

Работа с DOM

Урок 7





Кадочников Алексей

Frontend-разработчик

- ☀ Веб-разработчик со стажем более 9 лет
- ☀ Преподаватель GeekBrains с 2015 года
- ☀ Автор курсов по Frontend на портале Geekbrains
- ☀ Работал в таких компаниях, как VK и Wizard-C



План курса

1

Знакомство с JavaScript

2

Основы JavaScript

3

Функции в JavaScript

4

Циклы и массивы

5

Работа с объектами в JavaScript

6

Введение в DOM

7

Работа с DOM

8

Основы событий в JavaScript

9

Работа с событиями в JavaScript

10

Основы шаблонизации, работа с JSON

11








Работа с медиафайлами

12

Основы API, итоги курса



Что будет на уроке сегодня

-  Управление стилями
-  Навигация по элементам DOM-дерева
-  Иерархия интерфейсов DOM
-  Интерфейсы коллекций в DOM
-  Операторы spread и rest
-  Работа с коллекциями
-  Методы для навигации по дереву DOM



Управление стилями





Управление стилями

Есть несколько способов управления стилями при использовании JavaScript.

Первый способ — добавить встроенные стили прямо на элементы, которые мы хотим динамически стилизовать. Для этого применяется свойство **HTMLElement.style**. Оно содержит встроенную информацию о стиле для каждого элемента документа. Можно установить свойства этого объекта для прямого обновления стилей элементов.

Важно! Версии свойств JavaScript стилей CSS пишутся в нижнем регистре верблюжьего стиля (lower camel case), в то время как версии свойств стилей CSS используют дефисы кебаб-стиля (kebab case), например, `backgroundColor` и `background-color`. Их нельзя путать!

Пример:

```
1 const divElement = document.createElement('div')
2 const paragraphElement = document.createElement('p')
3 divElement.appendChild(paragraphElement)
4
5 paragraphElement.style.color = 'white'
6 paragraphElement.style.backgroundColor = 'black'
7 paragraphElement.style.padding = '10px'
8 paragraphElement.style.width = '250px'
9 paragraphElement.style.textAlign = 'center'
10
```

Перезагрузим страницу и увидим, что стили применяются к абзацу. Если посмотреть на этот параграф в инспекторе своего браузера, окажется, что эти строки действительно добавляют встроенные стили в документ:

```
1 <p style="color: white; background-color: black; padding: 10px;
  width: 250px; text-align: center;">... </p>
2
```



Управление стилями

Второй способ — динамического управления стилями документа.

1. Удалим предыдущие пять строк, добавленных в JavaScript.
2. Добавим в элемент `<head>` следующее содержимое:

```
1
2 <style>
3   .paragraph {
4     color: white;
5     background-color: black;
6     padding: 10px;
7     width: 250px;
8     text-align: center;
9   }
10 </style>
11
```

Теперь перейдём к очень полезному методу для общего манипулирования HTML:

Element.setAttribute().

Этот метод принимает два аргумента: имя и значение атрибута, устанавливаемый для элемента. Укажем в нашем абзаце имя класса выделения:

```
1
2 paragraphElement.setAttribute('class', 'paragraph')
3
```

Установка атрибута `class` через метод `setAttribute` — не единственный и, возможно, не самый идиоматичный способ. Современные браузеры поддерживают свойства `Element.className` и более мощный и функциональный `Element.classList`.



Преимущества и недостатки

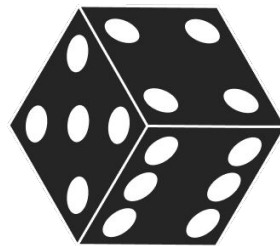
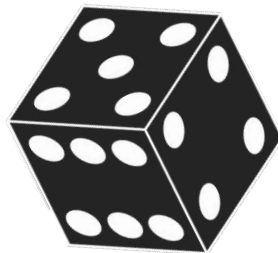
1. Добавить встроенные стили прямо на элементы.

Преимущества:

- Принимает меньше настроек
- Хорош для простого использования

Недостатки:

- Смешивает работу CSS и JavaScript





Преимущества и недостатки

1. Добавить встроенные стили прямо на элементы.

