JavaScript (лекция 2)

О чем будем говорить?

Сухая теория: как получать доступ к элементам на странице с помощью JavaScript? Как их удалять, добавлять и изменять их свойства?

На практике: как узнать, какую опцию в форме выбрал пользователь? Как в зависимости от пользовательских действий (или вне зависимости от них, а, например, по истечению времени) модифицировать контент на странице?

DOM, BOM и JS

Внутри JavaScript иерархия объектов выстроена следующим образом:

- Window
  - **JavaScript** (стандартные инструменты языка)
    (Object, Array, Function, …)
  - **DOM**
    (Document, …)
  - **BOM**
    (Navigator, Screen, Location, Frames, XMLHttpRequest, …)

Корневой объект Window

Является глобальным объектом в JS, содержит функционал для управления окном браузера (не, ну разве это легально?)

```
window.open('http://www.yandex.ru'); // откроется в новой вкладке
```

Объектная модель документа (DOM)

Глобальный объект `document` позволяет взаимодействовать с содержимым страницы: изменять цвет, положение, размеры и другие свойства элементов, создавать и удалять их.

```
document.body.style.backgroundColor = 'red';
```

Объектная модель браузера (BOM)

Сюда входят объекты `navigator`, `location` и наши любимые функции `alert`, `confirm` и `prompt`.

```
alert(location['href']); // отображение текущего URL'a
```

Подробнее о DOM

Основные понятия

Любой (правильно составленный) XML/HTML документ представляет собой **иерархию тэгов**:

```
<!doctype html>
<html>
    <head>
        <meta charset='utf-8'>
        <title>Test DOM tree presentation page</title>
    </head>
    <body>
        <h1>JS is terrible!</h1>
        <p>Please, let me unsee it!</p>
    </body>
</html>
```

Тэги образуют **узлы-элементы**, из них и строится дерево DOM. Внутри них находятся **текстовые узлы** (`#text`), содержащие исключительно строки (включая пробелы и символы перевода каретки), и **узлы-комментарии** (`#comment`).

Все, что есть в HTML, попадает в DOM! Существует **12 типов узлов**, однако большинство операций производится с четырьмя вышеуказанными.

Самый удобный способ посмотреть DOM-структуру — средства разработчика в браузере (**F12**).

```
$1.style.backgroundColor = 'blue';
```

`$1` — предпоследний выбранный на вкладке Elements узел. Чисто для удобства.

Навигация по DOM

Первые шаги

Доступ к любому элементу можно получить, используя объект `document`.

Как получить, например, объект `<html>`?

```
document.documentElement // результат можно присвоить переменной
```

Круто, но если я хочу посмотреть, что внутри? **Как получить потомков?**

```
<element>.childNodes // все потомки первого уровня
<element>.childNodes[i] // i-тый потомок в студию!
<element>.lastChild // последний потомок в коллекции, firstChild — 1й
```

Коллекции – особый тип массивов, в них нет стандартных `pop`, `push`, `forEach`, `map`, зато есть `item(n)` и `length`, которые видны во время итераций по ним (серьезно?).

А что с соседями?

```
<element>.previousSibling // предыдущий сосед, nextSibling — следующий
<element>.parentNode // родитель (мамка)
```

Навигация только по элементам

Принцип работы схож, имена функций содержат подстроку **Element**.

```
<element>.children // все дочерние узлы-элементы
<element>.firstElementChild // первый дочерний узел-элемент
<element>.previousElementSibling // предыдущий элемент-сосед
<element>.parentElement // элемент-родитель
```

Всё, по сути, аналогично. Применяется там, где нет нужды учитывать текст и комментарии.

Более крутая навигация по DOM

getElementById(<id>)

Как получить нужный элемент откуда-то **из глубины документа**? Если известен `id` элемента, можно использовать функцию `getElementById(<'id'>)`:

```
document.getElementById('main_info');
```

Внимание! Следите за **уникальностью** идентификатора! Поведение JavaScript в случае её нарушения неопределено!

getElementsByTagName(<tag>)

А если `id` не знаем, но точно уверены, что нужно найти, например, картинку? Можно получить коллекцию элементов-потомков с подходящим тэгом, используя `getElementsByTagName(<'tag'>)`:

```
document.getElementsByTagName('img');
```

Внимание! Функции далее возвращают `NodeList` или `HTMLCollection`, а значит для получения дальнейшего доступа к объектам необходимо **обращаться к ним по индексу!** Результаты обновляются **динамически**.

getElementsByClassName(<className>)

Исходя из сигнатуры — позволяет получать все элементы, **принадлежащие указанному классу**, даже если классов у них несколько.

```
document.getElementsByClassName('card');
```

Есть еще `getElementsByName(<name>)`, который возвращает элементы с указанным значением атрибута `name`, но на практике он почти не используется.

Хай-левел!

Настоящие профи, которые владеют магией CSS, могут искать элементы **прямо по селекторам**, используя функции `querySelectorAll(<selector>)`, `querySelector(<selector>)` и `closest(<selector>)`:

```
document.querySelectorAll('ul > li:first-child'); // все li, являющиеся первыми потомками у ul'ов

document.querySelector('.goods'); // так же как и getElementsByClassName('goods')[0]

<element>.closest('.chapter') // ближайший .chapter вверх по иерархии, включая стартовый элемент
```

`closest` не поддерживается IE и Edge!

