JavaScript - jQuery

INTRODUCTION : Qu'est ce que jQuery ?

jQuery est un Framework JavaScript sous licence libre qui permet de faciliter des fonctionnalités communes de JavaScript. L'utilisation de cette bibliothèque permet de gagner du temps de développement lors de l'interaction sur le code HTML d'une page web, l'AJAX (signifiant "Asynchronous Javascript And Xml" et désignant un nouveau type de conception de pages Web permettant l'actualisation de certaines données d'une page sans procéder au rechargement total de cette page), ou la gestion des évènements.

JavaScript - jQuery

DEROULEMENT DU COURS

JavaScript - jQuery

I.] Installation de jQuery

jQuery est distribué sous la forme d'un fichier Javascript que vous pouvez télécharger sur le site officiel : http://jquery.com/

Quelle version ?

Il en existe 2 versions : DEVELOPPEMENT ou PRODUCTION.

Je vous conseille de prendre la version PRODUCTION, de toute façon on n'ira jamais modifier jQuery.

Note : Celle qui est en production est illisible, mais elle est très légère. Dans les 2 cas, vos programmes fonctionnent de la même façon.

JavaScript - jQuery

II.] Le Sélecteur jQuery

La principale fonctionnalité est le sélecteur. Il permet de sélectionner un ou plusieurs éléments du DOM très facilement, beaucoup plus facilement qu'en javascript natif, puisqu'il utilise la même syntaxe que les sélecteurs CSS (grâce au moteur Sizzle, pour ceux que ça intéresse).

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>
Je vous expliquerai qu'il vaut mieux télécharger le fichier "jquery.min.js" en local, plutot que de faire un appel externe permanent, comme ceci :
<script type="text/javascript" src="js/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>
</body>
</html>
```

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

```
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 pouvons faire disparaitre le lien grâce à la fonction jQuery fadeOut() :

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

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*" :

```
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.

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 :

```
jQuery('#handle').click(function(){

   jQuery('#header .description').fadeOut();

});
```

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**.

JavaScript - jQuery

Click n'est pas le seul événement, il en existe beaucoup d'autres, comme nous avons pu e voir précédemment dans les cours de JavaScript :

- **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 **mouseover** au lieu de **click** :

```
jQuery('#handle').mouseover(function(){

   jQuery('#header .description').fadeOut();

});
```

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** :

```
jQuery(document).ready(function($){

   // ... votre code ici ...

});
```

Cet événement ne se déclenche qu'une seule fois pour chaque page, lorsque le navigateur a fini de transformer le code HTML en DOM, 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 :

JavaScript - jQuery

- *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 :

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Je débute en jQuery</title>
<script type="text/javascript" src="js/jquery.min.js"></script>
</head>

<body>

<script type="text/javascript">
jQuery('#header .description').fadeOut();
</script>

<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>

</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 !

JavaScript - jQuery

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* :

```
<script type="text/javascript">

jQuery(document).ready(function($){

   $('#header .description').fadeOut();

});

</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.

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().

JavaScript - jQuery

III.] Les animations

jQuery propose par défaut une série de fonctions d'animation et de transition qui permettent d'animer des éléments du DOM.

Les effets

Les effets (*effects*) sont des animations prédéfinies, on en a déjà utilisé un dans les chapitres précédents : fadeOut, qui permet de faire disparaitre un élément avec un fondu.

Il existe différents effets que je vous invite à tester :

```
jQuery(document).ready(function($){
  // SHOW/HIDE
  $('#exemple-show span').hide();
  $('#exemple-show a').click(function(e){
    $('#exemple-show span').show();
  });

  $('#exemple-hide a').click(function(e){
    $('#exemple-hide span').hide();
  });

  // FADE
  $('#exemple-fadetoggle a').click(function(e){
    $('#exemple-fadetoggle span').fadeToggle();
  });
});
```

```
<div id="exemple-show">
   <a href="#">show</a>
   <span>Du texte qui apparaît brutalement</span>
</div>

<div id="exemple-hide">
   <a href="#">hide</a>
   <span>Du texte qui disparaît brutalement</span>
</div>

<div id="exemple-fadetoggle">
   <a href="#">fadeToggle</a>
   <span>Du texte qui apparaît/disparaît en fondu à chaque clic</span>
</div>
```

Animations personnalisées

Les effets que vous avez vus ci-dessus sont simples et pratiques, mais on peut encore faire mieux avec jQuery et la fonction **animate()**.

