html academy ■ Практикум ▼ 😚 Профессии ▼ У Курсы ▼ У Моё обучение ▼ Работа с DOM в JavaScript Выполнен на О%

В Введение

🖪 Общая теория

📴 Углублённая теория

В Что же такое DOM?

В Поиск элементов в DOM

Навигация по элементам

**DOM** элементов

JavaScript

В Методика работы с DOM

№ Кейс 1, лёгкий уровень

№ Кейс 2, лёгкий уровень

№ Кейс 5, средний уровень

🖸 Кейс 6, средний уровень

№ Кейс 7, средний уровень

🖸 Кейс 8, сложный уровень

№ Кейс 9, сложный уровень

🖸 Кейс 10, сложный уровень

Кейс 3, лёгкий уровень

В Работа со свойствами и содержимым

📴 Реализация плавной прокрутки на

Создание и наполнение DOM элементов

Главная / Моё обучение / Работа с DOM в JavaScript / Углублённая теория /

## 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».

ह Виталий Ризун 🔻

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  $\langle body \rangle$ .

document.documentElement — доступ к элементу html.

document.forms — доступ ко всем формам на странице.

// document.forms — коллекцию (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:

console.log(item); // выводится Полезный текст с описанием товара

Некоторые типы DOM-элементов содержат дополнительные полезные свойства, специфичные только для них,

const item = 5; // теперь item равен 5, а не Полезный текст с описанием товара

например, тег . Вот некоторые из этих свойств:

— table.rows — коллекция строк таблицы;

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

Этот способ поддерживается в основном для совместимости.

Полезный текст с описанием товара

console.log(item); // 5

элементы и соседи.

соответственно.

(предыдущий и следующий сосед).

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

console.log(document.documentElement.parentNode); // выведет document

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

Родители, дети 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

Следующая группа свойств помогает добраться до родственных элементов в DOM-дереве: родитель, дочерние

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

За исключением document.documentElement:

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

<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"> 1000 <h2 class="product\_\_title">Селфи-ложка</h2> <img class="product\_\_image" src="img/item-2.jpg" alt="Селфи-ложка"> 1500 <h2 class="product\_\_title">Ушки - детектор настроения</h2> <img class="product\_\_image" src="img/item-3.jpg" alt="Ушки – детектор настроения"> 2500 const cardList = document.querySelector('.products');

console.log(cardList.children); Есть аналогичное свойство childNodes, которое возвращает коллекцию типа NodeList.

«li»Выпилить ... лобзиком хоть что-нибудь. «li»Повисеть ... вниз головой под брюхом бешено скачущей лошади, паля при этом из двух пистолетов. -li>Ворваться ... в пылающее здание и вынести оттуда угоревшего пожарного. <!-- План-минимум того, что нужно сделать в наступившем году -->

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

const todoList = document.querySelector('.todo'); // вывести содержимое последнего дочернего элемента console.log(todoList.lastElementChild.textContent); // Результат: Ворваться ... в пылающее здание и вынести оттуда угоревшего пожарного.

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

previous Element Sibling, next Element Sibling — возвращает предыдущий и следующий элемент того же родителя

Element (другие типы узлов игнорируются). <!-- Прошло --> <time datetime="2021-01-01">Byepa</time> <!-- Сейчас -->

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

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

Mетод closest

<!-- Будет -->

Метод удобно использовать при делегировании. Делегирование — это такой приём разработки, суть которого заключается в следующем: если у нас есть много элементов, события на которых нужно обрабатывать одним

исключить обработку события на «лишних» элементах.

Панель с кнопками для форматирования текста

container.addEventListener('click', settingButtonClickHandler);

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

<time id="today" datetime="2021-01-02">Сегодня</time>

<time datetime="2021-01-03">Завтра</time>

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

Разберём на примере. У нас есть панель с кнопками для форматирования текста. Каждая кнопка состоит из картинки и текста, который обёрнут в тег <span>.

</> •/A **•** 0 Видео HTML Разделитель Картинка Текст Кнопка Ссылка

Создадим один обработчик для всех кнопок панели <div class="tools js-tools-container">, используем делегирование. const container = document.querySelector('.js-tools-container');

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

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

придётся дополнительно выбирать родительский элемент, чтобы добавить класс. А если добавить в код метод

произошло событие, используем свойство evt.target. Если клик пришёлся на область тега (<span>), нам

Чтобы проще было в разметке отделять классы, предназначенные для JS кода и классы, которые влияют

<div class="tools js-tools-container"> <button class="button button-img" type="button"> <svg>...</svg> <span>Kaртинкa</span> </button> <button class="button button-text" type="button">

<svg>...</svg> <span>Tekct</span> </button> <button class="button button-button" type="button">

button.classList.add(`active`);

<svg>...</svg>

closest, этого делать не нужно.

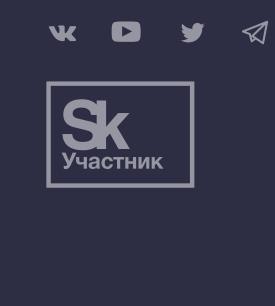
Итак, разметка для нашего примера:

<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;

container.addEventListener(`click`, toolsButtonClickHandler);

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

Ознакомились со статьёй? Сохранить прогресс **<** Поиск элементов в DOM Работа со свойствами и содержимым DOM элементов >



Практикум Курсы Блог HTML и CSS. Профессиональная вёрстка сайтов Тренажёры С чего начать HTML и CSS. Адаптивная вёрстка и автоматизация Шпаргалки для разработчиков Подписка Отчеты о курсах Для команд и компаний JavaScript. Профессиональная разработка веб-интерфейсов Учебник по РНР JavaScript. Архитектура клиентских приложений React. Разработка сложных клиентских приложений Профессии Информация РНР. Профессиональная веб-разработка Фронтенд-разработчик Об Академии React-разработчик РНР и Yii. Архитектура сложных веб-сервисов О центре карьеры Фулстек-разработчик Node.js. Разработка серверов приложений и API Остальное Бэкенд-разработчик

</>
⟨/⟩ → ♥ → ₫₫

Написать нам

Мероприятия

Форум

© ООО «Интерактивные обучающие технологии», 2013–2022

Анимация для фронтендеров

Vue.js для опытных разработчиков

Алгоритмы и структуры данных

Регулярные выражения для фронтендеров

Вёрстка email-рассылок

Шаблонизаторы HTML

Анатомия CSS-каскада

Услуги

Работа наставником

Для учителей

Стать автором