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 |

**[querySelectorAll](https://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>

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.

**[innerHTML: содержимое элемента](https://learn.javascript.ru/basic-dom-node-properties" \l "innerhtml-%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B8%D0%BC%D0%BE%D0%B5-%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B0)**

Свойство innerHTML описано в спецификации HTML 5 – [embedded content](http://www.w3.org/TR/html5/embedded-content-0.html).

Оно позволяет получить HTML-содержимое элемента в виде строки. В innerHTML можно и читать и писать.

Пример выведет на экран все содержимое document.body, а затем заменит его на другое:

<body>

<p>Параграф</p>

<div>Div</div>

<script>

alert( document.body.innerHTML ); // читаем текущее содержимое

document.body.innerHTML = 'Новый BODY!'; // заменяем содержимое

</script>

</body>

Значение, возвращаемое innerHTML – всегда валидный HTML-код. При записи можно попробовать записать что угодно, но браузер исправит ошибки:

<body>

<script>

document.body.innerHTML = '<b>тест'; // незакрытый тег

alert( document.body.innerHTML ); // <b>тест</b> (исправлено)

</script>

</body>

**[outerHTML: HTML элемента целиком](https://learn.javascript.ru/basic-dom-node-properties" \l "outerhtml-html-%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B0-%D1%86%D0%B5%D0%BB%D0%B8%D0%BA%D0%BE%D0%BC)**

Свойство outerHTML содержит HTML элемента целиком.

Пример чтения outerHTML:

<div>Привет <b>Мир</b></div>

<script>

var div = document.body.children[0];

alert( div.outerHTML ); // <div>Привет <b>Мир</b></div>

</script>

**Изменить outerHTML элемента невозможно.**

Здесь мы остановимся чуть подробнее. Дело в том, что технически свойство outerHTML доступно на запись. Но при этом элемент не меняется, а *заменяется на новый*, который тут же создаётся из нового outerHTML.

При этом переменная, в которой изначально был старый элемент, и в которой мы «перезаписали» outerHTML, остаётся со старым элементом.

Это легко может привести к ошибкам, что видно на примере:

**[nodeValue/data: содержимое текстового узла](https://learn.javascript.ru/basic-dom-node-properties" \l "nodevalue-data-%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B8%D0%BC%D0%BE%D0%B5-%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE-%D1%83%D0%B7%D0%BB%D0%B0)**

Свойство innerHTML есть только у узлов-элементов.

**Содержимое других узлов, например, текстовых или комментариев, доступно на чтение и запись через свойство data.**

Его тоже можно читать и обновлять. Следующий пример демонстрирует это:

<body>

Привет

<!-- Комментарий -->

<script>

for (var i = 0; i < document.body.childNodes.length; i++) {

alert( document.body.childNodes[i].data );

}

</script>

Пока

</body>

**[Текст: textContent](https://learn.javascript.ru/basic-dom-node-properties" \l "%D1%82%D0%B5%D0%BA%D1%81%D1%82-textcontent)**

Свойство textContent содержит только текст внутри элемента, за вычетом всех <тегов>.

Оно поддерживается везде, кроме IE8-.

Например:

<div>

<h1>Срочно в номер!</h1>

<p>Марсиане атакуют людей!</p>

</div>

<script>

var news = document.body.children[0];

// \n Срочно в номер!\n Марсиане атакуют людей!\n

alert( news.textContent );

</script>

**Иными словами, elem.textContent возвращает конкатенацию всех текстовых узлов внутри elem.**

Не сказать, чтобы эта информация была часто востребована.

**Гораздо полезнее возможность записать текст в элемент, причём именно как текст!**

В этом примере имя посетителя попадёт в первый div как innerHTML, а во второй – как текст:

<div></div>

<div></div>

<script>

var name = prompt("Введите имя?", "<b>Винни-пух</b>");

document.body.children[0].innerHTML = name;

document.body.children[1].textContent = name;

</script>

## [Исследование элементов](https://learn.javascript.ru/basic-dom-node-properties" \l "%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2)

У DOM-узлов есть и другие свойства, зависящие от типа, например:

* value – значение для INPUT, SELECT или TEXTAREA
* id – идентификатор
* href – адрес ссылки
* …многие другие…

Например:

<input type="text" id="elem" value="значение">

<script>

var input = document.body.children[0];

alert( input.type ); // "text"

alert( input.id ); // "elem"

alert( input.value ); // значение

</script>

Как узнать, какие свойства есть у данного типа элементов?

