons tous nos scripts
$(document).ready(function() {
    // Ici le DOM est prêt, appelons nos scripts
});
```

Cette fonction utilise avec la méthode `.ready()` qui écoute le chargement de la page.

L'écoute d'événements consiste à attacher une action à remplir à un élément pour déclencher une fonction que nous appellerons **écouteur d'événement**.

Cette fonction est la solution jQuery pour effectuer les tâches JavaScript déclenchées par l'événement `onload`. Les résultats sont équivalents mais ne sont pas exécutés au même moment.

L'événement `onload` est effectué quand le DOM est entièrement chargé (les images, les vidéos, etc), tandis que la méthode `.ready()` écoute uniquement le fait que le DOM est prêt (entièrement lu).

Si nous voulons écouter le chargement entier du DOM nous utiliserons la fonction suivante.

```
$(window).on("load", function(){
    // Ici le DOM est entièrement chargé, appelons nos scripts
});
```

Les événements de navigateur

API : http://api.jquery.com/category/events/browser-events/

jQuery permet d'écouter les événements déclenchés par le navigateur.

Méthode	Description
`.resize()`	Au redimensionnement de la fenêtre du navigateur
`.scroll()`	Au scroll dans un élément

Méthode .resize()

La méthode resize sert à écouter l'événement de redimensionnement du navigateur.

Prenons un exemple plus intrusif qui nous servira juste à voir le fonctionnement de cette méthode.

```
$(window).resize(function() {
    alert('Je redimensionne mon navigateur');
});
```

Cette méthode sera bien utile pour adapter certains de vos scripts au responsive webdesign. Par exemple, lors du calcul de hauteur de certains éléments. Nous le verrons d'ailleurs dans la partie expliquant la manipulation de dimensions.

Méthode .scroll()

La méthode `.scroll()` sert à écouter l'événement de scroll dans un élément.

Dans l'exemple suivant, nous ajoutons un message dans la console lors du scroll dans le navigateur.

```
$(window).scroll(function() {
    console.log('Je scroll dans ma page');
});
```

Cette méthode peut être utilisée pour lancer des animations CSS3 en ajoutant une classe en fonction de la distance du scroll par rapport au point haut de la page, comme dans l'exemple de CSS3 Animation Cheat Sheet.

```
$(window).scroll(function() {
    $('#animatedElement').each(function(){
    var imagePos = $(this).offset().top;
    var topOfWindow = $(window).scrollTop();
        if (imagePos < topOfWindow+400) {
            $(this).addClass("slideUp");
        }
    });
});
```

Gestionnaire d'attachement d'événement

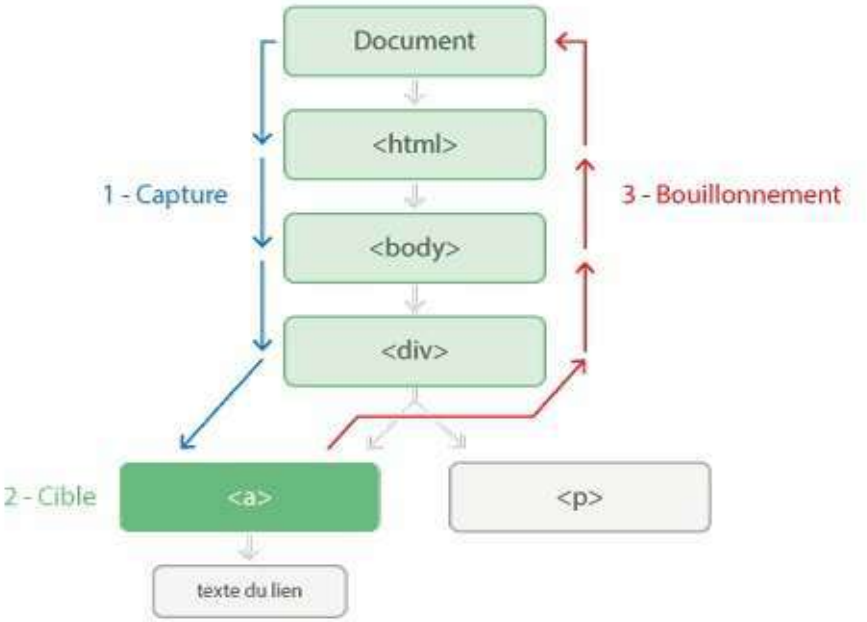
Dans la précédente partie, nous avons vu comment effectuer des tâches lors d'un événement sur le navigateur.

Dans cette partie, nous verrons comment attacher un événement comme le clic d'une souris, la modification d'un champ de formulaire, une touche clavier pressée …

Anatomie d'un événement

L'exécution d'un événement se fait en trois phases :

1. la capture
2. l'atteinte de la cible
3. le bouillonnement.



Méthode .on()

API : http://api.jquery.com/on/

La méthode `.on()` attache un ou plusieurs gestionnaires d'événements sur les éléments sélectionnés et les éléments enfants. Elle permet d'écouter les éléments créés dynamiquement.

Vous pouvez retrouver la liste des événements à cette adresse : http://api.jquery.com/category/events/

La méthode `.on()` prend comme événement :
blur, focus, focusin, focusout, load, resize, scroll, unload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup, error

Exemples

Dans l'exemple ci-dessous, nous écoutons l'événement au clic sur l'élément **.btn**, puis nous retournons la chaine de

caractères dans une fenêtre d'alerte.

```
$('.btn').on('click', function() {
    alert( $(this).text() );
});
```

See the Pen .on() by Sutterlity (@sutterlity) on CodePen.

Faisons de même, mais en sélectionnant le parent au départ.

```
$('.box').on('click', '.btn', function() {
    alert( $(this).text() );
});
```

See the Pen .on() -2 by Sutterlity (@sutterlity) on CodePen.

Prenons un autre exemple (plutôt tordu) : au clic ajoutons / supprimons-lui la classe **.border** , au survol de **.pod** ajoutons la classe **.hover** et pour finir, en sortant retirons la classe **.hover**.

```
$('.pod').on({
  click: function() {
    $(this).toggleClass('have-border');
  }, mouseenter: function() {
    $('body').css('background', '#212121')
  }, mouseleave: function() {
    $('body').css('background', '#fff')
  }
});
```

See the Pen .on() - Multiple event by Sutterlity (@sutterlity) on CodePen.

Les méthodes d'événements

API : http://api.jquery.com/category/events/

L'association d'un gestionnaire à un événement peut aussi s'écrire de façon plus simple mais perd l'avantage d'écouter en permanence la page. Par exemple, si un bouton est créé à l'aide d'un script après que le DOM soit chargé, la méthode `.click()` ne fonctionnera pas.