Преимущества:

- Принимает меньше настроек
- Хорош для простого использования

Недостатки:

- Смешивает работу CSS и JavaScript



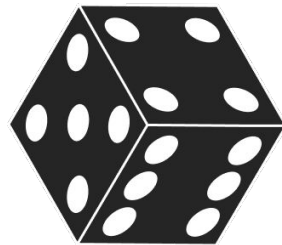
2. Управлять стилями документа динамически.

Преимущества:

- Более чистый, то есть без смешивания CSS и JavaScript
- Без встроенных стилей

Недостатки:

- Большое количество настроек для добавления стилей



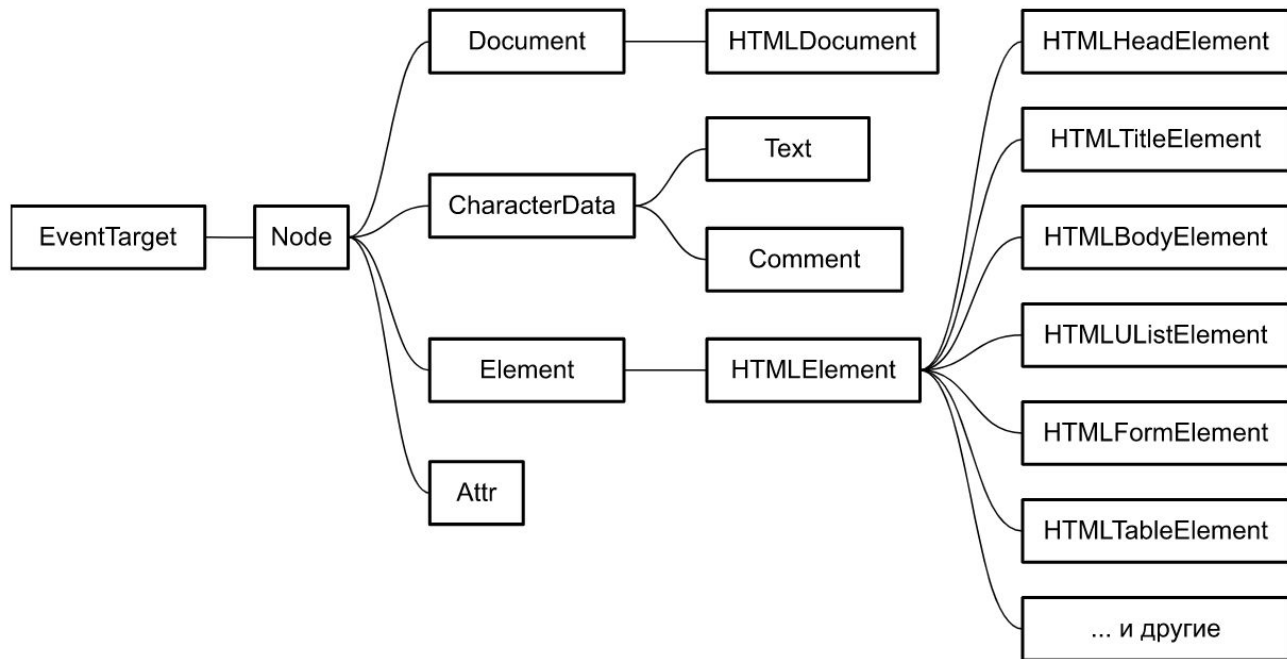


Навигация по элементам DOM- дерева





Иерархия интерфейсов DOM





Node

Node — это базовый класс для всех DOM-интерфейсов, в том числе для Document. У Node тоже есть родитель — **EventTarget**, интерфейс, реализуемый объектами, которые генерируют события и имеют подписчиков на эти события.

DOM-интерфейс разрабатывался как универсальный интерфейс не только для HTML, но и для любого XML-документа. **HTMLDocument** — это абстрактный интерфейс **DOM**, который обеспечивает доступ к специальным свойствам и методам, не представленным по умолчанию в регулярном XML-документе.