Это просто. Нужно либо посмотреть [список элементов HTML5](http://dev.w3.org/html5/markup/elements.html) и найти в нём интересующий вас элемент и прочитать секцию с interface.

Если же недосуг или интересуют особенности конкретного браузера – элемент всегда можно вывести в консоль вызовом console.dir(элемент).

Метод console.dir выводит аргумент не в «красивом» виде, а как объект, который можно развернуть и исследовать.

Например:

// в консоли можно будет увидеть все свойства DOM-объекта document

console.dir(document);

Основные свойства DOM-узлов:

**nodeType**

Тип узла. Самые популярные типы: "1" – для элементов и "3" – для текстовых узлов. Только для чтения.

**nodeName/tagName**

Название тега заглавными буквами. nodeName имеет специальные значения для узлов-неэлементов. Только для чтения.

**innerHTML**

Внутреннее содержимое узла-элемента в виде HTML. Можно изменять.

**outerHTML**

Полный HTML узла-элемента. При записи в elem.outerHTML переменная elem сохраняет старый узел.

**nodeValue/data**

Содержимое текстового узла или комментария. Свойство nodeValue также определено и для других типов узлов. Можно изменять. На некоторых узлах, где data нет, nodeValue есть и имеет значение null, поэтому лучше использовать data.

**textContent**

Содержит только текст внутри элемента, за вычетом всех тегов. Можно использовать для защиты от вставки произвольного HTML кода

**Свойство и атрибут hidden**

Скрыть элемент можно с помощью установки свойства hidden в true или с помощью атрибута

Узлы DOM также имеют другие свойства, в зависимости от тега. Например, у INPUT есть свойства value и checked, а у A есть href и т.д. Мы рассмотрим их далее.

**[трибуты](https://learn.javascript.ru/attributes-and-custom-properties" \l "%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B)**

Элементам DOM, с другой стороны, соответствуют HTML-теги, у которых есть текстовые атрибуты.

Конечно, здесь речь именно об узлах-элементах, не о текстовых узлах или комментариях.

Доступ к атрибутам осуществляется при помощи стандартных методов:

* elem.hasAttribute(name) – проверяет наличие атрибута
* elem.getAttribute(name) – получает значение атрибута
* elem.setAttribute(name, value) – устанавливает атрибут
* elem.removeAttribute(name) – удаляет атрибут

Эти методы работают со значением, которое находится в HTML.

Также все атрибуты элемента можно получить с помощью свойства elem.attributes, которое содержит псевдо-массив объектов типа [Attr](http://www.w3.org/TR/DOM-Level-3-Core/core.html" \l "ID-637646024).

В отличие от свойств, атрибуты:

* Всегда являются строками.
* Их имя *нечувствительно* к регистру (ведь это HTML)
* Видны в innerHTML (за исключением старых IE)

Рассмотрим отличия между DOM-свойствами и атрибутами на примере HTML-кода:

<body>

<div id="elem" about="Elephant" class="smiling"></div>

</body>

Пример ниже устанавливает атрибуты и демонстрирует их особенности.

<body>

<div id="elem" about="Elephant"></div>

<script>

alert( elem.getAttribute('About') ); // (1) 'Elephant', атрибут получен

elem.setAttribute('Test', 123); // (2) атрибут Test установлен

alert( document.body.innerHTML ); // (3) в HTML видны все атрибуты!

var attrs = elem.attributes; // (4) можно получить коллекцию атрибутов

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

alert( attrs[i].name + " = " + attrs[i].value );

}

</script>

</body>

При запуске кода выше обратите внимание:

1. getAttribute('About') – первая буква имени атрибута About написана в верхнем регистре, а в HTML – в нижнем, но это не имеет значения, так как имена нечувствительны к регистру.
2. Мы можем записать в атрибут любое значение, но оно будет превращено в строку. Объекты также будут автоматически преобразованы.
3. После добавления атрибута его можно увидеть в innerHTML элемента.
4. Коллекция attributes содержит все атрибуты в виде объектов со свойствами name и value.

