Programmation web
JavaScript - Applications 1 - JavaScript, côté client

Vincent Padovani, PPS, IRIF

Une page web peut être associée à un ou plusieurs scripts qui seront chargés et exécutés par le navigateur dans lequel elle est affichée. Ce chapitre explique la manière dont un script peut modifier dynamiquement l'aspect d'une page, ou encore réagir aux événements se produisant sur celle-ci.

1 Association d'un script

Un script peut être associé à une page en écrivant de manière directe ses instructions entre un ou plusieurs couples `<script><script/>`, qui seront considérés comme formant un unique script :

```
<script>
   console.log("Hello World!");
<script/>
```

Il est cependant plus courant et plus utile d'isoler le script dans un fichier distinct de celui de la page, et d'inclure son contenu dans la partie `head` de celle-ci par un élément de la forme suivante :

```
<script src="https://www.exemple.com/monScript.js"/>
```

2 Le DOM

Le DOM (Document Object Model) est une représentation arborescente du contenu d'une page web, construite par le navigateur après chargement de cette page. Chaque noeud de cet arbre représente un élément HTML, les liens de parenté entre les noeuds étant calqués sur les imbrications d'éléments. Les feuilles de cet arbre représentent les éléments textuels purs (ceux ne contenant aucune balise) ainsi que les valeurs d'attributs des éléments.

2.1 La propriété `document`

Dans un script asssocié à une page, le DOM est représenté par l'objet `document`. La structure arborescente dont cet objet représente la racine peut être examinée de manière directe dans la console du navigateur. Considérons par exemple la page web minimaliste suivante :

```html
<html lang="fr">
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <h1>Ma page web</h1>
    <p id="p1">Un paragraphe</p>
    <p id="p2">Un <em>autre</em> paragraphe</p>
  </body>
</html>
```

Après chargement de la page et construction du DOM, la console réagit de la manière suivante – noter que dans l'affichage de la propriété `children`, les "valeurs" `html`, `head`, `body`, `h1`, `p` ... ne sont pas de vraies références, mais simplement un affichage particulier par Firefox du type de balise de l'élément ayant engendré un noeud donné :

```
document;             //=> HTMLDocument file:///home/me/dom.html
document.children;    //=> HTMLCollection { 0: html, length: 1 }

// descente dans le fils 0 (élément <html>...</html>):
document.children[0].toString();
// => "[object HTMLHtmlElement]"
// accès à la valeur de l'attribut lang via une propriété de même nom :
document.children[0].lang;
// => "fr"
document.children[0].children;
// => HTMLCollection { 0: head, 1: body, length: 2 }

// descente dans le fils 1 (élément <body>...</body>) :
document.children[0].children[1].toString();
// => "[object HTMLBodyElement]"
document.children[0].children[1].children;
// => HTMLCollection { 0: h1, 1: p, 2: p, length: 3 }

// descente dans le fils 1 (premier élément en <p></p>) :
document.children[0].children[1].children[2].toString();
// => "[object HTMLParagraphElement]"
```

2.2 Déplacements dans le DOM

Chaque nœud du DOM est muni de propriétés permettant de se déplacer vers les nœuds voisins. Les noms de ces propriétés désignent clairement quels sont les nœuds désignés par leur valeur : `parentNode` (`null` pour la racine), `childNodes` (renvoyant une liste en `NodeList`, accessible comme un tableau), `firstChild`, `lastChild`, `previousSibling` et `nextSibling` (valant `null` en cas d'absence du noeud spécifié).

La fonction ci-dessous, par exemple, permet d'appliquer une fonction `f` à chaque nœud du DOM ou un à un sous-arbre du DOM :

```
function mapNodes(e, f) {
  f(e);
  for (let c of e.childNodes) {
    mapNodes(c, f);
  }
}
console.log(mapNodes(document, e => console.log(e)));
// affiche la représentation par Firefox de chaque noeud du DOM
```

2.3 Sélections dans le DOM via `css`

L'accès aux nœuds du DOM peut aussi se faire à l'aide d'invocations de méthodes de la forme suivante – la chaîne argument est dans chaque cas un sélecteur `css`, et les nœuds sélectionnés seront tous ceux représentant les éléments HTML de la page qui seraient sélectionnés par ce sélecteur dans une feuille de style :

```
// sélectionner le premier noeud du DOM spécifié par le sélecteur
document.querySelector(" ... ");
// sélectionner tous les noeuds du DOM spécifié,
// sous la forme d'une NodeList accessible comme un tableau
document.querySelectorAll(" ... ");
```

