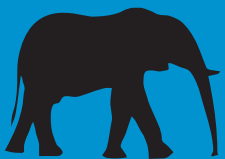
jQuery mémento

2e édition

Éric Sarrion



EYROLLES

Sélectionner des éléments HTML

Simple, efficace et complète, la bibliothèque JavaScript jQuery permet de manipuler les pages HTML au moyen de JavaScript de façon révolutionnaire. Finie la syntaxe compliquée : la fonction `jQuery()`, abrégée en `$()`, remplace toutes les méthodes JavaScript.

Les sélecteurs s'utilisent pour désigner des éléments HTML dans la page.

Sélecteurs simples

Ils consistent à désigner des éléments par leurs noms de balises, voire désigner tous les éléments à l'aide de `*`, qui désigne toutes les balises.

`p`	Sélection de tous les paragraphes de la page.
`img`	Sélection de toutes les images de la page.
`*`	Sélection de tous les éléments HTML de la page.

Sélecteurs d'attributs

On peut spécifier la valeur d'un ou plusieurs attributs pour l'élément. Il peut s'agir d'attributs HTML classiques (`name`, `title`, etc.) ou d'attributs créés dans le code JavaScript (donc avec des noms non standards).

`p[name]`	Sélection de tous les paragraphes ayant un attribut `name`.
`*[name]`	Sélection de toutes les balises ayant un attribut `name`.
`p[name="Sarrion"]`	Sélection de tous les paragraphes ayant un attribut `name="Sarrion"` (valeur exacte). Peut aussi s'écrire `p[name=Sarrion]`. L'attribut `name` doit correspondre exactement à la valeur indiquée. Par exemple, si la balise `<p>` possède l'attribut `name="Sarrion Eric"`, cela ne correspond pas.
`p[name~="Sarrion"]`	Sélection de tous les paragraphes ayant un attribut `name` contenant la valeur `"Sarrion"` délimitée par des espaces. Dans ce cas, si la balise `<p>` possède l'attribut `name="Sarrion Eric"`, cela correspond, car `"Sarrion"` est bien une des valeurs de l'attribut. Par contre, si la balise `<p>` possède l'attribut `name="SarrionEric"`, cela ne correspond pas.
`p[name][job]`	Sélection de tous les paragraphes ayant un attribut `name` et un attribut `job`.
`p[name^="Sar"]`	Sélection de tous les paragraphes ayant un attribut `name` dont la valeur *commence* par `Sar`.
`p[name$="ion"]`	Sélection de tous les paragraphes ayant un attribut `name` dont la valeur *se termine* par `ion`.
`p[name*="arrio"]`	Sélection de tous les paragraphes ayant un attribut `name` dont la valeur *contient* `arrio`.

Sélecteurs de classe

Lorsque le sélecteur d'attributs porte sur l'attribut `class`, il existe une notation spéciale permettant de raccourcir la syntaxe.

`p[class~="rotation"]`	Sélection de tous les paragraphes ayant un attribut `class` contenant la valeur `rotation` délimitée par des espaces. Peut aussi s'écrire `p.rotation`.
`p.rotation`	Identique au sélecteur précédent : sélectionne tous les paragraphes possédant la classe CSS `rotation`.
`p.rotation.translation`	Sélection de tous les paragraphes ayant les classes CSS `rotation` et `translation`. Peut aussi s'écrire `p.translation.rotation`.
`*.rotation`	Sélection de tous les éléments HTML ayant la classe CSS `rotation`. Peut aussi s'écrire `.rotation` (on omet le `*`).

Sélecteurs d'identifiants

Lorsque le sélecteur d'attributs porte sur l'attribut `id`, une notation spéciale permet une fois encore de raccourcir la syntaxe.

`p[id="p1"]`	Sélection de tous les paragraphes ayant un attribut `id` possédant la valeur *p1*. Peut aussi s'écrire `p[id=p1]`.
`p#p1`	Sélection de tous les paragraphes ayant un attribut `id` possédant la valeur *p1* (identique au précédent).
`*#p1`	Sélection de tous les éléments HTML ayant un attribut `id` possédant la valeur *p1*.
`#p1`	Sélection du *premier* élément ayant un attribut `id` possédant la valeur *p1*. Notez la différence avec `*#p1`.

Pseudo-classes

Les pseudo-classes s'utilisent pour préciser un peu plus les éléments indiqués par le sélecteur.

:first-child

Cette pseudo-classe permet de sélectionner les éléments qui sont le premier enfant d'un élément père.

`*:first-child`	Sélection du premier enfant de tous les éléments. Peut aussi s'écrire `:first-child`. Le signe `*` représente ici tous les éléments, tandis que la pseudo-classe `:first-child` spécifie de ne retenir, parmi les éléments représentés par `*`, que ceux qui sont le premier enfant de leur père.
`li:first-child`	Sélection du premier élément dans une liste. Le signe `li` représente ici tous les éléments `<li>`, tandis que la pseudo-classe `:first-child` spécifie de ne retenir, parmi les éléments `<li>`, que ceux qui sont le premier enfant de leur père (donc le premier `<li>` de chaque liste).
`div.rotation: first-child`	Sélection des éléments `<div>` de classe `rotation` qui sont premier enfant de leur père.

