jQuery

Table des matières

1. Introduction....2
1.1. L'histoire d'un framework....2
1.2. Les outils....2
2. Les Bases du framework....2
2.1. Mon premier programme....2
2.2. Les sélecteurs....4
2.2.1. Le cas des formulaires....6
2.2.2. Le sélecteur $(this)....6
2.2.3. Performance des sélecteurs....6
2.3. jQuery et les événements....7
2.3.1. L'écoute sur la souris....7
2.3.2. L'écoute sur le clavier....8
2.3.3. Le cas des formulaires....8
2.3.4. Les gestionnaires d'évènements....8
2.3.5. Les espaces de noms....11
2.4. Manipuler le code CSS....11
3. DOM et animations....13
3.1. Les effets....13
3.1.1. Les effets natifs....16
3.1.2. Le contrôle des effets....16
3.2. Manier les attributs....18
3.2.1. Gérer les classes....19
3.3. Parcourir les éléments du DOM....20
3.4. Manipuler le code HTML....22
3.4.1. Le contenu textuel....22
3.4.2. Le contenu HTML....23
3.4.3. Le contenu des éléments de formulaire....24
3.4.4. Manipulation des éléments HTML....24
3.4.5. Créer des éléments à la volée....25

jQuery est une bibliothèque JavaScript libre et multi-plateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web.

1. Introduction

1.1. L'histoire d'un framework[1]

JavaScript est un langage de programmation de scripts créé en 1995 et principalement employé dans les pages web interactives mais aussi pour les serveurs. Très rapidement, ses utilisateurs ont voulu faire évoluer JavaScript pour le rendre meilleur. C'est dans cet esprit que la communauté de JavaScript a développé des frameworks pour le langage ; l'un d'eux est jQuery !

jQuery n'est en fait qu'un seul et unique fichier JavaScript téléchargeable sur le web. C'est une énorme bibliothèque de fonctions JavaScript qui ont été écrites et regroupées pour plus de simplicité.

Cependant, jQuery est plus qu'une simple librairie même s'il en a les traits majeurs, il va vraiment vous faire coder en JavaScript d'une nouvelle manière. Et ceci à tel point qu'il est tout à fait possible de considérer jQuery comme un langage un peu à part, puisqu'il s'agit vraiment d'une redécouverte totale de JavaScript.

jQuery est un framework qui va nous simplifier la programmation concernant :

- la compatibilité inter-navigateurs
- l'utilisation d'AJAX[2]
- les animations
- les formulaires

jQuery est publié sous licence libre GNU GPL et a déjà subi beaucoup de mises à jour. La communauté est en effet très active, et les mises à jour se font toujours régulièrement.

1.2. Les outils

Pour coder à l'aide de jQuery, il faut quelques outils :

- un navigateur : FireFox : actuellment le meilleur de tous les navigateurs lorsqu'il s'agit de développer pour le web.
- le greffon Firebug : l'extension Firebug se caractérise en fait par toute une panoplie d'outils pour le développement web qui vont venir s'intégrer complètement à Firefox dans le but de le compléter et de le rendre beaucoup utile et agréable à l'utilisation.
- un éditeur de texte : Notepad++ : c'est un éditeur suffisament perfectionné pour gérer la coloration syntaxique.

2. Les Bases du framework

2.1. Mon premier programme

Pour commencer à travailler avec jQuery, il va falloir tout naturellement l'inclure dans les pages web.

1 Architecture logicielle : famille d'ensembles logiciels facilitant le développement de programmes et leur déploiement
2 Asynchronous JavaScript and XML : architecture informatique qui permet de construire des sites web dynamiques

La méthode la plus recommandée pour inclure jQuery dans la page web est encore d'inclure le fichier directement depuis un serveur distant. Google met à disposition de tous le framework jQuery hébergé directement sur ses serveurs :

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Test jQuery</title>
   </head>
   <body>
   <p>On va utiliser jQuery !</p>
   <script
       src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
   </script>
   </body>
</html>
```

Inclure jQuery depuis un serveur Google permet une mise en cache immédiate, pour toujours accélérer les temps de chargement de votre site.

L'ensemble du framework jQuery repose sur une seule fonction : jQuery() ! Sans cette fonction, le code jQuery peut ne pas être interprété correctement :

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Test jQuery</title>
   </head>
   <body>
   <p>On va utiliser jQuery !</p>
   <script
       src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
   </script>
   <script>
       jQuery(document).ready(function(){
           // Du code en jQuery va pouvoir être tapé ici !
       });
   </script>
   </body>
</html>
```

L'instruction « jQuery(document).ready(function() » peut être traduite par : "quand le document HTML est chargé, lance une fonction qui contient le code jQuery."

Pour vérifier que cela est fonctionnel il suffit d'ajouter un console.log à la place des commentaires dans le code ci-dessus pour voir si on reçoit quelque chose dans la console Firebug à l'exécution de ce code. Ou bien utiliser un alert().

Exemple :

```
/*
 * Avec un console.log
 */
jQuery(document).ready(function(){
      console.log("jQuery est prêt !");
});
```

```
/*
 * Avec un alert()
 */
jQuery(document).ready(function(){
      alert("jQuery est prêt !");
});
```

Remarque : on peut ne pas spécifier le ready() :

```
/*
 * La structure de base simplifiée.
 */
jQuery(function(){
});
```

De même, la fonction jQuery peut aussi s'appeller avec l'alias $() :

```
$(function(){
      alert('jQuery est prêt !');
});
```

La plupart des bibliothèques JavaScript utilisent elles aussi le $ (dollar), ce qui peut être la source de conflit entre les différentes librairies sur une même page. Pour éviter ce genre de problème, il est possible de passer la variable jQuery à une fonction anonyme pour être sûr que notre alias ne soit pas l'origine de conflits.

```
(function($){
    // notre code ici
})(jQuery);
```

Parallèlement, ready() accepte également un alias en paramètre de la fonction anonyme. Il est alors possible de simplifier le code au maximum :

```
jQuery(function($){
    // notre code ici
});
```

Le framework met en place une fonction dite principale. Ainsi, pour agir avec les éléments d'une page web, on réalisera ce qu'on appelle couramment un « ciblage d'élément » et on agira ensuite dessus grâce à des méthodes :

```
$('monElement').maMethode();
```

2.2. Les sélecteurs

jQuery a une syntaxe qui permet de sélectionner très facilement des éléments sur la page. Le framework fonctionne avec les sélecteurs de CSS :

```
$('p');           // je sélectionne tous les paragraphes
$('.maClasse');   // je sélectionne les éléments ayant .maClasse pour classe
$('#monId');      // je sélectionne l'élément qui possède l'identifiant #monId
```

Parmi les sélecteurs un peu plus complexes que les classes et les identifiants se trouvent les sélecteurs de descendance. Si on dispose d'un paragraphe, dans lequel il y a un lien absolu, la balise est alors imbriquée dans les balises propres au paragraphe : on dit que le paragraphe est un élément parent, et le lien un élément enfant.

