JavaScript
(часть 2)

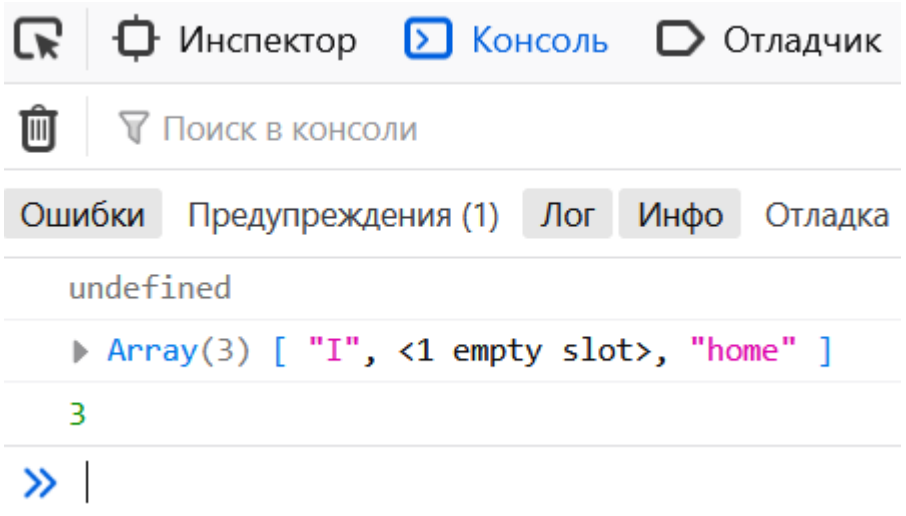
Татаринова А.Г., каф. ПМИ

2025

Методы массивов

- Массив – это объекты в JavaScript
- Пример

```
<html><head><script>
let arr = ["I", "go", "home"];
delete arr[1];
console.log(arr[1] );
console.log(arr);
console.log(arr.length );
</script></head><body></body></html>
```



- Специальные методы для массивов

Метод splice()

- Позволяет добавлять, удалять и заменять элементы
- Синтаксис

```
arr.splice(start [, deleteCount, elem1, …, elemN])
```

изменяет **arr** начиная с индекса **start**: удаляет **deleteCount** элементов и затем вставляет **elem1**, **. . .**, **elemN** на их место

- Возвращает массив из удалённых элементов
- Пример1

```
let arr = [“Я”, “изучаю”, “JavaScript”];
arr.splice(1, 1); //начиная с позиции 1, удалить 1 элемент
alert(arr); //будет выведено [“Я”, “JavaScript”]
```

Метод splice()

- Пример2 удаления 3 элементов и замена их 2 другими

```
let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 3 первых элемента и заменить их другими
arr.splice(0, 3, "Давай", "танцевать");

alert( arr );
// теперь ["Давай", "танцевать", "прямо", "сейчас"]
```

Метод splice()

- Пример3 получение массива из удалённых элементов

```
let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 2 первых элемента
let removed = arr.splice(0, 2);

alert( removed );
// "Я", "изучаю" <-- массив из удалённых элементов
```

Метод splice()

- Пример4 вставка элемента без удаления

```
let arr = ["Я", "изучаю", "JavaScript"];

// с позиции 2
// удалить 0 элементов
// вставить "сложный", "язык«
arr.splice(2, 0, "сложный", "язык");

alert( arr );
// "Я", "изучаю", "сложный", "язык", "JavaScript"
```

Метод splice(): отрицательные индексы

Допускается использование отрицательного индекса, который позволяет начать отсчёт элементов с конца массива

Пример5

```
let arr = [1, 2, 5];

// начиная с индекса -1 (перед последним элементом)
// удалить 0 элементов,
// затем вставить числа 3 и 4

arr.splice(-1, 0, 3, 4); alert( arr ); // 1,2,3,4,5
```

Метод slice

Похож на метод splice

Синтаксис

```
arr.slice([start], [end])
```

Возвращает новый массив, в который копирует элементы, начиная с индекса `start` и до `end` (не включая `end`)

Оба индекса `start` и `end` могут быть отрицательными. В таком случае отсчёт будет осуществляться с конца массива

Метод slice

Пример

```
let arr = ["t", "e", "s", "t"];

alert( arr.slice(1, 3) ); // e,s (копирует с 1 по 3)

alert( arr.slice(-2) ); // s,t (копирует с -2 до конца)
```

Метод concat

Создаёт новый массив, в который копирует данные из других массивов и дополнительные значения

Синтаксис

```
arr.concat(arg1, arg2...)
```

Принимает любое количество аргументов, которые могут быть как массивами, так и простыми значениями

Если аргумент **`argN`** – массив, то все его элементы копируются. Иначе скопируется сам аргумент.