:last-child

Fonctionne comme `:first-child`, mais sur le dernier enfant au lieu du premier.

:nth-child(an+b)

La pseudo-classe `:nth-child(an+b)` représente un élément qui est le `(an+b)` *ième* enfant de son élément père (`n` commençant à `0`).

- Si `an+b` vaut `1`, il représente le premier enfant.
- Si `an+b` vaut `2`, il représente le deuxième enfant.
- Etc.

Si `an+b` vaut au final `0`, aucun enfant n'est représenté par cette valeur, donc aucun élément n'est sélectionné dans ce cas.

`li:nth-child(2n+0)`	Sélection des éléments d'indice pair dans une liste (`2`, `4`, `6`...). Peut aussi s'écrire `li:nth-child(2n)` ou `li:nth-child(even)`.
`li:nth-child(2n+1)`	Sélection des éléments d'indice impair dans une liste (`1`, `3`, `5`...). Peut aussi s'écrire `li:nth-child(odd)`.
`li:nth-child(0n+5)`	Sélection du cinquième élément dans une liste. Peut aussi s'écrire `li:nth-child(5)`.
`li:nth-child(-1n+5)`	Sélection des cinq premiers éléments dans une liste. Peut aussi s'écrire `li:nth-child(-n+5)`.
`li:nth-child(1)`	Sélection du premier élément dans une liste. Peut aussi s'écrire `li:first-child`.

:not()

Cette pseudo-classe s'utilise pour exprimer une négation.

`li:not(:last-child)`	Sélection de tous les éléments d'une liste sauf le dernier.

Pseudo-classes

:eq(n)

Cette pseudo-classe permet de récupérer l'élément à l'indice `n` parmi les éléments résultats du sélecteur (`0` désigne le premier élément de la liste de résultats).

`div:eq(0)`	Sélection du premier `<div>` de la page. Peut aussi s'écrire `div:first`.
`div.rotation:eq(1)`	Sélection du deuxième `<div>` ayant la classe `rotation`.

:first

Cette pseudo-classe permet de récupérer le premier élément parmi les éléments résultats du sélecteur. C'est un raccourci de `:eq(0)`.

`div:first`	Sélection du premier `<div>` de la page. Peut aussi s'écrire `div:eq(0)`.
`div.rotation:first`	Sélection du premier `<div>` ayant la classe `rotation`.

:last

Cette pseudo-classe permet de récupérer le dernier élément parmi les éléments résultats du sélecteur.

`div:last`	Sélection du dernier `<div>` de la page.
`div.rotation:last`	Sélection du dernier `<div>` ayant la classe `rotation`.

:contains(text)

Cette pseudo-classe permet de récupérer les éléments qui contiennent le texte indiqué, en tenant compte de la casse (majuscules/minuscules).

`p:contains("Sarrion")`	Sélection de tous les paragraphes contenant `"Sarrion"`. Peut aussi s'écrire `p:contains(Sarrion)`.

:has(selector)

Cette pseudo-classe permet de sélectionner les éléments qui ont, dans leur *descendance*, des éléments correspondant au sélecteur indiqué.

`p:has(b)`	Sélection de tous les paragraphes qui ont du texte en gras (balise `<b>`).
`p:has(b:contains("Sarrion"))`	Sélection de tous les paragraphes contenant le texte `"Sarrion"` mis en gras.

Combinateurs

Un combinateur représente une succession de sélecteurs, qui indique ainsi un ordre de parenté dans les éléments décrits. En voici les différents types.

Combinateur de descendance

Il sert à représenter l'ordre dans lequel doivent se trouver les éléments dans l'arborescence du DOM, indépendamment du niveau hiérarchique. Les sélecteurs sont séparés par un espace.

`div h1`	Sélection de tous les éléments `<h1>` ayant un élément `<div>` dans la parenté.
`div h1.red`	Sélection de tous les éléments `<h1>` de classe CSS `red` ayant un élément `<div>` dans la parenté.
`div * h1`	Sélection de tous les éléments `<h1>` ayant un élément `<div>` dans la parenté, mais pas en fils direct (entre les deux, il y a au moins un élément quelconque, symbolisé par `*`).
`div ul span[selected="true"]`	Sélection de tous les éléments `<span>` ayant l'attribut `selected=true`, ayant un élément `<ul>` puis `<div>` dans la parenté.


Combinateurs

Combinateur filial

Il sert à indiquer qu'un élément est fils direct d'un autre, au contraire du combinateur de descendance qui ne s'occupe que de la parenté. On utilise le signe > (entouré ou non d'espaces) pour indiquer la filiation directe.

`div > h1`	Sélection de tous les éléments `<h1>` fils directs d'un élément `<div>`.