Cette fonction prend en paramètre un ensemble de propriétés CSS, et elle va se charger de faire une animation **linéaire** pour chacune de ces propriétés, entre leur état actuel, et l'état défini dans ses paramètres.

Petit exemple : j'ai une div carré situé en haut à gauche de la page :

```
<input type="button" id="handle" value="Lancer l'animation !" />
<div id="carre"></div>
```

JavaScript - jQuery

Ici j'y ajoute une animation avec les propriétés CSS left et top. JQuery va se charger d'animer chacun de ces paramètres au cours du temps, de sorte que le carré se déplace vers le bas et vers la droite.

```
jQuery(document).ready(function($){
   $('#handle').click(function(e){
      $('#carre').animate({
         left: '50px',
         top: '80px'
      });
   });
});
```

Avec plusieurs étapes

Une fois que vous avez compris le procédé, c'est toujours la même chose. Ci-dessous, j'ai ajouté quelques animations :

- Un boulet bleu est tiré par le carré à la fin de l'animation (première partie).
- Ensuite, ce boulet se déplace vers le haut au survol de la souris, puis revient à sa position initiale quand vous sortez le pointeur de la souris.

```
<div id="carre">
   <div id="ball"></div>
</div>
```

```
jQuery(document).ready(function($){
   $('#handle').click(function(e){
      $('#carre').animate({
         left: '50px',
```

```
        top: '80px'
    }, 600, function(){
        $('#ball').animate({
            opacity:'1',
            left: '400px'
        }, 300, 'swing');  // #1 : Easing
    });
});

$('#ball').mouseover(function(e){
    $(this).animate({
        top: '-=50px'   // #2 : Valeur relative
    });
});

$('#ball').mouseout(function(e){
    $(this).animate({
        top: '+=50px'
    });
});
});
```

C'est quoi $(this) ?

Note : Vous voyez que j'ai utilisé le sélecteur avec le paramètre this ("ça" en Anglais).

JavaScript - jQuery

En jQuery, `$(this)` désigne l'objet courant, on l'utilise souvent dans les fonctions anonymes qu'on bind à un événement.

Dans mon exemple ci-dessus, `$(this)` désigne l'élément portant l'id "ball", et `$(this)` est équivalent à `$('#ball')`.

C'est mieux que de réécrire le sélecteur de l'élément à chaque fois, car ça facilite les mises à jour, et c'est plus propre.

Attention : `$(this)` c'est pas tout à fait la même chose que `this` !
`$(this)` retourne un objet jQuery, alors que `this` est un objet Javascript de base (c'est du natif), ça signifie qu'avec `this` vous ne pourrez pas utiliser des fonctions jQuery comme fadeIn, fadeOut...

Bien sûr, javascript n'a pas été conçu pour créer ce genre d'animation, c'est beaucoup moins pratique de coder les différentes étapes que d'utiliser un système d'images clé comme on en trouver dans After Effects ou Flash.

Mais ça donne un aperçu de ce qu'on peut faire aujourd'hui avec un framework Javascript comme jQuery et un peu de **CSS3**...

Easing

Il existe un paramètre important dans ce système d'animations : Easing.

Il s'agit de la méthode qui contrôle l'animation. Par défaut, c'est le Easing linear qui est utilisé. Dans ce mode, l'animation est linéaire. Si par exemple vous animez le paramètre left de 0px à 100px pendant une durée de 2 secondes, à 1 seconde, left vaudra exactement 50px, et ainsi de suite.