```
$('.btn').on('click', function() {
    alert( $(this).text() );
});
```

Devient :

```
$('.btn').click(function() {
    alert( $(this).text() );
});
```

L'avantage de ces méthodes raccourcies et d'être moins gourmandes en mémoire puisque le DOM n'est pas écouté en permanance.

L'exemple suivant nous permet de montrer que la méthode `.click()` ne fonctionne pas sur le bouton créé après le chargement du DOM.

```
<div class="box">
  <p>
    <button class="btn btn-primary" id="add">Ajouter un bouton</button>
  </p>
  <p id="container" class="bg-muted">
    <button class="btn btn-default btn-sm">Bouton créer avant le chargement du DOM</button>
  </p>
</div>
```

```
$('#add').click(function() {
    var html = ' Bouton créer après le chargement du DOM ';
    $("button.btn:last").parent().append(html);
});

$("#container .btn").click(function() {
    alert('Méthode .click()');
});

$("#container").on('click', '.btn', function() {
    alert('Méthode .on()');
});
```

See the Pen .click() vs .on() by Sutterlity (@sutterlity) on CodePen.

Voici quelques unes des méthodes les plus utilisées.

Les événements de la souris

Événement	Description

`.click()`	Lors du clic sur l'élément
`.dblclick()`	Au double-clic sur l'élément
`.scroll()`	Lors du scroll sur l'élément
`.mousedown()`	Lorsque le bouton de la souris est enfoncé sur l'élément
`.mouseup()`	Lorsque le bouton de la souris est relâché au dessus de l'élément
`.mousemove()`	Le pointeur se déplace dans l'élément
`.mouseover()`	Lorsque le curseur entre dans l'élément
`.mouseout()`	Lorsque le curseur sort de l'élément
`.mouseenter()`	Lorsque le curseur entre dans l'élément, le bouillonnement n'a pas d'effet
`.mouseleave()`	Lorsque le curseur sort de l'élément, le bouillonnement n'a pas d'effet

Les événements du clavier

Événement	Description
`.keydown()`	Lorsqu'une touche du clavier est enfoncée
`.keypress()`	Lorsqu'une touche du clavier vient d'être enfoncée
`.keyup()`	Lorsqu'une touche vient d'être relachée

Les autres événements

Événement	Description
`.blur()`	Lorsque l'élément perd son focus
`.focus()`	Lorsque l'élément a le focus
`.load()`	Lorsque l'élément est chargé
`.resize()`	Lorsque l'élément est redimensionné
`.change()`	Lorsque l'élément subit un changement
`.select()`	Lorsque le texte est sélectionné dans un formulaire
`.submit()`	Lorsque le formulaire est soumis

La propagation d'événements

Lorsque que vous déclarez un événement, il se propage à travers le DOM jusqu'à rencontrer l'élément ciblé.

Prenons le cas d'un élément unique :

```
<a id="totop" href="#">Haut de page</a>
```

```
$("#totop").click(function(){
    alert($(this).html());
});
```

L'événement pointe sur un élément unique `id=&quot;totop&quot;` et le script est exécuté une seule fois.

Tandis que dans le cas suivant, le message d'alerte apparaîtra deux fois.

```
<ul>
    <li>Niveau 1</li>
    <li>Niveau 1</li>
    <li>
        <ul>
            <li>Niveau 2</li>
            <li>Niveau 2</li>
        </ul>
    </li>
    <li>Niveau 1</li>
    <li>Niveau 1</li>
</ul>
```

```
$("li").click(function(){
    alert($(this).html());
});
```

L'événement se propagera à travers tous les **li** du document suivant le **principe de bouillonnement**. Le script sera exécuté à chaque élément **li** rencontré, peu importe son niveau dans le DOM. C'est ce qu'on appelle la propagation d'événements.

Exemple de propagation : http://codepen.io/mueflo00/details/sAHFu

Éviter la propagation

Dans certains cas, la propagation est problématique. Heureusement jQuery propose deux moyens de les éviter.

event.stopPropagation()

API : http://api.jquery.com/event.stoppropagation/

`event.stopPropagation()` empêche l'évènement de se propager dans l'arbre DOM en évitant le bouillonnement. Dans l'exemple ci-dessous, le message d'alerte n'apparaît qu'une fois.

```
$("li").click(function(e){
```

```
    event.stopPropagation();
    alert($(this).html());
});
```

return false

`return false` empêche-lui aussi l'évènement de se propager dans l'arbre DOM en évitant le bouillonnement comme `event.stopPropagation()`. Il annule également l'action en cours.

Dans l'exemple suivant, nous affichons un message d'alerte et nous empêchons la modification de l'URL au clic sur l'ancre.

```
<a id="totop" href="#">Haut de page</a>
```

```
$("#totop").click(function(){
    alert($(this).html());
    return false
});
```

Attention
Les déclarations en dessous de `return false` ne sont pas traitées.

Les effets

API : http://api.jquery.com/category/effects/

Dans ce chapitre nous aborderons les effets ainsi que les animations avec jQuery. Nous allons voir comment afficher et cacher des éléments, faire des fondus, des effets de slide ainsi que des animations personnalisées.

Animation jQuery Vs CSS3

Avant toute chose, j'aimerais faire un point sur la pertinence des effets d'animation jQuery à l'heure du CSS3.

Animation CSS3

Comme dans cette démonstration nous pouvons nous rendre compte que les animations CSS3 sont assez puissantes. Elles sont aussi plus rapides car intégrées directement dans le navigateur.

L'inconvénient des animations CSS3 réside dans le manque de compatibilité avec les navigateurs anciens (IE8 etc) même si elles ont un *fallback*. En effet, si le navigateur ne les comprend pas, il affiche directement le résultat final laissant ainsi le site fonctionnel.

Lorsque trop d'animations CSS3 sont lancées en même temps, elles ne sont pas fluides.

Il est aussi à l'heure actuelle impossible d'ajouter un *callback* en fin d'animation CSS3, au moins tant que transitionend n'est pas complètement implémenté.

Animation jQuery

L'avantage majeur des animations jQuery est l'intercompatibilité des navigateurs même IE7 est en mesure de lire une animation jQuery. Bien sûr, nous pouvons aussi ajouter des *callback*.

Lorsqu'il y a beaucoup d'animations, jQuery reste fluide contrairement à CSS3, évitant ainsi les bugs et les artefacts visuels.

Je profite aussi de ce point pour vous parler de la librairie Velocity.js. Elle propose de remplacer les animations jQuery par une version plus optimisée. Elles sont même plus rapides que les animations CSS3.

Conclusion

Les animations CSS3 sont plus simples à mettre en place mais souffrent encore de quelques lacunes. Si vous avez besoin de *fallback* il faudra passer par jQuery.

Les animations jQuery sont plus complexes à mettre en place et nécessitent l'import de la librairie. En prenant en compte Velocity.js elles sont aussi plus performantes.

Cacher / Afficher

Dans cette partie, nous verrons les méthodes `.hide()`, `.show()` et `.toggle()`.

Elles permettenet de faire apparaître ou disparaître des éléments avec une animation.

Ces méthodes peuvent prendre comme paramètres :

1. une vitesse
2. un callback

Méthode .hide()

API : http://api.jquery.com/hide/

La méthode `.hide()` permet de cacher les éléments.

