Formation jQuery

Table des Matières

Introduction

jQuery est une bibliothèque JavaScript qui jQuery est une bibliothèque JavaScript qui retient l'attention en raison de sa syntaxe retient l'attention en raison de sa syntaxe astucieuse, de ses performances, de sa astucieuse, de ses performances, de sa compacité et de son approche modulaire à compacité et de son approche modulaire à base de plugins.

Premiers pas avec le sélecteur jQuery

On va prendre par exemple une page simple :

```
<html>
<head>
<title>Je débute en jQuery</title>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
</head>

<body>

<div id="header">
   <a id="handle" href="#">Je suis un lien</a><br/>
   <span class="description">La description de cette page qui ne sert à rien</span>
</div>
<div id="content">
   <div class="description">Une autre description</div>
</div>

<script type="text/javascript">
// ... votre code JS ici ...
</script>

</body>
</html>
```

Pour sélectionner le lien, on peut utiliser son **id** avec le sélecteur :

```
1 jQuery('#handle');
```

Cette ligne ne fait... rien, **en apparence** seulement. En fait, cette ligne permet simplement de sélectionner le lien, on va donc pouvoir agir dessus.

Par exemple, nous allons faire disparaitre le lien grâce à la fonction jQuery fadeOut() :

```
1 jQuery('#handle').fadeOut();
```

Comme vous pouvez le voir, le lien disparait au moment ou le javascript est exécuté (c'est à dire au chargement de la page).

De la même manière, on pourrait masquer ce même lien avec un sélecteur différent, qui est **#header a**, et qui signifie "Tous les éléments *a* (liens hypertexte) contenu dans l'élément portant l'id *header*" :

```
1 jQuery('#header a').fadeOut();
```

Les événements

Pour le moment, le code JavaScript (fadeOut) est exécuté au chargement de la page.

Heureusement, les possibilités offertes par javascript sont beaucoup plus vastes, puisqu'on peut "capter" (on dit aussi "écouter" ou "bind") une série d'actions à certains événements.

Mais comme on l'a vu au chapitre 1, javascript permet de "capter" (on dit aussi "écouter" ou "bind") des événements, et d'effectuer une série d'action lorsque ces événements sont déclenchés.

On va prendre un exemple concret : le clic sur le lien **Je suis un lien** est un événement. On pourrait très bien choisir de faire disparaitre la description lorsqu'on clique sur ce lien :

```
1    jQuery('#handle').click(function(){
2        jQuery('#header .description').fadeOut();
3    });
```

À la première ligne, on sélectionne notre lien : **#handle**, et on affecte une fonction anonyme, qui sera exécutée à chaque fois qu'on va cliquer sur ce lien.

À la seconde ligne, on spécifie l'action effectuée par la fonction anonyme, ici : faire disparaitre (**fadeOut**) les éléments de **classe description** contenus dans l'élément qui porte l'**id header**.

Click n'est pas le seul événement, il en existe beaucoup d'autres :

- **mouseover** : déclenché lorsque le pointeur de la souris survol l'élément
- **keypress** : déclenché lorsque l'utilisateur appui sur une touche (très utile pour les champs de formulaire : input, textarea...)
- **change** : lorsque la valeur d'un champ de formulaire est modifiée

Le même exemple avec **mousehover** au lieu de **click** :

```
1    jQuery('#handle').mouseover(function(){
2        jQuery('#header .description').fadeOut();
3    });
```

La liste de tous les événements ainsi que leur documentation est sur le site officiel de jQuery.

C'est beaucoup plus propre d'utiliser jQuery pour binder des actions à des événements, que d'utiliser les attributs javascript natifs (lien), car le code Javascript est totalement séparé du code HTML.

Javascript vient alors comme une surcouche à notre page web, qui l'enrichi par l'ajout de comportements dynamiques et intéractifs.

L'événement DOM Ready

Mais parmi tous ces événements, il y en a un qu'on va utiliser presque tout le temps, c'est l'événement**ready** de l'objet **document** :

```
1    jQuery(document).ready(function($){
2       // ... votre code ici ...
3    });
```

Cet événement ne se déclenche qu'une seule fois pour chaque page, lorsque le navigateur a fini de transformer le code HTML en DOM (cf chapitre 1) on dit alors que le DOM est chargé.

Pour ceux qui connaissent déjà Javascript, c'est un peu l'équivalent de **window.onload**, à quelques différences près :

- *window.onload* est déclenché plus tard que *jQuery(document).ready*, puisqu'il intervient après que le DOM soit chargé **ET** les ressources également (images, feuilles CSS, iframes...)
- *jQuery(document).ready* est plus pratique car il n'écrase pas les instructions précédemment bindées (contrairement à *window.onload*)

C'est quoi l'intérêt ?

Pour comprendre, on va tester cette page web :

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4  <title>Je débute en jQuery</title>
5  <script type="text/javascript"
6  src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
7  </head>
8
9  <body>
10
11 <script type="text/javascript">
12 jQuery('#header .description').fadeOut();
13 </script>
14
15 <div id="header">
```

```
16    <a id="handle" href="#">Je suis un lien</a><br/>
17    <span class="description">La description de cette page qui ne sert à rien</span>
18 </div>
19 <div id="content">
20    <div class="description">Une autre description</div>
21 </div>
22
23 </body>
   </html>
```

Normalement, la description devrait disparaitre au chargement de la page, comme à l'**exemple 1**.

Sauf que cette fois-ci, le code JavaScript est écrit avant le code HTML qui crée l'élément description...

Et comme le navigateur exécute le JavaScript immédiatement (en même temps qu'il le lit), au moment où le Javascript est exécuté l'élément description n'existe pas encore !

Pour régler ce problème, soit on place toujours le javascript après les éléments sur lesquels il agit, soit on encapsule notre code dans une fonction anonyme bindée à l'événement *jQuery(document).ready* :

```
1    <script type="text/javascript">
2    jQuery(document).ready(function($){
3       $('#header .description').fadeOut();
4    });
5    </script>
```

Et maintenant ça marche !

Pourquoi ?
Tout simplement parce que le code qui masque la description est exécuté une fois que le chargement du DOM est terminé.

$: le sélecteur raccourci

Encore une petite précision : dans mon extrait de code ci-dessus, vous voyez que j'ai ajouté un paramètre $ pour ma fonction anonyme, et que j'ai ensuite utilisé $() à la place de jQuery().

En fait, $() est un **alias** de la fonction jQuery() (un autre nom pour la même fonction), on l'utilise plus souvent que jQuery(), tout simplement parce que c'est plus court à écrire !

Le seul problème est que d'autres framework javascript (Mootools nottament) utilisent également $ comme raccourci, du coup si vous utilisez plusieurs frameworks sur la même page, il y aura des conflits.

Heureusement, jQuery propose un mode noConflict(). Dans ce mode, le raccourci $ n'est pas créé pour éviter les problèmes. Le mode noConflict est par exemple activé par défaut dans Wordpress, donc $ ne marche pas et il faut donc se rabattre sur jQuery() à la place, la loose '.

L'astuce que j'ai utilisée me permet d'utiliser $ comme raccourci SEULEMENT à l'intérieur de ma fonction anonyme. Comme ça, même sous Wordpress où le mode noConflict est activé, je pourrais utiliser $ à l'intérieur de mon jQuery(document).ready.

Et il n'y a aucun risque de conflit puisque le tout est encapsulé dans jQuery(), elle est pas belle la vie !