Rappelons qu'un sélecteur `css` décrit un ensemble d'élément en spécifiant leur balise, la valeur de leur attribut `id` ou encore une des mots de leur attribut `class` :

```
*                   // tout élément
p                   // tout élément en <p>
#p1                 // l'élément d'id ="p1"
```

On peut restreindre une sélection (y compris la sélection vide, équivalente dans ce cas à `*`) en ne gardant que ses éléments dont l'attribut `class` contient un certain mot :

```
.menu               // tout élément dont l'attribut
                    // class contient le mot "menu"
div.menu            // tout élément en <div> dont l'attribut
                    // class contient le mot "menu"
div.menu.gauche     // tout élément en <div> dont l'attribut
                    // class contient les mots "menu" et "gauche"
```

On peut restreindre une sélection en ne conservant que les éléments possédant un certain attribut, ou une valeur d'attribut donnée :

```
[lang]              // les éléments ayant un attribut lang
p[lang='fr']        // les paragraphes dont l'attribut lang vaut "fr"
```

On peut joindre deux sélections en les séparant par une virgule :

```
[lang], div.menu    // l'union de ces deux sélections"
```

Il existe des sélections plus fines encore qui tiennent compte du placement relatif des éléments, mais nous n'en auront pas besoin ici.

2.4 Altérations du DOM

Le contenu HTML d'une page web n'est lu qu'une fois au moment de la construction du DOM. Le DOM peut être altéré a posteriori. Ceci ne modifie pas le code-source de la page mais seulement le DOM, les modifications étant dynamiquement répercutées sur la structure du DOM et sur l'aspect de la page dans le navigateur, *comme si* le contenu de la page avait lui-même été altéré.

Dans cette section et les suivantes, nous commettrons un abus de langage fréquent dans les documentations du DOM : nous emploierons le terme *élément* pour désigner un noeud du DOM, et plus généralement un nœud d'une arborescence quelconque. Ces éléments *représentent* des éléments HTML, mais ne doivent pas pour autant être confondus avec eux. Nous parlerons *d'altération des attributs, du contenu HTML, etc.* d'un élément pour désigner une modification des propriétés d'un nœud du DOM ayant une répercussion comparable sur l'élément HTML qu'il représente.

Atérations d'attributs. Les invocations de méthodes suivantes permettent de consulter ou d'altérer les attributs d'un élément – dans chaque cas, `name` et `value` doivent être des chaînes de caractères formant respectivement un nom d'attribut valide et une valeur d'attribut valide :

```
e.getAttribute(name);          // => la valeur de l'attribut, ou null
e.setAttribute(name, value);   // modifie l'attribut ou ajoute un attribut
e.hasAttribute(name);          // => true/false
e.removeAttribute(name)        // supprime l'attribut.
```

Un attribut d'élément peut aussi être modifié de manière directe via la propriété de son nœud associé portant le même nom :

```
let e = document.querySelector("[lang='fr']");
e.lang;                        // => "fr"
e.lang = "en";
```

Le mot `class` étant réservé, l'attribut `class` d'un élément ne peut cependant pas être modifié de cette manière. La propriété `classList` du nœud associé dispose de méthodes évitant d'avoir à gérer de manière directe la chaîne formant les mots d'un attribut `class` :

```
e.classList.contains("gauche");  // => true/false
e.classList.remove("gauche");    // supprimer le mot "gauche"
e.classList.add("droite");       // ajouter le mot "droite"
```

Remplacement et insertion de contenu HTML. Le texte HTML entre les balises de `head`, de `body` ou de tout autre élément peut être consulté et modifié dynamiquement :

```
// texte entre les balises des éléments HTML représentés :
document.head.innerHTML; document.body.innerHTML; p.innerHTML;
// modification dynamique du texte d'un paragraphe :
p.innerHTML = "Encore un <em>autre</em> texte.";
// modification dynamique du corps de la page :
document.body.innerHTML = "<h1>Ooops.</h1>";
```

