DOM

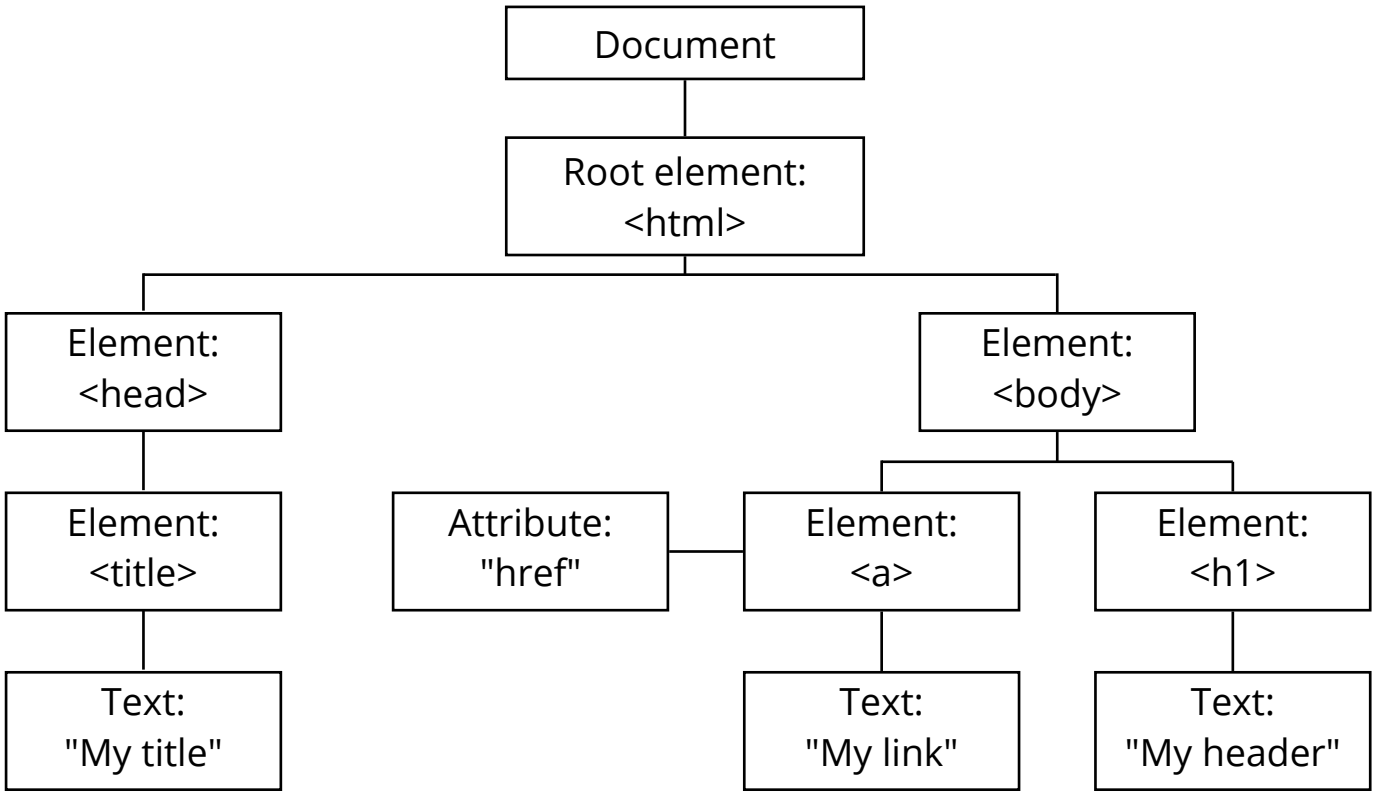
LoftSchool
от мыслителя к создателю

Содержание

1

DOM

Document Object Model (DOM) - объектная модель документа. Иными словами можно сказать что DOM - это некая иерархия (структура) документа, а каждый тег или текст образует ее.



На картинке изображена структура, на которой изображены узлы DOM-дерева. **Есть главный элемент, есть потомки,** у потомков (например, у head или body) есть свои потомки, а у тех, в свою очередь - свои.

Есть разные типы узлов DOM-дерева, но нас интересуют лишь два из них: **узел элемента** и **текстовый узел**.

Узел элемента - это тег. ***Текстовый узел*** - это обычный текст.

