Конспект по теме 1.2 «Способы поиска нужного HTML-элемента»

Содержание конспекта:

1. Методы получения DOM-элементов по их особенностям.
2. Селекторы
3. Методы получения родительских элементов
4. Объект HTMLCollection
5. Объект NodeList
6. Указатель $0 в консоли

1. Методы получения DOM-элементов по их особенностям

Изучая JavaScript и области его применения в браузере, мы нередко сталкиваемся с необходимостью изменить структуру документа или же атрибуты HTML-элементов. Но чтобы что-то сделать с элементом, его сначала необходимо найти.

Чтобы получить HTML-элементы со страницы, существуют следующие методы:

- **getElementsByTagName**;
- **getElementsByClassName**;
- **getElementsByName**;
- **getElementById**.

Важно: три первых метода есть у объекта **document** и у каждого HTML-элемента, который находится в глобальной области видимости.

Исключение — функция **getElementById**, которая находится только у объекта **document**.

Функцию **getElementById** нельзя применить к отдельному элементу HTML-страницы. Это связано с тем, что элемент с уникальным **ID** может быть только один на всей странице.

Методы получения HTML-элементов очень важны, так как именно они делают возможным взаимодействие с HTML-документом и обеспечивают интерактивность веб-страницы. А это основное назначение языка JavaScript.

1. Метод **getElementsByTagName**() позволяет получить все теги с одинаковым названием. Например, все изображения на странице. Функция **getElementsByTagName**() аргументом принимает название тега:

```
// Получение всех изображений со страницы
let images = document.getElementsByTagName("img");
```

2. Метод **getElementsByClassName**() позволяет загрузить все элементы с одинаковым атрибутом name:

```
let elements = document.getElementsByName('age');
```

Функция **getElementsByClassName**() аргументом принимает значение аргумента **name**:

```
const elementsRed = document.getElementsByClassName('red');
const elementsSelected = document.getElementsByClassName('selected');
const elementsBlue = document.getElementsByClassName('blue');
```

3. Метод **getElementById**() ищет элемент по его идентификатору. Чтобы получить элемент, необходимо передать его **ID** в качестве аргумента функции **getElementById**:

```
let loginButton = document.getElementById('loginBtn');
```

Помимо функции **getElementById**(), уникальный элемент находится в глобальной области видимости Window, что также позволяет найти его по ID.

Следующие вызовы будут идентичными:

```
let loginButton = document.getElementById('loginBtn');
let loginButtonSame = window.loginBtn;
```

Обратите внимание, функции для получения элементов начинаются с **getElements**, и только **getElementById** не содержит «s». Все методы получения элементов (кроме **getElementById**) возвращают коллекцию элементов. Метод **getElementById**() возвращает только один элемент, т. к. на странице может быть только один элемент с уникальным **ID**.

Методы поиска элемента в структуре HTML-документа представлены в JavaScript большим многообразием функций. Они — незаменимое подспорье в разработке функциональной части любого сайта. Некоторые функции позволяют найти множество объектов по какому-либо конкретному атрибуту, другие же способны выдавать результат поиска по заданному селектору.

2. Селекторы

Иногда нужно получить определённые элементы, для которых нельзя воспользоваться предыдущими функциями. Например, если нужно получить все элементы списка с классом **exclusive**.

Селекторы — вторая группа методов поиска элемента:

1. Функция **querySelector**() возвращает **null** или элемент по входному CSS-селектору.
2. Функция **querySelectorAll**() возвращает коллекцию элементов по входному CSS-селектору.

Пример:

```
// Получаем все элементы списка с классом "exclusive"
let exclusiveElements = document.querySelectorAll("ul.exclusive li");
// Получаем первый элемент списка, у которого есть класс "exclusive"
let exclusiveElements = document.querySelector("ul li.exclusive");
```

Методы поиска элементов расширяют список возможностей, позволяя искать элемент по селектору.

3. Методы получения родительского элемента

При разработке скрипта JavaScript разработчику необходимо получить ближайшего к элементу родителя, доступ к которому при помощи иных функций может быть затруднён. Существует несколько возможных способов решения и особенностей получения родительских элементов.