Sélecteurs multiples

Ils sont séparés par une virgule et signifient que les éléments HTML correspondants doivent satisfaire au moins l'un d'eux.

`p, img`	Sélection de tous les paragraphes et de toutes les images.

Principales formes de la fonction $()

Ces formes de la méthode `$()` permettent de construire un objet de classe `jQuery` qui sera associé à des éléments HTML de la page.

`$(`*`selector`*`, context)`	Construit un objet de classe `jQuery` à partir des éléments HTML correspondant au sélecteur indiqué, éventuellement dans la descendance de l'élément DOM `context` (optionnel).
`$(`*`element`*`)`	Construit un objet de classe `jQuery` à partir de l'élément DOM indiqué.
`$(html)`	Construit un objet de classe `jQuery` à partir des éléments inscrits dans le code HTML indiqué.
`$(jQueryObject)`	Retourne l'objet de classe `jQuery` passé en paramètre.

Manipuler la liste des éléments renvoyés par la fonction $()

La fonction `$()` permet d'associer à l'objet de classe jQuery retourné par la fonction, une liste d'éléments HTML sur lesquels s'appliqueront des méthodes de jQuery. Certaines de ces méthodes permettent de supprimer, ajouter ou remplacer des éléments dans cette liste.

Supprimer des éléments dans la liste

`filter (`*`selector`*`)`	Conserve uniquement les éléments qui satisfont au sélecteur indiqué. Les autres éléments sont supprimés de la liste.
`not (`*`selector`*`)`	Conserve uniquement les éléments qui ne satisfont pas au sélecteur indiqué. Les autres éléments sont supprimés de la liste. C'est l'inverse de `filter (`*`selector`*`)`.
`not (`*`element`*`)`	Conserve tous les éléments sauf l'élément DOM indiqué.
`has (`*`selector`*`)`	Conserve uniquement les éléments pour lesquels au moins un de leurs descendants correspond au sélecteur indiqué.
`eq (`*`index`*`)`	Conserve uniquement l'élément positionné à l'index indiqué (`index >= 0`).
`first ()`	Conserve uniquement le premier élément.
`last ()`	Conserve uniquement le dernier élément.

Ajouter des éléments dans la liste

`add (`*`selector`*`,context)`	Ajoute les éléments qui correspondent au sélecteur indiqué, éventuellement situés dans la descendance de l'élément DOM `context` (optionnel).
`add (`*`element`*`)`	Ajoute l'élément DOM indiqué.
`add (html)`	Ajoute les éléments indiqués à partir du code HTML.


Manipuler la liste des éléments renvoyés par la fonction $()

Remplacer les éléments de la liste

`find (selector)`	Récupère parmi les descendants des éléments de la liste, ceux qui correspondent au sélecteur indiqué.
`children (selector)`	Récupère parmi les enfants *directs* des éléments de la liste, ceux qui correspondent au sélecteur. Si le sélecteur n'est pas indiqué, récupère tous les enfants directs (cela correspond à `selector=*`).
`parents (selector)`	Récupère tous les parents des éléments de la liste (jusqu'à l'élément `<html>`) qui correspondent au sélecteur indiqué (optionnel).
`parentsUntil (selector)`	Récupère tous les parents des éléments de la liste jusqu'à tomber (éventuellement) sur un parent qui correspond au sélecteur indiqué. Ce parent n'est pas inclus dans la nouvelle liste.
`parent (selector)`	Récupère le parent *immédiat* de chaque élément de la liste, et ne conserve que ceux qui correspondent au sélecteur éventuellement indiqué (`selector` optionnel).
`closest (selector)`	Récupère le plus proche élément (parent ou déjà dans la liste) de chaque élément de la liste, correspondant au sélecteur indiqué.
`siblings (selector)`	Récupère les frères (précédents et suivants) des éléments de la liste, et ne conserve que ceux qui correspondent au sélecteur éventuellement indiqué (`selector` optionnel).
`prev (selector)`	Récupère le frère précédent de chaque élément de la liste, et ne conserve que ceux qui correspondent au sélecteur éventuellement indiqué (`selector` optionnel).
`prevAll (selector)`	Récupère tous les frères précédents de chaque élément de la liste, et ne conserve que ceux qui correspondent au sélecteur éventuellement indiqué (`selector` optionnel).
`prevUntil (selector)`	Récupère tous les frères précédents de chaque élément de la liste, jusqu'à rencontrer un élément (exclus) qui corresponde au sélecteur éventuellement indiqué (`selector` est optionnel). Cet élément n'est pas inséré dans la nouvelle liste. Si aucun élément précédent ne correspond au sélecteur, tous les frères précédents sont récupérés.
`next (selector)`	Récupère le frère suivant de chaque élément de la liste, en ne conservant que ceux qui correspondent au sélecteur éventuellement indiqué (`selector` est optionnel).
`nextAll (selector)`	Récupère tous les frères suivants de chaque élément de la liste, et ne conserve que ceux qui correspondent au sélecteur éventuellement indiqué (`selector` est optionnel).
`nextUntil (selector)`	Récupère tous les frères suivants de chaque élément de la liste, jusqu'à un élément (exclus) qui correspond au sélecteur éventuellement indiqué (`selector` est optionnel). Cet élément n'est pas inséré dans la nouvelle liste. Si aucun élément suivant ne correspond au sélecteur, tous les frères suivants sont récupérés.