Mais il y a d'autres easings qui ne sont pas du tout linéaires, certains accélèrent au fil du temps, d'autres simulent un rebond...

```
jQuery(document).ready(function($){

   $('#handle').click(function(e){

      $('#carre').slideToggle({ duration: 1000, easing: 'easeOutBounce'});

   });

});
```

JavaScript - jQuery

IV.] Manipuler le DOM

Javascript permet de manipuler le DOM, ça vous le savez puisque vous avez lu le chapitre 1 ^^. Du coup il est possible de modifier une page web sans la recharger : supprimer un paragraphe, ajouter une image, changer tout le contenu d'une div...

jQuery propose toute une série de fonctions qui permettent de manipuler facilement la structure DOM sans se prendre la tête avec du javascript natif dont la syntaxe est beaucoup plus complexe, tout en étant moins flexible !

Modifier les éléments

Chaque élément du DOM est caractérisé par plusieurs choses :

- son **tag** (<a>, , <p>, <img>, <iframe>...)
- ses **attributs** (id, class, style, name, href, src...)
- son **contenu** : du texte ou d'autres éléments
  Par exemple un élément contient du texte, alors qu'une liste <ul> contiens des éléments <li>.
  Certains éléments n'ont pas de contenu, c'est le cas de <img>,
 ou <hr> notamment.

Contenu

jQuery permet de modifier très facilement le contenu d'un élément avec la fonction .html() :

```
<a href="#" id="handle">Ajouter du texte</a>

<div id="mon_element">Texte par défaut</div>

jQuery(document).ready(function($){

   $('#handle').click(function(e){

      var texte = prompt("Entrez le texte à placer dans #mon_element");

      $('#mon_element').html(texte);

   });

});
```

Comme vous pouvez le voir, la fonction .html() efface tout le contenu de l'élément ( ) et ensuite y ajoute ce qu'on lui a passé en paramètre.

JavaScript - jQuery

C'est pratique, mais parfois on peut avoir besoin de simplement ajouter du contenu à un élément (à la fin de celui-ci, une sorte de concaténation quoi).

Pour obtenir cela, on pourrait simplement utiliser l'opérateur "+" qui sert à concaténer des chaines en Javascript :

```
$('#mon_element').html( $('#mon_element').html() + ' salut' );
```

Mais on ne va pas faire ça, parce qu'il y a mieux : .append().
La fonction append (*to append = ajouter*) sert justement à ajouter quelque chose dans l'élément sélectionné (avec le sélecteur $).

Et quelque chose, ça signifie :

- une chaîne de caractères (qui peut être du code HTML), exactement comme avec .html()
- ou un élément DOM !

Ah oui, je ne vous ai pas dit, jQuery permet de créer facilement des éléments avec une syntaxe très simple : jQuery(*<le code html de votre élément>*) :

```
// On créé un nouvel élément (div)
var new_element = jQuery('<div>X</div>');
// On ajoute un peu de style avec la fonction .css()
new_element.css({
   background: 'red',
   width:     '30px',
   height:    '30px'
});
// Pour le moment, notre élément existe, il est simplement stocké dans la mémoire,
// mais il n'est pas encore affiché à l'écran
// On ajoute ce nouvel élément à un élément existant (ici : body, le corps du document HTML)
$('body').append(new_element);
// Et maintenant on peut le voir !
```

JavaScript - jQuery

Attributs

La fonction .attr() permet d'accéder ou de modifier la valeur de n'importe quel attribut d'un élément.