```
$(selecteur).hide(vitesse,callback);
```

Exemple :

```
$('#hide').click(function(){
    $('#a').hide();
    $('#b').hide(1000);
    $('#c').hide('slow');
    $('#d').hide('slow', function(){
        console.log('Élément #d est caché');
    });
});
```

Un autre exemple :

8ee the Pen mehai by Sutterlity (@sutterlity) on CodePen.

Méthode .show()

API : http://api.jquery.com/show/

La méthode `.show()` permet d'afficher les éléments.

```
$(selecteur).show(vitesse,callback);
```

Exemple :

```
$('#show').click(function(){
    $('#a').show();
    $('#b').show(1000);
    $('#c').show('slow');
    $('#d').show('slow', function(){
        console.log('Élément #d est affiché');
    });
```

```
});
```

Un autre exemple :

See the Pen rfkcI by Sutterlity (@sutterlity) on CodePen.

Méthode .toggle()

API : http://api.jquery.com/toggle/

La méthode `.toggle()` permet de cacher / afficher les éléments.

```
$(selecteur).toggle(vitesse,callback);
```

Exemple :

```
$('#toggle').click(function(){
    $('#a').toggle();
    $('#b').toggle(1000);
    $('#c').toggle('slow');
    $('#d').toggle('slow', function(){
        console.log('Élément #d est caché/affiché');
    });
});
```

Un autre exemple :

See the Pen .toggle by Sutterlity (@sutterlity) on CodePen.

Les fondus

Dans cette partie nous verrons les méthodes `.fadeIn()`, `.fadeOut()`, .fadeToggle() `et` .fadeTo()`. Elles permettent de faire apparaître ou disparaître des éléments avec une animation de fondu.

Méthode .fadeIn()

API : http://api.jquery.com/fadeIn/

La méthode `.fadeIn()` permet de faire apparaître les éléments avec une animation de fondu.

```
$(selecteur).fadeIn(vitesse,callback);
```

Exemple :

```
$('#fadeIn').click(function(){
    $('#a').fadeIn();
    $('#b').fadeIn(1000);
    $('#c').fadeIn('slow');
    $('#d').fadeIn('slow', function(){
        console.log('Élément #d est apparu');
    });
});
```

See the Pen .fadeIn() by Sutterlity (@sutterlity) on CodePen.

Méthode .fadeOut()

API : http://api.jquery.com/fadeOut/

La méthode `.fadeOut()` permet de faire disparaître les éléments avec une animation de fondu.

```
$(selecteur).fadeOut(vitesse,callback);
```

Exemple :

```
$('#fadeOut').click(function(){
    $('#a').fadeOut();
    $('#b').fadeOut(1000);
    $('#c').fadeOut('slow');
    $('#d').fadeOut('slow', function(){
        console.log('Élément #d à disparu');
    });
});
```

See the Pen .fadeOut() by Sutterlity (@sutterlity) on CodePen.

Méthode .fadeToggle()

API : http://api.jquery.com/fadeToggle/

La méthode `.fadeToggle()` permet de faire apparaître / disparaître les éléments avec une animation de fondu.

```
$(selecteur).fadeToggle(vitesse,callback);
```

Exemple :

```
$('#fadeToggle').click(function(){
    $('#a').fadeToggle();
    $('#b').fadeToggle(1000);
    $('#c').fadeToggle('slow');
    $('#d').fadeToggle('slow', function(){
        console.log('Élément #d est apparu / à disparu');
    });
});
```

Un autre exemple :

See the Pen .fadeToggle by Sutterlity (@sutterlity) on CodePen.

Méthode .fadeTo()

API : http://api.jquery.com/fadeTo/

La méthode `.fadeTo()` permet de modifier l'opacité d'un élément avec un effet de transition.

```
$(selecteur).fadeTo(vitesse,opacité,callback);
```

Exemple :

```
$('#fadeTo').click(function(){
    $('#b').fadeTo(1000, 0.25);
    $('#c').fadeTo('slow', 0.25);
    $('#d').fadeTo('slow', 0.25, function(){
        console.log('Élément #d à une opacité de 25%');
    });
});
```

Plier / Déplier

Dans cette partie, nous verrons les méthodes `.slideDown()`, `.slideUp()` et `.slideToggle()`. Elles permettent de plier et déplier des éléments avec une animation de fondu.

Méthode .slideDown()

API : http://api.jquery.com/slideDown/

La méthode `.slideDown()` permet de déplier un élément avec une transition.

```
$(selecteur).slideDown(vitesse,callback);
```

Exemple :

```
$('#slideDown').click(function(){
    $('#a').slideDown();
    $('#b').slideDown(1000);
    $('#c').slideDown('slow');
    $('#d').slideDown('slow', function(){
        console.log('Élément #d est déplié');
    });
});
```

Un autre exemple :

See the Pen .slideDown() by Sutterlity (@sutterlity) on CodePen.

Méthode .slideUp()

API : http://api.jquery.com/slideUp/

La méthode `.slideUp()` permet de replier un élément avec une transition.