Autres méthodes

`is (selector)`	Retourne `true` si au moins un des éléments de la liste correspond au sélecteur indiqué en paramètres.
`end ()`	La liste revient à son état antérieur à l'application du dernier sélecteur. Si la dernière méthode l'avait modifiée, elle sera remise à l'état précédent. Plusieurs instructions `end ()` peuvent se succéder et remettre la liste dans les états antérieurs.
`andSelf ()`	Ajoute dans la liste les éléments qu'elle possédait avant l'application du dernier sélecteur.

Manipuler le DOM

Une fois que la liste des éléments HTML a été récupérée (voir les méthodes précédentes), des actions peuvent être effectuées sur ces éléments.

Parcours des éléments de la liste

`each (callback)`	Appelle la fonction `callback (index, element)` pour chacun des éléments de la liste. L'index commence à 0. Le paramètre `element` correspond à l'élément DOM de la liste, qui est aussi accessible par la valeur `this` dans le code de la fonction `callback ()`.
`length`	Propriété permettant de connaître le nombre d'éléments de la liste.

Gérer les attributs

`attr (name)`	Récupère la valeur de l'attribut *name*, seulement pour le *premier* élément de la liste.
`attr (name, value)`	Affecte à l'attribut *name* la valeur *value*, pour tous les éléments de la liste.

Gérer les propriétés

`data (key)`	Récupère la valeur de la propriété `key`, seulement pour le *premier* élément de la liste.
`data (key, value)`	Affecte à la propriété `key` la valeur `value`, pour tous les éléments de la liste.

Gérer les classes CSS

`addClass (name)`	Ajoute la classe CSS *name* à tous les éléments de la liste.
`removeClass (name)`	Supprime la classe CSS *name* à tous les éléments de la liste.
`toggleClass (name)`	Alterne (c'est-à-dire ajoute ou supprime) la classe CSS *name* à tous les éléments de la liste.
`hasClass (name)`	Retourne `true` si au moins un élément de la liste possède la classe CSS *name*.
`css (name)`	Retourne une chaîne de caractères correspondant à la valeur de la propriété CSS *name*, uniquement pour le *premier* élément de la liste.
`css (name, value)`	Affecte à la propriété CSS *name* la valeur indiquée (*value*), pour tous les éléments de la liste.

Gérer les dimensions

`height ()`	Retourne la hauteur en pixels du *premier* élément de la liste.
`height (value)`	Affecte une nouvelle hauteur pour les éléments de la liste.
`width ()`	Retourne la largeur en pixels du *premier* élément de la liste.
`width (value)`	Affecte une nouvelle largeur pour les éléments de la liste.

Gérer la position

`offset ()`	Retourne un objet `{ top, left }` contenant les coordonnées du *premier* élément de la liste, relativement aux bords de la page.
`offset (coords)`	Positionne les éléments de la liste aux coordonnées indiquées dans l'objet *coords* `{ left, top }`, relativement aux bords de la page.
`position ()`	Retourne un objet `{ top, left }` contenant les coordonnées du *premier* élément de la liste, relativement à l'élément parent.




Gérer le contenu

`html ()`	Récupère le contenu HTML du *premier* élément de la liste.
`html (html)`	Remplace le contenu des éléments de la liste avec le code HTML indiqué.
`text ()`	Récupère les éléments de type texte inclus dans la descendance de chacun des éléments de la liste, et les retourne dans une chaîne de façon concaténée. Les balises HTML sont donc ignorées.
`text (text)`	Remplace le contenu des éléments de la liste par le texte indiqué. Un élément texte (en texte brut, sans prise en compte des balises HTML) est inséré dans chacun des éléments de la liste.

Gérer les éléments de formulaire

`val ()`	Récupère la valeur saisie ou sélectionnée dans le *premier* élément de la liste. À utiliser pour les champs `<input type="text">`, `<textarea>` ou `<select>`.
`val (value)`	Remplace la valeur des éléments de la liste par la valeur `value` indiquée. À utiliser pour les champs `<input type="text">`, `<textarea>` ou `<select>`.
`attr ("checked")`	Récupère la valeur de l'attribut `"checked"`, seulement pour le *premier* élément de la liste. À utiliser pour déterminer si une case à cocher ou un bouton radio est sélectionné (valeur `"checked"`) ou pas (`undefined`/`false`, selon les versions de jQuery).
`attr ("checked", value)`	Affecte à l'attribut `"checked"` la valeur `value` (`"checked"` ou `undefined`/`false`), pour tous les éléments de la liste. À utiliser pour les cases à cocher et les boutons radio.

