Поиск: getElement*, querySelector*

Свойства навигации по DOM хороши, когда элементы расположены рядом. А что, если нет? Как получить произвольный элемент страницы?

Для этого в DOM есть дополнительные методы поиска.

document.getElementById или просто id

Если у элемента есть атрибут `id`, то мы можем получить его вызовом `document.getElementById(id)`, где бы он ни находился.

Например:

```
<div id="elem">
  <div id="elem-content">Element</div>
</div>

<script>
  // получить элемент
  let elem = document.getElementById('elem');

  // сделать его фон красным
  elem.style.background = 'red';
</script>
```

Также есть глобальная переменная с именем, указанным в `id`:

```
<div id="elem">
  <div id="elem-content">Элемент</div>
</div>

<script>
  // elem - ссылка на элемент с id="elem"
  elem.style.background = 'red';

  // внутри id="elem-content" есть дефис, так что такой
  // ...но мы можем обратиться к нему через квадратные
</script>
```

...Но это только если мы не объявили в JavaScript переменную с таким же именем, иначе она будет иметь приоритет:

```
<div id="elem"></div>

<script>
  let elem = 5; // теперь elem равен 5, а не <div id="e

  alert(elem); // 5
</script>
```

Пожалуйста, не используйте такие глобальные переменные для доступа к элементам

Это поведение соответствует стандарту, но поддерживается в основном для совместимости, как осколок далёкого прошлого.

Браузер пытается помочь нам, смешивая пространства имён JS и DOM. Это удобно для простых скриптов, которые находятся прямо в HTML, но, вообще говоря, не очень хорошо. Возможны конфликты имён. Кроме того, при чтении JS-кода, не видя HTML, непонятно, откуда берётся переменная.

В этом учебнике мы будем обращаться к элементам по `id` в примерах для краткости, когда очевидно, откуда берётся элемент.

В реальной жизни лучше использовать `document.getElementById`.

Значение `id` должно быть уникальным

Значение `id` должно быть уникальным. В документе может быть только один элемент с данным `id`.

Если в документе есть несколько элементов с одинаковым значением `id`, то поведение методов поиска непредсказуемо. Браузер может вернуть любой из них случайным образом. Поэтому, пожалуйста, придерживайтесь правила сохранения уникальности `id`.

Только `document.getElementById`, а не `anyElem.getElementById`

Метод `getElementById` можно вызвать только для объекта `document`. Он осуществляет поиск по `id` по всему документу.

querySelectorAll

Самый универсальный метод поиска – это `elem.querySelectorAll(css)`, он возвращает все элементы внутри `elem`, удовлетворяющие данному CSS-селектору.

Следующий запрос получает все элементы `<li>`, которые являются последними потомками в `<ul>`:

```
<ul>
  <li>Этот</li>
  <li>тест</li>
</ul>
<ul>
  <li>полностью</li>
  <li>пройден</li>
</ul>
<script>
  let elements = document.querySelectorAll('ul > li:las

  for (let elem of elements) {
    alert(elem.innerHTML); // "тест", "пройден"
  }
</script>
```

Этот метод действительно мощный, потому что можно использовать любой CSS-селектор.

Псевдоклассы тоже работают

Псевдоклассы в CSS-селекторе, в частности `:hover` и `:active`, также поддерживаются. Например, `document.querySelectorAll(':hover')` вернёт коллекцию (в порядке вложенности: от внешнего к внутреннему) из текущих элементов под курсором мыши.

querySelector

Метод `elem.querySelector(css)` возвращает первый элемент, соответствующий данному CSS-селектору.

Иначе говоря, результат такой же, как при вызове `elem.querySelectorAll(css)[0]`, но он сначала найдёт *все* элементы, а потом возьмёт первый, в то время как `elem.querySelector` найдёт только первый и остановится. Это быстрее, кроме того, его короче писать.

matches

Предыдущие методы искали по DOM.

Метод elem.matches(css) ничего не ищет, а проверяет, удовлетворяет ли `elem` CSS-селектору, и возвращает `true` или `false`.

Этот метод удобен, когда мы перебираем элементы (например, в массиве или в чём-то подобном) и пытаемся выбрать те из них, которые нас интересуют.

Например:

```
<a href="http://example.com/file.zip">...</a>
<a href="http://ya.ru">...</a>

<script>
  // может быть любая коллекция вместо document.body.ch
  for (let elem of document.body.children) {
    if (elem.matches('a[href$="zip"]')) {
      alert("Ссылка на архив: " + elem.href );
    }
  }
</script>
```

closest

Предки элемента – родитель, родитель родителя, его родитель и так далее. Вместе они образуют цепочку иерархии от элемента до вершины.

Метод `elem.closest(css)` ищет ближайшего предка, который соответствует CSS-селектору. Сам элемент также включается в поиск.

Другими словами, метод `closest` поднимается вверх от элемента и проверяет каждого из родителей. Если он соответствует селектору, поиск прекращается. Метод возвращает либо предка, либо `null`, если такой элемент не найден.

Например:

```
<h1>Содержание</h1>

<div class="contents">
  <ul class="book">
    <li class="chapter">Глава 1</li>
    <li class="chapter">Глава 2</li>
  </ul>
</div>

<script>
  let chapter = document.querySelector('.chapter'); //
```


```
12
13    alert(chapter.closest('.book')); // UL
14    alert(chapter.closest('.contents')); // DIV
15
16    alert(chapter.closest('h1')); // null (потому что h1
17  </script>
```

getElementsBy*

Существуют также другие методы поиска элементов по тегу, классу и так далее.