Pour illustrer ça, on va prendre un lien (<a>), un lien c'est bien parce qu'on peut lui refourguer pleins d'attributs !

```
<a href="http://www.finalclap.com" rel="internal" id="handle" target="_blank"
style="font: bolder 20pt Arial" title="Bookmark moi !">Finalclap</a>
jQuery(document).ready(function($){
   $('#handle').click(function(e){
      e.preventDefault();   // On désactive le lien

      $('#output').html('') // On lit quelques attributs de ce lien, et on les affiche
         .append( '<i>attribut target </i>: ' + $(this).attr('target')   + '<br/>' )
         .append( '<i>attribut rel    </i>: ' + $(this).attr('rel')      + '<br/>' )
         .append( '<i>attribut title  </i>: ' + $(this).attr('title')    + '<br/>' )
         .append( '<i>attribut style  </i>: ' + $(this).attr('style')    + '<br/>' )
   });
});
```

Comme vous pouvez le voir, pour lire un attribut avec la fonction .attr(), il suffit de lui passer en paramètre le nom de cet attribut, et la fonction retourne sa valeur.

Et si je veux modifier la valeur d'un attribut ?

C'est presque aussi simple, il suffit de passer un second paramètre à la fonction .attr(), et la valeur de l'attribut sera remplacée par ce second paramètre :

```
// Modification de l'attribut href de notre lien
$('#handle').attr('href', 'http://www.photoshoptuto.com');
```

JavaScript - jQuery

Modifier les structures

Après les éléments, passons maintenant aux structures.

Ce que j'appelle structure c'est simplement des groupes d'éléments HTML, comme par exemple un qui contiendrait plusieurs autres correspondant à des films :

```
<a href="#" id="handle-liste-films">Ajouter</a>

<div id="liste-films">

   <div><span>Pulp fiction</span> - <b>supprimer cette ligne</b></div>

   <div><span>Fight club</span> - <b>supprimer cette ligne</b></div>

   <div><span>Toy Story</span> - <b>supprimer cette ligne</b></div>

   <div>

      <span>Star Wars</span> - <b>supprimer cette ligne</b> - <a class="vider">vider</a>


      <div><span>Star Wars - episode I</span> - <b>supprimer cette ligne</b></div>

      <div><span>Star Wars - episode II</span> - <b>supprimer cette ligne</b></div>

   </div>

</div>
```

Grâce à jQuery, on va pouvoir manipuler la structure de cet ensemble (modifier le DOM en fait) :

- supprimer des éléments
- déplacer des éléments (dans le DOM, pas à l'écran)
- créer de nouveaux éléments

Et pour cela, on utilise des fonctions qui permettent de naviguer dans la structure :

- .parent()
  Retourne l'élément qui contient l'élément sélectionné

- .children([string: sélecteur])
  Retourne la liste des éléments (dans un tableau).

Si vous lui fournissez un paramètre une sélection (CSS, comme avec le sélecteur, ex: *.content p*), seuls les enfants correspondant à cette sélection seront retournés

- .siblings([string: sélecteur])
  Se mot signifie "Frères et soeurs" en Anglais. Son fonctionnement est identique à children, si ce n'est qu'elle ne s'applique pas aux enfants de l'éléments, mais aux autres éléments qui ont le même parent.
  $('#element').siblings('div') est donc équivalent à
  $('#element').parent().children('div').

Pour l'exemple, on va permettre de supprimer chaque ligne ( ) et ajouter de nouvelles lignes :

```
jQuery(document).ready(function($){

  // -------------------------------------------

  // Suppression de ligne lors du clic sur supprimer

  // -------------------------------------------

  $('#liste-films div b').click(function(e){

    $(this).parent().remove();

  });



  // ------------------------

  // Ajout de nouvelles lignes

  // ------------------------

  $('#handle-liste-films').click(function(e){

    e.preventDefault();



    // On demande à l'utilisateur d'entrer du texte

    var texte = prompt("Entrez le texte à placer dans #liste-films");



    // Création de l'élément nouvelle_ligne
```

```
    var nouvelle_ligne = $('<div><span>'+texte+'</span> - <b>supprimer cette ligne</b></div>');


    // On ajoute (bind) la fonction qui supprime cette nouvelle ligne quand on clique sur supprimer
    nouvelle_ligne.children('b').click(function(e){
        nouvelle_ligne.remove();
    });


    // Ajout de ce nouvel élément à notre liste
    $('#liste-films').append(nouvelle_ligne);
  });


  // -----
  // Vider
  // -----
  $('#liste-films div .vider').click(function(e){
    $(this).siblings('div').remove();
  });
});
```

Bien entendu, jQuery propose beaucoup d'autres fonctions qui permettent de manipuler le DOM. Je vous ai montré uniquement les 3 fonctions les plus utiles, celles qui permettent d'arriver à faire ce qu'on veut dans la plupart des cas.