```
$(selecteur).slideUp(vitesse,callback);
```

Exemple :

```
$('#slideUp').click(function(){
    $('#a').slideUp();
    $('#b').slideUp(1000);
    $('#c').slideUp('slow');
    $('#d').slideUp('slow', function(){
        console.log('Élément #d est plié');
    });
});
```

Un autre exemple :

See the Pen SlideUp by Sutterlity (@sutterlity) on CodePen.

Méthode .slideToggle()

API : http://api.jquery.com/slideToggle/

La méthode `.slideToggle()` permet de plier / déplier un élément avec une transition.

```
$(selecteur).slideToggle(vitesse,callback);
```

Exemple :

```
$('#slideToggle').click(function(){
    $('#a').slideToggle();
    $('#b').slideToggle(1000);
    $('#c').slideToggle('slow');
    $('#d').slideToggle('slow', function(){
        console.log('Élément #d est plié / déplié');
    });
});
```

Un autre exemple :

See the Pen .slideToggle() by Sutterlity (@sutterlity) on CodePen.

Animation

Dans cette partie, nous verrons la méthode `.animate()` utilisée pour céer des animations personnalisées.

Méthode .animate()

API : http://api.jquery.com/animate/

```
$(selecteur).animate({paramètres},vitesse,callback);
```

Par exemple, déplaçons un élément **class="pod"** de 250px vers la droite.

```
$('#play').click(function(){
    $('.pod').animate({left: '250px'}, 500);
});
```

See the Pen .slideToggle() by Sutterlity (@sutterlity) on CodePen.

Les animations peuvent aussi être plus complexes.
Comme dans l'exemple ci-dessus, nous déplaçons **class="pod"** vers la droite et nous agrandissons sa largeur de 150px en même temps.

```
$('#play').click(function(){
    $('.pod').animate({
        left: '250px',
        width: '+=150px'
    }, 500);
});
```

See the Pen .animate() - 2 by Sutterlity (@sutterlity) on CodePen.

Nous pouvons aussi rajouter un callback.

```
$('#play').click(function(){
    $('.pod').animate({
        left: '250px',
        width: '+=150px'
    }, 500, function(){
        alert("L'animation est terminée");
    });
});
```

See the Pen .animate() - 3 by Sutterlity (@sutterlity) on CodePen.

Chaîner deux animations

Nous avons la possibilité avec la méthode `.animate()` de chaîner les animations, c'est-à-dire de lancer une animation puis une autre.

Continuons de complexifier l'exemple du dessus. À la fin de l'animation **class="pod"** prend une opacité de 25%.

```
$('#play').click(function(){
    $('.pod').animate({
        left: '250px',
        width: '+=150px'
    }, 500)
    .animate({
        opacity: '0.25'
    }, 1000);
});
```

See the Pen .animate() - 4 by Sutterlity (@sutterlity) on CodePen.

Nous pouvons aussi lancer les deux animations en même temps en fixant le paramètre `queue` à `false`. Cela aura pour effet de ne plus mettre l'animation ciblée à la queue.

```
$('#play').click(function(){
    $('.pod').animate({
        left: '250px',
        width: '+=150px'
    }, 500)
    .animate({
        opacity: '0.25',
    }, {
        'queue': false,
        'duration': 1000
    };
});
```

See the Pen .animate() - 5 by Sutterlity (@sutterlity) on CodePen.

Stopper une animation / un effet

Dans cette partie, nous verrons la méthode `.stop()` servant à stopper **une animation** ou **un effet**.

Méthode .stop()

API : http://api.jquery.com/stop/

```
$(selecteur).stop(stopAll,goToEnd)
```

Cette méthode prend 2 paramètres :

Paramètre	Description
StopAll	**Optionnelle** : la valeur par défaut est **false**. Valeur booléenne indiquant **true** ou **false**. **false** permet d'arrêter toutes les animations en file d'attente.
goToEnd	**Optionnelle** : la valeur par défaut est **false**. Valeur booléenne indiquant **true** ou **false**. **false** permet de finir toutes les animations immédiatement.

Stop sans paramètres

Prenons l'exemple d'un carré qu'on anime avec la méthode `.animate()` au clic sur le bouton *Play*. Si nous cliquons sur le bouton *stop*, l'animation en cours se stoppe grâce à la méthode `.stop()` sans paramètres. Au click sur *play*, l'animation reprend là où elle en était.

```
// Déclaration de l'animation
$('#play').click(function(){
    $('.pod').animate({
        left: '250px',
        width: '+=150px'
    }, 1500)
    .animate({
        opacity: '0.25',
    }, 3000);
});

// Déclaration de la méthode .stop() au click
$('#stop').click(function(){
   $('.pod').stop();
});
```

See the Pen .stop() by Sutterlity (@sutterlity) on CodePen.

Prenons l'exemple d'un carré qu'on modifie au survol. Dans l'exemple si dessous nous voyons qu'au survol avec de multiples passages sur le carré rose, l'animation continue le nombre de fois survolée, ce qui peut s'avérer gênant. Tandis que le carré bleu cesse son animation lorsque le survol est terminé.

```
// Déclaration sans .stop()
$('#a').hover(function(){
    $(this).fadeTo('slow', 0.25);
  },function(){
    $(this).fadeTo('slow', 1);
});
```

```
// Déclaration avec .stop()
$('#b').hover(function(){
    $(this).stop().fadeTo('slow', 0.25);
  },function(){
    $(this).stop().fadeTo('slow', 1);
});
```

See the Pen .stop() - 2 by Sutterlity (@sutterlity) on CodePen.

Stop avec les paramètres 'stopAll' et 'goToEnd'

Reprenons le cas de notre premier exemple. Agrémentons le d'un bouton permettant de stopper les animations en cours et un autre bouton permettant de les terminer toutes.

```
// Stopper toutes les animations en cours
$('#stopAll').click(function(){
   $('.pod').stop(true);
});

// Finir toutes les animations en cours
$('#goToEnd').click(function(){
   $('.pod').stop(true, true);
});
```

See the Pen .stop() - 3 by Sutterlity (@sutterlity) on CodePen.

Ajouter un délais

Dans cette partie, nous verrons la méthode `.delay()` servant à ajouter un délais avant l'exécution d'un effet.

Méthode .delay()

API : http://api.jquery.com/delay/

```
$(selecteur).delay(Durée)
```

Prenons le cas de deux éléments **#a** et **#b**, qui reçoivent successivement une méthode `.slideUp()` puis `.slideDown()`. Nous ajoutons à l'élément **#b** une méthode `.delay(1000)` entre les deux méthodes de *slide*. Nous pouvons nous rendre compte qu'au clic sur le bouton, les effets de **#a** sont successifs, tandis que pour **#b** il y a une temporisation de 1s entre les deux.

