# Навигация по элементам

Методы поиска — это не единственный способ получить нужные элементы и коллекции. Добраться до элемента можно за счёт ссылок быстрого доступа и «родственных связей».

Начнём со ссылок быстрого доступа.

## Быстрый доступ к коллекциям и элементам

В объекте `document` есть свойства, которые содержат все формы страницы, все ссылки, заголовок документа и другие. Рассмотрим основные из них:

*document.body* — доступ к элементу `<body>`.

*document.documentElement* — доступ к элементу `html`.

*document.forms* — доступ ко всем формам на странице.

```
// document.forms — коллекция (HTMLCollection) форм в текущем документе
// document.forms[0] — первая по счёту форма на странице
const orderForm = document.forms[0];

const orderFormElements = orderForm.querySelectorAll('input');

// подписываемся на событие изменение значений всех полей для ввода нашей формы
Array.from(orderFormElements).forEach((element) => {
  element.addEventListener('change', (evt) => {
    console.log(evt.target.value);
  });
});
```

### Доступ к элементу по идентификатору

Есть возможность получить элемент с помощью быстрой ссылки `document.id` или просто `id`.
Но не смотря на то, что этот способ соответствует стандарту его категорически не рекомендуется использовать для доступа к элементам. Получается, что вы смешиваете пространства имён *DOM* и *JavaScript*.

Наглядный пример, что может произойти, если всё-таки воспользоваться свойством `document.id` и в коде случайно добавить переменную с таким же именем:

```
<p id="item">Полезный текст с описанием товара</p>
```

```
console.log(item); // выводится <p id="item">Полезный текст с описанием товара</p>
```

```
const item = 5; // теперь item равен 5, а не <p id="item">Полезный текст с описанием товара</p>

console.log(item); // 5
```

Этот способ поддерживается в основном для совместимости.

Некоторые типы *DOM*-элементов содержат дополнительные полезные свойства, специфичные только для них, например, тег `<table>`.

Вот некоторые из этих свойств:

— *table.rows* — коллекция строк `<tr>` таблицы;

— *table.caption* — ссылка на элемент `<caption>` таблицы;

— и другие.

Полный перечень свойств для быстрого доступа к коллекциям можно найти в спецификации.

## Родители, дети и соседи

Следующая группа свойств помогает добраться до родственных элементов в *DOM*-дереве: родитель, дочерние элементы и соседи.

*parentElement* — свойство возвращает родителя узла Element или `null`, если узел не имеет родителя или его родитель не Element.

*parentNode* — свойство возвращает «любого» родителя. Родитель элемента может быть Element узлом, Document узлом или DocumentFragment узлом.

Обычно свойства `parentElement` и `parentNode` возвращают один и тот же результат: они оба получают родителя.

За исключением `document.documentElement`:

```
console.log(document.documentElement.parentNode); // выведет document
console.log(document.documentElement.parentElement); // выведет null
```

*children* — вернёт дочерние элементы в виде коллекции *HTMLCollection*. Свойство: `children` мы разбираем в наших тренажёрах «JavaScript в браузере».

```
<ul class="products">
  <li class="product">
    <h2 class="product__title">Перепонки на руки для плавания</h2>
    <img class="product__image" src="img/item-1.jpg" alt="Перепонки на руки для плавания">
    <p class="product__price">1000</p>
  </li>
  <li class="product">
    <h2 class="product__title">Сенфи-ложка</h2>
    <img class="product__image" src="img/item-2.jpg" alt="Сенфи-ложка">
    <p class="product__price">1500</p>
  </li>
  <li class="product">
    <h2 class="product__title">Умки — детектор настроения</h2>
    <img class="product__image" src="img/item-3.jpg" alt="Умки — детектор настроения">
    <p class="product__price">150</p>
  </li>
</ul>
```

```
const cardList = document.querySelector('.products');

console.log(cardList.children);
```

Есть аналогичное свойство `childNodes`, которое возвращает коллекцию типа *NodeList*.