CSS, et par extension, jQuery, permet de manipuler ce concept de descendance grâce à des sélecteurs spéciaux. Le plus connu est sans doute le sélecteur qui fait suivre deux éléments au

minimum :

```
$('p .lien');
/* ici, nous sélectionnons tous les éléments ayant la classe .lien,
et contenus dans un paragraphe qui joue le rôle de parent */
```

Le problème de cette méthode-ci est qu'elle prend en compte tous les éléments possédant la bonne classe et contenus dans un parent. Pour sélectionner que les éléments qui descendent directement du bloc parent, il vous faut utiliser le sélecteur >.

```
$('p > .lien');
/* ici, nous sélectionnons seulement les éléments ayant la classe .lien,
et descendants directement du paragraphe ! */
```

Les sélecteurs + et ~ se chargent de représenter les frères de l'élément :

```
$('.lien + .visite');
/* la sélection s'effectue sur les éléments ayant pour classe .visite,
et qui sont immédiatement précédés d'un élément ayant pour classe .lien */

$('.lien ~ .visite');
/* dans ce cas-là, ce sont tous les éléments .visite,
précédés immédiatement ou non par un élément .lien */
```

Pour les besoins des développeurs, jQuery met en place une sélection d'éléments beaucoup plus poussée que CSS à l'aide de filtres qui ont une syntaxe particulière : deux points suivis du nom du sélecteur.

Parmis les très nombreux sélecteurs proposés par le framework figurent ceux qui permettent de sélectionner un élément suivant sa place sur la page :

- :first, qui sélectionnera le tout premier élément donné
- :eq(index), qui se chargera de retourner l'élément possédant l'indece spécifié
- :last, qui permettra de sélectionner le dernier élément

Exemple :

```
$('p:first');      // sélection du premier paragraphe trouvé
$('p:eq(2)');      // sélection de 3ème paragraphe trouvé (l'indice commence à 0)
$('a:last');       // ici, on sélectionne le dernier lien de la page
```

Un autre type de sélecteurs sont les sélecteurs fonctionnant grâce aux attributs des éléments du DOM[3]. Il est ainsi possible de réaliser de multiples actions sur les balises qui contiennent un attribut donné, tel qu'un identifiant ou une valeur :

```
$('p[id]'); // retourne seulement les paragraphes ayant un identifiant
```

Pour récupérer les éléments ayant une valeur (value) ou un nom (name), il est possible de le dire à jQuery grâce à la syntaxe suivante :

```
$('input[name=pseudo]'); // cible seulement l'élément de formulaire ayant « pseudo » pour nom
```

Pour cibler tous les autres éléments, ceux qui n'ont pas la bonne valeur dans leur attribut, il suffit de rajouter un point d'exclamation (!) devant le signe "égal" (=) :

```
$('input[name!=pseudo]'); // retourne les éléments n'ayant pas « pseudo » pour nom
```

3 Document Object Model

Le sélecteur :not() permet de cibler les éléments qui sont tout sauf ce qui a été indiqué à l'intérieur :

```
$('p:not(.rouge)');           // sélection de tous les paragraphes, sauf ceux ayant .rouge comme classe
$('input:not(.bleu, .vert)'); // on sélectionne tous les éléments de formulaire sauf ceux ayant .bleu et/ou .vert comme classe
```

Pour sauvegarder un objet jQuery, il est possible de rentrer une sélection dans une variable, cette dernière pouvant être utilisée à la place de la fonction principale :

```
var $element = $('monElement');
```

2.2.1. Le cas des formulaires

Les éléments de formulaire peuvent changer de type selon ce que l'on veut en faire. La balise ne change pas et reste la même, on utilisera dans 90% des cas <input />. En changeant l'attribut type de la balise, il est possible de spécifier que l'on veut une case à cocher, un champ de texte, un bouton...

Pour cela, il faut indiquer le type, précédé de deux points :

```
$('input:button'); // on sélectionne un input de type button
```

Et faire ceci pour chaque type d'input :

Type	Code
text (Texte)	input:text
password (Mot de passe)	input:password
file (Fichier)	input:file
checkbox (Case à cocher)	input:checkbox
radio (Bouton radio)	input:radio
button (Bouton normal)	input:button
submit (Bouton d'envoi)	input:submit

Pour connaître l'état des éléments d'un formulaire, il suffit d'utiliser les sélecteurs suivants :

- :checked vérifie qu'une case est cochée
- :disabled cible les éléments désactivés
- :enabled sélectionne les éléments activés

2.2.2. Le sélecteur $(this)

Dans la plupart des langages orientés objet, le mot-clé this représente l'objet courant, celui qui est actuellement traité par une fonction. jQuery permet la sélection de cet objet :

```
$('p').each(function(){
    $(this).html('Hello World !'); // $(this) représente le paragraphe courant
});
```

2.2.3. Performance des sélecteurs

Les sélecteurs ne ciblent pas tous les éléments à la même vitesse.

Ainsi, cibler un élément par un identifiant sera toujours plus rapide que cibler un élément par sa classe : c'est normal, il utilise directement la fonction getElementById() ! On note une rapidité 5

fois supérieure au deuxième sélecteur le plus rapide, qui est le sélecteur par balise.

Vient ensuite le sélecteur par classe, les autres enregistrant une forte baisse de performance à l'utilisation. Ces différences proviennent en fait du parcours du DOM : alors que l'identifiant, étant normalement unique, est ciblé directement par jQuery à cause de sa singularité, les classes vont être traitées une à une, en parcourant tout le document HTML pour les trouver. Pour optimiser au maximum une requête par classe, il suffit de cibler tout d'abord le parent de l'élément avec un identifiant :

```
$('#menu .sections');
// sélection plus rapide que :
$('.sections');
```

Les sélecteurs peuvent s'enchaîner, afin de cibler plus précisément des éléments. Cette astuce possède un lourd inconvénient : elle ralentit considérablement la vitesse de sélection des éléments du DOM. En effet, jQuery va devoir tout d'abord trouver le premier élément, puis va devoir passer au deuxième, et ainsi de suite... Il est alors possible d'augmenter exponentiellement le temps d'exécution du code :

```
$('div p a');
$('#lien');
```

Ces deux ciblages sont identiques, pourtant, la première sélection est environ 15 fois plus lente que la deuxième : plus la chaîne de sélecteurs est courte, plus rapide sera la sélection.

2.3. jQuery et les événements

La fonction de base de jQuery fonctionne autour de l'évènement ready() qui écoute le chargement du document. L'écoute d'un évènement, c'est tout simplement le fait d'attacher une action bien précise à remplir à un élément pour déclencher une fonction, appelée écouteur d'évènement.

En JavaScript, gérer des évènements ne se faisait pas du tout de la même manière d'un navigateur à l'autre. jQuery uniformise le tout !

La syntaxe en elle-même est simple :

```
$("#uneDiv").click(function(){
    // Le code a exécuter !
});
```

En cliquant sur l'élément qui possède l'id #uneDiv ( ), alors on déclenche du code JavaScript.

Parmis les nombreux évènements disponibles, les plus célèbres sont sans aucun doute ceux pouvant être déclenchés par l'utilisateur. Ce sont eux qui permettent une très grande interactivité avec le visiteur, c'est donc eux qui seront les plus utilisés en général.