1. Можно использовать свойство **parentElement**, которое позволяет получить ближайшего родителя к заданному HTML-элементу.
2. Метод **closest** позволяет получить доступ к ближайшему родителю элемента по CSS-селектору.

```
let listElements = document.getElementsByTagName("li");
let firstElement = listElements.item(0);
console.log(firstElement.parentElement); // <ul class="exclusive">...</ul>
console.log(firstElement.closest("ul")); // <ul class="exclusive">...</ul>
console.log(firstElement.closest("*")); // <li>Element 1</li>
console.log(document.parentElement); // null, т. к. document является корневым элементом
```

Методы поиска родительского элемента позволяют получить доступ к HTML-элементам и изменить их атрибуты даже в тех случаях, когда прямой доступ к ним затруднён по той или иной причине.

Итоги:

- функция **getElementById** получает один элемент по ID;
- функция **getElementsByTagName** получает коллекцию элементов по тегу;
- функция **getElementsByClassName** получает коллекцию элементов по классу;
- функция **getElementsByName** получает коллекцию элементов по имени;
- функция **querySelector** получает первый элемент по CSS-селектору;
- функция **querySelectorAll** получает все элементы по CSS-селектору;
- функция **closest** по CSS-селектору возвращает ближайшего родителя.

4. Объект HTMLCollection

HTMLCollection — это список узлов, каждый из которых может быть доступен по номеру или имени узла и атрибута.

Результат вызова таких функций, как **getElementsByTagName**, **getElementsByClassName**, — набор объектов-узлов, но это не массив. Такие структуры в JavaScript обычно называют псевдомассивами, например, **arguments**. В нашем случае это объект класса **HTMLCollection**, который представляет собой динамическую структуру, т. е. при изменении **DOM** изменяется и список узлов **HTMLCollection**.

У **HTMLCollection** нет привычных методов для обработки массива, поэтому для упрощения работы удобно пользоваться функцией **Array.from**(), чтобы превратить коллекцию в массив:

```
let links = document.getElementsByTagName("a");
console.log(Array.isArray(links)); // false
let arr = Array.from(links);
console.log(Array.isArray(arr)); // true
```

1. Получение определённого элемента.

Часто приходится взаимодействовать не с целой коллекцией, а с определённым элементом. Для этого у элемента **HTMLCollection** есть метод **item**() , позволяющий получить элемент по его позиции. В этом случае преобразование в массив не требуется.

```
let listElements = document.getElementsByTagName("li");
let firstElement = listElements.item(0);
// Получение элемента как из массива тоже возможно
let secondElement = listElements[1];
```

2. Получение определённого элемента по его имени.

Определённый элемент коллекций можно получить по его имени через метод **namedItem**():

```
let ageInput =
document.getElementsByTagName("input").namedItem("age");
```

3. Получение количества элементов.

Иногда необходимо узнать количество найденных элементов, для этого у коллекции **HTMLCollection** есть свойство **length** (доступно только для чтения). Свойство **length** показывает количество элементов коллекции:

```
let links = document.getElementsByTagName("a");
console.log(links.length);
```

Однако это свойство может изменить своё значение, если изменится **DOM.** Если структура **DOM** изменяется, эти изменения могут затронуть и объект **HTMLCollection**, а значит и свойство **length**. Например, если добавить элемент, который подходит под наши критерии поиска, то **length** увеличится. Поэтому для итерации по оригинальной коллекции лучше пользоваться массивом или сохранять значение свойства **length** в локальную переменную.

Пример:

1. Получим HTML-элементы.
2. Сделаем цикл по всем элементам коллекции (т. е. пока итератор не достигнет **length**).
3. Будем изменять HTML-элементы (например, добавлять новые).

В таком алгоритме есть проблемы, поскольку **length** обновляется динамически. При добавлении элемента **length** будет изменяться (увеличиваться), а значит это грозит проблемой бесконечного цикла.

Исправить проблему можно, определив перед циклом начальное количество элементов и сделав цикл до этой переменной.