La propriété `outerHTML` d'un nœud vaut quant à elle l'integralité du texte HTML de l'élément représenté, balises incluses, et sa valeur peut être modifiée de la même manière. On peut insérer du contenu HMTL au voisinage des balises d'un élément, sur le modèle suivant :

```
// insérer le contenu HTML de la chaîne content immédiatement...
e.insertAdjacentHTML("beforebegin", content);// avant la 1ère balise de e
e.insertAdjacentHTML("afterbegin",  content);// après la 1ère balise de e
e.insertAdjacentHTML("beforeend",   content);// avant la 2nde balise de e
e.insertAdjacentHTML("afterend",    content);// après la 2nde balise de e
```

Remplacement par du texte pur. Le contenu HTML d'un élément peut être remplacé par du texte pur via la propriété `textContent` - le rôle actif des caractères spéciaux d'HTML (’<’, ’>’ ...) est dans ce cas neutralisé en les remplaçant par ce qu'on appelle des *entités* – ils représentent bien ces caractères pour le navigateur, mais ne sont plus considérés comme faisant partie du balisage HTML :

```
e.innerHTML = "<div>left<div>center</div>right</div>";
e.textContent = "leftcenterright"; // contenu purement textuel de p
e.textContent = "<h1>title</h1>";
e.innerHTML;                       // => "&lt;h1&gt;title&lt;/h1&gt;"
```

Ajout et suppression de nouveaux éléments. Les méthodes de remplacement ou d'insertion de contenu HTML mentionnées ci-dessus ne sont pas très efficaces : elles obligent le navigateur à réinterpréter ce contenu pour mettre à jour l'arborescence dont le nœud modifié est la racine. Un autre technique est de modifier l'arborescence elle-même, par insertion d'autres arborescences construites dynamiquement, ou encore par suppression de sous-arbres.

La création d'un nouvel élément peut se faire à l'aide de la méthode `document.createElement`, en spécifiant le type de balise de l'élément HTML qu'il doit représenter :

```
let div = document.createElement("div");
div.textContent("Une div");
div.outerHTML;                  // "<div>Une div.</div>"
```

Un arbre B peut être inséré dans un arbre A en tant que premier ou dernier fils de sa racine. Cela revient à insérer les éléments HTML représentés par B après la première balise ou avant la seconde balise de l'élément HTML représenté par A :

```
e.prepend(div); // e.insertAdjacentHTML("afterbegin", div.outerHTML);
e.append(div);  // e.insertAdjacentHTML("beforeend", div.outerHTML);
```

Si un arbre A a pour fils B, un arbre C peut être inséré dans A en tant que frère précédent ou frère suivant de B. Cela revient à insérer les élements HTML représenté par C immédiatement avant la première balise ou immédiatement après la seconde balise de l'élément HTML représenté par A :

```
e.before(div); // e.insertAdjacentHTML("beforebegin", div.outerHTML);
e.after(div);  // e.insertAdjacentHTML("afterend", div.outerHTML);
```

Les quatre méthodes acceptent une suite d'arguments multiples, qui seront insérés successivement. Un argument peut être une chaîne de caractère, qui sera dans ce cas considéré comme du contenu textuel pur.

La suppression d'une sous-arborescence quelconque se fait par l'invocation de la méthode `remove` sur son nœud racine :

```
e.remove();
```

3 Gestion d'événements

Lorsqu'un script est associé à une page web, il peut demander au navigateur de l'informer des événements se produisant sur les éléments de cette page : fin de chargement d'une ressource, clic sur un élément, frappe d'une touche dans un champ texte, etc.

3.1 Types d'événements

Les différentes sortes d'événements pouvant se produire sur un élément dépendent bien sûr du type de cet élément, *i.e.* du nom de sa balise. Ce qu'on appelle *type* d'un évenement est une chaîne de caractères identifiant sa nature. Voici quelques exemples de types d'événements ainsi que des exmples d'éléments pouvant émettre ces événements :

- `"load"` émis par l'objet global `window` lorsque le DOM d'une page a été construit et que toutes ses ressources externes (`.css`, images...) ont été chargées.
- `"keydown"`, `"keypressed"` émis par un champ texte (`input`) ayant le focus lorsque l'événement clavier correspondant se produit.
- `"change"` émis par un champ texte ayant le focus lorsque l'utilisateur presse et relâche la touche d'entrée.
- `"input"` émis aussi par un champ texte ayant le focus à chaque fois que l'utilisateur presse et relâche une touche.
- `"click"` émis par tout élément sur lequel on clique (et pas seulement un bouton).
- `"mousedown"`, `"mouseenter"`, `"mousemove"`, `"mouseout"`, `"mouseover"`, `"mouseup"`, `"wheel"` émis par tout élément lorsque l'événement souris correspondant se produit au dessus cet élément.