JavaScript - jQuery

V.] jQuery API

http://api.jquery.com/

Attention, cette liste principale, suivie de la liste des incontournables, ne contiennent pas tout. La liste complète se trouve sur le site http://api.jquery.com/

Voici les principaux :

.val()

Get the current value of the first element in the set of matched elements or set the value of every matched element.

.width()

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

.eq()

Reduce the set of matched elements to the one at the specified index.

.each()

Iterate over a jQuery object, executing a function for each matched element.

.attr()

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

.click()

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

.change()

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

.toggle()

Display or hide the matched elements.

.size()

Return the number of elements in the jQuery object.

.show()

Display the matched elements.

.mouseover()

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

.mouseout()

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

.length

The number of elements in the jQuery object.

.select()

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

.prop()

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

.is()

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

.html()

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

ID Selector (“#id”)

Selects a single element with the given id attribute.

.hide()

Hide the matched elements.

.has()

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

.focus()

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

.find()

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

.css()

Get the value of a computed style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

.dblclick()

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Class Selector (“.class”)

Selects all elements with the given class.

Suivi d'autres incontournables :

.add()

Create a new jQuery object with elements added to the set of matched elements.

.addClass()

Adds the specified class(es) to each element in the set of matched elements.

.after()

Insert content, specified by the parameter, after each element in the set of matched elements.

All Selector (“*”)

Selects all elements.

.animate()

Perform a custom animation of a set of CSS properties.

Attribute Equals Selector [name=”value”]

Selects elements that have the specified attribute with a value exactly equal to a certain value.

.before()

Insert content, specified by the parameter, before each element in the set of matched elements.

.bind()

Attach a handler to an event for the elements.

:button Selector

Selects all button elements and elements of type button.

:checkbox Selector

Selects all elements of type checkbox.

:checked Selector

Matches all elements that are checked or selected.

Child Selector (“parent > child”)

Selects all direct child elements specified by “child” of elements specified by “parent”.

.children()

Get the children of each element in the set of matched elements, optionally filtered by a selector.

.clone()

Create a deep copy of the set of matched elements.

.contents()

Get the children of each element in the set of matched elements, including text and comment nodes.

.delay()

Set a timer to delay execution of subsequent items in the queue.

.die()

Remove event handlers previously attached using .live() from the elements.

:disabled Selector

Selects all elements that are disabled.

Element Selector (“element”)

Selects all elements with the given tag name.

.empty()

Remove all child nodes of the set of matched elements from the DOM.

.end()

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

:eq() Selector

Select the element at index n within the matched set.

.error()

Bind an event handler to the “error” JavaScript event.

:even Selector

Selects even elements, zero-indexed. See also odd.

event.currentTarget

The current DOM element within the event bubbling phase.

event.isDefaultPrevented()

Returns whether event.preventDefault() was ever called on this event object.

event.pageX

The mouse position relative to the left edge of the document.

event.pageY

The mouse position relative to the top edge of the document.

event.preventDefault()

If this method is called, the default action of the event will not be triggered.

event.result

The last value returned by an event handler that was triggered by this event, unless the value was undefined.

event.target

The DOM element that initiated the event.

.fadeIn()

Display the matched elements by fading them to opaque.

.fadeOut()

Hide the matched elements by fading them to transparent.

.filter()

Reduce the set of matched elements to those that match the selector or pass the function's test.