Интерфейсы коллекций в DOM





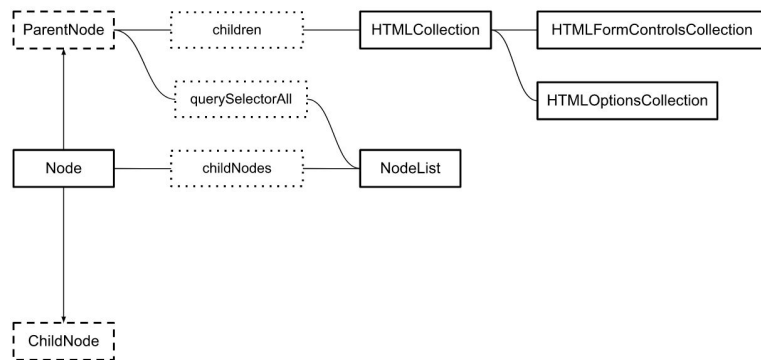
Интерфейсы коллекций в DOM

Коллекции в DOM представлены двумя базовыми интерфейсами. Рассмотрим их:

DOM-элементы, реализующие интерфейс **Node**, также реализуют два дополнительных интерфейса: **ParentNode** и **ChildNode**.

Интерфейс **ParentNode** содержит методы, относящиеся к Node-объектам, у которых могут быть потомки.

Интерфейс **ChildNode** включает в себя методы, специфичные для объектов Node и имеющие родителя.

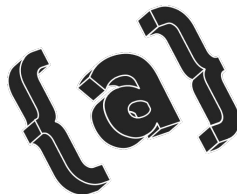
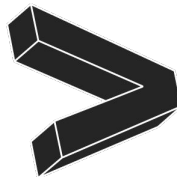




Интерфейсы коллекций в DOM

Между этими коллекциями есть несколько различий:

1. `NodeList` включает в себя любые типы дочерних узлов, например, `HTMLElement`, `Text`, `Comment`.
2. `HTMLCollection` содержит только узлы типа `HTMLElement`, соответствующие HTML-тегам, например, `<div>` и `<p>`, для которых поле `nodeType` равно 1.
3. `NodeList` может быть как динамическим, так и статическим. Например, поле `childNodes` — это динамический `NodeList`, а `NodeList`, возвращаемый методом `Node.querySelectorAll`, считается статическим: он не обновляет поле `length` при добавлении или удалении элемента из DOM-дерева.
4. `HTMLCollection` — это динамическая коллекция элементов.





Spread,
rest operator



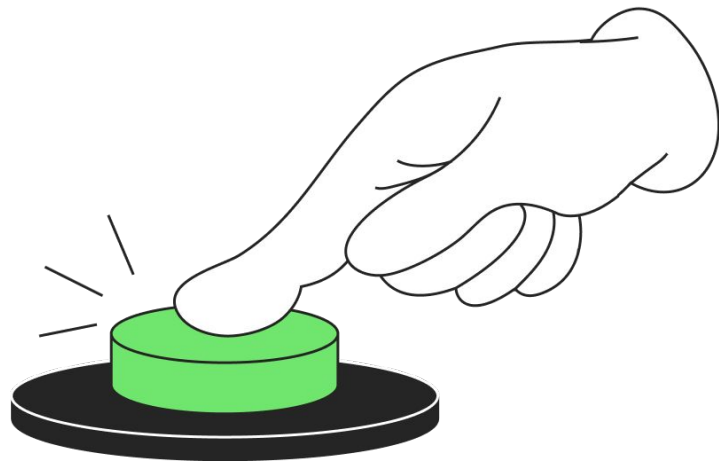


Spread operator

Spread (англ. «расширять») — оператор расширения, или, по-другому, распространения данных из массива в атомарные элементы.

Мы можем взять массив и вытащить все его элементы как отдельные переменные. Это бывает необходимо, когда мы хотим передать множество аргументов в функцию или перенести элементы одного массива в другой. Для этого перед массивом ставят многоточие (оператор spread).

Давайте рассмотрим примеры.