Эти знания нам пригодятся далее, в манипуляциях с DOM- элементами. Например, нам нужно будет создать новый узел, удалить, поменять местами, добавить класс и т.д.

Есть одна интересная особенность - текстовые узлы появляются между тегами, если в коде есть пробелы или переводы строки.

Например,

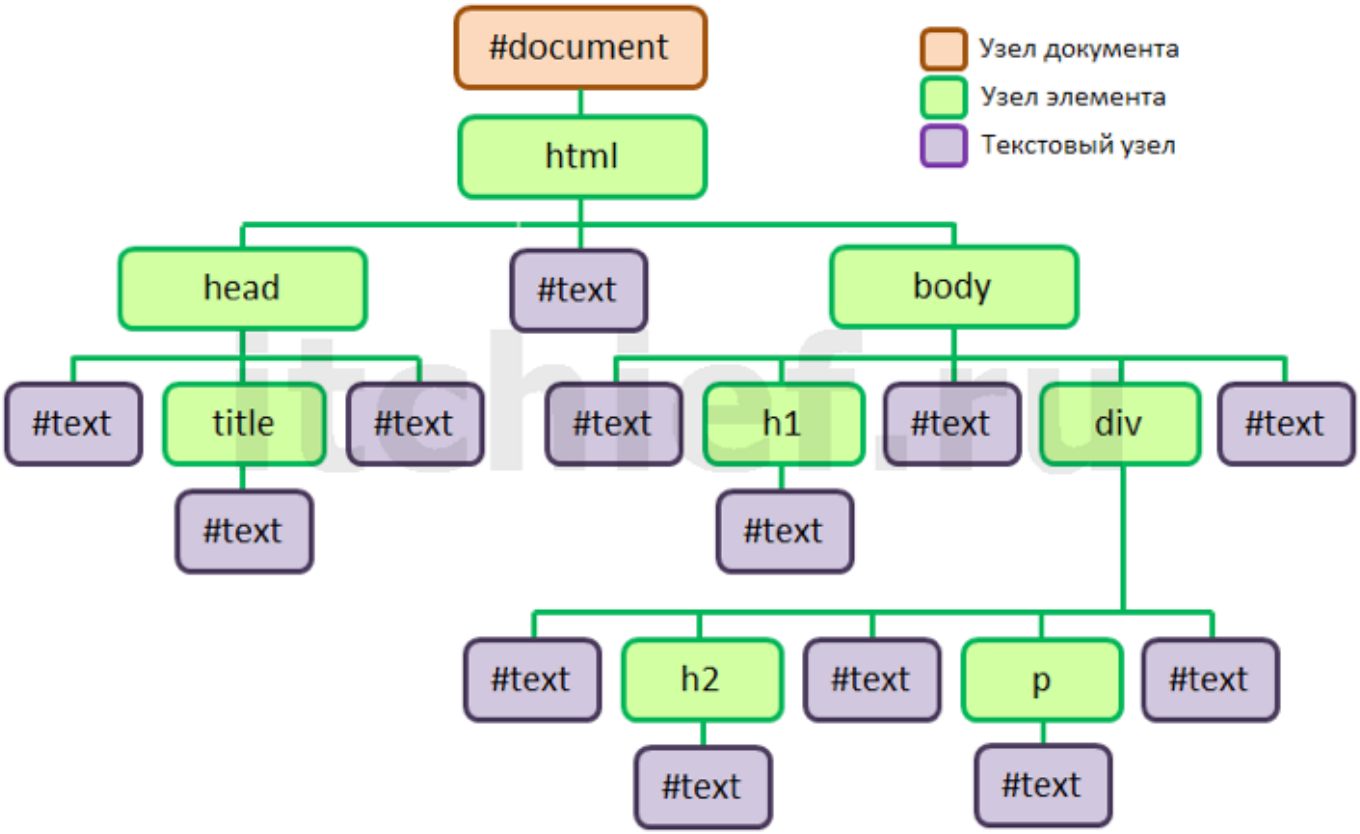
```
<p>Element</p> // здесь – перевод строки, а значит, между двумя узлами P будет создан еще один, текстовый узел

<p>Element</p>
```

В этом примере будет два (не берем в расчет все родительские узлы - body, html и т.д.) узла элемента и 3 узла текста. Два из текстовых узлов - это содержимое тега <p>, третий - это перевод строки, он-то и создаёт нам еще один текстовый узел.