```
$('#play').click(function(){

    // Pas de .delay()
    $('#a').slideUp().slideDown();

    // Avec .delay()
    $('#b').slideUp().delay(1000).slideDown();

});
```

See the Pen .delay by Sutterlity (@sutterlity) on CodePen.

Pour information la méthode `.delay()` ne s'applique qu'aux effets. Pour temporiser d'autres méthodes, il convient d'employer la fonction JavaScript setTimeout.

Ajax

Ajax ou **Asynchronous JavaScript and XML** est une technologie se basant sur l'objet XMLHttpRequest.

XMLHttpRequest est un objet ActiveX ou JavaScript qui permet d'obtenir des données au format XML, JSON, mais aussi HTML, ou encore texte simple à l'aide de requêtes HTTP.

Ajax va pouvoir récupérer ou transmettre sur un serveur des données sans rafraichir notre navigateur.

Exemples d'applications utilisant AJAX : Gmail, Google Maps, Youtube et les onglets Facebook.

Le principe

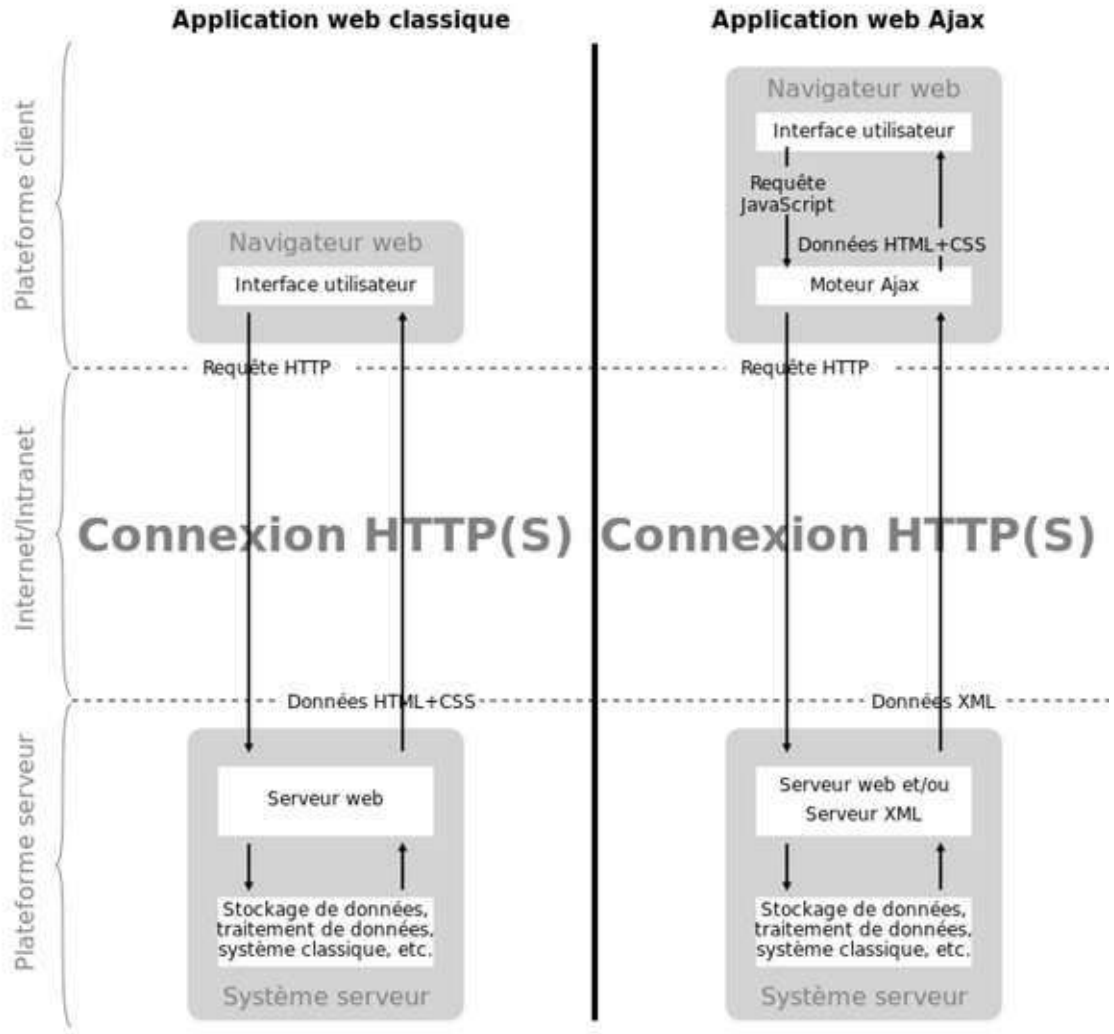
Source : Wikipédia

Dans une application Web, la méthode classique de dialogue entre un navigateur et un serveur est la suivante : lors de chaque manipulation faite par l'utilisateur, le navigateur envoie une requête contenant une référence à une page Web, puis le serveur Web effectue des calculs, et envoie le résultat sous forme d'une page Web à destination du navigateur. Celui-ci affichera alors la page qu'il vient de recevoir. Chaque manipulation entraîne la transmission et l'affichage d'une nouvelle page. L'utilisateur doit attendre l'arrivée de la réponse pour effectuer d'autres manipulations.

En utilisant Ajax, le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante : un programme écrit en langage de programmation JavaScript, incorporé dans une page web, est exécuté par le navigateur. Celui-ci envoie en arrière-plan des demandes au serveur Web, puis modifie le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçu du serveur, évitant ainsi la transmission et l'affichage d'une nouvelle page complète.

La méthode classique de dialogue utilise des mécanismes propres au World Wide Web, qui sont incorporés dans tous les navigateurs ainsi que les robots d'indexation, et ne nécessite pas de programmation. Au contraire, le fonctionnement d'Ajax nécessite de programmer en JavaScript les échanges entre le navigateur et le serveur Web. Il nécessite également de programmer les modifications à effectuer dans la page Web à la réception des réponses, sans quoi les dialogues se font à l'insu de l'utilisateur.

En Ajax, comme le nom l'indique, les demandes sont effectuées de manière asynchrone : le navigateur Web continue d'exécuter le programme JavaScript alors que la demande est partie, il n'attend pas la réponse envoyée par le serveur Web et l'utilisateur peut continuer à effectuer des manipulations pendant ce temps.



Prérequis

Avant de vous lancer dans l'aventure AJAX

Vous devez travailler sur un serveur pour que vos échanges Ajax fonctionnent. Vous pouvez travailler en local et instaler un logiciel du type WAMP, MAMP, XAMPP …

La méthode .load()

API : http://api.jquery.com/load/

La méthode `.load()` nous permettra de récupérer du contenu à insérer dans notre page. Nous pouvons charger des fichiers du type : HTML, PHP, TXT, XML et JSON.

Elle est un raccourci de la fonction `.ajax()`.

La méthode en détail