3.2 Écouteurs d'événements

Par convention, un élément pouvant émettre un certain type d'événement est muni d'une propriété dont le nom commence par `on` suivi littéralement des caractères formant le type de cet événement : `onchange`, `onclick` ... Par défaut, cette propriété vaut `null`, mais sa valeur peut être remplacée par un *écouteur d'événement* (event listener), une fonction à un argument de la forme `function (e) { /* ... */ }`.

Après ce remplacement, cet écouteur sera appelé à chaque fois qu'un événement du type associé à la propriété se produira sur l'élément, en lui passant en argument un objet encapulant des informations sur cet événement :

```
let h = document.querySelector("h1");
h.onclick;              // null
h.onclick = function (e) { console.log(e); };
// après un clic sur le titre :
// => click { target: h1, buttons: 0, clientX: 137, clientY: 43, ... }
h.onclick = function (e) { console.log(e.target === h); };
// après un clic sur le titre : //=> true

// dans une page contenant un input d'id "nom" :
let t = document.querySelector("#nom");
t.oninput = function (e) { console.log(e); }
// après entrée du caractère 'a':
// => input { target: input#nom, ..., data: "a", ... }
// après entrée du caractère 'b':
// => input { target: input#nom, ..., data: "b", ... }
t.value;
// "ab"
```

Remarque. Dans un écouteur de nœud en `function` (resp. en `=>`), `this` désigne le nœud lui-même (resp. l'objet global `window`). L'écriture en `function` est en pratique préférable.

3.3 Ecouteurs multiples

La technique précédente ne permet de définir un seul écouteur pour un élément donné et un type d'événement donné. Si l'on souhaite ajouter plusieurs écouteurs à un même élément, il faut dans ce cas se servir de la méthode `addEventListener` de son nœud associé, en spécifiant un type d'événement et un écouteur.

```
let listener1 = function () { console.log("msg1"); };
let listener2 = function () { console.log("msg2"); };
h.addEventListener("click", listener1);
h.addEventListener("click", listener2);
// après un clic :
// => msg1
// => msg2
```

On peut aussi retirer un écouteur de la liste des écouteurs d'un type d'évenement donné à l'aide de la méthode `removeEventListener` :

```
h.removeEventListener("click", listener1);
// après un clic :
// => msg2
```

4 Chronologie de la manipulation du DOM

L'accès au DOM, l'ajout d'écouteurs, etc., ne sont possible que lorsque le DOM est entièrement construit. L'accès via le DOM à toutes les ressources de la page n'est possible que lorsque toutes ces ressources sont entièrement chargées.

Pour ces raisons, on peut fixer pour règle que toutes les opérations manipulant le DOM, mais pas les ressources externes de la page, doivent être effectuées par un unique écouteur de l'objet `document`, dont l'exécution ne commencera qu'à la notification d'un événement en `"DOMContentLoaded"` émis à la complétion de la construction du DOM :

```
/* ...
 * définitions d'objets, de classes, de fonctions,
 * d'écouteurs non encore liés...
 * ...
 */
 // écouteur principal
 function main() {
   // toute manipulation effective du DOM sera écrite ici.
 }

 // ajout du listener principal à document.
 // il n'y a pas de raccourci en on.. pour ce type d'événement :
 document.addEventListener('DOMContentLoaded', main);
```

S'il est en revanche nécessaire d'attendre le chargement complet de la page pour manipuler à la fois son DOM et ses ressources, il suffit de remplacer l'invocation ci-dessus par celle-ci (l'usage de `function` ou de `=>` ne change rien ici) :

```
window.onload(() => { /* accès au DOM et aux ressources */ });
```

5 Documentation du DOM

Une documentation du DOM parfois un peu imprécise mais souvent suffisante se trouve sur le site des développeurs de Mozilla. La documentation actuellement la plus complète se trouve sur le site du WHATWG, le "Web Hypertext Application Technology Working Group", une collaboration non officielle des différents développeurs de navigateurs web (Apple, Microsoft, Google, Mozilla). Elle est cependant assez difficile à lire sans avoir au préalable appris les bases du Web IDL – pour "Interface Definition Language", un langage permettant de spécifier les types attendus des propriétés des objets du DOM, les types attendus pour les arguments et la valeur de retour de leurs méthodes, ou même le comportement attendu de certaines méthodes.