Метод concat

Пример:

```
let arr = [1, 2];

// создать массив из: arr и [3,4]
alert( arr.concat([3, 4]) ); // 1,2,3,4

// создать массив из: arr и [3,4] и [5,6]
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// создать массив из: arr и [3,4],
// потом добавить значения 5 и 6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

Переборный метод forEach

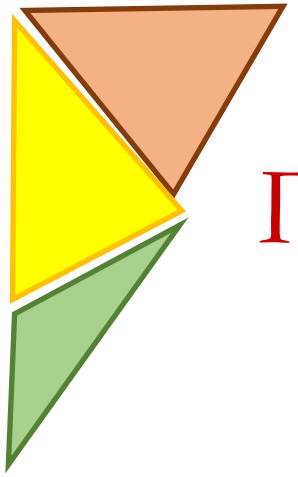
Позволяет выполнять функцию для каждого элемента

Синтаксис:

```
arr.forEach(function(item, index, array) {
  // ... делать что-то с item
});
```

Пример:

```
// Вызов alert для каждого элемента
["Bilbo", "Gandalf", "Nazgul"].forEach(alert);
```



Поиск в массиве

`arr.indexOf(item, from)` ищет `item`, начиная с индекса `from`, и возвращает индекс, на котором был найден искомый элемент, в противном случае -1

`arr.lastIndexOf(item, from)` – то же самое, но ищет справа налево

`arr.includes(item, from)` – ищет `item`, начиная с индекса `from`, и возвращает `true`, если поиск успешен

Поиск в массиве

Пример

```
let arr = [1, 0, false];
alert(arr.indexOf(0)); // 1
alert(arr.indexOf(false)); // 2
alert(arr.indexOf(null)); // -1
alert(arr.includes(1)); // true
```

(!) отличием includes является то, что метод правильно обрабатывает **NaN** в отличие от методов **indexOf**/**lastIndexOf**:

```
const arr = [NaN];
alert( arr.indexOf(NaN) ); // -1 (неверно)
alert( arr.includes(NaN) );// true (верно)
```

Поиск в массиве

Метод **find** позволяет найти объект с определённым условием

Синтаксис:

```
let result = arr.find(function(item, index, array) {
 // если true - возвращается текущий элемент и перебор прерывается
 // если все итерации оказались ложными, возвращается undefined
});
```

Поиск в массиве

Например, есть массив пользователей, каждый из которых имеет поля **id** и **name**.
Требуется найти пользователя с **id == 1**:

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"}
];

let user = users.find(function(x){return x.id == 1});

alert(user.name); // Вася
```

Метод filter

Возвращает элементы массива для которых выполняется условие (фильтрует)

Синтаксис:

```
let results = arr.filter(function(item, index, array) {
  // если true - элемент добавляется к результату,
  // и перебор продолжается
  // возвращается пустой массив в случае, если ничего не найдено
});
```

Метод filter

Пример:

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"}
];

// возвращает массив, состоящий из двух первых пользователей
let users2 = users.filter(function(x){return x.id < 3});

alert(users2.length); // 2
```

Метод map

Вызывает функцию для каждого элемента массива и возвращает массив результатов выполнения этой функции

Синтаксис:

```
let result = arr.map(function(item, index, array) {
  // возвращается новое значение вместо элемента
});
```

Пример:

```
let lengths=["Bilbo","Gandalf","Nazgul"].map(function(x){return x.length});
alert(lengths); // 5,7,6
```

Метод reduce

Вычисляет общее значение на основе всего массива

Синтаксис:

```
let value = arr.reduce(function(preValue, item, index, array) {
  // ...
}, [initial]);
```

Аргументы:

- **preValue** – результат предыдущего вызова функции, равен initial при первом вызове,
- **item** – очередной элемент массива,
- **index** – его индекс,
- **array** – сам массив

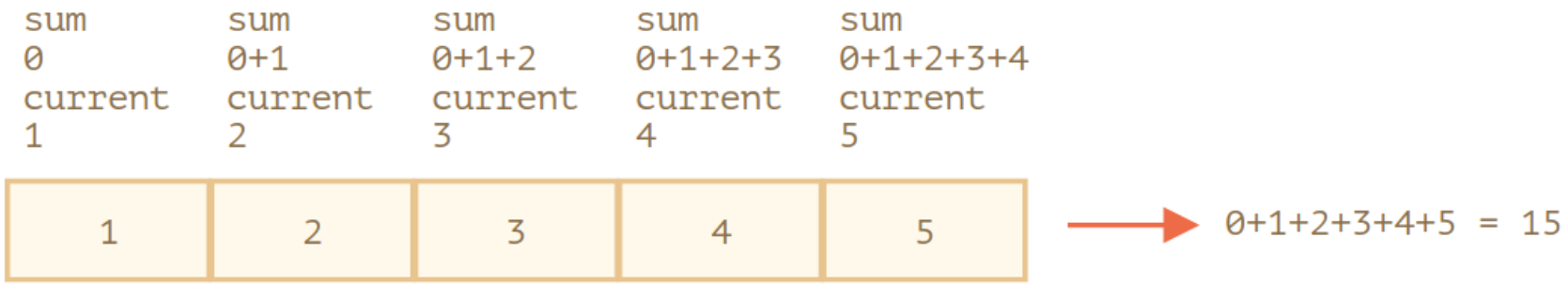
Функция применяется по очереди ко всем элементам массива и «переносит» свой результат на следующий вызов

Метод reduce

Пример нахождения суммы всех элементов массива

```
let arr = [1, 2, 3, 4, 5];
let result = arr.reduce(
                    function(sum, current){return sum + current}, 0
             );
alert(result); // 15
```

Поток вычислений



Метод reduce

Пример нахождения суммы всех элементов массива

```
let arr = [1, 2, 3, 4, 5];
let result = arr.reduce(
                    function(sum, current){return sum + current}, 0
             );
alert(result); // 15
```

Трассировка

	sum	current	result
первый вызов	0	1	1
второй вызов	1	2	3
третий вызов	3	3	6
четвёртый вызов	6	4	10
пятый вызов	10	5	15

Итого по методам массивов

Для добавления/удаления элементов:

push(...items) – добавляет элементы в конец, **pop**() – извлекает элемент с конца, **shift**() – извлекает элемент с начала, **unshift**(...items) – добавляет элементы в начало, **splice**(pos, deleteCount, ...items) – начиная с индекса pos, удаляет deleteCount элементов и вставляет items, **slice**(start, end) – создаёт новый массив, копируя в него элементы с позиции start до end (не включая end), **concat**(...items) – возвращает новый массив: копирует все члены текущего массива и добавляет к нему items. Если какой-то из items является массивом, тогда берутся его элементы.