Если бы было вот так:

```
<p>Element</p><p>Element</p>
```

то здесь уже не было бы перевода строк, поэтому у нас было бы 2 узла элемента <p> и 2 текстовых узла (их содержимое).


#document
html
head
#text
body
#text
title
#text
#text
#text
h1
#text
#text
div
#text
#text
#text
h2
#text
#text
p
#text
#text
Узел документа
Узел элемента
Текстовый узел

2

Поиск элементов

Для манипуляции с теми или иными узлами DOM-дерева существуют специальные методы, которые мы и рассмотрим.

```
var element = document.getElementById(elementId); //
найдет в документе элемент, у которого id = elementId
и вернет его (либо вернет null, если такой элемент не
найден)
```

Следующие методы возвращают коллекцию элементов. ***Node-коллекции или NodeList*** - это некое подобие массивов, но не массив, а объект. Общим у node-коллекций с массивами является только свойство **length**, и то, что к их содержимому также можно обращаться по индексу. Но таких методов, как pop, push - у node-коллекций нет.

```
var elements = document.getElementsByTagName('li'); //
Вернет все элементы <li>

var items = document.getElementsByClassName('item'); //
Вернет все элементы, у которых есть класс item
```

Есть более универсальные методы - querySelector и querySelectorAll. Им, в качестве параметра, можно передать любой css-селектор. Разница между ними только в том, что первый метод вернет только один элемент, удовлетворяющий селектору, а второй метод вернет все элементы удовлетворяющие селектору.

Например:

```
var phone = document.querySelector('.phone') // Вернет
первый элемент, у которого есть класс phone

var phones = document.querySelectorAll('#someId .phone')
// вернет все элементы с классом phone, которые находятся
в элементе с id '#someId
```

В **querySelector** и **querySelectorAll** можно подставить любой css-селектор, поддерживаемый текущим браузером.

Мы рассмотрели то, как выбрать элемент по его **id**, **class**, **name** и **два универсальных метода**. На самом деле методов значительно больше, но на практике чаще всего применяются именно эти.

3

Отношения между элементами

Теперь давайте посмотрим, как можно находить элементы относительно выбранного.

Если посмотреть на DOM-дерево, то почти у любого узла может быть соседний, дочерний и родительский узлы.

Каждый узел в дереве имеет ссылки на эти узлы.
Эти ссылки хранятся в свойствах узла:

Ссылки на соседние узлы

previousSibling - ссылка на предыдущий узел-сосед
nextSibling - ссылка на предыдущий узел-сосед

Здесь нужно обратить внимание на то, что если предыдущий или следующий соседи будут текстовыми узлами, то именно текстовый узел и выберется.

Если же мы хотим получить именно элемент-сосед, тогда нужно воспользоваться **previousElementSibling** или **nextElementSibling**

previousElementSibling - ссылка на предыдущий элемент-сосед (узел с типом “элемент”)
nextElementSibling - ссылка на следующий элемент-сосед (узел с типом “элемент”)

Посмотрим следующий пример.
Разметка:

```
<p>first elem</p>
<p class="second">second elem</p>
<p>third elem</p>
```

Код:

```
var second = document.querySelector('.second');

console.log(second.previousSibling); //ссылка
на текстовый узел между <p>first elem</p> <p
class="second">second elem</p>

console.log(second.nextSibling); //ссылка на текстовый
узел между <p class="second">second elem</p> и <p>third
elem</p>

console.log(second.previousElementSibling); //ссылка на
<p>first elem</p>

console.log(second.nextElementSibling); //ссылка на
<p>third elem</p>
```

Выполните код и убедитесь сами.

Ссылки на дочерние элементы

firstChild - ссылка на первый дочерний узел
lastChild - ссылка на последний дочерний узел

firstElementChild - ссылка на первый дочерний элемент(узел с типом “элемент”)
lastElementChild - вытащит последний дочерний элемент(узел с типом “элемент”)

childNodes - ссылки на дочерние узлы
children - ссылка на дочерние элементы(узелы с типом “элемент”)

Вернемся к нашему примеру, только обернем параграфы в тег div:

```
<div>
  <p>first elem</p>
  <p class="second">second elem</p>
  <p>third elem</p>
</div>
```

Теперь выполните в консоли такой код:

```
var div = document.querySelector('div');

console.log(div.childNodes);
console.log(div.children);

console.log(div.firstChild);
console.log(div.lastChild);

console.log(div.firstElementChild);
console.log(div.lastElementChild);
```

И внимательно посмотрите на то, что вывелось в каждом случае.