2.3.1. L'écoute sur la souris

Voici une liste d'évènements adaptés à la souris :

Action	**Fonction**
Clic	click()
Double-clic	dblclick()
Passage de la souris	hover()

Rentrer dans un élément	mouseenter()
Quitter un élément	mouseleave()
Presser un bouton de la souris	mousedown()
Relâcher un bouton de la souris	mouseup()
Scroller (utiliser la roulette)	scroll()

2.3.2. L'écoute sur le clavier

Les évènements associés ne sont pas très nombreux :

- keydown(), qui se lance lorsqu'une touche du clavier est enfoncée ;
- keypress(), qui se lance lorsqu'on maintient une touche enfoncée ;
- keyup(), qui se lance lorsqu'on relâche une touche préalablement enfoncée.

Pour connaître la touche enfoncée par l'utilisateur, il faut employer une fonction anonyme équipée d'un argument représentant le code de la touche, sur lequel on appliquera la propriété keyCode. Cependant, le principal problème de cette propriété est qu'elle ne fonctionne pas sur tous les navigateurs (certains ont des moteurs JavaScript différents). Il faut donc rendre notre code compatible tous navigateurs :

```
$(document).keyup(function(touche){ // on écoute l'évènement keyup()
    // le code est compatible tous navigateurs grâce à ces deux propriétés
    var appui = touche.which || touche.keyCode;

    if ( appui == 13 ){ // si le code de la touche est égal à 13 (Entrée)
        alert('Vous venez d\'appuyer sur la touche Entrée !'); // on affiche une
alerte
    }
});
```

2.3.3. Le cas des formulaires

Les éléments de formulaire possèdent eux aussi leur lot d'évènements associés :

Action	**Fonction**
Focalisation	focus()
Sélection (p.e. dans une liste)	select()
Changement de valeur	change()
Envoi du formulaire	submit()

2.3.4. Les gestionnaires d'évènements

En jQuery, et même plus globalement en JavaScript, on peut faire appel aux gestionnaires d'évènements. Ce sont des fonctions auxquelles on donne un type d'évènement à écouter, ainsi qu'une fonction à exécuter à chaque fois que l'évènement est déclenché.

Remarque : jQuery permet de simuler le déclenchement d'évènements sans attendre que l'utilisateur remplisse une action précise pour lancer du code, grâce à trigger(). Il suffit de donner le type de l'évènement en tant qu'argument :

```
$('p').click(function(){
```

```
    alert('Cliqué !');
});
$('p').trigger('click'); // déclenche l'action au chargement du script
```

Chaque évènement possède son propre groupe d'éléments spécifiques à traiter (la soumission de formulaire ne s'applique pas à tous les cas). Cependant, certains éléments ont un comportement par défaut, défini par le navigateur. Le cas le plus courant est le lien hypertexte : son comportement va être de rediriger l'utilisateur vers l'adresse donnée.

Une méthode en jQuery permet d'annuler tous comportement par défaut. Il s'agit de preventDefault().

```
$('a').click(function(e){
    e.preventDefault(); // annule l'action du lien
});
```

La méthode utilisée pour écouter un événement avec jQuery est la méthode on() en indiquant le type d'évènement, puis la fonction de callback à exécuter :

```
$('button').on('click', function(){
    alert('Ce code fonctionne !');
});
```

- on cible un bouton ;
- on initialise un gestionnaire d'évènement ;
- on écoute le clic de l'utilisateur ;
- et on exécute le code de la fonction de retour.

L'écoute peut se faire sur plusieurs évènements en même temps, sans avoir à créer un gestionnaire pour chacun d'eux, en séparant les deux types par un espace :

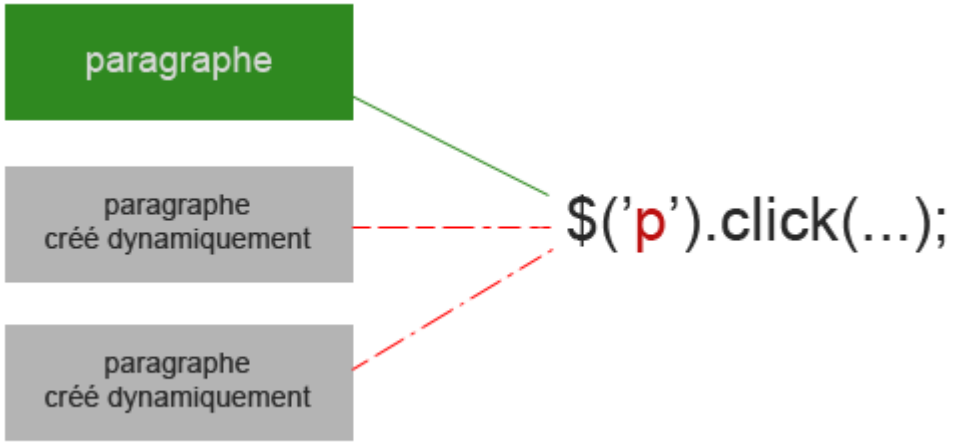
```
$('button').on('click dblclick', function(){
    // on écoute le clic et le double-clic !
    alert('Ce code fonctionne !');
});
```

Il est possible de passer un objet en tant qu'argument à cette méthode, afin d'exécuter des fonctions différentes pour chaque évènement ! Le concept est très simple, il suffit de donner le type d'évènement en tant qu'identifiant, auxquel vous attachez une fonction de retour à chaque fois :

```
$('button').on({
    click : function(){
        alert('Vous avez cliqué !');
    },
    mouseup : function(){
        alert('Vous avez relâché le clic !');
    }
});
```

La délégation d'évènements permet de créer un écouteur d'évènements sur un élément, et de s'adapter automatiquement aux éléments similaires créés plus tard, de façon dynamique !

Exemple : on dispose d'un paragraphe simple, sur lequel on applique un évènement. Si d'autres paragraphes sont créés dans la page, ceux-ci ne seront pas pris en compte par l'écouteur !



La syntaxe pour déléguer un évènement est très simple. Il faut donner trois arguments :

- le type d'évènement ;
- l'élément sur lequel on veut faire une délégation ;
- et la fonction de retour.

```
$('div').on('click', 'p', function(){
    alert('Les paragraphes créés peuvent être cliqués !');
});
```

Pour des raisons de performance, il est conseillé de lancer la méthode sur le parent non dynamique le plus proche des éléments créés à la volée.

S'il est possible d'écouter un évènement avec on(), il est également possible de le supprimer. La méthode off() permet de supprimer tous les gestionnaires et écouteurs mis en place précédemment avec on().

```
$('p').on('click', function(){
    // du code ici
});
$('p').off('click');     // supprime tous les gestionnaires écoutant le clic
$('p').off();            // supprimer tous les gestionnaires de n'importe quel
évènement
```

Les évènements directs, mais aussi délégués, ne seront plus écoutés.

Pour supprimer tous les gestionnaires d'évènements délégués seulement, il faut donner un second argument à la méthode, qui est l'élément créé dynamiquement. La valeur « ** » désigne tous les éléments qui profitent de la délégation d'évènements :

```
$('body').on('click', 'p', function(){
    // du code ici
});
// supprime tous les gestionnaires d'évènements délégués sur les paragraphes
$('body').off('click', 'p');
// supprime tous les gestionnaires d'évènements délégués
$('body').off('click', '**');
```

Les bases du framework se mettent de plus en plus en place dans votre tête, et c'est une très bonne

chance. Bientôt vous serez étonnez de ce qu'il est possible de faire avec votre site web ! La prochaine étape : la manipulation de votre code CSS. Modifier des éléments de votre design dynamiquement, un rêve inavoué ? :-°

2.3.5. Les espaces de noms

Les espaces de noms (namespaces) ont la capacité de désigner un évènement bien précis. Le nommage d'évènement n'étant pas possible avec une fonction, les développeurs de jQuery ont préféré mettre en place ce système. Les espaces de noms ont une syntaxe bien particulière :

event.namespace

- « event » désigne le type d'évènement qui doit subir un espace de nom.
- Le point permet de faire la jonction avec l'espace de nom.
- « namespace » désigne le nom.