```
$(selecteur).load(url,data,function(reponse,status,xhr));
```

Paramètre	Description
url	**Requis** Une chaîne contenant l'URL vers laquelle la demande est envoyée
data	**Optionnel** Spécifie les données à envoyer vers le serveur en même temps que la demande
function(reponse,status,xhr)	**Optionnel** Indique une fonction à exécuter quand la méthode est terminée Paramètres supplémentaires : **- data** - contient les données résultant de la demande **- status** - contient l'état de la demande ("success", "notmodified", "error", "timeout", ou "parsererror") **- xhr** - contient l'objet XMLHttpRequest

Exemples

Charger un document

Voici un exemple simple pour comprendre le fonctionnement.

```
$('#result').load('result.txt');
```

Nous avons chargé le contenu du ficher *result.txt* dans la balise **#result**.

Faisons de même avec un fichier PHP mais cette fois-ci au clic.

```
$('#a').click(function(){
    $('#result').load('inc/result.php');
});
```

Bien sûr, nous pouvons ajouter des effets de fondu. La méthode consiste à cacher le contenu, le charger puis faire le fondu en *callback*.

```
$('#a').on('click', function(){
    var box = $('#result');
    box.hide().load('inc/result.php', function() {
```

```
        box.fadeIn('750');
    });
});
```

Charger une partie d'un document

Allons encore plus loin et chargeons une seule partie du fichier suivant.

```
<div id="box-1">
    <h2>Titre numéro 1</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nam, magnam, repellat dicta nesciunt aperiam cupiditat
</div>

<div id="box-2">
    <h2>Titre numéro 2</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.</p>
</div>
```

Voici le script pour charger **#box-2** dans **result** au clic sur **#a**.

```
$('#a').click(function(){
    $('#result').load('inc/result.php #box-2');
});
```

Passer des paramètres à un fichier PHP

Il nous est possible de demander en PHP à la base de données de nous retourner des informations necessitant un paramètre.

Dans l'exemple ci-dessous, nous récupérons la valeur d'un champ **#search**, puis nous la passons en paramètre.

```
$('#a').click(function(){
    var param = 'id=' + $('#search').val();
    $('#result').load('inc/result.php', param);
});
```

La méthode .get()

API : http://api.jquery.com/jQuery.get/

La méthode `.get()` permet de recevoir des données. Contairemement à la méthode `.load()`, elle nous permet non pas de recevoir du contenu directement dans l'élément ciblé mais de nous laisser le choix sur les actions à faire.

La méthode en détail

```
$.get( url, data, function(data,status,xhr), dataType );
```

Paramètre	Description
url	**Requis** Une chaîne contenant l'URL vers laquelle la demande est envoyée
data	**Optionnel** Spécifie les données à envoyer vers le serveur en même temps que la demande
function(data,status,xhr)	**Optionnel** Indique une fonction à exécuter lorsque la méthode est terminée Paramètres supplémentaires : **- data** - contient les données résultant de la demande **- status** - contient l'état de la demande ("success", "notmodified", "error", "timeout", ou "parsererror") **- xhr** - contient l'objet XMLHttpRequest
dataType	**Optionnel** Spécifie le type de données attendu. Par défaut jQuery effectue une estimation automatique (HTML, PHP, TXT, XML et JSON)

Exemples

Charger un document

Prenons l'exemple suivant, le fichier **'inc/test.html'** se charge dans **#result**.

```
$.get('inc/test.html', function( data ) {
    $('#result').html( data );
});
```

Elle est un raccourci de la fonction `.ajax()` suivante. Nous la verrons plus en détails dans la prochaine partie.

```
$.ajax({
    url: 'inc/test.html',
    success: function (data) {
        $('#result').html( data );
    }
});
```

Maintenant, affichons le statut dans la console.

```
$.get('inc/test.html', function( data, status ) {
    $('#result').html( data );
    console.log(status);
});
```

Charger des données de type JSON

Voici un exemple de chargement de données issues du fichier *JSON* à l'aide de `.get()`.

Prenons l'exemple de fichier JSON.

```
{
    "name": "Jean-Michel",
    "email": "jeanmich@caramail.com"
}
```

Voici le script nécessaire à l'affichage des informations.

```
$.get('inc/user.json', function( data ) {
    $('#result').html( data.name + ' : ' +  data.email );
}, 'json');
```

JQuery à aussi une méthode raccourcie pour le JSON `.getJSON()`

```
$.getJSON('inc/user.json', function( data ) {
    $('#result').html( data.name + ' : ' +  data.email );
});
```

Ce dernier script est le raccourci de la méthode ajax suivante.

```
$.ajax({
    dataType: "json",
    url: 'inc/test.html',
    success: function (data) {
        $('#result').html( data.name + ' : ' +  data.email );
    }
});
```

La méthode .post()

API : http://api.jquery.com/jQuery.post/

La méthode `.post()` permet d'envoyer des données. Par exemple, vous pouvez l'utiliser pour envoyer des données saisies dans un formulaire.

La méthode consiste à envoyer des données vers un fichier *Php* qui se chargera de les transmettre au serveur. Cette méthode peut également retourner des informations en *callback* dans la page.