Ссылка на родительский узел

parentNode - ссылка на узел-родитель

Для примера возьмем разметку из примера выше и выполним такой код:

```
var second = document.querySelector('.second');

console.log(second.parentNode); //выведет div
```

4

Атрибуты и содержимое элементов

Атрибуты

Давайте разберем следующие методы для работы с атрибутами:

element.hasAttribute(attributeName) - возвращает **true** либо **false** в зависимости от того, есть ли у элемента атрибут с указанным именем:

```
<div class="first"></div>

var element = document.querySelector('.first');
console.log(element.hasAttribute('class')); // true
```

element.getAttribute(attributeName) - возвращает значение переданного атрибута

```
var element = document.querySelector('.first');

console.log(element.getAttribute('class'));  // first
```

element.setAttribute(attributeName, attributeValue) - создает/изменяет указанный атрибут с указанным значением

```
var element = document.querySelector('.first');

element.setAttribute('id', 'new');
console.log(element.getAttribute('id')); // new
```

element.removeAttribute(attributeName) - удаляет заданный атрибут

Содержимое

Итак, мы разобрали то, как можно найти тот или иной элемент и как работать с атрибутами.

Теперь давайте поговорим о том, как работать с содержимым.
Например, у нас есть такая разметка:

```
<div class="first" id="last">
  <p>text first</p>
  <p>text second</p>
</div>
```

Для того, чтобы получить весь внутренний код вместе с тегами, воспользуемся свойством **innerHTML**:

```
var element = document.querySelector('.first');

console.log(element.innerHTML);
```

Так же, с помощью этого свойства, мы можем заменить внутренний код элементам:

```
element.innerHTML='<span>new text</span>';
```

5

Создание, вставка и удаление DOM-узлов

С помощью Javascript мы можем создавать узлы, добавлять их в другие узлы, удалять, менять.

document.createElement(tagName) - создает новый html-элемент

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

Мы создали новый html-элемент **div** и присвоили его переменной **div**. После этого мы можем производить дальнейшие действия с этим элементом.

```
div.innerHTML = '<p>Это только что созданный div с
текстом внутри</p>';
```

Для того, чтобы добавить узел в определенное место, мы можем воспользоваться следующими методами:

parent.appendChild(node) - добавить **node** в конец **parent**

```
document.body.appendChild(div); //добавит в конец body
элемент div
```

Следующий метод для вставки, это - ***parent.insertBefore(node, before)***
Если переводить на человеческий язык, то получится примерно следующее: вставить **node** в **parent** перед элементом **before**

Например:

```
<div class="section">
  <p>text-1</p>
  <p>text-2</p>
  <p>text-3</p>
</div>
```

```
var section = document.querySelector('.section'),
    secondP = section.children[1],
    newDiv = document.createElement('div');

section.insertBefore(newDiv, secondP);
```

Приведенный выше код вставить новый **div** перед ** text-2 **.

Для того, чтобы удалить узел, можно воспользоваться методом ***removeChild***

Для этого нужно применить его к родительскому блоку, у которого мы удаляем элемент и передать удаляемый элемент:

```
section.removeChild(newDiv); // удалили наш созданный
узел из блока с классом section
```

6

HTML-формы

После того как мы научились работать с узлами, давайте немного поговорим о формах.

Как вы помните, мы можем добраться до элементов по классам, идентификаторам и других селекторов.

К формам, помимо выше указанных вариантов, мы можем обратиться с помощью свойства **forms** объекта **document**.

Например:

```
var secondForm = document.forms[1]; //выбрали вторую форму
```

Так же мы можем выбрать форму по ее имени, например:

```
<form action="" name="registration"></form> // форма в html разметке
```

```
var formRegistration = document.forms.registration;

console.log(formRegistration);
```

Как вы наверняка знаете, элементы форм - это **input**, **select** и т.д. К ним так же очень легко получить доступ.

```
<form action="" name="registration">
<input type="radio" name="lang" value="ru">ru
<input type="radio" name="lang" value="en">en
</form>
```

```
var formRegistration = document.forms.registration;

console.log(formRegistration.lang[0]); // выводим 1-й
input
console.log(formRegistration.lang[1]); // выводим 2-й
input
```

То есть мы просто обращаемся к нужной форме, указываем значение атрибута name и порядковый номер начиная с 0, как в массивах.

