**Поиск: getElement\* и querySelector\* и не только**

Прямая навигация от родителя к потомку удобна, если элементы рядом. А если нет?

Как достать произвольный элемент откуда-то из глубины документа?

Для этого в DOM есть дополнительные методы поиска.

**[document.getElementById или просто id](http://learn.javascript.ru/searching-elements-dom" \l "document-getelementbyid-ili-prosto-id)**

Если элементу назначен специальный атрибут id, то можно получить его прямо по переменной с именем из значения id.

Например:

<div id="content-holder">

<div id="content">Элемент</div>

</div>

<script>

alert( content ); // DOM-элемент

alert( window['content-holder'] ); // в имени дефис, поэтому через [...]

</script>

Это поведение соответствует [стандарту](http://www.whatwg.org/specs/web-apps/current-work/#dom-window-nameditem). Оно существует, в первую очередь, для совместимости, как осколок далёкого прошлого и не очень приветствуется, поскольку использует глобальные переменные. Браузер пытается помочь нам, смешивая пространства имён JS и DOM, но при этом возможны конфликты.

**Более правильной и общепринятой практикой является доступ к элементу вызовом document.getElementById("идентификатор").**

Например:

<div id="content">Выделим этот элемент</div>

<script>

var elem = document.getElementById('content');

elem.style.background = 'red';

alert( elem == content ); // true

content.style.background = ""; // один и тот же элемент

</script>

**Должен остаться только один**

По стандарту значение id должно быть уникально, то есть в документе может быть только один элемент с данным id. И именно он будет возвращён.

Если в документе есть несколько элементов с уникальным id, то поведение неопределено. То есть, нет гарантии, что браузер вернёт именно первый или последний – вернёт случайным образом.

Поэтому стараются следовать правилу уникальности id.

Далее в примерах я часто буду использовать прямое обращение через переменную, чтобы было меньше букв и проще было понять происходящее. Но предпочтительным методом является document.getElementById.

**[getElementsByTagName](http://learn.javascript.ru/searching-elements-dom" \l "getelementsbytagname)**

Метод elem.getElementsByTagName(tag) ищет все элементы с заданным тегом tag внутри элемента elem и возвращает их в виде списка.

Регистр тега не имеет значения.

Например:

// получить все div-элементы

var elements = document.getElementsByTagName('div');

**Обратим внимание: в отличие от getElementById, который существует только в контексте document, метод getElementsByTagName может искать внутри любого элемента.**

Например, найдём все элементы input внутри таблицы:

<table id="age-table">

<tr>

<td>Ваш возраст:</td>

<td>

<label>

<input type="radio" name="age" value="young" checked> младше 18

</label>

<label>

<input type="radio" name="age" value="mature"> от 18 до 50

</label>

<label>

<input type="radio" name="age" value="senior"> старше 60

</label>

</td>

</tr>

</table>

<script>

var tableElem = document.getElementById('age-table');

var elements = tableElem.getElementsByTagName('input');

for (var i = 0; i < elements.length; i++) {

var input = elements[i];

alert( input.value + ': ' + input.checked );

}

</script>

**Можно получить всех потомков, передав звездочку '\*' вместо тега:**

// получить все элементы документа

document.getElementsByTagName('\*');

// получить всех потомков элемента elem:

elem.getElementsByTagName('\*');

**Не забываем про букву "s"!**

Одна из самых частых ошибок начинающих (впрочем, иногда и не только) – это забыть букву "s", то есть пробовать вызывать метод getElementByTagName вместо getElement**s**ByTagName.

Буква "s" не нужна там, где элемент только один, то есть в getElementById, в остальных методах она обязательна.

**Возвращается коллекция, а не элемент**

Другая частая ошибка – это код вида:

// не работает

document.getElementsByTagName('input').value = 5;

То есть, вместо элемента присваивают значение коллекции. Работать такое не будет.