```
$.post('send.php',
    {
        name: 'Jean-Michel',
        email: 'jeanmich@caramail.com'
    }, function(data) {
        alert(data);
});
```

La méthode en détail

```
$.get( url, data, function(data,status,xhr), dataType );
```

Paramètre	Description
url	**Requis** Une chaîne contenant l'URL vers laquelle la demande est envoyée
data	**Optionnel** Spécifie les données à envoyer vers le serveur en même temps que la demande
function(data,status,xhr)	**Optionnel** Indique une fonction à exécuter lorsque la méthode est terminée Paramètres supplémentaires : **- data** - contient les données résultant de la demande **- status** - contient l'état de la demande ("success", "notmodified", "error", "timeout", ou "parsererror") **- xhr** - contient l'objet XMLHttpRequest
dataType	**Optionnel** Spécifie le type de données attendu. Par défaut jQuery effectue une estimation automatique (*xml*, *json*, *script*, ou *html*)

Exemple d'envoi de données

Dans l'exemple suivant, nous envoyons une requête permettant de faire une recherche puis nous affichons le résultat de la recherche.

Cet exemple ne contient pas le fichier PHP de traitement (ce n'est pas le sujet de ce cours).

La structure html

```
<form action="/" id="searchForm">
    <input type="text" name="s" placeholder="Rechercher…">
    <input type="submit" value="Ok">
```

```
</form>
<div id="result"></div>
```

Le script jQuery de traitement

```
$('#searchForm').submit(function(event) {

  // Stop la propagation par défaut
  event.preventDefault();

  // Récupération des valeurs
  var $form = $(this),
       term = $form.find( "input[name='s']" ).val(),
       url = $form.attr( "action" );

  // Envoie des données
  var posting = $.post( url, { s: term } );

  // Reception des données et affichage
  posting.done(function(data) {
    var content = $(data).find('#content');
    $('#result').empty().append(content);
  });

});
```

La méthode .ajax()

La méthode `.ajax()` est l'artillerie lourde. Elle permet de maîtriser l'ensemble des paramètres de requête. Les autres méthodes vues juste avant sont seulement des raccourcis d' `.ajax()` .

Cette méthode peut s'écrire de deux façons:

```
$.ajax(url, {options});
```

ou

```
$.ajax({options});
```

Paramétrer sa requête

Voici les options les plus utilisées :

Option	Description
URL	Adresse à laquelle la requête est envoyée
type	Le type de requête *GET* (par défaut) ou *$POST*
data	Les données à envoyer au serveur
datatype	Le type de données pouvant être transmises au serveur : *php*, *html*, *script*, *json* et *xml*
success	La fonction à appeler si la requête a abouti
error	La fonction à appeler si la requête n'a pas abouti
timeout	Le délai maximum en millisecondes de traitement de la demande. Passé ce délai, elle retourne le paramètre **error**.

Bien sûr, il existe bien d'autres options que vous pouvez retrouver dans l'API jQuery.

Chargement de fichier

Voici un exemple simple permettant de comprendre le fonctionnement.

```
// Au clic sur le bouton #search je lance la fonction
$('#search').on('click', function(){

    // J'initialise le variable box
    var box = $('#result');

    // Je définis ma requête ajax
    $.ajax({

        // Adresse à laquelle la requête est envoyée
        url: '../inc/search.php',

        // Le délai maximun en millisecondes de traitement de la demande
        timeout: 4000,

        // La fonction à apeller si la requête aboutie
```

```
        success: function (data) {
            // Je charge les données dans box
            box.html(data);
        },

        // La fonction à appeler si la requête n'a pas abouti
        error: function() {
            // J'affiche un message d'erreur
            box.html("Désolé, aucun résultat trouvé.");
        }

    });

});
```

Créer un plugin

Dans cette partie nous allons voir comment créer son propre plugin jQuery.

Rien de compliqué, il suffit de déclarer le plugin, puis de l'appeler.

Déclarer un plugin

```
(function($) {
    $.fn.nomDuPlugin = function(parametre, …) {
        // Ici vos scripts
    };
})(jQuery);
```

Appeler un plugin

```
$('#elem').nomDuPlugin(58);
```

Bien entendu, notre plugin peut prendre :

- aucun paramètre ;
- un paramètre ;
- plusieurs paramètres.

Exemple de plugin sans paramètre

Dans cet exemple nous allons créer un plugin nous permettant d'ajouter un attribut `target=&quot;_blank&quot;` aux éléments ayant un un attribut `rel=&quot;external&quot;`.

Création du plugin

```
(function($) {
    $.fn.newWindow = function() {
        $(this).attr('target', '_blank');
    };
})(jQuery);
```

Appel du plugin

```
$('a[rel="external"]').newWindow();
```

Exemple de plugin avec paramètres

Dans cet exemple nous allons créer un plugin permettant d'ajouter les propriétés CSS `color` et `background-color` avec des valeurs par défaut à un élément.

Création du plugin

```
(function($) {
    $.fn.myPlugin = function( options ) {

        // Déclarations de valeurs par défaut
        var settings = $.extend({
            color           : '#09f',
            backgroundColor : "#f90"
        }, options );

        // Application des valeurs 'par défaut si non renseignées'
        return this.css({
            color           : settings.color,
            backgroundColor : settings.backgroundColor
        });