На данный момент, они скорее исторические, так как `querySelector` более чем эффективен.

Здесь мы рассмотрим их для полноты картины, также вы можете встретить их в старом коде.

- `elem.getElementsByTagName(tag)` ищет элементы с данным тегом и возвращает их коллекцию. Передав `"*"` вместо тега, можно получить всех потомков.
- `elem.getElementsByClassName(className)` возвращает элементы, которые имеют данный CSS-класс.
- `document.getElementsByName(name)` возвращает элементы с заданным атрибутом `name`. Очень редко используется.

Например:

```
1  // получить все элементы div в документе
2  let divs = document.getElementsByTagName('div');
```

Давайте найдём все `input` в таблице:

```
1  <table id="table">
2    <tr>
3      <td>Ваш возраст:</td>
4
5      <td>
6        <label>
7          <input type="radio" name="age" value="young" ch
8        </label>
9        <label>
10         <input type="radio" name="age" value="mature">
11       </label>
12       <label>
13         <input type="radio" name="age" value="senior">
14       </label>
15     </td>
16   </tr>
17 </table>
18
19 <script>
20   let inputs = table.getElementsByTagName('input');
21
22   for (let input of inputs) {
23     alert( input.value + ': ' + input.checked );
24   }
25 </script>
```

Не забываем про букву "s"!

Одна из самых частых ошибок начинающих разработчиков (впрочем, иногда и не только) – это забыть букву `"s"`. То есть пробовать вызывать метод `getElementByTagName` вместо `getElementsByTagName`.

Буква `"s"` отсутствует в названии метода `getElementById`, так как в данном случае возвращает один элемент. Но `getElementsByTagName` вернёт список элементов, поэтому `"s"` обязательна.

Возвращает коллекцию, а не элемент!

Другая распространённая ошибка – написать:

```
// не работает
document.getElementsByTagName('input').value = 5;
```

Попытка присвоить значение *коллекции*, а не элементам внутри неё, не сработает.

Нужно перебрать коллекцию в цикле или получить элемент по номеру и уже ему присваивать значение, например, так:

```
// работает (если есть input)
document.getElementsByTagName('input')[0].value =
```

Ищем элементы с классом `.article`:

```
<form name="my-form">
  <div class="article">Article</div>
  <div class="long article">Long article</div>
</form>

<script>
  // ищем по имени атрибута
  let form = document.getElementsByName('my-form')[0];

  // ищем по классу внутри form
  let articles = form.getElementsByClassName('article')
  alert(articles.length); // 2, находим два элемента с
</script>
```

Живые коллекции

Все методы `"getElementsBy*"` возвращают *живую* коллекцию. Такие коллекции всегда отражают текущее состояние документа и автоматически обновляются при его изменении.

В приведённом ниже примере есть два скрипта.

1. Первый создаёт ссылку на коллекцию `<div>`. На этот момент её длина равна `1`.
2. Второй скрипт запускается после того, как браузер встречает ещё один `<div>`, теперь её длина – `2`.

```
<div>First div</div>

<script>
  let divs = document.getElementsByTagName('div');
  alert(divs.length); // 1
</script>
```

```
7
8  <div>Second div</div>
9
10 <script>
11   alert(divs.length); // 2
12 </script>
```

Напротив, `querySelectorAll` возвращает *статическую* коллекцию. Это похоже на фиксированный массив элементов.

Если мы будем использовать его в примере выше, то оба скрипта вернут длину коллекции, равную `1`:

```
1  <div>First div</div>
2
3  <script>
4    let divs = document.querySelectorAll('div');
5    alert(divs.length); // 1
6  </script>
7
8  <div>Second div</div>
9
10 <script>
11   alert(divs.length); // 1
12 </script>
```

Теперь мы легко видим разницу. Длина статической коллекции не изменилась после появления нового `div` в документе.

Итого

Есть 6 основных методов поиска элементов в DOM:

Метод	Ищет по...	Ищет внутри элемента?	Возвращает живую коллекцию?
`querySelector`	CSS-selector	✔	-
`querySelectorAll`	CSS-selector	✔	-
`getElementById`	`id`	-	-
`getElementsByName`	`name`	-	✔
`getElementsByTagName`	tag or `'*'`	✔	✔
`getElementsByClassName`	class	✔	✔

Безусловно, наиболее часто используемыми в настоящее время являются методы `querySelector` и `querySelectorAll`, но и методы `getElement(s)By*` могут быть полезны в отдельных случаях, а также встречаются в старом коде.

Кроме того:

- Есть метод `elem.matches(css)`, который проверяет, удовлетворяет ли элемент CSS-селектору.
- Метод `elem.closest(css)` ищет ближайшего по иерархии предка, соответствующему данному CSS-селектору. Сам элемент также включён в поиск.

И, напоследок, давайте упомянем ещё один метод, который проверяет наличие отношений между предком и потомком:

- `elemA.contains(elemB)` вернёт `true`, если `elemB` находится внутри `elemA` (`elemB` потомок `elemA`) или когда `elemA==elemB`.

Задачи

Поиск элементов

важность: 4

Вот документ с таблицей и формой.

Как найти?...

1. Таблицу с `id="age-table"`.
2. Все элементы `label` внутри этой таблицы (их три).
3. Первый `td` в этой таблице (со словом «Age»).
4. Форму `form` с именем `name="search"`.
5. Первый `input` в этой форме.
6. Последний `input` в этой форме.

Откройте страницу table.html в отдельном окне и используйте для этого браузерные инструменты разработчика.

решение

Комментарии

перед тем как писать...