*firstElementChild*, *lastElementChild* — свойства возвращают первый и последний дочерний элемент (Element) соответственно.

```
<ul class="todo">
  <li>Выполнить … лобзиком хоть что-нибудь.</li>
  <li>Повесить … вниз головой под брюхом бешено скачущей лошади, пля при этом из двух пистолетов.</li>
  <li>Ворваться … в пылающее здание и вынести оттуда угорелого пожарного.</li>
  <!-- План минимум тоже, что нужно сделать в наступающем году -->
</ul>
```

```
const todoList = document.querySelector('.todo');

// вывести содержимое последнего дочернего элемента
console.log(todoList.lastElementChild.textContent);

// Результат: Ворваться … в пылающее здание и вынести оттуда угорелого пожарного.
```

*previousElementSibling*, *nextElementSibling* — возвращает предыдущий и следующий элемент того же родителя (предыдущий и следующий соседа).

Есть ещё свойства `previousSibling` и `nextSibling`. Различия между `previousElementSibling` и `previousSibling`, `nextElementSibling` и `nextSibling` схожи с различиями между `parentElement` и `parentNode`, то есть `previousSibling` возвращает предыдущий одноуровневый узел любого типа (элемент, текстовый узел, узел комментария и так далее), а `previousElementSibling` возвращает предыдущий одноуровневый узел-элемент Element (другие типы узлов игнорируются).

```
<!-- Прошел -->
<time datetime="2021-01-01">Вчера</time>
<!-- Сейчас -->
<time id="today" datetime="2021-01-02">Сегодня</time>
<!-- Будет -->
<time datetime="2021-01-03">Завтра</time>
```

```
const today = document.getElementById('today');

// вывести содержимое предыдущего элемента
console.log(today.previousElementSibling.textContent);
// Вчера

// вывести содержимое предыдущего элемента
console.log(today.nextElementSibling.textContent);
// Завтра
```

## Метод `closest`

Ещё один метод, который может быть полезен при работе с DOM — `closest`.

*element.closest(...)* — возвращает ближайший родительский элемент (или сам элемент), который соответствует заданному CSS-селектору или `null`, если таких элементов нет.

Метод удобно использовать при *делегировании*. Делегирование — это такой прием разработки, суть которого заключается в следующем: если у нас есть много элементов, события на которые нужно обрабатывать одним и тем же способом, вместо того, чтобы назначать обработчик каждому, мы назначаем один обработчик на их общий родительский элемент.

Если дочерние элементы имеют сложную структуру (вложенные, дочерние элементы), за счёт `closest` можно исключить обработку события на «лишних» элементах.

Разберём на примере.

У нас есть панель с кнопками для форматирования текста. Каждая кнопка состоит из картинки и текста, который обернут в тег `<span>`.


Панель с кнопками для форматирования текста

Создадим один обработчик для всех кнопок панели: `<div class="tools js-tools-container">`, используем делегирование.

```
const container = document.querySelector('.js-tools-container');

container.addEventListener('click', settingButtonClickHandler);
```

### Наименования классов для элементов, которые обрабатываются в JavaScript-коде.

Чтобы проще было в разметке отделять классы, предназначенные для JS-кода и классы, которые влияют на вёрстку страницы, рекомендуем в названия классов для JS добавлять префикс `js-*`. Так можно разделить зоны ответственности и уменьшить вероятность поломать вёрстку или сделать код не рабочим.

```
<div class="js-tools-container">
...
</div>
```

В обработчике клика будем для кнопки `<button>` добавлять класс `active`. Чтобы отловить элемент, на котором произошло событие, используем свойство `evt.target`. Если клик пришёлся на область тега `<span>`, нам придётся дополнительно выбирать родительский элемент, чтобы добавить класс. А если добавить в код метод `closest`, этого делать не нужно.

Итак, разметка для нашего примера:

```
<div class="tools js-tools-container">
  <button class="button button-img" type="button">
    <svg>...</svg>
    <span>Картинка</span>
  </button>
  <button class="button button-text" type="button">
    <svg>...</svg>
    <span>Текст</span>
  </button>
  <button class="button button-button" type="button">
    <svg>...</svg>
    <span>Кнопка</span>
  </button>
  ...
</div>
```

Код:

```
const container = document.querySelector('.js-tools-container');

const toolsButtonClickHandler = (evt) => {
  // ищем ближайшего родителя, тег button
  const button = evt.target.closest('button');

  if (!button) {
    return;
  }

  button.classList.add('active');
  ...
};

container.addEventListener('click', toolsButtonClickHandler);
```

Подробнее о том, как работает метод, `closest`, читайте в спецификации.

## Ознакомились со статьёй?

Сохранить прогресс