Коллекцию нужно или перебрать в цикле или получить элемент по номеру и уже ему присваивать value, например так:

// работает

document.getElementsByTagName('input')[0].value = 5;

**[document.getElementsByName](http://learn.javascript.ru/searching-elements-dom" \l "document-getelementsbyname)**

Вызов document.getElementsByName(name) позволяет получить все элементы с данным атрибутом name.

Например, все элементы с именем age:

var elems = document.getElementsByName('age');

До появления стандарта HTML5 этот метод возвращал только те элементы, в которых предусмотрена поддержка атрибута name, в частности: iframe, a, input и другими. В современных браузерах (IE10+) тег не имеет значения.

Используется этот метод весьма редко.

**[getElementsByClassName](http://learn.javascript.ru/searching-elements-dom" \l "getelementsbyclassname)**

Вызов elem.getElementsByClassName(className) возвращает коллекцию элементов с классом className. Находит элемент и в том случае, если у него несколько классов, а искомый – один из них.

Поддерживается всеми современными браузерами, кроме IE8-.

Например:

<div class="article">Статья</div>

<div class="long article">Длинная статья</div>

<script>

var articles = document.getElementsByClassName('article');

alert( articles.length ); // 2, найдёт оба элемента

</script>

Как и getElementsByTagName, этот метод может быть вызван и в контексте DOM-элемента, и в контексте документа.

**[querySelectorAll](http://learn.javascript.ru/searching-elements-dom" \l "querySelectorAll)**

Вызов elem.querySelectorAll(css) возвращает все элементы внутри elem, удовлетворяющие CSS-селектору css.

Это один из самых часто используемых и полезных методов при работе с DOM.

Он есть во всех современных браузерах, включая IE8+ (в режиме соответствия стандарту).

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>

var elements = document.querySelectorAll('ul > li:last-child');

for (var i = 0; i < elements.length; i++) {

alert( elements[i].innerHTML ); // "тест", "пройден"

}

</script>

**Псевдо-класс тоже работает**

Псевдо-классы в CSS-селекторе, в частности :hover и :active, также поддерживаются. Например, document.querySelectorAll(':hover') вернёт список, в порядке вложенности, из текущих элементов под курсором мыши.

**[querySelector](http://learn.javascript.ru/searching-elements-dom" \l "querySelector)**

Вызов elem.querySelector(css) возвращает не все, а только первый элемент, соответствующий CSS-селектору css.

Иначе говоря, результат – такой же, как и при elem.querySelectorAll(css)[0], но в последнем вызове сначала ищутся все элементы, а потом берётся первый, а в elem.querySelector(css) ищется только первый, то есть он эффективнее.

Этот метод часто используется, когда мы заведомо знаем, что подходящий элемент только один, и хотим получить в переменную сразу его.

**[matches](http://learn.javascript.ru/searching-elements-dom" \l "matches)**

Предыдущие методы искали по DOM.

Метод [elem.matches(css)](http://dom.spec.whatwg.org/" \l "dom-element-matches) ничего не ищет, а проверяет, удовлетворяет ли elem селектору css. Он возвращает trueлибо false.

Не поддерживается в IE8-.

Этот метод бывает полезным, когда мы перебираем элементы (в массиве или по обычным навигационным ссылкам) и пытаемся отфильтровать те из них, которые нам интересны.

Ранее в спецификации он назывался matchesSelector, и большинство браузеров поддерживают его под этим старым именем, либо с префиксами ms/moz/webkit.

Например:

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

<a href="http://ya.ru">...</a>

<script>

var elems = document.body.children;

for (var i = 0; i < elems.length; i++) {

if (elems[i].matches('a[href$="zip"]')) {

alert( "Ссылка на архив: " + elems[i].href );

}

}

</script>

**[closest](http://learn.javascript.ru/searching-elements-dom" \l "closest)**

Метод elem.closest(css) ищет ближайший элемент выше по иерархии DOM, подходящий под CSS-селектор css. Сам элемент тоже включается в поиск.

