

Цели занятия

- вспомнить что такое DOM, но уже с позиции JavaScript;
- разобраться с тем, как искать элементы на странице с помощью JavaScript;
- научиться работать с событиями и управлять значением полей ввода на странице;

Полезные материалы

- [Подробнее про DOM-дерево](#)
- [Шпаргалка по работе с методами DOM](#)
- [Простым языком про событийную модель](#)

DOM-дерево

DOM (от англ. Document Object Model — «объектная модель документа») — это независимый от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML - и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Основой HTML-документа являются теги.

В соответствии с объектной моделью документа, каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

Все эти объекты доступны при помощи JavaScript, мы можем использовать их для изменения страницы. Например, `document.body` – объект для тега `<body>`.

Если запустить этот код, то `<body>` станет красным на 3 секунды:

```
// сделать фон красным
document.body.style.background = "red";

setTimeout(() => {
  // вернуть назад
  document.body.style.background = "";
}, 3000);
```

Это был лишь небольшой пример того, что может DOM. Скоро мы изучим много способов работать с DOM, а подробнее о структуре DOM можно узнать [вот здесь](#).

Поиск элементов на странице

Но прежде чем каким-либо образом видоизменять элементы с помощью JavaScript или добавлять интерактивность нам необходимо научиться их находить на странице. И если всё содержимое страницы доступно с помощью `document.body`, то получить отдельные элементы может быть чуть сложнее, благо для этого в DOM есть много полезных методов:

- **Поиск по идентификатору** — позволяет найти первый элемент на странице, у которого есть указанный атрибут `id`

```
const element = document.getElementById('someId');
```

- **Поиск по тегу** — позволяет найти сразу все элементы с указанным тегом

```
let divs = document.getElementsByTagName('div');
```

- **Поиск по классу** — находит сразу все элементы, которые содержат указанный класс (атрибут `class`)

```
let buttons = document.getElementsByClassName('button');
```

- **Поиск по селектору** — позволяет гибко искать элементы, основываясь на указанном CSS-селекторе (самый универсальный и удобный способ, в большинстве случаев лучше использовать его)

```
// Вернёт только первый найденный элемент
let element = document.querySelector('ul > li:last-child');

// Вернёт ВСЕ кнопки, которые находятся внутри .sidebar
let elements = document.querySelectorAll('.sidebar .button');
```

Обратите внимание, что есть методы, которые возвращают только один элемент и его сразу можно изменять, а есть методы, которые возвращают несколько элементов, так называемую HTML-коллекцию. В этом случае с коллекцией необходимо работать с помощью циклов:

```
let elements = document.querySelectorAll(".button");

for (let elem of elements) {
    // Выведет текст каждой найденной кнопки
    alert(elem.innerText);
}
```

Смена стилей элемента через JS

После того как мы нашли необходимый элемент на странице и записали в переменную, можно делать с ним что угодно — все изменения будут сразу же применяться на странице. Например, мы можем редактировать стили с помощью JS:

```
const element = document.querySelector(".some-element");

// добавили красную границу в 1 px
element.style.border = "1px solid #ff0000";
```

```
// Скрыли элемент, добавив display: none
element.style.display = "none";

// Убрали значение display: теперь оно
// по умолчанию для этого элемента, или то, которое задано в
// классах
element.style.display = "";
```

Соответственно, эти значения переопределяют стили из классов, потому что стили в атрибуте style самые приоритетные. Но мы также можем добавлять/удалять классы, чтобы применить необходимые стили, записанные в CSS-файле, либо менять любые атрибуты у элемента.

```
let element = document.querySelector(".some-element");

// Добавили класс some-class к элементу element
// Если класс описан в CSS, то его стили применяются к этому
// элементу
element.classList.add("some-class");

// Добавили класс some-other-class к элементу element
element.classList.add("some-other-class");

// Удалили класс some-other-class
element.classList.remove("some-other-class");

// Добавили атрибут к элементу
element.setAttribute("name", "phone");

// Удалили атрибут у элемента
element.removeAttribute("name");
```

В DOM есть метод практически на каждое действие, которое вы можете захотеть выполнить с элементом, но нет смысла заучивать и разбирать их все, гораздо лучше запомнить самые полезные, а

остальные всегда можно подсмотреть в документации или [шпаргалке по работе с DOM](#).