Méthodes d'insertion dans le DOM

Ces méthodes permettent d'insérer des éléments HTML relativement aux éléments de la liste associée à l'objet de classe jQuery. L'insertion peut se faire avant, après, en début, en fin, autour des éléments de la liste, ou autour de leur contenu.

Insertion avant

`before (html)`	Insère le code HTML avant chacun des éléments de la liste.
`before (element)`	Insère une copie de l'élément DOM indiqué avant chacun des éléments de la liste. L'élément DOM d'origine est supprimé de son emplacement.
`before (jQueryObject)`	Insère une copie des éléments indiqués dans `jQueryObject`, avant chacun des éléments de la liste. Les éléments correspondant au `jQueryObject` sont supprimés de l'emplacement d'origine.
`insertBefore (selector)`	Insère les éléments de la liste avant chacun des éléments repérés par le sélecteur indiqué.
`insertBefore (element)`	Insère les éléments de la liste avant l'élément DOM indiqué.
`insertBefore (jQueryObject)`	Insère les éléments de la liste avant chacun des éléments présents dans l'objet `jQueryObject`.

EXEMPLE

```
$("div#div1").before ("<div>Avant les paragraphes</div>");
```

```
<div> Avant les paragraphes </div>
<div id="div1">
  <p> Paragraphe 1 </p>
  <p> Paragraphe 2 </p>
  <p> Paragraphe 3 </p>
</div>
```



Méthodes d'insertion dans le DOM

Insertion après

`after (html)`	Insère le code HTML après chacun des éléments de la liste.
`after (element)`	Insère une copie de l'élément DOM indiqué après chacun des éléments de la liste. L'élément DOM d'origine est supprimé de son emplacement.
`after (jQueryObject)`	Insère une copie des éléments indiqués dans `jQueryObject`, après chacun des éléments de la liste. Les éléments correspondant au `jQueryObject` sont supprimés de l'emplacement d'origine.
`insertAfter (selector)`	Insère les éléments de la liste après chacun des éléments repérés par le sélecteur indiqué.
`insertAfter (element)`	Insère les éléments de la liste après l'élément DOM indiqué.
`insertAfter (jQueryObject)`	Insère les éléments de la liste après chacun des éléments présents dans l'objet `jQueryObject`.

EXEMPLE

```
$("div#div1").after ("<div>Après les paragraphes</div>");
```

```
<div id="div1">
  <p> Paragraphe 1 </p>
  <p> Paragraphe 2 </p>
  <p> Paragraphe 3 </p>
</div>
<div> Après les paragraphes </div>
```

Insertion en début

`prepend (html)`	Insère le code HTML au début de chacun des éléments de la liste.
`prepend (element)`	Insère une copie de l'élément DOM indiqué au début de chacun des éléments de la liste. L'élément DOM d'origine est supprimé de son emplacement.
`prepend (jQueryObject)`	Insère une copie des éléments indiqués dans `jQueryObject`, au début de chacun des éléments de la liste. Les éléments correspondant au `jQueryObject` sont supprimés de l'emplacement d'origine.
`prependTo (selector)`	Insère les éléments de la liste au début de chacun des éléments repérés par le sélecteur indiqué.
`prependTo (element)`	Insère les éléments de la liste au début de l'élément DOM indiqué.
`prependTo (jQueryObject)`	Insère les éléments de la liste au début de chacun des éléments présents dans l'objet `jQueryObject`.

EXEMPLE

```
$("div#div1").prepend ("<p>Paragraphe 0</p>");
```

```
<div id="div1">
  <p> Paragraphe 0 </p>
  <p> Paragraphe 1 </p>
  <p> Paragraphe 2 </p>
  <p> Paragraphe 3 </p>
</div>
```

Insertion en fin

`append (html)`	Insère le code HTML à la fin de chacun des éléments de la liste.
`append (element)`	Insère une copie de l'élément DOM indiqué à la fin de chacun des éléments de la liste. L'élément DOM d'origine est supprimé de son emplacement.
`append (jQueryObject)`	Insère une copie des éléments indiqués dans `jQueryObject`, à la fin de chacun des éléments de la liste. Les éléments correspondant au `jQueryObject` sont supprimés de l'emplacement d'origine.
`appendTo (selector)`	Insère les éléments de la liste à la fin de chacun des éléments repérés par le sélecteur indiqué.
`appendTo (element)`	Insère les éléments de la liste à la fin de l'élément DOM indiqué.

Méthodes d'insertion dans le DOM

appendTo (jQueryObject) — Insère les éléments de la liste à la fin de chacun des éléments présents dans l'objet `jQueryObject`.

