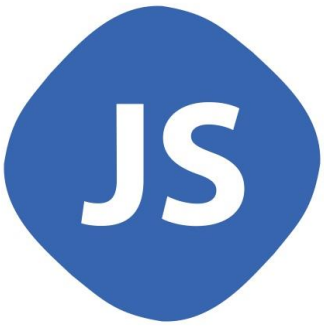


Выбор элементов DOM

- Введение
- Выбор элементов по CSS селектору
- Методы getElement*
- Дополнительные способы получения коллекций

Введение

Работа со страницей, так или иначе, связана с **манипулированием DOM элементами**. Но для того, чтобы с этими элементами работать их необходимо сначала **выбрать** или получить.

Рассмотренные нами в предыдущих уроках свойства узлов дают скорее общее представление о структуре HTML-документа, чем способ навигации по нему. **Пошаговая выборка** узлов слишком трудоемкий способ получения доступа.

Как достать произвольный элемент откуда-то из документа? Как быстро получить информацию о свойствах и атрибутах элемента?

Для этого в DOM есть свойства и методы для выборки **одного** или **коллекции** элементов.

Найти несколько DOM-элементов и получить к ним доступ из JavaScript можно разными способами: querySelectorAll, getElementsByTagName, children и так далее.

В итоге в каждом случае будет возвращена **коллекция** — сущность, которая похожа на массив объектов, но при этом им не является, на самом деле это набор DOM-элементов. Стоит учесть, что фактически разные методы возвращают **разные коллекции**:

- **HTMLCollection** — коллекция непосредственно HTML-элементов.

- **NodeList** — коллекция узлов, более абстрактное понятие. Например, в DOM-дереве есть не только узлы-элементы, но также текстовые узлы, узлы-комментарии и другие, поэтому NodeList может содержать другие типы узлов.

Разница между HTMLCollection и NodeList

NodeList и **коллекция HTML** — это одно и то же (почти). Оба представляют собой массивоподобные коллекции (списки) узлов (элементов), извлеченных из документа. Доступ к узлам можно получить по порядковым номерам. Индекс начинается с 0.

Оба имеют свойство **длины**, которое возвращает количество элементов в списке (наборе).

HTMLCollection — это коллекция **элементов документа**.

NodeList — это набор **узлов документа** (узлов элементов, узлов атрибутов и текстовых узлов).

Доступ к элементам HTMLCollection можно получить по их **имени**, **идентификатору** или **номеру индекса**.

Доступ к элементам NodeList возможен только по их **порядковому номеру**.

Наиболее часто используемые JavaScript методы манипулирования объектами:

- **querySelector**
- **querySelectorAll**

Они выполняют поиск элементов в DOM по CSS-селектору. Они отличаются друг от друга только тем, что querySelectorAll возвращает все найденные элементы, а querySelector - только первый из них.

Кроме этого, в API DOM имеются ещё другие методы для выбора элементов. Это так называемые старые методы доступа к элементам:

- **getElementById**
- **getElementsByClassName**

- **getElementsByTagName**
- **getElementsByName**

Сейчас они используются крайне редко, т.к. их действия можно очень просто выполнить с помощью **querySelectorAll** и **querySelector**.

Выбор элементов по CSS селектору

querySelectorAll

querySelectorAll – метод, который позволяет найти все элементы (коллекцию) по CSS селектору внутри **всей страницы** или **внутри конкретного элемента**, для которого он вызывается.

Синтаксис

```
elements = document.querySelectorAll('selector');
elements = element.querySelectorAll('selector');
```

Параметр

- **selector** – это строковый аргумент, представляющий собой **CSS селектор**, в соответствии с которым нужно найти элементы.

Пример:

```
// выбор элементов div
var elements = document.querySelectorAll('div');
// выбор элементов li которые находятся внутри элемента ul
var elements = document.querySelectorAll('ul li');
```

Метод **querySelectorAll** возвращает все найденные элементы в виде статической коллекции (живые и статические коллекции рассматриваются позже).

Узнать количество элементов в коллекции можно с помощью свойства **length**.

example_1. Получение коллекции

```
<script>
	// выберем все элементы p внутри div#citate
	var p = document.querySelectorAll('div#citate p');
	// коллекция NodeList узлов p
	console.log(p);
	// узнаем количество найденных элементов
	var length = p.length;
	console.log("Количество элементов в коллекции: ", length);
</script>
```

example_2. Еще один способ получения коллекции

```
<script>
	// выберем блок div#citate
	var div = document.querySelector('#citate');
	// выберем все элементы p внутри div#citate
	var p = div.querySelectorAll("p");
	// коллекция NodeList узлов p
	console.log(p);
	// узнаем количество найденных элементов
	var length = p.length;
	console.log("Количество элементов в коллекции: ", length);
</script>
```

Важно. **Метод querySelectorAll всегда возвращает коллекцию**. И даже если элемент, соответствующий селектору на странице, будет один, будет

возвращена **коллекция из одного элемента**. Это очень важно. К элементу коллекции необходимо обращаться по индексу.