.finish()

Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.

.first()

Reduce the set of matched elements to the first in the set.

:first-child Selector

Selects all elements that are the first child of their parent.

.get()

Retrieve the DOM elements matched by the jQuery object.

:has() Selector

Selects elements which contain at least one element that matches the specified selector.

.hasClass()

Determine whether any of the matched elements are assigned the given class.

.height()

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

:hidden Selector

Selects all elements that are hidden.

.hover()

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

:image Selector

Selects all elements of type image.

.innerHeight()

Get the current computed inner height (including padding but not border) for the first element in the set of matched elements or set the inner height of every matched element.

.innerWidth()

Get the current computed inner width (including padding but not border) for the first element in the set of matched elements or set the inner width of every matched element.

:input Selector

Selects all input, textarea, select and button elements.

.insertAfter()

Insert every element in the set of matched elements after the target.

.insertBefore()

Insert every element in the set of matched elements before the target.

jQuery()

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

.jquery

A string containing the jQuery version number.

jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. This property was removed in jQuery 1.9 and is available only through the jQuery.migrate plugin. Please try to use feature detection instead.

jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

.keydown()

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

.keypress()

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

.keyup()

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

:lang() Selector

Selects all elements of the specified language.

.last()

Reduce the set of matched elements to the final one in the set.

:last-child Selector

Selects all elements that are the last child of their parent.

:last Selector

Selects the last matched element.

.load()

Load data from the server and place the returned HTML into the matched element.

.mousedown()

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

.mouseenter()

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

.mouseleave()

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

.mousemove()

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

.mouseup()

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Multiple Attribute Selector [name=”value”][name2=”value2″]

Matches elements that match all of the specified attribute filters.

Multiple Selector (“selector1, selector2, selectorN”)

Selects the combined results of all the specified selectors.

.next()

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

.not()

Remove elements from the set of matched elements.

:not() Selector

Selects all elements that do not match the given selector.

:odd Selector

Selects odd elements, zero-indexed. See also even.

.off()

Remove an event handler.

.offset()

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

.offsetParent()

Get the closest ancestor element that is positioned.

.on()

Attach an event handler function for one or more events to the selected elements.

.outerHeight()

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns a number (without “px”) representation of the value or null if called on an empty set of elements.

.outerWidth()

Get the current computed width for the first element in the set of matched elements, including padding and border.

.parent()

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

.position()

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

.prev()

Get the immediately preceding sibling of each element in the set of matched elements. If a selector is provided, it retrieves the previous sibling only if it matches that selector.

:radio Selector

Selects all elements of type radio.

.ready()

Specify a function to execute when the DOM is fully loaded.

.remove()

Remove the set of matched elements from the DOM.

.removeAttr()

Remove an attribute from each element in the set of matched elements.

.removeClass()

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

.removeProp()

Remove a property for the set of matched elements.

.replaceAll()

Replace each target element with the set of matched elements.

.replaceWith()

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

.resize()

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

.scroll()

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

.scrollLeft()

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

.scrollTop()

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

:selected Selector

Selects all elements that are selected.

.selector

A selector representing selector passed to jQuery(), if any, when creating the original set.

.slice()

Reduce the set of matched elements to a subset specified by a range of indices.

.stop()

Stop the currently-running animation on the matched elements.

.submit()

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

:submit Selector

Selects all elements of type submit.

:target Selector

Selects the target element indicated by the fragment identifier of the document’s URI.

.text()

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

:text Selector

Selects all input elements of type text.

.toArray()

Retrieve all the elements contained in the jQuery set, as an array.

.unbind()

Remove a previously-attached event handler from the elements.

.unload()

Bind an event handler to the “unload” JavaScript event.

:visible Selector

Selects all elements that are visible.

JavaScript - jQuery

PUIS MISE EN PRATIQUE AVEC LE PROJET FIL ROUGE

- en mode test, et non en mode version finale -

- Voir avec le formateur avant de commencer -