Il est possible d'exécuter différentes fonctions à partir d'un même type d'évènement. Il suffit de spécifier l'espace de nom à utiliser, et seule la fonction correspondante sera exécutée.

```
$('button').on('click.nom', function(){
    alert('Premier évènement');
});
$('button').on('click.prenom', function(){
    alert('Second évènement');
});

// exécute le clic, MAIS ne lance que la première alerte !
$('button').trigger('click.nom');
```

Quatre règles doivent être respectées pour utiliser les espaces de noms :

1. il est possible de donner plusieurs espaces de noms à un évènement ;
2. en revanche, il n'est pas possible d'appeler plusieurs espaces de noms d'un seul coup !

   Exemple : on('click.nom.prenom', ...) définira en même temps click.nom et click.prenom ; mais trigger('click.nom.prenom') ne marchera pas.
3. On peut seulement préciser le type d'évènement, sans les espaces de noms, pour tous les déclencher en même temps ;
4. en revanche, il n'est pas possible de ne préciser qu'un espace de nom sans type d'évènement !

   Exemple : trigger('click') déclenchera toutes les fonctions associées aux espaces de noms sur le clic ; mais trigger('.nom') ne marchera pas.

2.4. Manipuler le code CSS

La manipulation du code CSS en jQuery se fait de manière rapide et efficace. La méthode css() permet de modifier le style d'une page et de récupérer la valeur d'une propriété.

Pour lire une propriété, le plus simple est de spécifier le nom de la propriété. Exemple :

```
$('p').css('color'); // ma méthode ira chercher la valeur de la propriété "color" et retournera sa valeur
```

Pour modifier la valeur d'une propriété CSS spécifique, il suffit de passer un deuxième argument à la méthode, qui contiendra la valeur à donner à l'attribut :

```
$('p').css('color', 'red'); // ma méthode modifiera la propriété "color" et la définira à "red"
```

Il est également possible de définir plusieurs propriétés CSS en même temps, grâce à un objet JavaScript que l'on passera en tant qu'argument. Il suffira de séparer les différents attributs par une virgule :

```
$('p').css({
    color : 'red',      // couleur rouge
    width : '300px',    // largeur de 300px
    height : '200px'    // hauteur de 200px
});
```

NB : il ne faut pas oublier que les tirets ne sont pas acceptés dans les identifiants d'un objet. Pour parer à ce problème, il vous faut soit mettre la première lettre de chaque mot en majuscule, sans séparer ceux-ci, soit mettre le nom de la propriété entre guillemets ou apostrophes :

```
$('p').css({
    borderColor : 'red', // bordure rouge
    paddingRight : '30px', // marge intérieure de 30px
    'margin-left' : '10px' // marge extérieure de 10px
});
```

Remarque : la propriété float doit être mise entre guillemets ou apostrophes, car c'est un mot-clé en JavaScript et il risque donc d'y avoir un conflit.

La position des éléments d'une page web peut se définir avec la méthode css() ou par des méthodes en jQuery :

- offset(), qui définit la position d'un élément par rapport au document, c'est donc une position absolue
- position(), qui définit la position d'un élément par rapport à son parent, c'est donc une position relative

Ces méthodes ne fonctionnent qu'avec deux objets, qui sont left (axe horizontal x) et top (axe vertical y). Souvenez-vous que les données ont pour origine le coin en haut à gauche de la page. Ainsi, pour récupérer la valeur de la position d'un élément :

```
$('p').offset().left;   // retourne la valeur "left" de l'élément (position absolue)
$('p').position().top;  // retourne la valeur "top" de l'élément (position relative)
```

Il est possible de spécifier une nouvelle position à un élément, en passant par les méthodes précédentes. Il suffit de passer un objet en tant qu'argument, en donnant les nouvelles valeurs (en pixels) aux identifiants left et top :

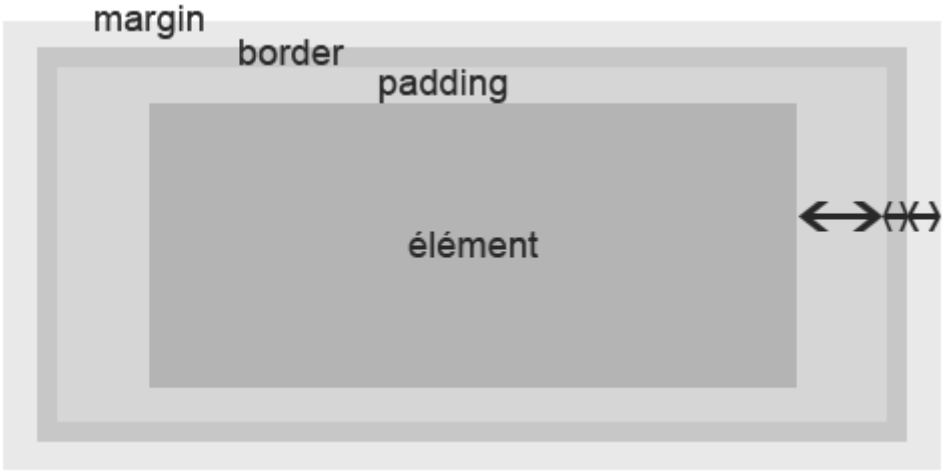
```
$('p').offset({
    left : 30,
    top : 200
});
$('p').position({
    left : 200
});
```

Chaque méthode transforme l'objet donné (l'élément de la page web), et lui définit automatiquement une position absolue ou relative.

De même que pour les positions d'un élément, les dimensions peuvent être gérées directement avec css(). Cependant, plusieurs autres méthodes sont très pratiques pour gérer la hauteur et la largeur d'un bloc :

- height(), qui retourne la hauteur formatée en pixels
- width(), qui fait la même chose avec la largeur
- innerHeight() et innerWidth(), qui prennent en compte les marges intérieures
- outerHeight() et outerWidth(), qui prennent en compte les marges intérieures et la bordure d'un élément

Le box model est un concept à connaître lorsque l'on manipule les dimensions avec jQuery. Conrètement, il faut retenir qu'il y a plusieurs types de largeur et d'hauteur. Les marges intérieures, les bordures et les marges extérieures sont des éléments à prendre en compte, et c'est pourquoi jQuery a créé plusieurs méthodes pour répondre à ce système.



Pour prendre en compte les marges extérieures, il faut passer la valeur true aux méthodes outerHeight() et outerWidth().

```
// retourne la hauteur stricte du paragraphe
$('p').height();
// retourne la largeur (avec marges intérieures) du paragraphe
$('p').innerWidth();
// retourne la largeur (avec marges intérieures + bordures) du paragraphe
$('p').outerWidth();
// retourne la hauteur (avec marges intérieures + bordures + marges extérieures)
du paragraphe
$('p').outerHeight(true);
```

3. DOM et animations

3.1. Les effets

Une des plus puissantes fonctionnalités de jQuery est sans doute l'animation. Le mécanisme est en fait très simple : c'est simplement une propriété CSS qui se déroulera pendant un interval de temps, donné en millisecondes.