Для поиска среди элементов:

indexOf/lastIndexOf(item, pos) – ищет item, начиная с позиции pos, и возвращает его индекс или -1, если ничего не найдено, **includes**(value) – возвращает true, если в массиве имеется элемент value, в противном случае false, **find/filter**(func) – фильтрует элементы через функцию и отдаёт первое/все значения, при прохождении которых через функцию возвращается true, findIndex похож на find, но возвращает индекс вместо значения.

Для перебора элементов:

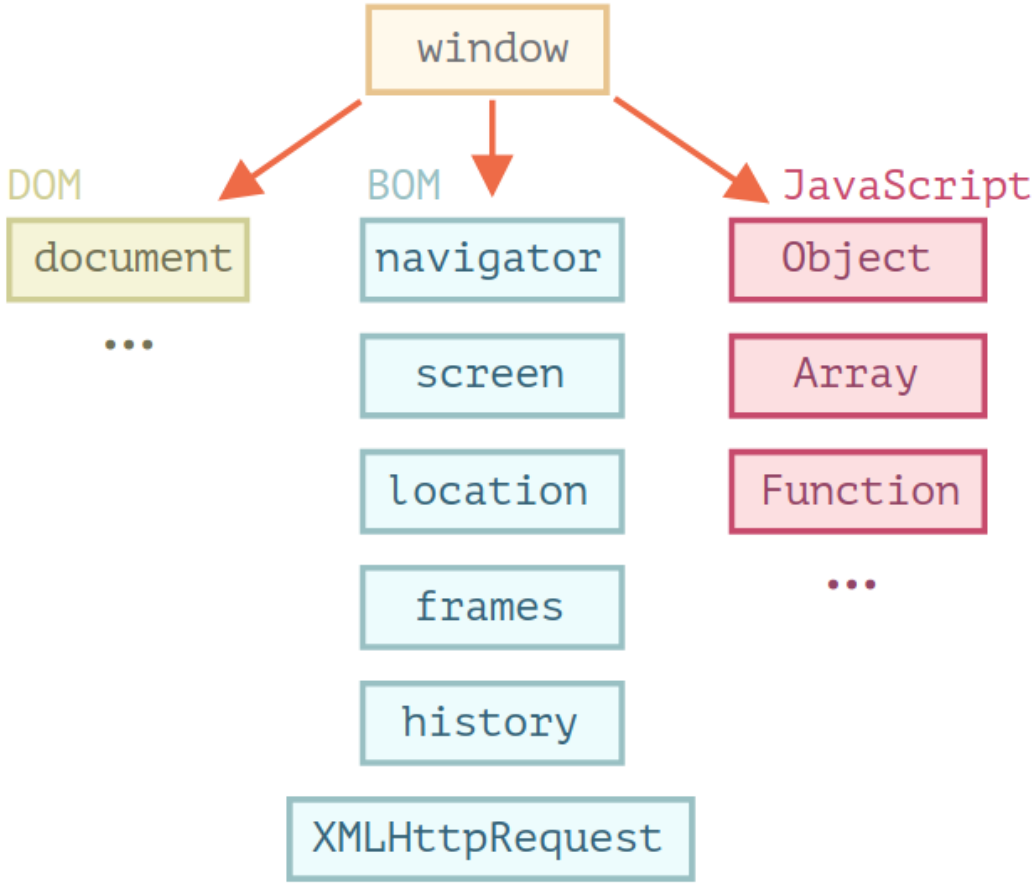
forEach(func) – вызывает func для каждого элемента. Ничего не возвращает.

Для преобразования массива:

map(func) – создаёт новый массив из результатов вызова func для каждого элемента, **sort**(func) – сортирует массив «на месте», а потом возвращает его, **reverse**() – «на месте» меняет порядок следования элементов на противоположный и возвращает изменённый массив, **split/join** – преобразует строку в массив и обратно, **reduce**(func, initial) – вычисляет одно значение на основе всего массива, вызывая func для каждого элемента и передавая промежуточный результат между вызовами.

Обратите внимание, что методы **sort**, **reverse** и **splice** изменяют исходный массив.

Браузерное окружение

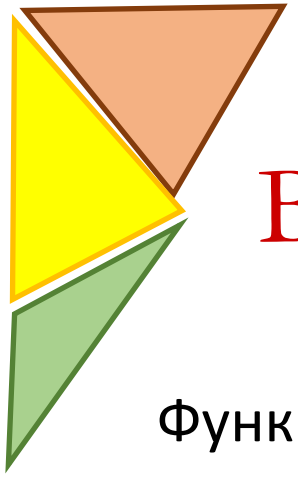


BOM (Browser Object Model)

Объектная модель браузера (Browser Object Model, BOM) – это дополнительные объекты, предоставляемые браузером (окружением), чтобы работать со всем, кроме документа

Пример:

```
alert(location.href); // показывает текущий URL
if (confirm("Перейти на Wikipedia?")) {
  location.href = "https://wikipedia.org";
}
```



BOM (Browser Object Model)

Функции `alert`/`confirm`/`prompt` тоже являются частью BOM: они не относятся непосредственно к странице, но представляют собой методы объекта окна браузера для коммуникации с пользователем

DOM (Document Object Model)

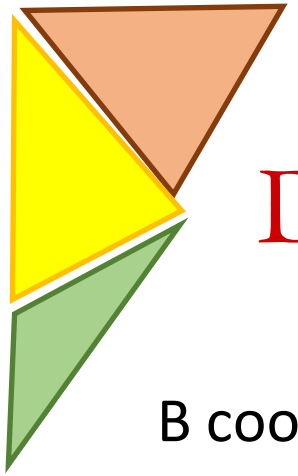
Объектная модель документа, которая представляет все содержимое страницы в виде объектов, которые можно менять

Объект `document` – основная «входная точка». С его помощью можно создавать или менять элементы на странице

Пример:

```
document.body.style.background = "red";

setTimeout(() => document.body.style.background = "", 1000);
```



DOM-дерево

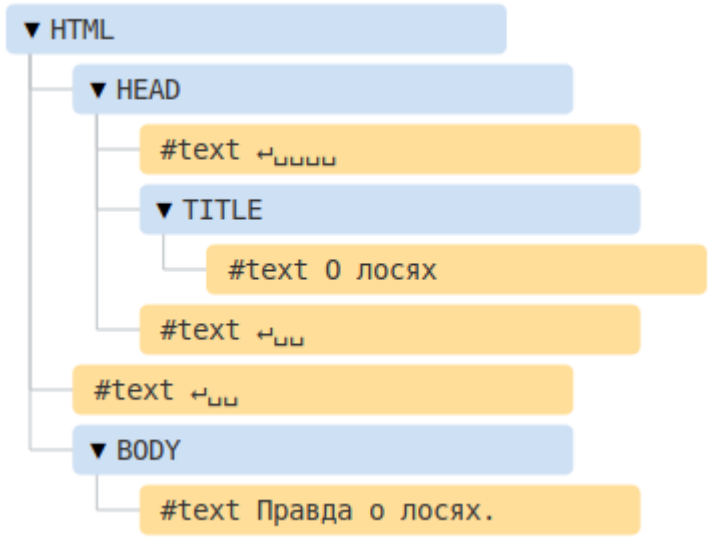
В соответствии с DOM, каждый HTML-тег является объектом

Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом

Данные объекты доступны при помощи JavaScript, их можно использовать для изменения страницы

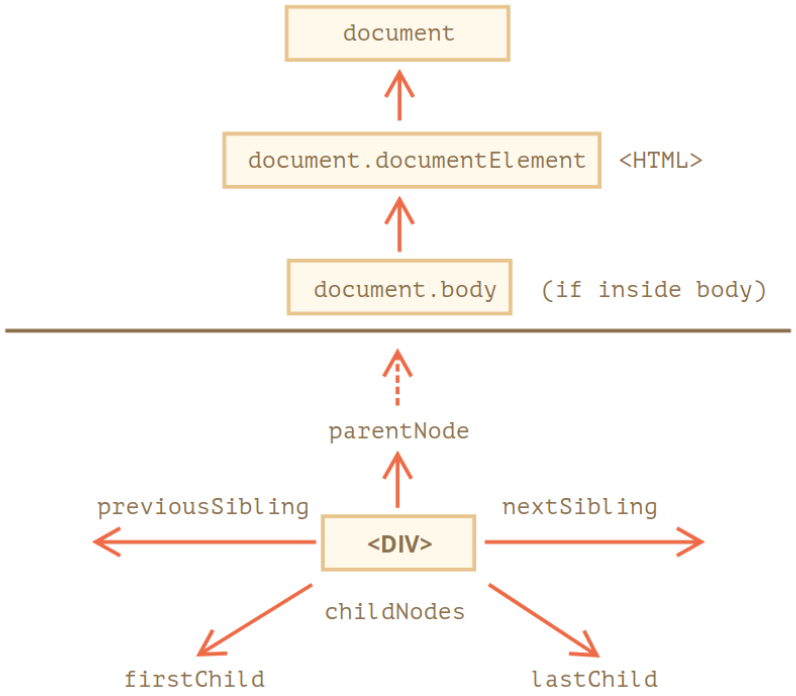
DOM-дерево

```
<!DOCTYPE HTML>
<html>
  <head>
      <title>
         О лосях
      </title>
  </head>
  <body>
      Правда о лосях.
  </body>
</html>
```



В дерево попадают комментарии, директива <!DOCTYPE...>
Выполнение автоисправления

Навигация по DOM-элементам



- Все операции с DOM начинаются с объекта **document** - это главная «точка входа»

Самые верхние элементы дерева доступны как свойства объекта **document**:

<html> = document.documentElement
Самый верхний узел документа: **document.documentElement**
В DOM он соответствует тегу **<html>**

<body> = document.body
Другой часто используемый DOM-узел – узел тега **<body>**:
document.body

<head> = document.head
Тег **<head>** доступен как **document.head**

Навигация по DOM-элементам

- **Дочерние узлы** (**дети**) – элементы, которые лежат непосредственно внутри данного. Например, `<head>` и `<body>` являются детьми элемента `<html>`
- **Потомки** – все элементы, которые лежат внутри данного, включая детей, их детей и т. Д

.

Пример:

```
<html><body>
  <div>Начало</div>
  <ul>
    <li>Информация</li>
  </ul>
  <div>Конец</div>
  <script>
    for (let i = 0; i < document.body.childNodes.length; i++) {
      console.log( document.body.childNodes[i] );
    }
  </script>
  ...какой-то HTML-код...
</body></html>
```

Навигация по DOM-элементам

В примере ранее последним будет выведен элемент `<script>`

Но на самом деле, в документе есть ещё «какой-то HTML-код», но на момент выполнения скрипта браузер ещё до него не дошёл, поэтому скрипт не видит его

Свойства `firstChild` и `lastChild` обеспечивают быстрый доступ к первому и последнему дочернему элементу

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

DOM-коллекции

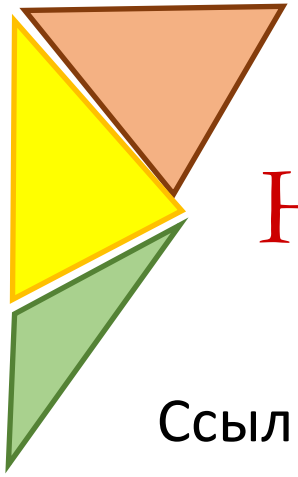
`childNodes` - не массив, а коллекция – особый перебираемый объект-псевдомассив

Для перебора коллекции можно использовать `for..of`:

```
for (let node of document.body.childNodes) {
  alert(node); // покажет все узлы из коллекции
}
```

Методы массивов **не будут** работать, потому что коллекция – это не массив

В `childNodes` находятся и текстовые узлы и узлы-элементы и узлы-комментарии, если они есть



Навигация только по элементам

Ссылки, которые указывают на элементы в DOM-дереве:

- `children` – коллекция детей, которые являются элементами
- `firstElementChild`, `lastElementChild` – первый и последний дочерний элемент
- `previousElementSibling`, `nextElementSibling` – соседи-элементы
- `parentElement` – родитель-элемент

Поиск элементов в DOM – getElementById()

Если у элемента есть атрибут `id`, то можно получить элемент посредством метода `document.getElementById(id)`

Пример:

```
<div id="elem">
  <div id="elem-content">Element</div>
</div>
<script>
  // получить элемент
  let elem = document.getElementById('elem');
  // сделать его фон красным
  elem.style.background = 'red';
</script>
```

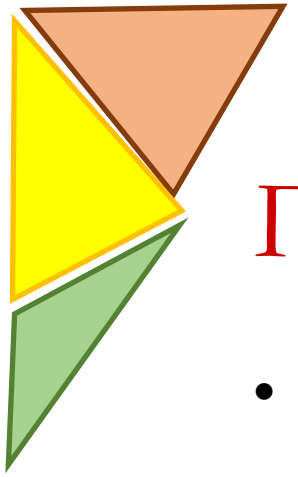
Поиск элементов в DOM – querySelectorAll()

Универсальным методом поиска является `elem.querySelectorAll(css)` - возвращает все элементы внутри `elem`, удовлетворяющие указанному CSS-селектору

Пример: запрос получает все элементы `<li>`, которые являются последними потомками в `<ul>`

```
<ul>
  <li>Этот</li>
  <li>тест</li>
</ul>
<ul>
  <li>полностью</li>
  <li>пройден</li>
</ul>
<script>
  let elements = document.querySelectorAll('ul > li:last-child');

  for (let elem of elements) {
    alert(elem.innerHTML); // "тест", "пройден"  }
</script>
```



Поиск элементов в DOM - querySelector

- Метод `elem.querySelector(css)` возвращает не все, а только **первый элемент**, соответствующий CSS-селектору css
- Аналогично `elem.querySelectorAll(css)[0]`, **но** в последнем сначала ищутся все элементы, а потом берётся первый, а в `elem.querySelector(css)` ищется только первый
- **Используется**, когда заведомо известно, что подходящий элемент только один

Поиск элементов в DOM – matches()

Проверяет, удовлетворяет ли **elem** селектору CSS возвращает **true**, либо **false**

Ранее в спецификации назывался **matchesSelector**, и большинство браузеров поддерживают его под этим старым именем, либо с префиксами ms/moz/webkit

```
<a href="http://example.com/file.zip">...</a>
<a href="http://ya.ru">...</a>

<script>
  var elems = document.body.children;

  for (var i = 0; i < elems.length; i++) {
    if (elems[i].matches('a[href$="zip"]')) {
      alert( "Ссылка на архив: " + elems[i].href );
    }
  }
</script>
```

Поиск элементов в DOM – closest()

Ищет ближайший элемент выше по иерархии DOM, подходящий под CSS-селектор

Сам элемент тоже включается в поиск

Старые браузеры его плохо поддерживают

```
<ul>
  <li class="chapter">Глава I
    <ul>
      <li class="subchapter">Глава <span class="num">1.1</span></li>
      <li class="subchapter">Глава <span class="num">1.2</span></li>
    </ul>
  </li>
</ul>

<script>
  var numberSpan = document.querySelector('.num');

  // ближайший элемент сверху подходящий под селектор li
  alert(numberSpan.closest('li').className) // subchapter

  // ближайший элемент сверху подходящий под селектор .chapter
  alert(numberSpan.closest('.chapter').tagName) // LI