То же самое справедливо и для **option**:

```
<form action="" name="registration">
  <select name="country">
<option value="USA">USA</option>
<option value="RU">RU</option>
<option value="UKR">UKR</option>
  </select>
</form>
```

```
var formRegistration = document.forms.registration;

console.log(formRegistration.county[1]; // выбрали option
RU
```

Для **input** и **textarea** существует свойство **value**:

```
var input = document.querySelector('.input-name');

input.value = 'Alex';
```

7

jQuery. Работа с DOM

jQuery - это JS-библиотека, которая упрощает работу с DOM элементами, а также предоставляет множество дополнительных удобств для разработки, помогает решить часть проблем, которые возникают из-за различий браузеров (кроссбраузерность).

Чтоб обратиться к jQuery, достаточно написать **jQuery** либо **$**.

Для того, чтобы выбрать тот или иной элемент (или группу элементов) на странице, обращаемся к jQuery и передаем нужный селектор.

Давайте посмотрим это на примерах:

`jQuery('p');`	выбрать все элементы p
`jQuery('.className');`	выбрать все элементы с классом className
`jQuery('#idName');`	выбрать все элементы с id idName
`jQuery('ul li');`	выбрать все элементы с li, которые находятся в элементе ul

То же самое можно сделать с помощью знака $:

`$('p');`	выбрать все элементы p
`$('.className');`	выбрать все элементы с классом className
`$('#idName');`	выбрать все элементы с id idName
`$('ul li');`	выбрать все элементы с li, которые находятся в элементе ul

Зная css-селекторы, мы просто передаем их объекту jQuery.

Мы можем применять к элементу те или иные методы: изменить содержимое, задать ширину, высоту, добавить/удалить/изменить атрибут и т.д.

Ранее мы посмотрели, как выбирать соседей, родителей, потомков на чистом Javascript, теперь давайте посмотрим, как это делается с помощью jQuery.

closest() - Находит ближайший, соответствующий заданному селектору элемент, из числа следующих: сам выбранный элемент, его родитель, его прародитель, и так далее, до начала дерева DOM.

prev() - Находит элементы, которые лежат непосредственно перед каждым из выбранных элементов.

next() - Находит элементы, которые лежат непосредственно после каждого из выбранных элементов.

siblings() - Находит все соседние элементы (под соседними понимаются элементы с общим родителем).

first() - Возвращает первый элемент в наборе.

last() - Возвращает последний элемент в наборе.

Давайте посмотрим, как это может нам пригодится.

Например, нам нужно скрыть все соседние элементы, но оставить видимым второй элемент.

```
<div>first</div>
<div class="second">second</div>
<div>third</div>
<div>fourth</div>
```

```
$('.second').siblings().hide();
```

Давайте посмотрим, как можно вытащить текст из выбранного элемента:

```
<p>Lorem ipsum dolor sit amet, consectetur adipisicing
elit. Dolorem laudantium, modi molestiae consectetur! At
dolore possimus optio temporibus dolor, eum est debitis
qui, totam soluta ratione veritatis, quos tenetur.
Eum necessitatibus vero, nam accusamus eos minima ea
quibusdam deserunt laboriosam autem. At cum quod,
laboriosam ab dolorum, alias illo iste.</p>
```

```
var text = $('p').text();

console.log(text);
```

И еще некоторые методы, которые могут пригодится:

attr() - возвращает/изменяет (в зависимости от числа параметров) значение атрибута у элементов на странице;

removeAttr() - удаляет атрибут у элементов на странице;

addClass() - добавляет класс элементам на странице;

removeClass() - удаляет класс(ы) у элементов на странице;

toggleClass() - изменяет наличие класса у элементов на противоположное (добавляет/удаляет);

css() - Возвращает/изменяет (в зависимости от числа входных параметров) CSS параметры элемента;

html() - Возвращает/изменяет (в зависимости от числа параметров) html-содержимое элементов на странице;

text() - Возвращает/изменяет (в зависимости от числа параметров) текст, находящийся в элементах на странице;

remove() - Удаляет элементы на странице.