Обратиться к определённому элементу в коллекции можно по его индексу. Индексы начинаются с 0.

Обращение к коллекции по индексу возвращает DOM элемент, находящийся под указанным индексом или **undefined**, если элемента с таким индексом нет.

example_3. Доступ к элементам коллекции

```
<script>
    // выбор всех элементов tr в документе
    var set_tr = document.querySelectorAll('tr');
    // выбор четвертой строки
    var tr = set_tr[3];
    // выбор всех элементов td внутри выбранной строки tr
    var set_td = tr.querySelectorAll('td');
    // выбор второй ячейки
    var td = set_td.item(1);
    // выведем информацию по выбранному узлу
    console.log("Узел: ", td);
    console.log("Тип узла: " + td.nodeType);
    console.log("Имя узла: " + td.nodeName);
    console.log("Значение узла: ", td.innerText);
</script>
```

example_4. HTMLCollection и NodeList

```
<script>
```

```
    // выберем элемент div#citate
    var div = document.querySelectorAll('div#citate')[0];
    nodes = div.querySelectorAll("p");
    // коллекция NodeList
    console.log(nodes);
    elements = div.children;
    // коллекция HTMLCollection
    console.log(elements);
</script>
```

querySelector

querySelector – метод, который также как **querySelectorAll** осуществляет поиск по **CSS селектору**, но в отличие от него возвращает не все найденные элементы, а только **первый из них**.

Этот метод часто используется, когда мы заведомо знаем, что подходящий элемент только один, и хотим получить его в переменную.

Синтаксис

```
element = document.querySelector('selector');

element = element.querySelector('selector');
```

Параметр

- **selector** – строка, содержащая CSS селектор, в соответствии с которым необходимо найти элемент.

Результат метода **querySelector** аналогичен **querySelectorAll('selector')[0]**, но в отличие от него он выполняет это намного быстрее. Это происходит потому, что **querySelector** сразу же останавливает процесс поиска, как только находит соответствующий элемент. В то время как **equerySelectorAll('selector')[0]** сначала находит все элементы, и только

после того как он все их нашёл мы уже можем обратиться к этим элементам и с помощью оператора доступа (квадратные скобки) взять первый из них.

В качестве результата метод **querySelector** возвращает ссылку на **объект** или **null** (если элемент не найден).

Метод **querySelector** возвращает **статический** элемент.

example_5. Выбор одного элемента по селектору

```
<script>
	// поиск по имени тега, тег - p
	let el1 = document.querySelector('p');
	console.log(el1); // найден первый элемент p
</script>
```

example_6. Способы составления селектора

```
<script>
	// поиск по имени тега, тег - p
	let el1 = document.querySelector('p');
	console.log(el1); // найден первый элемент p
	// поиск по идентификатору, id='auth'
	let el2 = document.querySelector('#auth');
	console.log(el2);
	// поиск по классу, class='sel'
	let el3 = document.querySelector(".sel");
	console.log(el3); // null - не найдено элементов
	// поиск по типу элемента, type='email'
	let el4 = document.querySelector("[type='email']");
	console.log(el4);
```

```
    // поиск по имени элемента, name='theme'
    let el5 = document.querySelector("[name='theme']");
    console.log(el5);
    // поиск по свойству value, value='master@mail.ru'
    let el6 =
    document.querySelector("[value='master@mail.ru']");
    console.log(el6);
</script>
```

Методы getElement*

getElementById

Метод **getElementById**() возвращает **элемент** (тег) страницы по его атрибуту id. В качестве результата метод **getElementById** возвращает ссылку на **объект** или значение **null**, если элемент с указанным идентификатором не найден.

С полученным элементом можно будет производить различные манипуляции: менять его текст, атрибуты, css стили и так далее.

Синтаксис:

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

Параметр:

- **tagID** - аргумент, который представляет атрибут **id** тега.

Указания значения **id** необходимо выполнять с учётом регистра, т.к., например, **main** и **Main** – это разные значения.

Кроме этого, в соответствии со стандартом в документе **не может быть** несколько элементов с одинаковым значением атрибута **id**, т.к. значение идентификатора должно быть уникальным.