Une méthode native de jQuery permet de gérer l'animation des éléments : il s'agit de animate(). Il

suffit de passer un objet à la méthode, contenant les propriétés CSS à animer.

La méthode peut définir :

- duration : le temps de déroulement de l'animation en millisecondes
- easing : la façon d'accélerer de l'animation
- complete : une fonction de callback, qui est l'action appelée lorsque l'animation est terminée

Ces arguments sont facultatifs ; par défaut, ils sont réglés respectivement à normal, swing et null.

Le temps de déroulement d'une animation est un facteur important : en effet, si l'animation est trop rapide, l'utilisateur risque de ne pas la voir. D'un autre côté, si elle est trop lente, il s'ennuiera et trouvera votre site web lourd et sans intérêt. Il va donc vous falloir trouver la bonne durée pour lancer votre animation.

L'argument peut prendre deux types de valeur : une chaîne de caractère (string) ou un nombre entier (int), qui représentera le temps en millisecondes. La chaîne de caractère ne peut être qu'un de ces trois mots :

- slow : équivaut à une durée de 600 millisecondes
- normal : valeur par défaut, qui est égale à 400 millisecondes
- fas : représente une durée de 200 millisecondes seulement

Exemple :

```
$('p').animate({
    width : '150px'
}, 'fast'); // premier exemple avec la valeur fast (200ms)
$('p').animate({
    width : '150px'
}, 1000); // second exemple avec 1000ms (= 1s)
```

L'évolution de l'animation est la façon dont va se dérouler celle-ci au cours du temps. jQuery propose deux façons de dérouler l'animation :

- swing : valeur par défaut, fait aller l'animation de plus en plus vite au cours du temps, et ralentit à la fin
- linear : force l'animation à se dérouler à la même vitesse durant toute l'opération

Pour user de plus de fonctions, il faudra passer par un plugin externe tel que jQuery UI.

```
$('p').animate({
    width : '150px'
}, 'linear'); // l'animation se déroulera de façon linéaire
```

La fonction de retour, plus communément appelée callback, est une action qui se lancera une fois l'animation terminée. Il suffit de donner une fonction anonyme en guise d'argument. La fonction de retour se lancera autant de fois qu'il y a d'éléments animés sur la page.

```
$('p').animate({
    width : '150px'
}, function(){
    alert('Animation terminée !');
});
```

Le premier argument de la méthode est un objet, contenant chaque propriété à animer. Ce qui est intéressant, c'est que les autres arguments peuvent se trouver sous forme d'objet eux aussi. Ainsi, ces deux animations feront exactement la même chose :

```
$('p').animate({
    width : '150px'
}, 1000, 'linear', function(){
    alert('Animation terminée !');
});

// ce code est égal à celui-ci :

$('p').animate({
    width : '150px'
}, {
    duration : 1000,
    easing : 'linear',
    complete : function(){
        alert('Animation terminée !');
    }
});
```

Il existe deux arguments supplémentaires

- step : lance une fonction à chaque étape de l'animation, c'est-à-dire à chaque propriété CSS traitée
- queue : détermine si une animation doit se terminer avant d'en lancer une seconde, et prend un booléen en tant que valeur

Ce second argument est utile si vous avez un chaînage d'animation ou plusieurs animations à la suite ; si vous voulez les lancer toutes en même temps, vous devrez mettre la valeur false : ainsi, les méthodes n'attendront pas la fin de la précédente avant de se lancer. Dans le cas contraire, queue devra être défini à true.

```
$('p')
    .animate({
        width : '150px'
    }, {
        duration : 1000,
        queue : false
    })
    .animate({
        fontSize : '35px'
    }, 1000);
// les deux animations se lanceront en même temps
```

Pour attribuer une accélération différente à chaque propriété CSS animée, il faut donner la nouvelle valeur de la propriété CSS dans un tableau, suivie du type d'accélération :

```
$('p').animate({
    fontSize : ['50px', 'linear'], // cette propriété s'animera de façon
linéaire
    width : '200px' // les autres s'animeront de la façon définie ensuite :
swing
}, 'swing');
```

jQuery permet d'agir sur les barres de défilement :

- scrollTop, qui agit sur la barre de défilement verticale
- scrollLeft, qui agit sur la barre horizontale (si elle existe)

Elles s'animent exactement de la même façon que les autres propriétés.

3.1.1. Les effets natifs

jQuery possède trois effets natifs :

- show()
- hide()
- toggle()

Ces méthodes fonctionnent toutes de la même façon, et prennent le même type d'arguments : duration (durée de déroulement en millisecondes) et complete (fonction de retour). Ils sont facultatifs.

```
$('p').hide('slow'); // cache le paragraphe en 600ms
$('p').show('fast', function(){
    alert('Paragraphe affiché !');
}); // affiche le paragraphe en 200ms, et lance une alerte à la fin de
l'animation
```

La méthode toggle() est un peu particulière, étant donné qu'elle agit sur l'objet jQuery en fonction de son état courant : s'il est caché, elle l'affiche, s'il est affiché, elle le cache. Elle accepte un argument de condition :

```
$('p').toggle(true);     // aura le même rôle que show()
$('p').toggle(false);    // aura le même rôle que hide()
```

jQuery propose six méthodes permettant d'afficher et cacher des éléments defaçon plus esthétique :

- slideDown() déroule l'élément pour l'afficher
- slideUp() enroule l'élément pour le cacher
- slideToggle() enroule ou déroule selon l'état courant de l'élément
- fadeIn() affiche l'élément progressivement
- fadeOut() cache l'élément, en ajustant l'opacité également
- fadeToggle() affiche ou cache l'élément, grâce à l'opacité.

Le framework met également à disposition une fonction permettant de contrôler l'opacité finale de l'élément animé. Il s'agit de fadeTo(), qui prend en argument obligatoire un nombre décimal (float) entre 0 et 1 indiquant l'opacité.

```
$('p').fadeTo('normal', 0.5); // ajuste l'opacité et la fixe à 0.5
```

L'argument de durée de l'animation est défini comme étant facultatif ; néanmoins, vous devez l'indiquer, sans quoi la méthode ne fonctionnera pas.

3.1.2. Le contrôle des effets

jQuery porpose des méthodes qui ont pour rôle de maîtriser les effets : quand les déclencher, les arrêter, les ordonner, et même les influencer !

Le concept de file d'attenteest utilisé à chaque fois qu'un élément d'une page web est animé. Les

méthodes permettant d'animer sont toutes basées dessus : ainsi, si vous avez quatre animations différentes, elles se dérouleront dans l'ordre chronologique, en attendant que la précédente se termine avant de se lancer. C'est un tableau qui va lister les fonctions et les exécuter une à une jusqu'à arriver au terme. En anglais, on l'appelera queue.

Il est possible de manipuler le tableau de fonctions, la queue, grâce à la méthode queue(). Cette fonction est à utiliser sur un élément qui subit une ou plusieurs animations :

```
$('p').fadeOut();
$('p').fadeIn();
$('p').slideUp();
$('p').slideDown();
var fileAttente = $('p').queue('fx');      // la file d'attente est stockée dans une variable
alert(fileAttente.length);                 // renvoie 4
```

NB : on peut spécifier le nom de la file d'attente, en donnant un argument comme dans l'exemple ci-dessus (le nom par défaut est « fx »).