Spread operator

```
1 const studentsGroup1PracticeTime = [
2   {
3     firstName: "Ivanov",
4     practiceTime: 56
5   },
6   {
7     firstName: "Petrov",
8     practiceTime: 120
9   },
10  {
11    firstName: "Sidorov",
12    practiceTime: 148
13  },
14  {
15    firstName: "Belkin",
16    practiceTime: 20
17  },
18  {
19    firstName: "Avdeev",
20    practiceTime: 160
21  }
22 ];
23
24 const studentsGroup2PracticeTime = [
25   {
26     firstName: "Mankov",
27     practiceTime: 87
28   },
29   {
30     firstName: "Kistin",
31     practiceTime: 133
32   },
33   {
34     firstName: "Kotlyarov",
35     practiceTime: 140
36   },
37   {
38     firstName: "Peskov",
39     practiceTime: 10
40   },
41 ];
42
43 // Напишем не очень удобную, но показательную функцию, которая умеет
44 // принимать неограниченное число аргументов, и находить максимум среди
45 // них. Функция должна вызываться подобным образом: const maximum =
46 // findMax(4, 7, 10);
47 function findMax() {
48   const values = arguments; // arguments - переменная доступная
49   // внутри каждой функции, которая содержит в себе все аргументы,
50   // переданные в функцию. Является псевдомассивом.
51   let maxValue = -Infinity;
52   // Так как arguments является псевдомассивом, мы не можем
53   // применять к нему новые методы массивов такие как forEach или reduce,
54   // поэтому будем итерировать по старинке.
55   for (let index = 0; index < values.length; index++) {
56     if (values[index] > maxValue) maxValue = values[index];
57   }
58   return maxValue;
59 };
60
```

```
1 // Мы должны передавать в нашу функцию только числа, а в наших
2 // массивах содержатся объекты, поэтому сначала создадим массивы
3 // только со значениями времени отработанными студентами.
4 const group1PracticeTime = studentsGroup1PracticeTime.map((student)
5   => student.practiceTime);
6 const group2PracticeTime = studentsGroup2PracticeTime.map((student)
7   => student.practiceTime);
8 // Теперь можем вызывать нашу функцию поиска максимального значения.
9 // Она принимает множество числовых аргументов, а у нас есть только
10 // массив, вот тут нам и поможет оператор spread.
11
12 const maxTimeFromGroup1 = findMax(...group1PracticeTime); //
13 // ...group1PracticeTime - вытянет из массива все элементы и передаст
14 // их в функцию как отдельные переменные.
15 // Это аналогично страшной и неудобной записи:
16 // findMax(group1PracticeTime[0], group1PracticeTime[1],
17 // group1PracticeTime[2], group1PracticeTime[3], group1PracticeTime[4])
18 console.log(maxTimeFromGroup1); // 160
19
20 const maxTimeFromGroup2 = findMax(...group2PracticeTime);
21 console.log(maxTimeFromGroup2); // 140
22
23 // Давайте также найдем максимально отработанное время среди двух
24 // групп. Мы можем сделать это передав данные обоих массивов в функцию
25 // таким образом:
26 // findMax(...group1PracticeTime, ...group2PracticeTime);
27 // А можем объединить два массива в один - это очень частая операция
28 // и оператор расширения (spread) очень в этом помогает.
29
30 const bothGroupsTime = [...group1PracticeTime,
31   ...group2PracticeTime];
32
33 // Для объединения двух массивов нам нужно вытянуть их элементы в
34 // один общий массив, поэтому мы объявляем новый массив, а в качестве
35 // его элементов делаем расширение элементов первого и второго массива.
36 // Также мы могли бы добавить в него и другие элементы.
37
38 const maxTimeBothGroups = findMax(...bothGroupsTime);
39 console.log(maxTimeBothGroups); // 160
40
```



Rest operator

Оператор `rest` (англ. «остальные», «оставшиеся») позволяет собрать оставшиеся аргументы функции в массив. Звучит немного странно, однако этот оператор позволяет не перечислять все аргументы функции как отдельные переменные, а получить их все одним массивом.

Для его использования нужно в функции, принимающей несколько аргументов, перечислить необходимые аргументы, а все оставшиеся, которые мы хотим собрать в один массив, — записать как `...<имя массива>`. Часто пишут `...rest`. Давайте перепишем наш предыдущий пример, используя оператор `rest` и тем самым избавляясь от псевдомассива `arguments`.