**[Классы в виде объекта: classList](https://learn.javascript.ru/attributes-and-custom-properties" \l "%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B-%D0%B2-%D0%B2%D0%B8%D0%B4%D0%B5-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%B0-classlist)**

Атрибут class – уникален. Ему соответствует аж целых два свойства!

Работать с классами как со строкой неудобно. Поэтому, кроме className, в современных браузерах есть свойство classList.

**Свойство classList – это объект для работы с классами.**

Оно поддерживается в IE начиная с IE10, но его можно эмулировать в IE8+, подключив мини-библиотеку [classList.js](https://github.com/eligrey/classList.js).

Методы classList:

* elem.classList.contains("class") – возвращает true/false, в зависимости от того, есть ли у элемента класс class.
* elem.classList.add/remove("class") – добавляет/удаляет класс class
* elem.classList.toggle("class") – если класса class нет, добавляет его, если есть – удаляет.

Кроме того, можно перебрать классы через for, так как classList – это псевдо-массив.

Например:

<body class="main page">

<script>

var classList = document.body.classList;

classList.remove('page'); // удалить класс

classList.add('post'); // добавить класс

for (var i = 0; i < classList.length; i++) { // перечислить классы

alert( classList[i] ); // main, затем post

}

alert( classList.contains('post') ); // проверить наличие класса

alert( document.body.className ); // main post, тоже работает

</script>

</body>

## [Нестандартные атрибуты](https://learn.javascript.ru/attributes-and-custom-properties" \l "%D0%BD%D0%B5%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D0%B5-%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B)

У каждого элемента есть некоторый набор стандартных свойств, например для <a> это будут href, name, а для <img>это будут src, alt, и так далее.

Точный набор свойств описан в стандарте, обычно мы более-менее представляем, если пользуемся HTML, какие свойства могут быть, а какие – нет.

Для нестандартных атрибутов DOM-свойство не создаётся.

Например:

<div id="elem" href="http://ya.ru" about="Elephant"></div>

<script>

alert( elem.id ); // elem

alert( elem.about ); // undefined

</script>

Свойство является стандартным, только если оно описано в стандарте именно для этого элемента.

То есть, если назначить элементу <img> атрибут href, то свойство img.href от этого не появится. Как, впрочем, и если назначить ссылке <a> атрибут alt:

<img id="img" href="test">

<a id="link" alt="test"></a>

<script>

alert( img.href ); // undefined

alert( link.alt ); // undefined

</script>

Нестандартные атрибуты иногда используют для CSS.

В примере ниже для показа «состояния заказа» используется атрибут order-state:

<style>

.order[order-state="new"] {

color: green;

}

.order[order-state="pending"] {

color: blue;

}

.order[order-state="canceled"] {

color: red;

}

</style>

<div class="order" order-state="new">

Новый заказ.

</div>

<div class="order" order-state="pending">

Ожидающий заказ.

</div>

<div class="order" order-state="canceled">

Заказ отменён.

</div>

Почему именно атрибут? Разве нельзя было сделать классы .order-state-new, .order-state-pending, order-state-canceled?

Конечно можно, но манипулировать атрибутом из JavaScript гораздо проще.

Например, если нужно отменить заказ, неважно в каком он состоянии сейчас – это сделает код:

div.setAttribute('order-state', 'canceled');

Для классов – нужно знать, какой класс у заказа сейчас. И тогда мы можем снять старый класс, и поставить новый:

div.classList.remove('order-state-new');

div.classList.add('order-state-canceled');

…То есть, требуется больше исходной информации и надо написать больше букв. Это менее удобно.

Проще говоря, значение атрибута – произвольная строка, значение класса – это «есть» или «нет», поэтому естественно, что атрибуты «мощнее» и бывают удобнее классов как в JS так и в CSS.

## [Свойство dataset, data-атрибуты](https://learn.javascript.ru/attributes-and-custom-properties" \l "%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE-dataset-data-%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B)

С помощью нестандартных атрибутов можно привязать к элементу данные, которые будут доступны в JavaScript.