Pour rajouter une fonction dans la file d'attente, il suffit de passer ladite fonction en tant qu'argument.

```
$('p').fadeOut();
$('p').fadeIn();
$('p').slideUp();
$('p').slideDown();
$('p').queue(function(){
    alert('Nouvelle fonction dans la file !'); // alerte s'affichant à la fin de la file
});
```

Pour annuler les animations, il suffit de remplacer le tableau par un nouveau :

```
$('p').queue('fx', []); // fait disparaître le paragraphe, puis vide la file d'attente
```

La méthode dequeue() permet de stopper l'animation en cours de la file d'attente, et de passer à la suivante :

```
$('p')
    .animate({
        fontSize : '+=100px'
    })
    .queue(function(){
        alert('Bonjour !');
        $(this).dequeue();
    })
    .animate({
        fontSize : '-=50px'
    })
    .queue(function(){
        alert('Au revoir !');
        $(this).dequeue();
    });
```

1. la taille de la police augmente de 100 pixels
2. une nouvelle fonction est ajoutée à la file
3. une alerte affiche « Bonjour ! »

4. la méthode dequeue() permet de lancer l'animation suivante
5. la taille de la police baisse de 50 pixels
6. une nouvelle fonction est ajoutée à la file
7. une alerte affiche « Au revoir ! »
8. la méthode dequeue() permet de ne pas entraver les futures animations sur l'élément.

NB : on utilisera le sélecteur $(this) pour lancer la méthode dequeue().

La méthode clearQueue() permet de supprimer toutes les fonctions de la file d'attente qui n'ont pas encore été exécutées :

```
$('p').animate({
    fontSize : '100px'
})
.queue(function(){ // on ajoute une fonction à la file d'attente
    alert('Bonjour !');
})
.clearQueue(); // empêche l'alerte de s'afficher
```

La méthode stop() permet de stopper une animation. On l'utilise le plus souvent pour éviter de lancer le même effet plusieurs fois de suite sans pouvoir l'arrêter.

Arrêter une animation est donc une sécurité, l'assurance qu'elle ne se lancera pas des dizaines et des dizaines de fois sans pouvoir rien y faire. Les deux arguments possibles permettent :

- de stopper une ou toutes les animations suivantes (booléen à indiquer)
- de laisser l'animation courante se dérouler jusqu'à son terme (également un booléen)

Les cas que l'on peut définir sont donc les suivants :

- `$('p:animated').stop();` `// arrête l'animation courante`
- `$('p:animated').stop(true);` `// annule toutes les animations suivantes, dont l'animation courante`
- `$('p:animated').stop(false, true);` `// arrête l'animation courante, mais laisse l'élément aller à son état final`
- `$('p:animated').stop(true, true);` `// annule toutes les animations suivantes, mais laisse l'élément courant aller à son état final`

Le sélecteur :animated cible tous les objets jQuery actuellement animés.

Remarque : pour désactiver toutes les animations de la page, il suffit de passer à true la propriété jQuery.fx.off.

3.2. Manier les attributs

En HTML, les attributs sont contenus dans des balises. La méthode attr() permet de récupérer et de modifier la valeur d'un attribut d'une balise.

Pour lire les valeurs d'un attribut d'une balise, il suffit de passer le nom de l'attribut souhaité en tant qu'argument :

```
var cheminImage = $('img').attr('src'); // affecte la valeur de l'attribut src dans une variable
```

Pour redéfinir un attribut, il suffit de passer la valeur de l'attribut souhaité en deuxième argument. Si l'attribut donné n'existe pas dans la balise, il sera créé automatiquement :

```
$('img').attr('src', 'nouveauChemin/photo.png'); // change l'attribut src en écrasant l'ancienne valeur
$('img').attr('title', 'Nouvelle photo'); // créé l'attribut title dans l'élément s'il n'existe pas
```

NB : l'attribut type des éléments de formulaires ne peut pas être changé, car Internet Explorer l'interdit. La fonction est donc bloquée sur tous les navigateurs.

Par souci de performance, on préféra passer par un objet si l'on a plusieurs attributs à influencer en même temps. Il est possible de donner un objet en tant que paramètre à la méthode attr(), qui contiendra chaque attribut à modifier et leurs nouvelles valeurs respectives :

```
$('img').attr({
    src : 'nouveauChemin/photo.png',
    alt : 'Nouvelle photo',
    title : 'Nouvelle photo'
});
```

Une fonction anonyme peut être déclarée en tant que valeur de l'attribut afin de récupérer :

- l'index de l'élément traité (commence par 0), représenté par un premier argument dans la fonction de retour ;
- la valeur actuelle de l'attribut traité, représenté par un second argument.

```
$('img').attr('alt', function(index, valeur){
    return index + 'ème élément - ' + valeur;
});
```

NB : il faut obligatoirement retourner la nouvelle valeur de l'attribut.

Pour supprimer un attribut, utiliser la méthode removeAttr() en lui passant le nom de l'attribut en argument, et elle se charge de le supprimer définitivement du DOM.

```
$('img').removeAttr('title'); // supprime l'attribut title des images
```

3.2.1. Gérer les classes

La méthode addClass() permet d'ajouter une classe à l'élément spécifié. L'ajout de classe signifie que la fonction prend en compte la ou les classe(s) déjà en place. Ainsi, on peut donner plusieurs classes dynamiquement à un élément. Il est possible d'ajouter plusieurs classes en même temps en les séparant par des espaces.

```
$('.vert').attr('class', 'rouge');          // cet élément aura la classe .rouge
$('.vert').addClass('rouge');               // cet élément aura les classes .vert et .rouge
$('.vert').addClass('rouge bleu jaune');    // cet élément aura les classes .vert, .rouge, .bleu et .jaune
```

La suppression de classe se fait avec la méthode removeClass() qui prend le nom d'une ou de plusieurs classe(s) en paramètre, et les supprime de l'élément sans influencer les autres.

```
$('p').removeClass('vert');                 // supprime la classe .vert de l'élément
$('p').removeClass('vert rouge bleu');      // supprimer les classes .vert, .rouge
```

```
et .bleu
```

Pour vérifier qu'un élément possède bien une classe, il existe la fonction hasClass(). Elle ne peut prendre qu'un seul argument, et qu'une seule classe à la fois et retourne un booléen.

```
if( $('p').hasClass('vert') ){ // si l'élément possède la classe .vert
    alert('Ce paragraphe est vert !'); // on affiche une alerte
}
```

La fonction toggleClass() permet d'ajouter ou de supprimer une classe donnée !

```
$('p').toggleClass('vert'); // ajoute la classe .vert si elle n'existe pas, sinon, la supprime
```

Vous pouvez contrôler l'action de cette méthode grâce à un second argument, sous la forme de booléen : true permettra d'ajouter la classe, alors que false la supprimera.

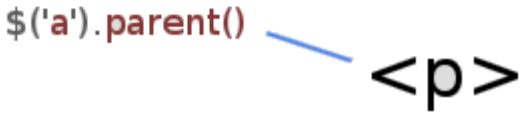
3.3. Parcourir les éléments du DOM

Le parcours du DOM permet de se déplacer aisément sur la page, afin de récupérer certains éléments HTML. Il s'agit d'une interface de programmation, utilisée exclusivement pour les documents XML et HTML, qui se base sur la structure.