EXEMPLE
```
$("div#div1").append ("<p>Paragraphe 4</p>");
```
```
<div id="div1">
 <p> Paragraphe 1 </p>
 <p> Paragraphe 2 </p>
 <p> Paragraphe 3 </p>
 <p> Paragraphe 4 </p>
</div>
```

Insertion autour

wrap (html) — Entoure chacun des éléments de la liste par l'élément HTML créé avec le code indiqué. L'élément HTML dans le paramètre `html` est créé dynamiquement par jQuery pour chacun des éléments de la liste et en devient le père.

wrap (*element*) — Entoure chacun des éléments de la liste par l'élément DOM indiqué. L'élément DOM est dupliqué dynamiquement par jQuery et devient le père de chacun des éléments de la liste.

wrap (jQueryObject) — Entoure chacun des éléments de la liste par le premier élément représenté dans la liste contenue dans `jQueryObject`. Le *premier* élément de `jQueryObject` est dupliqué dynamiquement par jQuery et devient le père de chacun des éléments de la liste.

wrap (*selector*) — Entoure chacun des éléments de la liste par le premier élément résultat du sélecteur indiqué. Le *premier* élément résultat du sélecteur est dupliqué dynamiquement par jQuery et devient le père de chacun des éléments de la liste.

EXEMPLE
```
$("div#div1").wrap ("<div id="wrap">");
```
```
<div id="wrap">
 <div id="div1">
  <p> Paragraphe 1 </p>
  <p> Paragraphe 2 </p>
  <p> Paragraphe 3 </p>
 </div>
</div>
```

Insertion autour du contenu

wrapInner (html) — Entoure le contenu de chacun des éléments de la liste par l'élément HTML créé avec le code indiqué. L'élément HTML dans le paramètre `html` est créé dynamiquement par jQuery dans chacun des éléments de la liste et devient le père du contenu.

wrapInner (*element*) — Entoure le contenu de chacun des éléments de la liste par l'élément DOM indiqué. L'élément DOM est dupliqué dynamiquement par jQuery et devient le père du contenu de chacun des éléments de la liste.

wrapInner (jQueryObject) — Entoure le contenu de chacun des éléments de la liste par le *premier* élément représenté dans la liste contenue dans `jQueryObject`. Le premier élément de `jQueryObject` est dupliqué dynamiquement par jQuery et devient le père du contenu de chacun des éléments de la liste.

wrapInner (*selector*) — Entoure le contenu de chacun des éléments de la liste par le *premier* élément résultat du sélecteur indiqué. Le premier élément résultat du sélecteur est dupliqué dynamiquement par jQuery et devient le père du contenu de chacun des éléments de la liste.

EXEMPLE
```
$("div#div1").wrapInner ("<div id="wrapinner">");
```
```
<div id="div1">
 <div id="wrapinner">
  <p> Paragraphe 1 </p>
  <p> Paragraphe 2 </p>
  <p> Paragraphe 3 </p>
 </div>
</div>
```

Autres méthodes de gestion du DOM

`clone (withEvents)`	Duplique les éléments de la liste et retourne un nouvel objet de classe jQuery les référençant. Si le paramètre `withEvents` est positionné à `true` (la valeur est `false` par défaut), les gestionnaires d'événements sont également dupliqués. Les éléments dupliqués devront ensuite être insérés dans l'arborescence.
`empty ()`	Supprime le contenu de chacun des éléments de la liste qui restent présents mais vides.
`remove (selector)`	Supprime de l'arborescence du DOM les éléments de la liste qui satisfont l'éventuel sélecteur indiqué.
`unwrap ()`	Effectue l'opération inverse de `wrap ()`. Les éléments de la liste sont remontés d'un niveau dans l'arborescence du DOM, car leurs parents sont supprimés. Les éléments de la liste deviennent donc rattachés au parent de leur parent, tandis que le parent est supprimé.

Gestion des événements

jQuery permet de gérer les événements survenant dans la page HTML, en uniformisant la gestion de ceux-ci indépendamment du navigateur utilisé.

Principales méthodes

`bind (eventName, callback)`	Définit une fonction de traitement de la forme `callback (event)` pour l'événement *eventName* indiqué (`"click"`, `"mousemove"`, etc.), dans laquelle `event` est un objet correspondant à l'événement qui se produit (voir plus loin).
`unbind ()`	Supprime les gestionnaires d'événements de tous les éléments de la liste.
`unbind (eventName)`	Supprime les gestionnaires de l'événement *eventName* de tous les éléments de la liste.
`unbind (eventName, refCallback)`	Supprime le gestionnaire `refCallback` associé à l'événement *eventName* de tous les éléments de la liste.
`on (eventName, callback)`	Identique à `bind (eventName, callback)`.
`on (eventName, selector, callback)`	Attache un gestionnaire d'événements à tous les éléments correspondant au *selector* indiqué, et situés dans la descendance des éléments de la liste. L'intérêt est que ces éléments peuvent ne pas être encore présents dans l'arborescence du DOM (on simule l'ancienne méthode `live ()` qui a été supprimée de jQuery à partir de la version 1.8). Par exemple : `$(document).on ("click", "p", callback)` déclenchera la méthode `callback` lors d'un clic sur tous les paragraphes situés dans la descendance de l'élément `document`, que ces paragraphes soient déjà présents dans la page ou insérés ultérieurement.
`off ()`	Supprime les gestionnaires d'événements de tous les éléments de la liste.
`off (eventName)`	Supprime les gestionnaires de l'événement *eventName* de tous les éléments de la liste.
`off (eventName, refCallback)`	Supprime le gestionnaire `refCallback` associé à l'événement *eventName* de tous les éléments de la liste.
`off (eventName, selector)`	Supprime les gestionnaires précédemment ajoutés par `on (eventName, selector, callback)`.
`off (eventName, selector, refCallback)`	Supprime les gestionnaires associés à `refCallback` et précédemment ajoutés par `on (eventName, selector, callback)`.
`trigger (eventName)`	Déclenche l'événement `eventName` sur les éléments de la liste.