  // ближайший элемент сверху, подходящий под селектор span
  // это сам numberSpan, так как поиск включает в себя сам элемент
  alert(numberSpan.closest('span') === numberSpan) // true
</script>
```

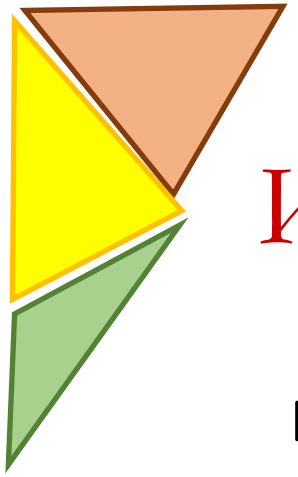
Поиск элементов в DOM

Ещё методы для поиска элементов:

`elem.getElementsByTagName(tag)` ищет элементы с данным тегом и возвращает их коллекцию. Передав "*" вместо тега, можно получить всех потомков

`elem.getElementsByClassName(className)` возвращает элементы, которые имеют данный CSS-класс

`document.getElementsByName(name)` возвращает элементы с заданным атрибутом `name`. Очень редко используется



Изменение страницы

DOM-узел можно **создать** двумя методами:

```
document.createElement(tag)
```

Создаёт новый элемент с заданным тегом:

```
let div = document.createElement('div');
```

```
document.createTextNode(text)
```

Создаёт новый текстовый узел с заданным текстом:

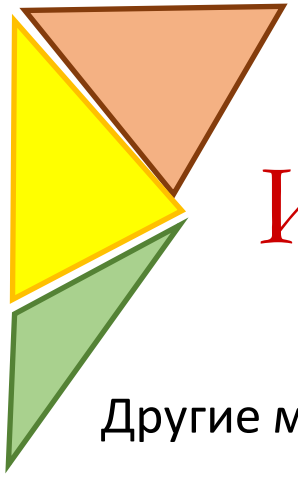
```
let textNode = document.createTextNode('HI');
```

Изменение страницы

Чтобы элемент вставить в **document** используется метод **append**

Пример:

```
<style>
.alert {  padding: 15px;   border: 1px solid #d6e9c6;   border-radius: 4px;
  color: #3c763d;   background-color: #dff0d8; }
</style>
<script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Всем привет!</strong> Вы прочитали важное
  сообщение.";
  document.body.append(div);
</script>
```



Изменение страницы

Другие методы вставки:

`node.append(...nodes or strings)` – добавляет узлы или строки в конец **`node`**;

`node.prepend(...nodes or strings)` – вставляет узлы или строки в начало **`node`**;

`node.before(...nodes or strings)` – вставляет узлы или строки до **`node`**;

`node.after(...nodes or strings)` – вставляет узлы или строки после **`node`**;

`node.replaceWith(...nodes or strings)` – заменяет **`node`** заданными узлами или строками

Изменение страницы: пример

```
<html><body>
<ol id="ol1">   <li>0</li>   <li>1</li>   <li>2</li> </ol>
<script>
  ol1.before('before');//вставить строку "before" перед <ol>
  ol1.after('after'); //вставить строку "after" после <ol>
  let liFirst = document.createElement('li');
  liFirst.innerHTML = 'prepend';
  ol1.prepend(liFirst); //вставить liFirst в начало <ol>
  let liLast = document.createElement('li');
  liLast.innerHTML = 'append';
  ol1.append(liLast); //вставить liLast в конец <ol>
</script></body></html>
```

before

1. prepend
2. 0
3. 1
4. 2
5. append

after

Изменение страницы

Для **удаления** узла есть методы **`node.remove()`**

Например, пусть сообщение будет удаляться через 1 с.:

```
<style>
.alert {  padding: 15px;  border: 1px solid #d6e9c6;  border-radius:
  4px;  color: #3c763d;  background-color: #dff0d8; }
</style>
<script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Всем привет!</strong> Вы прочитали важное
 сообщение.";
  document.body.append(div);
  setTimeout(() => div.remove(), 1000);
</script>
```

Задача

Пусть дан массив

```
[{name:"Alice", age:25, job:"developer"},
 {name:"Bob", age:23, job:"developer"},
 {name:"Eva", age:22, job:"tester"},
 {name:"Mike", age:26, job:"admine"}]
```

Выведите на страницу маркированный список из элементов массива

- Alice
  - age - 25
  - job - developer
- Bob
  - age - 23
  - job - developer
- Eva
  - age - 22
  - job - tester
- Mike
  - age - 26
  - job - admine