6 jQuery

jQuery est une librairie javascript dont un des buts est de faciliter l'écriture de manipulations du DOM. Sa documentation, presque purement alphabétique, est malheureusement assez confuse.

Ce tutorial montre comment optimiser l'organisation du code utilisant jQuery, mais les conseils qu'il fournit ne se limitent pas à cet usage : tout code basé sur la définition d'écouteurs (et plus généralement, de fonctions invoquées de manière asynchrone) gagne à être organisé de cette manière (définition des futurs écouteurs/fonctions de rappels, puis dans un second temps, câblage).

6.1 Accès à la librairie

Il y a plusieurs manières de rendre jQuery accessible à un script lié une page web. La plus simple est encore d'inclure dans le `<head>` une demande de chargement de la version la plus récente de son script sur le site des développeurs :

```
<script src="http://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="monScript.js"></script>
```

Le script peut bien sûr être aussi téléchargé localement. Noter que la version chargée ici est en `"min"` c'est'a-dire sans retours à la ligne. Le fichier `"jquery-3.6.0.js"` contient les mêmes éléments mais avec une mise en page plus lisible.

6.2 Les objets `jQuery`

Ce que la documentation de jQuery appelle "objet `jQuery`" est un objet renvoyé par un appel de la fonction `jQuery` soit sur un sélecteur `css`, soit sur du contenu HTML :

```
let e = JQuery("h1");
let f = JQuery("<h1>Mon titre</h1>");
```

L'usage des fonctionnalités de la librairie nécessitant presque toujours un appel préliminaire de cette fonction, il existe un raccourci d'écriture pour celle-ci : un simple `$`.

```
jQuery === $;                          // => true
let e = $("h1");
let f = $("<h1>Mon titre</h1>");
```

Un objet `JQuery` encapsule un ou plusieurs nœuds du DOM, ou dans le cas d'un appel à `$` sur du contenu HTML, un noeud racine représentant ce contenu. L'invocation de cette fonction sur un sélecteur renvoie un objet itérable utilisable comme un tableau et encapsulant tous les noeuds du DOM sélectionnés par ce sélecteur :

```
// trois paragraphes d'ids "#p0", "#p1", "#p2".
let a = $("p");
// => Object { 0: p#p0, 1: p#p1, 2: p#p2, length: 3, ... }
a[0];                                  // p#p0
for(div in $("div")) { /* ...  */ }  // pour chaque div ...
```

6.3 Altérations de propriétés CSS en jQuery

Les propriétés CSS des éléments du DOM associés à une objet jQuery peuvent être librement modifiées à l'aide de la méthode `css`, sur le modèle suivant :

```
// tous les paragraphes seront de texte rouge
// et de taille de fonte double.
$("p").css ({
  color: "red",
  fontSize: "2em"
});
```

La même méthode permet de récupérer la valeur d'une propriété CSS du premier objet sélectionné par un appel de `$`, sous forme de chaîne de caractères :

```
$("p").css("left");   // => "10px"
```

Noter que les fonctions effectuant une altération des propriétés d'un objet jQuery renvoient en général l'objet lui-même, ce qui permet de chaîner les invocations de méthodes :

```
$("p").css({color: "red"}).css({fontSize: "2em"});
```

6.4 Écoute d'événements en jQuery

jQuery utilise les mêmes noms de types d'événements que le DOM, mais autorise plusieurs forme d'écritures équivalentes pour ajouter un écouteur aux éléments encapsulés dans un objet `jQuery`.

1. Il existe pour chaque type d'événement une méthode portant le même nom et prenant en argument un écouteur. Invoquée sur les objets `jQuery` appropriés, ces méthodes permettent de leur ajouter des écouteurs pour un type d'événements donné :

   ```
   $("#button1").click(function (e) {
     /* gestion d'un clic */
   });
   ```

   Noter que dans le corps de l'écouteur, `this` désigne le premier noeud du DOM sélectionné par la fonction `$`.