Propriétés du paramètre event

Le paramètre `event` passé en arguments des fonctions de *callback* utilisées dans les méthodes `bind (eventName, callback)` ou `on (eventName, callback)` contient des propriétés permettant d'avoir des informations sur cet événement.

Gestion globale de l'événement

`type`	Correspond au nom de l'événement (`"click"`, `"mousemove"`, etc.).
`timeStamp`	Permet de dater l'événement, en indiquant le nombre de millisecondes écoulées depuis le 1er janvier 1970.
`handler`	Référence à la fonction de traitement de l'événement (à savoir la fonction `callback (event)`).
`originalEvent`	Permet d'accéder à l'événement d'origine fourni par le navigateur. Cela permet d'accéder à de nouvelles informations qui ne sont pas nécessairement stockées directement dans l'objet `event`. Par exemple, pour une application mobile utilisant la fonctionnalité de multi-touch, on utilisera `event.originalEvent.touches` qui correspond à un tableau des doigts touchant l'écran du smartphone ou de la tablette. Pour connaître le nombre de doigts touchant simultanément l'écran, utiliser `event.originalEvent.touches.length` (la valeur est `1` pour un doigt, `2` pour deux doigts, etc.).

Éléments DOM impliqués dans l'événement

`target`	Correspond à l'élément DOM sur lequel s'est produit l'événement, qui se trouve au plus profond niveau d'imbrication.
`currentTarget`	Correspond à l'élément DOM qui est à l'écoute de cet événement. Il correspond à `this` dans le code de la fonction `callback (event)`. Il est égal à `target` ou est situé dans son ascendance.

Gestion des touches

`shiftKey`	Indique si la touche *Shift* est enfoncée (`true` si oui, `false` sinon).
`ctrlKey`	Indique si la touche *Ctrl* est enfoncée (`true` si oui, `false` sinon).
`altKey`	Indique si la touche *Alt* est enfoncée (`true` si oui, `false` sinon).
`keyCode`	Indique le code de la touche du clavier qui a été utilisée.

Gestion de la souris

`which`	Indique quel bouton de la souris a participé à l'événement (`1` : gauche, `2` : milieu, `3` : droit).
`screenX et screenY`	Coordonnées de la souris, relativement aux bords haut et gauche de l'écran.
`pageX et pageY`	Coordonnées de la souris, relativement aux bords haut et gauche de la page.
`clientX et clientY`	Coordonnées de la souris, relativement aux bords haut et gauche de la zone affichée à l'écran. Ce sont les mêmes que `pageX` et `pageY` dans le cas où il n'y a eu aucun défilement horizontal ou vertical de la page.

Méthodes du paramètre event

`stopPropagation ()`	Empêche la propagation de l'événement aux niveaux supérieurs de l'arborescence.
`preventDefault ()`	Empêche le traitement par défaut de l'événement. Par exemple, si l'événement est un clic sur un lien, l'instruction `event.preventDefault ()` empêchera l'affichage de la nouvelle page HTML correspondant à l'URL du lien.

Requêtes Ajax

jQuery permet d'émettre des requêtes Ajax vers un serveur, en effectuant l'appel Ajax via la méthode `$.ajax (options)`, et ceci indépendamment du navigateur utilisé.

Options de la méthode $.ajax (options)

`url`	Correspond à l'URL du programme sur le serveur qui traitera la requête. Cette option est obligatoire, sinon aucun traitement sur le serveur ne peut avoir lieu.
`data`	Objet ou chaîne de caractères qui sera transmis au serveur. Si on utilise une chaîne de caractères, elle doit être de la forme `name1=value1&name2=value2...`, chaque `name` étant le nom d'un paramètre et `value` la valeur correspondante, encodée en UTF-8. Si on utilise un objet, jQuery encode lui-même en UTF-8 chacune des valeurs et transmet au serveur une chaîne de la forme `name1=value1&name2=value2...`
`type`	Mode de transmission des paramètres indiqués dans l'option `data` précédente. Deux valeurs sont possibles : `"get"` (valeur par défaut) ou `"post"`.
`complete`	Méthode de la forme `complete (xhr, textStatus)` appelée à la fin de la requête Ajax, que celle-ci ait réussie ou non. Le paramètre `textStatus` contient `"success"` si la requête a réussi, sinon un texte d'erreur, comme `"error"`, `"timeout"` ou `"parseerror"`. Dans le cas où la requête a réussi, le paramètre `xhr` donne accès à l'objet `XMLHttpRequest` contenant la réponse du serveur (dans `xhr.responseText`).
`username`	Nom d'utilisateur à indiquer si le serveur nécessite une autorisation.
`password`	Mot de passe associé à l'option `username`.