Топ по скорости работы

1. `getElementsById(<id>);`
2. `querySelector(<selector>);` (иногда работает как №1)
3. `querySelectorAll(<selector>);`
4. `getElementsBy*(<smth>).`

Свойства узлов

Получение подробной информации

Лайфхак! Используйте `console.dir(<element>)`: выводится список всех свойств и их значений, удобно для анализа и исследования.

Как отфильтровать только **узлы определенного типа**? `nodeType` содержит информацию о типе узла (см. спецификацию DOM). Только для чтения.

```
if (childNodes[i].nodeType == 8) { ... } // работа только с комментариями
```

`tagName` + `nodeName` — **сведения об имени элемента.** Только для чтения.

```
document.body.firstChild.nodeName );
document.body.firstChild.tagName ); // может быть undefined, если первый потомок — комментарий
```

`innerHTML` — **содержимое элемента** в виде строки. Можно читать и писать.

```
document.body.innerHTML = 'Some content';
```

Важно! **JS-скрипты не выполнятся**, если они будут записаны в `innerHTML` напрямую.

`outerHTML` — **весь элемент** в виде строки. При записи создается новый элемент в DOM, переменные не изменяются.

```
<div>Hello, world!</div>

<script>
    var div = document.body.children[0];
    div.outerHTML = '<p>Some new content</p>';
    alert(div.outerHTML); // <div>Hello, world!</div>
</script>
```

`data` хранит **содержимое текстовых узлов и комментариев**, его можно использовать там, где нет `innerHTML`. Доступно на чтение и на запись.

`<element>.textContent` — **конкатенация всех текстовых узлов (без тэгов)**, полезно использовать для защиты от вставки произвольного HTML-кода.

Свойства vs. атрибуты

Как связаны HTML и JS?

В общем случае **можно создать свои атрибуты** для существующих объектов в JavaScript, однако на отображение в документе они не повлияют. Для работы непосредственно с HTML можно использовать следующие методы:

```
<element>.hasAttribute(name); // есть атрибут?
<element>.getAttribute(name); // получить значение
<element>.setAttribute(name, value); // установить атрибут
<element>.removeAttribute(name); // удалить атрибут
```

Атрибуты и свойства называются по-разному, но между ними происходит постоянная синхронизация. В качестве примера рассмотрим свойства, управляющие классами у элемента:

`className` — **строка с перечислением классов узла**:

```
document.body.className = 'main default';
```

`classList` — **псевдомассив со строками - названиями классов.** Поддерживает дополнительные методы и перебор в цикле:

```
<element>.clsList.contains('className'); // вернуть true/false, в
зависимости от того, есть ли у элемента класс className
<element>.classList.add/remove('className'); // добавить/удалить класс
className
<element>.classList.toggle('className'); // contains + add/remove
```

dataset, data-* атрибуты

Можно в HTML задавать особые атрибуты, которые будут доступны по специальному ключу в JS:

```
<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
    alert(elem.dataset.about); // Elephant
    alert(elem.dataset.userLocation); //
street

</script>
```

Добавлено в HTML5.

Добавление и удаление узлов

Создание узла в JavaScript

Попробуем **создать новый элемент** в коде, для чего воспользуемся методом `document.createElement(<tag>)`, а после определим несколько его свойств:

```
var div = document.createElement('div');
div.className = "alert alert-success";
div.innerHTML = "<strong>Success!</strong> Tests successfully passed";
```

Добавление узла в HTML

Для добавления **в список дочерних узлов** выбранного родителя нужно использовать следующие функции:

```
<parent>.appendChild(div); // добавление в конец
<parent>.insertBefore(div, <parent>.children[2]); // вставка перед
третьим элементом
```

Лень заново создавать?

Похожие элементы можно клонировать, причем со всеми потомками и параметрами:

```
var newDiv = div.cloneNode(true); // true – флаг глубокого копирования
```

Я передумал(а), как удалить?

Очень просто, вызови вот эти функции:

```
<parent>.removeChild(div); // это с концами
<parent>.replaceChild(newDiv, div); // это на случай, если есть чем
заменить
```

Немного о динозаврах

Метод `document.write(<smth>)` работает во время загрузки документа, быстрый и не преобразует передаваемый в него код. Используется, в основном, для отображения рекламы на сайтах.

Доступ к стилям

Так делать не надо!

Вообще модифицирование стилей напрямую через JavaScript считается **плохой практикой**. Гораздо правильнее **переключать с его помощью классы**, определенные в CSS, такое решение более гибкое и не требует дополнительных навыков и знаний.

Как установить CSS свойство?

```
<element>.style.borderLeftWidth = '2px';
document.body.style.display = 'none';
```

Внимание! Единицы измерения обязательны!

Добавление нескольких свойств сразу

```
div.style.cssText = "color: red !important; \
    background-color: yellow; \
    width: 100px; \
    text-align: center; \
";
```

Внимание! Используя эту функцию, имеющиеся свойства заменяются, это целесообразно только на новых элементах, где стилей еще точно нет.

Получение свойств

Имеет смысл учитывать каскадирование, для этого используйте такую функцию:

```
var computedStyle = getComputedStyle(document.body);
alert( computedStyle.marginTop ); // отступ в px
alert( computedStyle.color );
```