Пример:

```
// получим элемент с id="list" и сохраним ссылку в переменную
// list
var list = document.getElementById('list');
```

Тем не менее, если вы допустили ошибку и в документе существуют несколько элементов с одинаковым id, то метод **getElementById** более вероятно вернёт **первый** элемент, который он встретит в коде (DOM). Но на это полагаться нельзя, т.к. такое поведение не прописано в стандарте.

example_7. Метод getElementById

```
<script>
    // получение элемента через getElementById
    var el = document.getElementById('citate');
    console.log(el);
    // получение коллекции дочерних элементов
    var elements = el.children;
    console.log(elements);
    console.log("Всего выбрано элементов: ", elements.length);
</script>
```

Действие метода **getElementById** можно выполнить с помощью **querySelector**.

```
// получение элемента через getElementById
var form = document.getElementById('form');
// получение элемента через querySelector
var form = document.querySelector('#form');
```

getElementsByClassName

Метод **getElementsByClassName**() возвращает **коллекцию** (HTMLCollection) элементов с определенным классом. Находит элемент и в том случае, если у него **несколько классов**, а искомый – один из них.

Этот метод позволяет найти все элементы с указанными классами во всём документе или в некотором элементе.

Синтаксис:

```
elements = document.getElementsByClassName('names');
elements = element.getElementsByClassName('names');
```

Параметр:

- **names** - строка, состоящая из одного или нескольких классов разделённых между собой с помощью пробела.

Метод **getElementsByClassName** позволяет искать элементы не только по одному имени класса, но и по нескольким, которые должны быть у элемента.

example_8. Метод getElementByClassName

```
<script>
    // получение коллекции через getElementsByClassName
    var elements = document.getElementsByClassName('selected');
    console.log("Количество элементов: ", elements.length);
    console.log("1 элемент коллекции: ", elements[0].innerText);
    console.log("2 элемент коллекции: ", elements[1].innerText);
    console.log("3 элемент коллекции: ", elements[2].innerText);
</script>
```

Действие метода **getElementsByClassName** можно очень просто выполнить с помощью **querySelector**.

```
// получение элемента через getElementsByClassName
```

```
var elem = document.getElementsByClassName('second');
// получение элемента через querySelector
var elem = document.querySelector('.second');
```

getElementsByTagName

Метод **getElementsByTagName**() предназначен для получения **коллекции** элементов по имени тега.

Синтаксис:

```
elements = document.getElementsByTagName('tagName');
elements = element.getElementsByTagName('tagName');
```

Параметр:

- **tagName** - строка с названием тега, который нужно найти. Регистр тега не имеет значения. Чтобы выбрать все элементы можно использовать символ *.

example_9. Метод getElementsByTagName

```
<script>
    // получение коллекции по имени тега - h2
    var h2 = document.getElementsByTagName('h2');
    console.log(h2);
    // получение коллекции по имени тега - table
    var table = document.getElementsByTagName('table');
    // выбор тегов tr внутри таблицы
    var set = table.item(0).getElementsByTagName('tr');
    console.log(set);
</script>
```

Действие метода **getElementsByTagName** можно очень просто выполнить с помощью **querySelector**.

```
// получение элемента через getElementsByTagName
var elem = document.getElementsByTagName('li');
// получение элемента через querySelector
var elem = document.querySelector('li');
```

getElementsByName

Метод **getElementsByName**() может применяться, когда вам нужно выбрать **коллекцию** элементов, имеющих атрибут **name** с указанным значением.

Синтаксис:

```
element = document.getElementsByName('value');
```

Параметр:

- **value** - строка со значением атрибута **name** в соответствии с которым нужно найти элементы.

example_10. Метод getElementsByName

```
<script>
    // выбрать элементы с name="prog"
    var elems = document.getElementsByName('prog');
    // коллекция элементов
    // к коллекции обращаться необходимо по индексу
    console.log(elems);
    // вывод значения второго чекбокса
    console.log(elems[1].value);
    // выбор элемента с нестандартным атрибутом
```

```
    var elems = document.getElementsByName('test');
    console.log(elems[0].id);  // citate
    console.log(elems[0].name);  // undefined
</script>
```

Действие метода **getElementsByName** можно выполнить с помощью **querySelector**.

```
// получение элемента через getElementsByName
var elem = document.getElementsByName('login');
// получение элемента через querySelector
var elem = document.querySelector("[name='login']");
```

Дополнительные способы получения коллекций

forms

Свойство **forms** возвращает **коллекцию** всех элементов **form** в документе.

Синтаксис

```
// возвращает коллекцию всех элементов form в документе
document.forms;
// возвращает количество элементов в коллекции
document.forms.length;
// возвращает элемент с указанным индексом
// или пустое значение, если индекс находится вне диапазона
document.forms[index];
// возвращает элемент с указанным индексом
```

```
// или пустое значение, если индекс находится вне диапазона
document.forms.item(index);
```

example_11. Коллекция forms

```
<script>
    alert("Количество форм на странице: " +
    document.forms.length);
    console.log("Первая форма: ", document.forms[0]);
    console.log("Вторая форма: ", document.forms.item(1));
    // получение коллекции форм
    let frms = document.forms;
    // вывод коллекции
    console.log(frms);
    // коллекция элементов второй формы
    let set = frms[1].children;
    console.log(set);
</script>
```

images

Свойство **images** возвращает **коллекцию** всех элементов **img** в документе.