Voici un exemple d'utilisation d'Ajax. On saisit un nom dans un champ de saisie, puis on envoie le nom saisi vers le serveur qui le transforme en majuscules et le retourne vers le navigateur, qui affiche alors le nom transformé.
La page HTML contenant le code JavaScript s'écrit :

```
<script src=jquery.js></script>

<body>
Entrez votre nom :
<input type=text id=nom><br>
<input type=button id=valider value=Valider><br><br>
</body>

<script>
$("#valider").bind("click", function() {
 var valnom = $("#nom").val();
 $.ajax({
  url : "action.php",
  data : { nom : valnom },
  complete : function(xhr, textStatus) {
   if (textStatus != "success") return;
   $("body").append("<p> Le nom saisi est : " + xhr.responseText + "</p>");
  }
 });
});
</script>
```

Le code de la partie serveur se trouve dans le fichier `action.php` :

```
<?
$nom = $_REQUEST["nom"];
echo strtoupper($nom); // retourner au navigateur le nom saisi, en majuscules
?>
```

Effets visuels standards

jQuery autorise les effets visuels. Ils sont de deux types : standards et personnalisés. Les effets visuels standards de jQuery manipulent les propriétés CSS `height` et `opacity`, soit de façon séparée, soit de façon simultanée. On les utilise avec les méthodes suivantes.

Effets visuels standards

`slideUp (duration)`	Fait remonter les éléments de la liste sur l'écran en les *cachant* progressivement par le bas, pendant la durée `duration` (en millisecondes). Par défaut 400 ms.
`slideDown (duration)`	Fait descendre les éléments de la liste sur l'écran en les *affichant* progressivement par le bas, pendant la durée `duration` (en millisecondes). Par défaut 400 ms.
`slideToggle (duration)`	Alterne `slideUp ()` ou `slideDown ()` sur chaque élément de la liste selon que l'élément est visible ou caché.
`fadeIn (duration)`	Fait apparaître les éléments de la liste en amenant progressivement leur opacité à `1`, pendant la durée `duration` (en millisecondes). Par défaut 400 ms.
`fadeOut (duration)`	Fait disparaître les éléments de la liste en amenant progressivement leur opacité à `0`, pendant la durée `duration` (en millisecondes). Par défaut 400 ms.
`fadeToggle (duration)`	Alterne `fadeOut ()` ou `fadeIn ()` sur chaque élément de la liste selon que l'élément est visible ou caché.
`show (duration)`	Fait apparaître les éléments de la liste progressivement, pendant la durée `duration` (en millisecondes). Par défaut 0 ms, provoquant un affichage instantané.
`hide (duration)`	Fait disparaître les éléments de la liste progressivement, pendant la durée `duration` (en millisecondes). Par défaut 0 ms, les cachant instantanément.
`toggle (duration)`	Alterne `hide ()` ou `show ()` sur chaque élément de la liste selon que l'élément est visible ou caché.

Effets visuels personnalisés

On peut créer ses propres effets en utilisant la méthode `animate (objStyle, options)` qui produit un effet sur les éléments de la liste.

`animate (objStyle, options)`	Produit un effet visuel personnalisé sur les éléments de la liste. Le paramètre `objStyle` représente un objet contenant le nouveau style des éléments à la fin de l'effet. Cet objet est de la forme `{ prop1 : value1, prop2: value2, ... }`, chaque `prop` étant une propriété CSS que l'on désire voir évoluer jusqu'à la valeur indiquée par `value`. Le paramètre `options` représente un objet contenant les options de l'effet, décrites ci-après.

Options de la méthode `animate (objStyle, options)`

`duration`	Indique le nombre de millisecondes de l'effet. Une valeur de `0` amène l'élément directement dans le nouveau style, sans progressivité.
`complete`	Méthode `complete ()` appelée pour chacun des éléments de la liste lorsque l'effet est terminé sur celui-ci. La valeur `this` dans la fonction représente l'élément DOM pour lequel l'effet est terminé.
`step`	Méthode `step ()` appelée pour chacun des éléments de la liste, après chaque mise à jour des propriétés CSS du style de l'élément. Elle peut donc être appelée plusieurs centaines de fois pour une même animation, à la différence de `options.complete` appelée une seule fois (pour chaque élément de la liste) à la fin de l'animation.
`queue`	Par défaut, deux effets ne peuvent pas s'exécuter simultanément sur le même élément, car le second effet est mis dans une file d'attente (il sera exécuté lorsque l'effet précédent sur cet élément sera terminé). Si `options.queue` est positionnée à `false`, l'effet n'est pas mis dans la file d'attente et peut donc s'exécuter immédiatement.

Code éditeur : G13964
ISBN : 978-2-212-13964-8
Conception : Nord Compo