Работа с событиями

Чтобы приложение было интерактивным, нам нужно понимать, что пользователь совершил то или иное действие на странице. Браузер распознает действия пользователя и создаёт *событие*.

События — это сигналы, которые браузер посылает разработчику, а разработчик может на сигнал реагировать. По аналогии со светофором: видим зелёный свет, едем дальше 🚦

События бывают разных типов: клик, нажатие клавиши на клавиатуре, прокрутка страницы и так далее.

Происходящие события можно обрабатывать и выполнять код, когда нужное событие происходит. Например, при клике на кнопку показывать всплывающее окно.

Большинство событий связаны с DOM-элементами. Если пользователь кликнул на кнопку, то событие `click` связано с конкретным DOM-элементом — кнопкой, на которой кликнул пользователь. Причём абсолютно каждый элемент имеет метод `addEventListener`, который позволяет *подписаться на его события*.

```
let button = document.querySelector(".add-to-cart");

button.addEventListener("click", function () {
  alert("Пользователь нажал добавить в корзину!");
});
```

Данный метод позволяет указать **тип события** и функцию, которая выполнится когда произойдёт указанное событие.

Функция-обработчик

Функция-обработчик, или просто обработчик, — это функция, которая вызывается браузером при наступлении события.

При вызове браузер передаёт в обработчик объект события с помощью аргумента.

Объект события — это JavaScript-объект с информацией о событии. В объекте события есть как общие свойства (тип события, время события), так и свойства, которые зависят от типа события (например, на какую кнопку нажал пользователь).

Чтобы работать с объектом события, нужно добавить параметр в объявление обработчика. Обработаем нажатие на кнопки клавиатуры и получим из объекта события информацию о нажатой кнопке:

```
window.addEventListener("keydown", function (event) {  
    // используем объект события, чтобы получить информацию о  
    нажатой клавише  
    alert(`Вы нажали на кнопку: ${event.key}`);  
});
```

Часто используемые типы событий

События мыши

- `click` — происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании).

- `contextmenu` – происходит, когда кликнули на элемент правой кнопкой мыши.
- `mouseenter` / `mouseleave` – когда мышь наводится на / покидает элемент.
- `mousedown` / `mouseup` – когда нажали / отжали кнопку мыши на элементе.
- `mousemove` – при движении мыши.

События на элементах управления

- `submit` – пользователь отправил форму `<form>`.
- `focus` – пользователь фокусируется на элементе, например нажимает на `<input>`
- `blur` — пользователь “уходит” из поля ввода, например, нажали на `<input>`, а потом на какой-то другой элемент
- `input` — пользователь вводит новое значение в поле ввода

```
let input = document.querySelector("input[name=phone]");

// вызывается каждый раз при изменении значения input
input.addEventListener("input", (event) => {
  // event.target.value - текущее значение input
  console.log(event.target.value);

  // Удалили все пробелы и записали как новое значение input
  event.target.value = event.target.value.replaceAll(" ",
  "");
});
```

События документа

- `DOMContentLoaded` – когда HTML загружен и обработан, DOM документа полностью построен и доступен (желательно чтобы ваш скрипт начинал выполняться только когда произошло данное событие)

Поведение по умолчанию

У некоторых элементов есть поведение по умолчанию, которое иногда может вам мешать. Например, при нажатии на ссылку ваш браузер открывает эту ссылку, а при отправке формы браузер перейдёт по адресу, указанному в атрибуте `action` и страница будет обновлена.

Но это можно легко отменить в тех случаях, когда вы хотите назначить своё поведение. Например, вы хотите сделать валидацию формы перед её отправкой и тогда первое, что нужно сделать это выключить поведение по умолчанию. Делается это с помощью метода `preventDefault`:

```
let form = document.querySelector("form");

// вызывается при отправке формы
form.addEventListener("submit", (event) => {
  // Отменяем поведение по умолчанию
  event.preventDefault();

  console.log("Делаем что угодно");
});
```