Иначе говоря, метод closest бежит от текущего элемента вверх по цепочке родителей и проверяет, подходит ли элемент под указанный CSS-селектор. Если подходит – останавливается и возвращает его.

Он самый новый из методов, рассмотренных в этой главе, поэтому старые браузеры его слабо поддерживают. Это, конечно, легко поправимо, как мы увидим позже в главе [Современный DOM: полифиллы](http://learn.javascript.ru/dom-polyfill).

Пример использования (браузер должен поддерживать closest):

<ul>

<li class="chapter">Глава I

<ul>

<li class="subchapter">Глава <span class="num">1.1</span></li>

<li class="subchapter">Глава <span class="num">1.2</span></li>

</ul>

</li>

</ul>

<script>

var numberSpan = document.querySelector('.num');

// ближайший элемент сверху подходящий под селектор li

alert(numberSpan.closest('li').className) // subchapter

// ближайший элемент сверху подходящий под селектор .chapter

alert(numberSpan.closest('.chapter').tagName) // LI

// ближайший элемент сверху, подходящий под селектор span

// это сам numberSpan, так как поиск включает в себя сам элемент

alert(numberSpan.closest('span') === numberSpan) // true

</script>

**[XPath в современных браузерах](http://learn.javascript.ru/searching-elements-dom" \l "xpath-v-sovremennyh-brauzerah)**

Для полноты картины рассмотрим ещё один способ поиска, который обычно используется в XML. Это [язык запросов XPath](http://www.w3.org/TR/xpath/).

Он очень мощный, во многом мощнее CSS, но сложнее. Например, запрос для поиска элементов H2, содержащих текст "XPath", будет выглядеть так: //h2[contains(., "XPath")].

Все современные браузеры, кроме IE, поддерживают XPath с синтаксисом, близким к [описанному в MDN](https://developer.mozilla.org/en/XPath).

Найдем заголовки с текстом XPath в текущем документе:

var result = document.evaluate("//h2[contains(., 'XPath')]", document.documentElement, null,

XPathResult.ORDERED\_NODE\_SNAPSHOT\_TYPE, null);

for (var i = 0; i < result.snapshotLength; i++) {

alert( result.snapshotItem(i).outerHTML );

}

IE тоже поддерживает XPath, но эта поддержка не соответствует стандарту и работает только для XML-документов, например, полученных с помощью XMLHTTPRequest (AJAX). Для обычных же HTML-документов XPath в IE не поддерживается.

Так как XPath сложнее и длиннее CSS, то используют его очень редко.

**[Итого](http://learn.javascript.ru/searching-elements-dom" \l "itogo)**

Есть 6 основных методов поиска элементов DOM:

| Метод | Ищет по... | Ищет внутри элемента? | Поддержка |
| --- | --- | --- | --- |
| getElementById | id | - | везде |
| getElementsByName | name | - | везде |
| getElementsByTagName | тег или '\*' | ✔ | везде |
| getElementsByClassName | классу | ✔ | кроме IE8- |
| querySelector | CSS-селектор | ✔ | везде |
| querySelectorAll | CSS-селектор | ✔ | везде |

Практика показывает, что в 95% ситуаций достаточно querySelector/querySelectorAll. Хотя более специализированные методы getElement\* работают чуть быстрее, но разница в миллисекунду-другую редко играет роль.

Кроме того:

* Есть метод elem.matches(css), который проверяет, удовлетворяет ли элемент CSS-селектору. Он поддерживается большинством браузеров в префиксной форме (ms, moz, webkit).
* Метод elem.closest(css) ищет ближайший элемент выше по иерархии DOM, подходящий под CSS-селектор css. Сам элемент тоже включается в поиск.
* Язык запросов XPath поддерживается большинством браузеров, кроме IE, даже 9-й версии, но querySelectorудобнее. Поэтому XPath используется редко.