2. La méthode `on` fournit la forme équivalentes suivante – la forme précédente n'est qu'un raccourci d'écriture pour celle-ci :

   ```
   $("#button1").on("click", function (e) {
     /* gestion d'un clic */
   });
   ```

3. On peut aussi donner en argument à la méthode `on` un objet littéral dont les noms de propriétés sont des types d'événements, leur valeurs étant des écouteurs pour chacun de ces types – comme précédemment, dans le corps de chaque écouteur, `this` désignera le premier noeud du DOM sélectionné par la fonction `$` [1].

```
$("#div1").on ({
  mouseenter: function () {
    /* ... */
  },
  mouseleave: function () {
    /* ... */
  },
  mousedown:  function () {
    /* ... */
  },
  mouseup: function () {
    /* ... */
  }
});
```

6.5 Effets spéciaux

L'aspect probablement le plus intéressant de jQuery est la possibilité d'effectuer des *animations*, des modifications visuellement continues d'attributs CSS. Ces animations peuvent être enchaînées sur un ou plusieurs éléments associés à un objet jQuery (*c.f.* la remarque à la fin de la Section 6.3 sur les chaînages d' invocation de méthodes) :

```
let o = $('.moving');           // sélection d'un unique élément
let o_top  = o.css("top");      // mise en mémoire de ses
let o_left = o.css("left");     // coordonnées courantes.

o.animate({             // déplacer l'élément sélectionné
  top : "20em",         // vers (x, y) = (20em, 10em)
  left : "10em"         // en 1000ms === 1s
}, 1000)
.animate({              // puis, translater le même élément
  top: "-=10em",        // de (-10, 10)
  left: "+=10em"        // en 2s
}, 2000)
.animate({              // puis revenir aux coordonnées initiales
  top: o_top,           // en 1.5s
  left: o_left
}, 1500);
```

1. Et non l'objet littéral, *i.e.* tous ces écouteurs sont extraits de l'objet, qui ne sert qu'à abréger l'écriture d'une suite d'invocations de `on`.

Dans cet exemple, les trois animations spécifiées seront ajoutées à une file (“queue”) liée à l'objet `o` : la terminaison d'une animation entraîne son retrait de la file puis le passage, si elle existe, à l'animation suivante.

Animations et fonctions de rappel. Chaque appel de `animate` peut recevoir en argument supplémentaire une fonction de rappel, qui sera appelée une et une seule fois une fois à la complétion de l'animation spécifiée. Après les trois déclarations de l'exemple précédent, on pourrait par exemple écrire :

```
o.animate({           // déplacer l'élément séléctionné
  top : "20em",       // vers (x, y) = (20em, 10em) en 1s
  left : "10em"
}, 1000,
function () {         // fonction de rappel :
  o.ccs({             // après l'animation précédente,
    top: o_top,       // retour immédiat aux coordonnées initiales
    left: o_left
  });
});
```

Ajouts de fonctions à une file. On peut aussi ajouter explicitement une fonction (sans argument) à la file des animations d'un objet jQuery, en la passant à sa méthode `queue` : la fonction sera exécutée après l'exécution de tous les éléments précédents de la file, mais à la différence des animations précédentes, il faut explicitement supprimer cette fonction de la file de l'objet par un appel de sa méthode `dequeue` pour passer à l'élément suivant. L'exemple précédent est équivalent à :

```
o.animate({
  top : "20em",
  left : "10em"
}, 1000)
.queue(function () {
  o.ccs({ top: o_top, left: o_left }).dequeue();
});
```

Remarques. Certaines animations peuvent être appliquées à *tous* les éléments du DOM associés à un objet jQuery, *e.g.* une modification de `fontSize`. D'autres ne fonctionnent que pour le *dernier* élément associé, *e.g.* une modification des coordonnées.

La documentation ne précise pas quelles sont les animations applicables à plusieurs objets plutôt qu'à un seul, ce qui suggère qu'elles sont plutôt conçues pour être appliquées à des éléments isolés.

Dans le corps d'une fonction argument de la méthode `queue`, `this` désigne par ailleurs le dernier élément du DOM associé à l'objet jQuery sur lequel est invoqué cette méthode - ou l'unique élément associé à l'objet le cas échéant, ce qui tent à renforcer la suggestion précédente.