```
<html><head><script>
  var persons = [{name:"Alice", age:25, job:"developer"},
{name:"Bob", age:23, job:"developer"}, {name:"Eva", age:22,
job:"tester"},{name:"Mike", age:26, job:"admine"}];
   function showPersons(){
     let t = document.createElement('ul');
        persons.forEach(function(p){
           let l = document.createElement('li');
           l.innerHTML = p.name;
              let subul = document.createElement('ul');
            for(let k in p){
                 if(k != "name"){
                     let subli = document.createElement("li");
                     subli.innerHTML = k + " - " + p[k];
                     subul.appendChild(subli)}
            }
            l.appendChild(subul);
            t.appendChild(l);
     });
     document.body.appendChild(t);}
</script></head><body onload="showPersons()"></body></html>
```

Обработчики событий

- Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM)
- Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло (именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя)
- Примеры событий:
  - 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
  - mousemove – при движении мыши
  - DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен
  - keydown и keyup – когда пользователь нажимает / отпускает клавишу

Обработчики событий

- **Использование атрибута элемента HTML**
  Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>
  ```
  <input value="Нажми меня" onclick="alert('Клик!')"
  type="button">
  ```
- **Использование свойства DOM-объекта**
  Можно назначать обработчик, используя свойство DOM-элемента on<событие>
  (!) Обработчик всегда хранится в свойстве DOM-объекта, а атрибут – лишь один из способов его инициализации
  ```
  button.onclick = function() {
      alert('Клик!');
  };
  ```
  Внутри обработчика события `this` ссылается на текущий элемент, то есть на тот, на котором назначен обработчик
  Основной недостаток описанных выше способов назначения обработчика – невозможность «повесить» несколько обработчиков на одно событие

```
<html><head><script>
function add_person(){
   person_list = document.getElementById("persons");
   new_person_elem = document.createElement('li');
   new_person_elem.innerText = `${document.forms[0].elements.person_firstname.value}
                        (${document.forms[0].elements.person_lastname.value})`;
   document.forms[0].elements.person_firstname.value = "";
   document.forms[0].elements.person_lastname.value = "";
   let btn_remove_person = document.createElement('input');
   btn_remove_person.type = "button";
   btn_remove_person.value = "Remove";
   btn_remove_person.onclick = function(){this.parentNode.remove()};
   new_person_elem.append(btn_remove_person);
   person_list.append(new_person_elem);
}
</script></head><body>
<form>
<input name="person_firstname" type="text" placeholder="First Name" />
<input name="person_lastname" type="text" placeholder="Last Name" />
<input name="btn_add" type="button" value="Add" onclick="add_person()"/>
</form>
<ul id="persons"></ul>
</body></html>
```

Обработчики событий

- **addEventListener**
  `element.addEventListener(event, handler, [options]);`
  `event` - имя события
  `handler` - ссылка на функцию-обработчик
  `options` - дополнительный объект со свойствами:

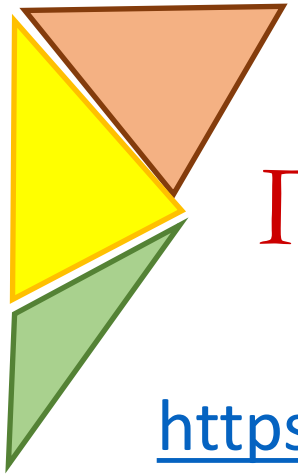
- Существуют события, которые нельзя назначить через DOM-свойство, но можно через `addEventListener`.
  Например, событие `DOMContentLoaded`, которое срабатывает, когда завершена загрузка и построение DOM документа

Обработчики событий

- Чтобы хорошо обработать событие, могут понадобиться детали того, что произошло. Не просто «клик» или «нажатие клавиши», а также – какие координаты указателя мыши, какая клавиша нажата и так далее.
- Когда происходит событие, браузер создаёт объект события, записывает в него детали и передаёт его в качестве аргумента функции-обработчику.
- Пример:

```
<input type="button" value="Нажми меня" id="elem">
<script>
  elem.onclick = function(event) {
    // вывести тип события, элемент и координаты клика
    alert(event.type + " на " + event.currentTarget);
    alert("Координаты: " + event.clientX + ":" + event.clientY);
  };
</script>
```

```
<html><head><script>
    function add_person(){
        persons_list = document.getElementById("persons");
        new_person_elem = document.createElement('li');
        new_person_elem.innerText = `${document.forms[0].elements.person_firstname.value}
        (${document.forms[0].elements.person_lastname.value})`;
        document.forms[0].elements.person_firstname.value = "";
        document.forms[0].elements.person_lastname.value = "";
        let btn_remove_person = document.createElement('input');
        btn_remove_person.type = "button";
        btn_remove_person.value = "Remove";
        btn_remove_person.addEventListener('click', (event) =>
        event.currentTarget.parentNode.remove());
        new_person_elem.append(btn_remove_person);
        persons_list.append(new_person_elem);
    }
    </script></head><body>
    <form><input name="person_firstname" type="text" placeholder="First Name" />
    <input name="person_lastname" type="text" placeholder="Last Name" />
    <input name="btn_add" type="button" value="Add" onclick="add_person()"/></form>
    <ul id="persons"></ul>
    </body></html>
```



Полезные источники

https://learn.javascript.ru/array-methods