Как правило, это делается при помощи атрибутов с названиями, начинающимися на data-, например:

<div id="elem" data-about="Elephant" data-user-location="street">

По улице прошёлся слон. Весьма красив и толст был он.

</div>

<script>

alert( elem.getAttribute('data-about') ); // Elephant

alert( elem.getAttribute('data-user-location') ); // street

</script>

[Стандарт HTML5](http://www.w3.org/TR/2010/WD-html5-20101019/elements.html#embedding-custom-non-visible-data-with-the-data-attributes) специально разрешает атрибуты data-\* и резервирует их для пользовательских данных.

При этом во всех браузерах, кроме IE10-, к таким атрибутам можно обратиться не только как к атрибутам, но и как к свойствам, при помощи специального свойства dataset:

<div id="elem" data-about="Elephant" data-user-location="street">

По улице прошёлся слон. Весьма красив и толст был он.

</div>

<script>

alert( elem.dataset.about ); // Elephant

alert( elem.dataset.userLocation ); // street

</script>

Обратим внимание – название data-user-location трансформировалось в dataset.userLocation. Дефис превращается в большую букву.

## [Свойство dataset, data-атрибуты](https://learn.javascript.ru/attributes-and-custom-properties#%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE-dataset-data-%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B)

С помощью нестандартных атрибутов можно привязать к элементу данные, которые будут доступны в JavaScript.

Как правило, это делается при помощи атрибутов с названиями, начинающимися на data-, например:

<div id="elem" data-about="Elephant" data-user-location="street">

По улице прошёлся слон. Весьма красив и толст был он.

</div>

<script>

alert( elem.getAttribute('data-about') ); // Elephant

alert( elem.getAttribute('data-user-location') ); // street

</script>

[Стандарт HTML5](http://www.w3.org/TR/2010/WD-html5-20101019/elements.html#embedding-custom-non-visible-data-with-the-data-attributes) специально разрешает атрибуты data-\* и резервирует их для пользовательских данных.

При этом во всех браузерах, кроме IE10-, к таким атрибутам можно обратиться не только как к атрибутам, но и как к свойствам, при помощи специального свойства dataset:

<div id="elem" data-about="Elephant" data-user-location="street">

По улице прошёлся слон. Весьма красив и толст был он.

</div>

<script>

alert( elem.dataset.about ); // Elephant

alert( elem.dataset.userLocation ); // street

</script>

Обратим внимание – название data-user-location трансформировалось в dataset.userLocation. Дефис превращается в большую букву.

## [Полифилл для атрибута hidden](https://learn.javascript.ru/attributes-and-custom-properties" \l "%D0%BF%D0%BE%D0%BB%D0%B8%D1%84%D0%B8%D0%BB%D0%BB-%D0%B4%D0%BB%D1%8F-%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B0-hidden)

Для старых браузеров современные атрибуты иногда нуждаются в полифилле. Как правило, такой полифилл включает в себя не только JavaScript, но и CSS.

Этот атрибут должен прятать элемент, действие весьма простое, для его поддержки в HTML достаточно такого CSS:

<style>

[hidden] { display: none }

</style>

<div>Текст</div>

<div hidden>С атрибутом hidden</div>

<div id="last">Со свойством hidden</div>

<script>

last.hidden = true;

</script>

Если запустить в IE11- пример выше, то <div hidden> будет скрыт, а вот последний div, которому поставили свойство hidden в JavaScript – по-прежнему виден.

Это потому что CSS «не видит» присвоенное свойство, нужно синхронизировать его в атрибут.

Вот так – уже работает:

<style>

[hidden] { display: none }

</style>

<script>

if (document.documentElement.hidden === undefined) {

Object.defineProperty(Element.prototype, "hidden", {

set: function(value) {

this.setAttribute('hidden', value);

},

get: function() {

return this.getAttribute('hidden');

}

});

}

</script>

<div>Текст</div>

<div hidden>С атрибутом hidden</div>

<div id="last">Со свойством hidden</div>

<script>

last.hidden = true;

</script>