В итоге свойство **length** будет изменяться, но это никак не повлияет на значение, которое мы получили заранее, а значит итератор будет изменяться до начального значения **length** , которое мы получили до всех изменений.

5. Объект NodeList

Объект **NodeList** представляет собой статическую коллекцию узлов, которую возвращает функция **querySelectorAll**. Это означает, что при изменении **DOM**, эта коллекция узлов по-прежнему будет содержать в своём составе изначальный список элементов. **NodeList** обеспечивает уровень абстракции над коллекцией узлов, не определяя и не ограничивая, как эта коллекция используется.

Объект **NodeList** можно получить методами **Node.childNodes** и **document.querySelectorAll**.

Коллекция HTML всегда находится в **DOM**, в то время как **NodeList** — более универсальная конструкция, которая может или не может быть в **DOM**.

6. Указатель $0 в консоли

Рассмотрим, что значит $0 в консоли, который мы так часто встречаем в инструментах разработчика в браузере. Иногда нужно поэкспериментировать, выполняя какие-либо действия с HTML-элементом.

Указатель $0 возвращает последний выбранный элемент или объект JavaScript.

Во время разработки достаточно часто нужно вызывать методы какого-то конкретного HTML-элемента, но вот писать сложные селекторы для его поиска не хочется. В этом случае можно выбрать элемент через инструменты разработчика. В консоли он станет доступен под названием $0.

```
> $0
<- ▶<div class="About-module-about-2tLLvF" id="about">…</div>
> $0.id
<- "about"
> $0.nextSibling
<- ▶<div class="PopularPrograms-module-programs-3PB6i0" id="programs">…</div>
> $0.nodeName
<- "DIV"
> $0.hasChildNodes()
<- true
>
```

Чтобы не писать специальные запросы для получения элемента, его можно просто выбрать, а в консоли обращаться к нему с помощью $0.

```
▼<section>
  ▼<div class="presentation_wrapper" id="presentation">
    ▼<div class="Presentation-module-presentation-1ovvxW">
      ▶<video autoplay loop class="Presentation-module-video-1uaI8o" style="display: block; width: 1570px; height: 882px;">…</video>
       <div class="Presentation-module-videoOverlay-3Ywu8T" style="opacity: 0.6;"></div>
      ▶<div class="Presentation-module-content-nWYi5F" style="transform: translateY(0px);">…</div>
      </div>
    </div>
...  ▶<div class="About-module-about-2tLLvF" id="about">…</div> == $0
  ▶<div class="PopularPrograms-module-programs-3PB6i0" id="programs">…</div>
  ▶<div class="Mentors-module-wrapper--kgknn" data-name="reviews" id="mentors">…</div>
  ▶<div class="direction-module-directions-1kLsLQ" id="direction">…</div>
  ▶<div class="subscriptionForm-module-root-10_I-k subscriptionForm-module-common-3kmnMJ">…</div>
  </section>
</section>
▶<div class="footer-module-root-2P1nxC">…</div>
▶<div class="cookiesAgreement-module-root-2INjMM">…</div>
</div>
</div>
```

Над выбранным элементом можно производить такие же действия, как над любым элементом HTMLElement.

Недостаток $0 в использовании

Использовать элемент $0 можно, только выбрав его вручную, а значит это удобно только в процессе разработки (в реальном коде использовать нельзя).

$1 , $2 , $3 и т. д.

Иногда необходимо получить элементы, которые были выбраны ранее. Для этого в консоли можно использовать $1 — элемент, который был выбран до текущего. $2 — элемент, который был выбран до элемента $1 . $3 — элемент, который был выбран до элемента $2 и т. д.

Итоги по теме:

- объект **HTMLCollection** является псевдомассивом;
- длина **HTMLCollection** обновляется динамически и всегда показывает количество элементов;
- $0 упрощает отладку кода и взаимодействие с DOM-деревом.

Весь код, используемый в лекции, — листинг кода.

Материалы, использованные при подготовке:

- «Поиск: getElement* и querySelector*»;
- «Работа с DOM из консоли»;
- «Навигация по DOM-элементам»;
- «Внутреннее устройство поисковых методов».