Rest operator

```
1 const studentsGroup1PracticeTime = [
2   {
3     firstName: "Ivanov",
4     practiceTime: 50
5   },
6   {
7     firstName: "Petrov",
8     practiceTime: 120
9   },
10  {
11    firstName: "Sidorov",
12    practiceTime: 140
13  },
14  {
15    firstName: "Belkin",
16    practiceTime: 20
17  },
18  {
19    firstName: "Avdeev",
20    practiceTime: 160
21  }
22 ];
23
24 const studentsGroup2PracticeTime = [
25   {
26     firstName: "Mankov",
27     practiceTime: 87
28   },
29   {
30     firstName: "Kistin",
31     practiceTime: 133
32   },
33   {
34     firstName: "Kotlyarov",
35     practiceTime: 140
36   },
37   {
38     firstName: "Peskov",
39     practiceTime: 10
40   },
41 ];
42
43 // Напишем не очень удобную, но показательную функцию, которая умеет
44 // принимать неограниченное число аргументов, и находить максимум среди
45 // них. Функция должна выглядеть подобным образом: const maxTime =
46 // findMax(4, 7, 10);
47
48 function findMax(...values) { // тут мы принимаем все переданные
49   // аргументы и с помощью rest оператора упаковываем их в массив values.
50   // На этот раз values уже настоящий массив и мы можем использовать
51   // reduce для итерации по нему и нахождения максимального числа.
52   return values.reduce((acc, value) => {
53     if (value > acc) return value;
54     return acc;
55   }, -Infinity);
56 };
```

```
1 // Создадим массивы только со значениями времени отработанными
2 // студентами.
3 const group1PracticeTime = studentsGroup1PracticeTime.map((student)
4   => student.practiceTime);
5 const group2PracticeTime = studentsGroup2PracticeTime.map((student)
6   => student.practiceTime);
7
8 // Вызовем нашу функцию поиска максимума, используя оператор spread.
9 const maxTimeFromGroup1 = findMax(...group1PracticeTime);
10 console.log(maxTimeFromGroup1); // 160
11
12 const maxTimeFromGroup2 = findMax(...group2PracticeTime);
13 console.log(maxTimeFromGroup2); // 140
14
15 // Давайте также найдем максимально отработанное время среди двух
16 // групп.
17 const bothGroupsTime = [...group1PracticeTime,
18   ...group2PracticeTime];
19
20 const maxTimeBothGroups = findMax(...bothGroupsTime);
21 console.log(maxTimeBothGroups); // 160
```



Rest operator

Давайте рассмотрим ещё один пример:

```
1 const saveFullNameInDB = (firstName, lastName, ...additional) => {  
2   saveFirstName(firstName);  
3   saveLastName(lastName);  
4   saveAdditional(additional); // Благодаря rest оператору мы  
   смогли собрать все дополнительные данные, которые были переданы для  
   сохранения в базе данных, и можем передать их одним массивом в  
   функцию сохранения дополнительных данных.  
5 }
```



Работа с коллекциями





Работа с коллекциями

Создадим HTML-файл со следующим содержимым и откроем его в браузере:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta name="viewport" content="width=device-width,initial-
      scale=1">
5     <title>Пример использования DOM коллекций</title>
6   </head>
7   <body>
8     <div>
9       <p>Первый параграф</p>
10      <p>Второй параграф</p>
11      <p>Третий параграф</p>
12    </div>
13  </body>
14 </html>
```



Работа с коллекциями

Добавим следующий JavaScript-код на страницу
или используем JavaScript-консоль браузера:

```
1 const divElement = document.querySelector('div')
2 console.log(divElement.childNodes.length) // 7
3 console.log(divElement.children.length) // 3
4
5
```




Работа с коллекциями

Коллекции **childNodes** и **children** имеют разную длину.

Посмотрим, какие элементы содержатся в каждой коллекции. Чтобы перебрать элементы, сначала преобразуем коллекции в массивы с помощью статического метода [Array.from](#) или оператора [spread](#).

```
1 Array.from(divElement.childNodes).forEach((childNode) => {
2   console.log('childNode "%s" типа "%d"', childNode.nodeName,
3     childNode.nodeType)
4 })
5 [...divElement.children].forEach((child) => {
6   console.log('child "%s" типа "%d"', child.nodeName, child.nodeType)
7 })
8
9
```



Работа с коллекциями

Коллекция `children` содержит только элементы `P`, в отличие от `childNodes`, где также есть текстовые ноды (переносы строк).

Рассмотрим разницу между динамическими и статическими коллекциями.