Синтаксис

```
// возвращает коллекцию всех элементов img в документе
document.images;
// возвращает количество элементов в коллекции
document.images.length;
// возвращает элемент с указанным индексом
// или пустое значение, если индекс находится вне диапазона
```

```
document.images[index];
// возвращает элемент с указанным индексом
// или пустое значение, если индекс находится вне диапазона
document.images.item(index);
```

example_12. Коллекция images

```
<script>
    document.write("Количество изображений на странице: ",
    document.images.length);
    // изменим размер изображений
    document.images.item(0).style.width = "250px";
    document.images.item(1).style.width = "250px";
    document.images.item(2).style.width = "250px";
    // выберем блок-контейнер изображений
    let div = document.querySelector("#img-container");
    // применим к блоку flex-позиционирование
    div.style.cssText = "display:flex; flex-direction:row;
    justify-content:center;";
</script>
```

links

Свойство **links** возвращает **коллекцию** всех **a** и **area** элементов в документе , которые имеют атрибут **HREF**.

Синтаксис

```
// возвращает коллекцию элементов которые имеют атрибут href
document.links;
// возвращает количество элементов в коллекции
document.links.length;
```

```
// возвращает элемент с указанным индексом
// или пустое значение, если индекс находится вне диапазона
document.links[index];
// возвращает элемент с указанным индексом
// или пустое значение, если индекс находится вне диапазона
document.links.item(index);
```

example_13. Коллекция links

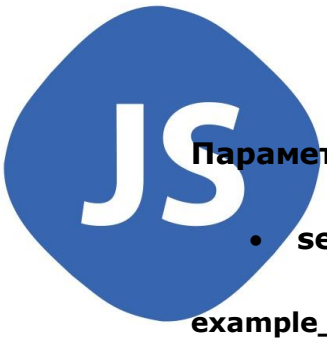
```
<script>
    document.write("Количество ссылок на странице: ",
    document.links.length);
    // изменим свойства ссылок
    document.links.item(0).style.color = "red";
    document.links.item(1).style.textDecoration = "none";
    document.links.item(2).style.cursor = "alias";
    document.links.item(3).style.fontFamily = "Arial";
    // получим в переменную коллекцию ссылок
    let set = document.links;
    // скроем вторую ссылку
    // set[1].hidden = true;
</script>
```

closest

Метод **closest** ищет ближайший родительский элемент, подходящий под указанный CSS селектор (сам элемент тоже включается в поиск).

Синтаксис

```
let parent = element.closest('selector');
```



Параметр

- **selector** – строка, представляющая CSS-селектор для поиска родителя.

example_14. Метод Closest

```
<script>
	// выбираем элемент
	let login = document.querySelector('[name="login"]');
	// определяем родителя, получаем его id
	let id = login.closest("form").getAttribute("id");
	// выводим id родителя
	console.log(id);
</script>
```

example_15. Практика применения метода Closest

```
<script>
	// выбираем элемент
	let el = document.querySelector('.sel');
	// находим ближайшего родителя tr
	let set = el.closest("tr");
	// получаем коллекцию потомков
	console.log ("Вы выбрали следующую строку:");
	console.log (set.children);
</script>
```

contains

Метод **contains** позволяет проверить, содержит ли один элемент внутри себя другой.

Синтаксис

```
let bool = parent.contains(element);
```

Параметр

- **element –** элемент для поиска внутри родительского.

Возвращаемое значение

Возвращает **true**, если узел является потомком узла, в противном случае **false**.

example_16. Метод Contains

```
<script>
    // выбираем первый элемент
    let pict = document.querySelector('#picture');
    // выбираем второй элемент
    let sel = document.querySelector('.selected');
    // определяем, является ли второй элемент потомком первого
    console.log(pict.contains(sel));
</script>
```

Самостоятельные задания к уроку:

- Выбор элементов с использованием свойств и методов DOM.
- Создание JS-объекта, комментарии в DOM.
- Выбор родительских элементов.
- Динамическая стилизация объектов DOM.
- Фрагмент модуля **Клиентское программирование** региональных чемпионатов по компетенции Веб-технологии.

P.S.

Для отработки и закрепления учебного курса **донам** группы предоставляется следующий раздаточный материал.

К каждому уроку курса:

- Файлы **демонстрационного кода** (example);
- **Задачи** с решениями в контексте рассматриваемых вопросов урока (task).

К каждой теме курса:

- **Практические** работы.