La méthode parent() permet d'accéder au bloc parent de l'élément ciblé. Lorsque nous agirons sur notre objet jQuery, ce ne sera donc plus ce dernier qui sera influencé, mais bien le bloc qui l'entoure.

```
$('a').css('color', 'blue'); // rend le lien ciblé seulement de couleur bleue

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



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

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

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

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

```
bloc div
```

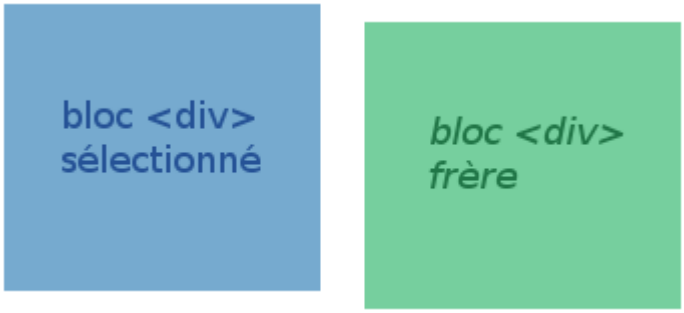
La fonction find() va se charger de trouver tous les enfants d'un élément, quelle que soit leur position par rapport à ce dernier :

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

La dernière méthode usant de ce principe de descendance est parents(). N'oubliez pas le "s", car elle est légèrement différente de la fonction parent(), et ne s'utilise pas tout à fait de la même façon !

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

Remarque : le frère d'un élément, c'est tout simplement la balise présente directement à côté de celui-ci.



Il y a quatre façons de cibler les élément frères :

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

Les filtres et les conditions permettent de cibler des éléments de manière plus précise et concise que les méthodes apparentées à la descendance.

La méthode principale permettant de filtrer des éléments se nomme filter(). Elle permet de supprimer tous les objets ne correspondant pas à la recherche de la sélection. La requête est à spécifier en tant qu'argument, et il est possible de donner plusieurs sélecteurs d'un seul coup en les séparant de virgules :

```
// supprime de la sélection tous les paragraphes n'ayant pas la classe .texte
$('p').filter('.texte');

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

La méthode not() permet de faire la même chose, mais prend en argument les objets à ne pas prendre en compte. Elle va donc parcourir le tableau d'occurrences trouvées et sélectionner seulement celui qui possède l'index indiqué en argument.

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

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

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

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

La fonction is() renvoie un booléen, true si elle déduit que l'argument donné correspond au type de l'élément analysé, false si ce n'est pas le cas :

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

La méthode each() va agir sur chaque objet trouvé, en effectuant une boucle. Cela va vous permettre par exemple d'exécuter des fonctions qui, normalement, s'arrête au premier élément rencontré.

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

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

Si la fonction retourne false, alors la boucle s'arrête sinon elle passe directement à l'élément suivant.

3.4. Manipuler le code HTML

Il existe trois sortes de contenu :

- le contenu textuel, qui correspond à du texte
- le contenu HTML, qui représente le contenu textuel avec les balises structurelles
- le contenu des éléments de formulaire, qui est la valeur des différents champs de texte, de mot de passe, de textarea...

Ces différentes sortes de contenu ne se manipule pas tout à fait de la même manière.

3.4.1. Le contenu textuel

Il n'est vraiment pas difficile de le manipuler car il n'existe qu'une seule fonction pour le faire : text(). Cette méthode permet soit de récupérer le contenu textuel d'un élément s'il existe, soit de le modifier en donnant la nouvelle version en argument.

NB : les balises données ne fonctionneront pas, car les chevrons (< et >) seront convertis automatiquement en entités HTML (respectivement < et >).

Pour récupérer le contenu d'un élément, il suffit d'utiliser la méthode text() : elle ne retournera qu'une chaîne de caractère (string), qui contiendra le contenu textuel sans les balises.

```
$('p').text(); // renvoie le texte contenu à l'intérieur du paragraphe
```

NB : la fonction s'arrête à la première occurrence trouvée.

La définition de contenu avec text() se fait par argument. Cette méthode écrase le contenu actuel pour le remplacer par le nouveau :

```
$('p').text('Nouveau contenu !'); // remplace le contenu actuel du paragraphe par "Nouveau contenu !"
```

3.4.2. Le contenu HTML

La fonction html() fonctionne comme la propriété innerHTML de JavaScript :

```
// renvoie le code HTML contenu dans ce bloc div
$('div').html();

// remplace le code HTML actuel par celui-ci
$('div').html('<p>Nouveau <strong>code</strong> !</p>');
```

Pour la récupération de code HTML, la méthode s'arrête à la première occurrence trouvée.

La méthode prepend() ajoute le contenu HTML passé en argument avant le contenu HTML actuellement en place, tout en restant dans la limite des balises :

```
$('p').prepend('<strong>Texte inséré avant le contenu actuel.</strong> ');
```

donnera :

```
<p>
    <strong>Texte inséré avant le contenu actuel.</strong> Contenu actuel.
</p>
```

Au lieu d'une chaîne de caractère, on peut également passer en argument un objet jQuery qui aura pour effet d'aller récupérer l'élément indiqué, et de l'insérer directement à l'intérieur :

```
$('p').prepend( $('h1') );
```

donnera :

```
<p>
    <h1>Un titre H1</h1>
    Contenu actuel.
</p>
```

Parallèlement à la méthode prepend(), append() ajoute le contenu HTML passé en argument après le contenu HTML actuellement en place. L'insertion s'effectue toujours à l'intérieur des balises de l'élément :

```
$('p').append(' <strong>Texte inséré après le contenu actuel.</strong>');
$('p').append( $('h1') );
```

Remarque : il existe deux méthodes qui permettent de faire exactement la même chose que celles que l'on vient de voir, mais de façon inverse. Il s'agit de prependTo() et appendTo(), qui fonctionnent à l'envers : on va tout d'abord sélectionner l'objet à déplacer, puis l'objet receveur, celui qui va accueillir le contenu souhaité :

```
$('p').append( $('h1') );
// ici, on ajoute le contenu du titre après avoir sélectionné notre élément
$('h1').appendTo( $('p') );
/* alors qu'ici, on sélectionne d'abord le contenu du titre,
et on le déplace après le contenu actuel de notre élément receveur */
$('p').prepend( $('.description') );
// on ajoute le contenu de .description avant le contenu de notre paragraphe
$('.description').prependTo('p');
// on peut spécifier directement l'élément, sans passer par un objet
```

Pour insérer du code HTML à l'extérieur d'un élément, cela implique alors une modification du parent de cet élément, et non de l'élément lui-même : celui-ci n'est pas modifié, mais la balise qui l'entoure voit son contenu changé. De même que pour prepend() et append(), before() et after() permettent respectivement d'ajouter du contenu HTML avant et après l'élément ciblé.

```
$('p').before('<p>Paragraphe précédent</p>');
$('p').after('<p>Paragraphe suivant</p>');
```

donnera :

```
<p>Paragraphe précédent</p>
<p>Paragraphe ciblé</p>
<p>Paragraphe suivant</p>
```