```
1 const allParagraphElements = divElement.querySelectorAll('p')
2
3 console.log('Static NodeList длина до: %d',
4   allParagraphElements.length)
5 console.log('Dynamic NodeList длина до: %d',
6   divElement.childNodes.length)
7 console.log('HTMLCollection длина до: %d',
8   divElement.children.length)
9
10
11 const fourthParagraphElement = document.createElement('p')
12 fourthParagraphElement.textContent = 'Четвертый параграф'
13 divElement.appendChild(fourthParagraphElement)
14
15 console.log('Static NodeList длина после: %d',
16   allParagraphElements.length)
17 console.log('Dynamic NodeList длина после: %d',
18   divElement.childNodes.length)
19 console.log('HTMLCollection длина после: %d',
20   divElement.children.length)
```

Статичный `NodeList`, возвращаемый из метода `querySelectorAll`, не меняет размер при добавлении новых нод в DOM, в отличие от динамического `NodeList` и `HTMLCollection`.



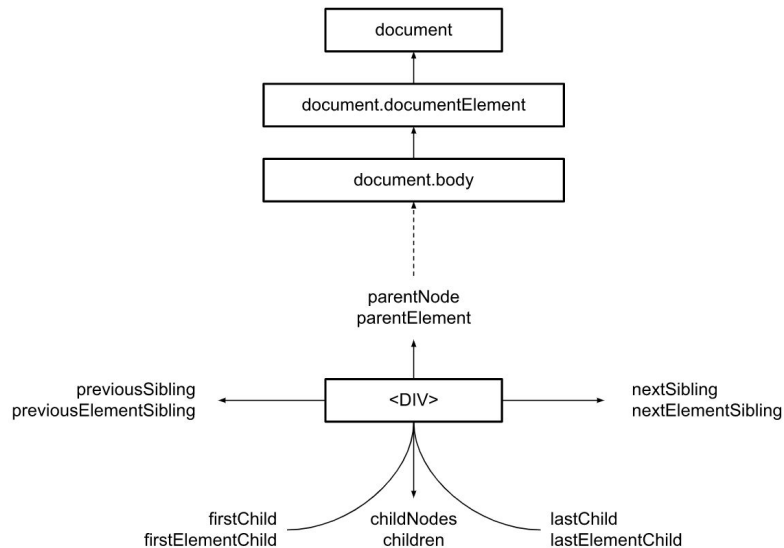
Методы для навигации по дереву DOM





Методы для навигации по дереву DOM

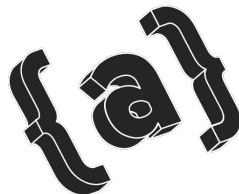
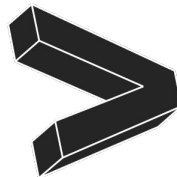
По аналогии с полями `childNodes` и `children` интерфейсы `Node` и `Element` позволяют получить доступ к элементам в дереве, напрямую окружающим исходный элемент.





К таким элементам относятся

1. **Родительский элемент** — `Node.parentNode` и `Node.parentElement`.
2. **Соседи исходного элемента** — `Node.nextSibling` / `Node.previousSibling` и `Element.nextElementSibling` / `Element.previousElementSibling`.
3. **Первый и последний дочерние элементы** — `Node.firstChild` / `Node.lastChild` и `ParentNode.firstChild` / `ParentNode.lastElementChild`.



Как и поля коллекций, эти поля используются только для чтения.








Type	Element	Node
Parent	parentElement	parentNode
Children	children firstElementChild lastElementChild	childNodes firstChild lastChild
Siblings	nextElementSibling previousElementSibling	nextSibling previousSibling


Для родительского элемента `parentNode` и `parentElement` практически всегда возвращают один и тот же элемент, за исключением случаев, когда `parentNode` элемента — не `DOM Element`. В таком случае `parentElement` возвращает `null`.

```
1 document.body.parentNode // Элемент <html>
2 document.body.parentElement // Элемент <html>
3 document.documentElement.parentNode // Нода document
  document.documentElement.parentElement // null
4 (document.documentElement.parentNode === document) // true
  (document.documentElement.parentElement === document) // false
5
```



Итоги урока

-  Управление стилями
-  Навигация по элементам DOM-дерева
-  Иерархия интерфейсов DOM
-  Интерфейсы коллекций в DOM
-  Операторы spread и rest
-  Работа с коллекциями
-  Методы для навигации по дереву DOM

Спасибо 
за внимание

