Получив DOM-узел, мы можем перейти к его ближайшим соседям используя навигационные ссылки.

Есть два основных набора ссылок:

* Для всех узлов: parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling.
* Только для узлов-элементов: parentElement, children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling.

Некоторые виды DOM-элементов, например таблицы, предоставляют дополнительные ссылки и коллекции для доступа к своему содержимому.

**Таблицы всегда содержат <tbody>**

Важный «особый случай» – работа с таблицами. По стандарту DOM у них должен быть <tbody>, но в HTML их можно написать (официально) без него. В этом случае браузер добавляет <tbody> в DOM самостоятельно.

<table id="table"><tbody><tr><td>1</td></tr></tbody></table>

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.

**Каждый DOM-узел принадлежит определённому классу. Классы формируют иерархию. Весь набор свойств и методов является результатом наследования.**

Главные свойства DOM-узла:

nodeType

Свойство nodeType позволяет узнать тип DOM-узла. Его значение – числовое: 1 для элементов,3 для текстовых узлов, и т.д. Только для чтения.

nodeName/tagName

Для элементов это свойство возвращает название тега (записывается в верхнем регистре, за исключением XML-режима). Для узлов-неэлементов nodeName описывает, что это за узел. Только для чтения.

innerHTML

Внутреннее HTML-содержимое узла-элемента. Можно изменять.

outerHTML

Полный HTML узла-элемента. Запись в elem.outerHTML не меняет elem. Вместо этого она заменяет его во внешнем контексте.

nodeValue/data

Содержимое узла-неэлемента (текст, комментарий). Эти свойства практически одинаковые, обычно мы используем data. Можно изменять.

textContent

Текст внутри элемента: HTML за вычетом всех <тегов>. Запись в него помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст. Можно использовать для защиты от вставки произвольного HTML кода.

hidden

Когда значение установлено в true, делает то же самое, что и CSS display:none.

В зависимости от своего класса DOM-узлы имеют и другие свойства. Например у элементов <input> (HTMLInputElement) есть свойства value, type, у элементов <a> (HTMLAnchorElement) есть href и т.д. Большинство стандартных HTML-атрибутов имеют соответствующие свойства DOM.

Впрочем, HTML-атрибуты и свойства DOM не всегда одинаковы, мы увидим это в следующей главе.

* Атрибуты – это то, что написано в HTML.
* Свойства – это то, что находится в DOM-объектах.

Небольшое сравнение:

|  | **Свойства** | **Атрибуты** |
| --- | --- | --- |
| Тип | Любое значение, стандартные свойства имеют типы, описанные в спецификации | Строка |
| Имя | Имя регистрозависимо | Имя регистронезависимо |

Методы для работы с атрибутами:

* elem.hasAttribute(name) – проверить на наличие.
* elem.getAttribute(name) – получить значение.
* elem.setAttribute(name, value) – установить значение.
* elem.removeAttribute(name) – удалить атрибут.
* elem.attributes – это коллекция всех атрибутов.

В большинстве ситуаций предпочтительнее использовать DOM-свойства. Нужно использовать атрибуты только тогда, когда DOM-свойства не подходят, когда нужны именно атрибуты, например:

* Нужен нестандартный атрибут. Но если он начинается с data-, тогда нужно использовать dataset.
* Мы хотим получить именно то значение, которое написано в HTML. Значение DOM-свойства может быть другим, например, свойство href – всегда полный URL, а нам может понадобиться получить «оригинальное» значение.
* Методы для создания узлов:
  + document.createElement(tag) – создаёт элемент с заданным тегом,
  + document.createTextNode(value) – создаёт текстовый узел (редко используется),
  + elem.cloneNode(deep) – клонирует элемент, если deep==true, то со всеми дочерними элементами.
* Вставка и удаление:
  + node.append(...nodes or strings) – вставляет в node в конец,
  + node.prepend(...nodes or strings) – вставляет в node в начало,
  + node.before(...nodes or strings) – вставляет прямо перед node,
  + node.after(...nodes or strings) – вставляет сразу после node,
  + node.replaceWith(...nodes or strings) – заменяет node.
  + node.remove() – удаляет node.
* Устаревшие методы:
  + parent.appendChild(node)
  + parent.insertBefore(node, nextSibling)
  + parent.removeChild(node)
  + parent.replaceChild(newElem, node)

Все эти методы возвращают node.

* Если нужно вставить фрагмент HTML, то elem.insertAdjacentHTML(where, html) вставляет в зависимости от where:
  + "beforebegin" – вставляет html прямо перед elem,
  + "afterbegin" – вставляет html в elem в начало,
  + "beforeend" – вставляет html в elem в конец,
  + "afterend" – вставляет html сразу после elem.

Также существуют похожие методы elem.insertAdjacentText и elem.insertAdjacentElement, они вставляют текстовые строки и элементы, но они редко используются.

* Чтобы добавить HTML на страницу до завершения её загрузки:
  + document.write(html)

После загрузки страницы такой вызов затирает документ. В основном встречается в старых скриптах.

Для управления классами существуют два DOM-свойства:

* className – строковое значение, удобно для управления всем набором классов.
* classList – объект с методами add/remove/toggle/contains, удобно для управления отдельными классами.

Чтобы изменить стили:

* Свойство style является объектом со стилями в формате camelCase. Чтение и запись в него работают так же, как изменение соответствующих свойств в атрибуте "style". Чтобы узнать, как добавить в него important и делать некоторые другие редкие вещи – смотрите [документацию](https://developer.mozilla.org/ru/docs/Web/API/CSSStyleDeclaration).
* Свойство style.cssText соответствует всему атрибуту "style", полной строке стилей.

Для чтения окончательных стилей (с учётом всех классов, после применения CSS и вычисления окончательных значений) используется:

* Метод getComputedStyle(elem, [pseudo]) возвращает объект, похожий по формату на style. Только для чтения.