Équivalents de prependTo() et appendTo(), insertBefore() et insertAfter() permettent d'inverser la logique des choses : pour utiliser ces méthodes, vous devez d'abord cibler l'objet à déplacer, puis spécifier l'objet receveur en argument :

```
$('h1').insertBefore('p');
// insère un titre H1 et son contenu avant un paragraphe
$('.description').insertAfter('h1');
// insère un élément .description et son contenu après un titre H1
```

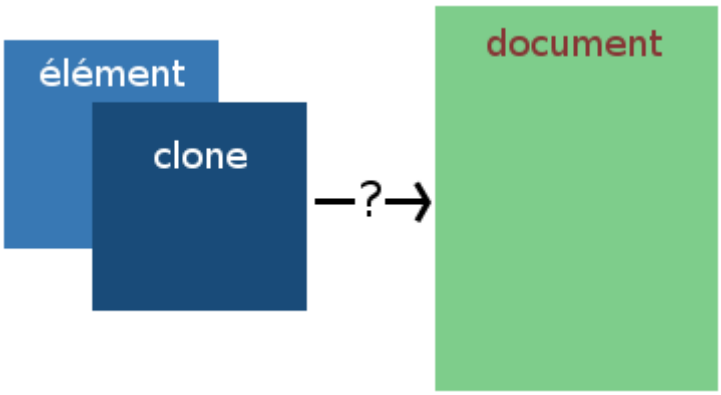
3.4.3. Le contenu des éléments de formulaire

Équivalente à la propriété value en JavaScript, la méthode val() sert à manipuler le contenu des éléments de formulaire. Elle fonctionne exactement de la même manière que text() et n'admet aucune balise structurelle.

```
$('input').val();
// retourne le contenu de l'élément input
$('input').val('Nouvelle valeur !');
// modifie le contenu actuel de l'élément (écrase puis écrit)
```

3.4.4. Manipulation des éléments HTML

Parmi les manipulations possibles existe le clonage d'élément. La méthode clone() a pour effet de dupliquer l'élément ciblé. Cependant, le clone d'élément ne sera pas ajouté au documen.



Il faut lui indiquer où se placer et lui insérer du contenu :

```
var $clone = $('.objet').clone();    // on clone l'élément
$('body').append($clone);            // puis on l'ajoute à la fin du document
// on peut également tout écrire sur une seule ligne grâce au chaînage de
méthode :
$('.objet').clone().appendTo('body');
```

Cette fonction clone l'élément original dans sa totalité : les attributs, le contenu, etc. Seulement, en l'utilisant ainsi, il est impossible de cloner également l'évènement éventuellement associé. Pour faire cela, il suffit de passer true en argument à la méthode.

Pour vider ou supprimer un élément, on utilisera les méthodes empty() et remove() : vider un élément supprimera simplement son contenu, qu'il soit textuel ou structurel, alors que supprimer un élément fera disparaître ce dernier en totalité, c'est-à-dire son contenu, mais aussi lui-même !

```
$('p').empty();
// vide l'élément, seules subsisteront les balises <p></p> et leurs attributs
$('p').remove();
// supprime l'élément, avec son contenu, rien n'en restera
$('div').remove('.suppression');
// supprime les éléments portant la classe .suppression parmi les blocs div
trouvés
```

Les méthodes replaceWith() et replaceAll() permettent de remplacer un élément par un autre. Cela va fonctionner comme un couper-coller : l'élément ciblé va être remplacé par un autre, dans le cas d'un objet, mais celui ci va se déplacer au lieu de se cloner. Il n'est pas nécessaire de spécifier un objet en tant qu'argument, on peut tout aussi bien mettre du nouveau contenu :

```
$('.pomme').replaceWith('<p>Cet élément a été remplacé !</p>');
// on remplace l'élément .pomme par un nouvel élément créé pour l'occasion
$('.pomme').replaceWith( $('.poire') );
// ici, l'élément .pomme est remplacé par l'élément .poire, ce dernier va se
déplacer
$('.poire').replaceAll('.pomme');
// inversement, on ordonne aux éléments .poire de remplacer tous les éléments
.pomme trouvés
```

La méthode wrap() permet d'insérer les balises indiquées autour de l'élément ciblé, il suffit d'écrire une balise HTML normalement, en la fermant directement.

Pour déballer vos éléments, il suffit d'utiliser unwrap() sans argument. Cela aura pour effet de détruire l'élément parent de celui qui est ciblé.

```
$('strong').wrap('<p />');
// entoure les balises <strong></strong> de balises de paragraphe
$('span').wrap('<p class="description" />');
// il est possible de donner des attributs au nouveau parent
$('span').unwrap();
// détruit le parent de tous les span
```

Pour entourer plusieurs éléments frères, utiliser la méthode wrapAll() :

```
$('li').wrapAll('<ul />'); // entoure tous les <li></li> d'un seul parent commun
```

La fonction wrapInner() permet d'entourer le contenu seulement d'un nouveau parent

```
$('p').wrapInner('<strong />'); // entoure le contenu de balises <strong></strong>
```

Donnera :

```
<p>
    <strong>Le ciel est bleu.</strong>
</p>
```

3.4.5. Créer des éléments à la volée

On va fabriquer de nouveaux éléments depuis le code jQuery, sans passer par la case chargement du

DOM. En JavaScript pur, la bonne façon de procéder, ou du moins la plus rapide, était d'utiliser la fonction native createElement(). De notre côté, seule la fonction principale jQuery() va nous servir !

La création de nouveaux éléments en jQuery se fait de manière extrêmement simple : on inscrit une ou plusieurs balises dans l'argument de la fonction principale, et celles-ci sont automatiquement créées. L'élément se trouve vaquant, et il faut utiliser les méthodes de placement.

```
$('<div />').appendTo('body');

// créé un nouveau bloc div, et le place à la fin du corps du document
```

Du contenu peut également être créé en même temps, de même que des attributs peuvent être initialisés (cela est possible grâce à la propriété native innerHTML de JavaScript) :

```
var $lien = $('<a href="http://www.siteduzero.com/">Le Site du Zéro !</a>');
// la variable contiendra l'élément lui-même, ainsi que son contenu
// n'oubliez pas de placer l'élément ensuite
```

La création d'attributs pour ce nouvel élément peut se faire directement lors de la fabrication de celui-ci. Seulement, le code peut vite devenir redondant et illisible. jQuery donne la possibilité de donner un objet en second argument à la méthode principale, contenant la liste de tous les attributs du nouvel élément :

```
$('<div />', {
    id : "bloc", // identifiant de l'élément
    css : { // style css de l'élément
        color : "red",
        fontSize : "30px"
    }
    // etc...
});
```

JavaScript est un langage orienté objet, il utilise donc le concept de classes, que l'on peut définir grâce au mot-clé class. Pour définir la classe d'un élément, structurellement parlant, il faudra mettre le nom de cette propriété entre guillemets ou apostrophes pour ne pas entrer en conflit avec JavaScript.

NB : Internet Explorer ne supporte pas de voir ces propriétés dans un objet, à la création de l'élément. Pour contourner ce problème, il faut passer par la création directe dans la balises, ou par la méthode attr() avant d'insérer l'élément dans le document :

```
$('<input />', {
    type : "text",
    name : "pseudo"
}); // non supporté par IE

$('<input type="text" />').attr({
    name : "pseudo"
}); // bonne façon de procéder
```