    };
})(jQuery);
```

Appel du plugin

```
$('.box').myPlugin({color: 'orange'});
```

jQuery UI

jQuery UI (User Interface) est une librairie de composants prêt à l'emploi permettant aux développeurs d'ajouter des effets visuels, des interactions et des widgets en quelques lignes.

jQuery UI est développé par la communauté. Il fait partie du projet jQuery.org.

Les widgets sont conçus pour être paramétrables au niveau du style et du comportement. Ils peuvent être installés indépendamment les uns des autres.

Un des grands avantages de jQuery UI est qu'il fonctionne de la même manière sur tous les navigateurs.

Les fonctions

Éléments d'intéraction :

Fonction	Description
Draggable	Autorise les éléments devant être déplacés à l'aide de la souris
Droppable	Crée des cibles pour *Draggable*
Resizable	Modifie la taille d'un élément à l'aide de la souris
Selectable	Utilise la souris pour sélectionner les éléments individuellement ou en groupe
Sortable	Réorganise les éléments d'une liste ou d'une grille à l'aide de la souris

Les widgets

Fonction	Description
Accordion	Plie/déplie des éléments de contenu pour la présentation d'informations dans un espace limité
Autocomplete	Permet aux utilisateurs de choisir parmi une liste pré-remplie de valeurs en s'appuyant sur la recherche et le filtrage
Button	Améliore l'aspect visuel et ajoute des classes suivant les états
Datepicker	Permet d'afficher un calendrier et de sélectionner une date
Dialog	Affiche une boite de dialogue 'popin'
Menu	Menu avec des interactions à la souris et au clavier
Progressbar	Affiche une progress-bar
Selectmenu	Affiche un élément select avec des options avancées
Slider	Afficher un élément similaire à un input range
Spinner	Améliore le champ *input text* en lui ajoutant des boutons permettant d'ajouter ou de diminuer une valeur
Tabs	Affiche une zone de contenu unique, avec un menu filtrant la partie du contenu à afficher
Tooltip	Affiche des info-bulles personnalisables

Les effets

Fonction	Description
Class	Ajoute des classes en animant le contenu
Color Animation	Anime le changement de couleur
Easing	Ajoute de l'accélération aux animations
Effect	Ajoute de nouvelles animations
Hide	Cache les éléments avec une animation
Remove Class	Supprime des classes en animant le contenu, et ajoute une autre classe
Show	Affiche les éléments avec une animation
Switch Class	Ajoute/supprime des classes en animant le contenu
Toggle	Affiche/cache les éléments avec une animation
Toggle Class	Ajoute/supprime des classes en animant le contenu

Les utilitaires

Fonction	Description
Position	Positionne un élément par rapport à la fenêtre, le document, un autre élément ou le curseur.
Widget Factory	Créer des plugins jQuery avec état utilisant la même abstraction que tous les widgets jQuery UI.

ThemeRoller

Url : http://jqueryui.com/themeroller/

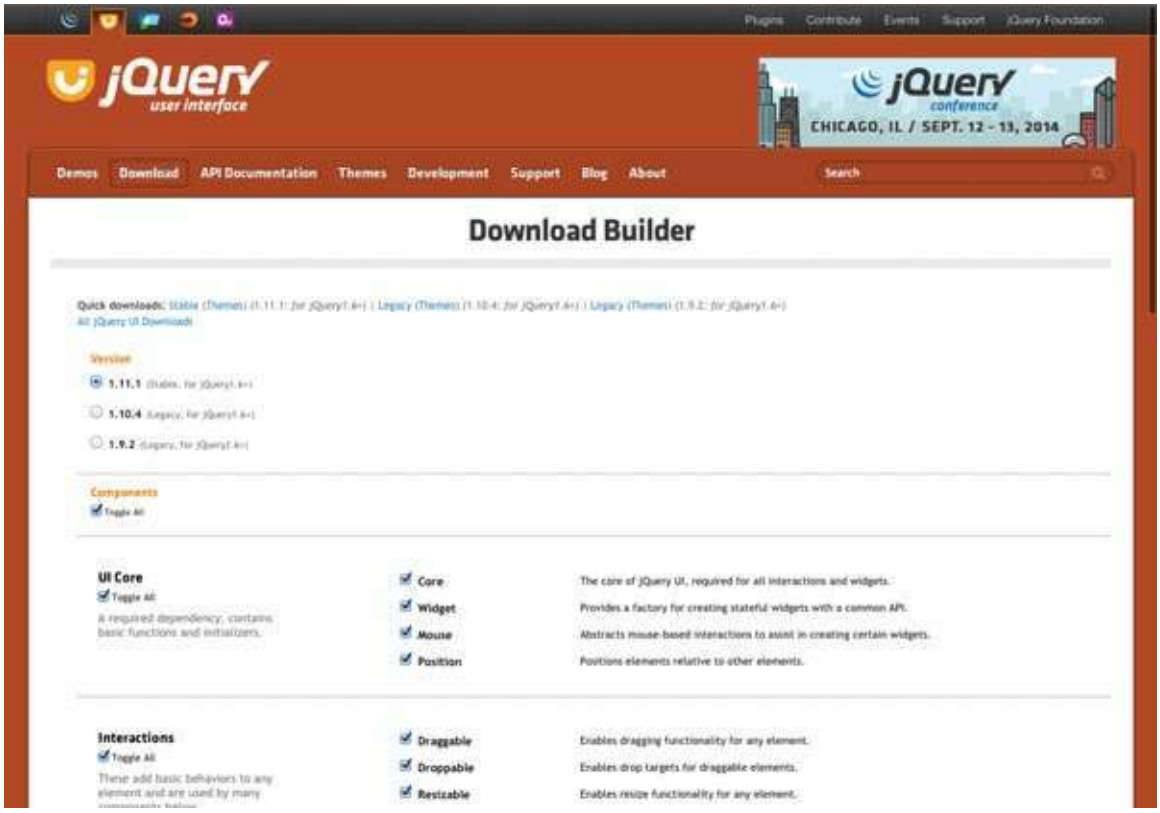
À la différence de jQuery, jQuery UI propose des éléments d'interface ayant besoin d'une couche graphique. Cet aspect visuel est pris en compte à travers un thème CSS paramétrable et personnalisable.

Installation jQuery UI

Comme la bibliothèque jQuery, jQuery UI n'est pas installé par défaut dans votre navigateur, il va donc falloir l'importer de la même manière que jQuery.

Télécharger jQuery UI

Aller sur lien : http://jqueryui.com/download/



Vous pouvez importer la bibliothèque en entier ou seulement les modules indispensables à votre projet, afin d'optimiser la vitesse de chargement de votre site.

Installation

De la même manière que pour jQuery, jQuery UI est un fichier JS à insérer dans votre document. Vous pouvez faire l'importation dans le header ou en bas de la page.

jQuery UI a aussi besoin d'un fichier CSS pour fonctionner. Bien entendu, pour la mise en production il est conseillé de concaténer les CSS

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Installation de jQuery</title>
    <link rel="stylesheet" href="css/styles.css">
    <link rel="stylesheet" href="css/jquery-ui.min.css">
    <script src="js/jquery-2.1.1.min.js"></script>
    <script src="js/jquery-ui.min.js"></script>
```

```
</head>
<body>
    <!-- Contenu de la page -->
</body>
</html>
```

Glossaire

DOM

Le DOM ou Document Object Model est un standard du W3C qui définit l'arborescence du document HTML.

11. Gestion des événements 11.3. Les méthodes d'événements 11.4. La propagation d'événements 6. Le DOM 7. Sélectionner des éléments 10. Manipuler le DOM 10.4. Les dimensions 10.2. Supprimer 9. Parcourir le DOM 9.2. Les descendants 9.1. Les ancêtres 5. Premiers pas 1. Présentation de jQuery

callback

Fonction de rappel qui s'éxécute lorsque l'autre fonction se termine.

8. Notions indispensables 13.1. La méthode .load() 13.3. La méthode .post() 12. Les effets 12.4. Animation 12.1. Cacher / Afficher

fallback

Solution alternantive

12. Les effets

getter

Permet d'obtenir la valeur de l'élément

10.3. Manipuler le CSS