Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 24**

**Инструкционно-технологическая карта**

Тема: способы поиска в DOM-дереве: getElement\*, querySelector\*.

Цель: научиться производить поиск в DOM-дереве при помощи операторов: getElement\*, querySelector\*.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Краткие теоретические сведения

Если у элемента есть атрибут **id**, то можно получить его вызовом **document.getElementById(id),** где бы он ни находился.

Например:

<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>

Также есть глобальная переменная с именем, указанным в id:

<div id="elem">

    <div id="elem-content">Элемент</div>

</div>

<script>

    // elem - ссылка на элемент с id="elem"

    elem.style.background = 'red';

    // внутри id="elem-content" есть дефис, так что такой id не может служить именем переменной

    // ...но можно обратиться к нему через квадратные скобки: window['elem-content']

</script>

Но это только если не была объявлена в JavaScript переменная с таким же именем, иначе она будет иметь приоритет:

<div id="elem"></div>

<script>

    let elem = 5; // теперь elem равен 5, а не <div id="elem">

    alert(elem); // 5

</script>

## querySelectorAll

Самый универсальный метод поиска – это **elem.querySelectorAll(css)**, он возвращает все элементы внутри **elem**, удовлетворяющие данному CSS-селектору.

Следующий 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>

Этот метод действительно мощный, потому что можно использовать любой CSS-селектор.

## querySelector

Метод **elem.querySelector(css**) возвращает первый элемент, соответствующий данному CSS-селектору.

Иначе говоря, результат такой же, как при вызове **elem.querySelectorAll(css)[0]**, но он сначала найдёт все элементы, а потом возьмёт первый, в то время как **elem.querySelector** найдёт только первый и остановится. Это быстрее, кроме того, его короче писать.

## matches

Предыдущие методы искали по DOM.

Метод **elem.matches(css)** ничего не ищет, а проверяет, удовлетворяет ли **elem** CSS-селектору, и возвращает **true** или **false**.

Этот метод удобен, когда идет перебирание элементов (например, в массиве или в чём-то подобном) и попытка выбрать те из них, которые больше всего интересуют.

Например:

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

<a href="http://ya.ru">...</a>

<script>

    // может быть любая коллекция вместо document.body.children

    for (let elem of document.body.children) {

        if (elem.matches('a[href$="zip"]')) {

            alert("Ссылка на архив: " + elem.href);

        }

    }

</script>

## closest

*Предки элемента* – родитель, родитель родителя, его родитель и так далее. Вместе они образуют цепочку иерархии от элемента до вершины.

Метод **elem.closest(css)** ищет ближайшего предка, который соответствует CSS-селектору. Сам элемент также включается в поиск.

Другими словами, метод **closest** поднимается вверх от элемента и проверяет каждого из родителей. Если он соответствует селектору, поиск прекращается. Метод возвращает либо предка, либо **null**, если такой элемент не найден.

Например:

<h1>Содержание</h1>

<div class="contents">

    <ul class="book">

        <li class="chapter">Глава 1</li>

        <li class="chapter">Глава 2</li>

    </ul>

</div>

<script>

    let chapter = document.querySelector('.chapter'); // LI

    alert(chapter.closest('.book')); // UL

    alert(chapter.closest('.contents')); // DIV

    alert(chapter.closest('h1')); // null (потому что h1 - не предок)

</script>

## getElementsBy

Существуют также другие методы поиска элементов по тегу, классу и так далее.

На данный момент, они скорее исторические, так как **querySelector** более чем эффективен.

Рассмотрим их для полноты картины, также можно встретить их в старом коде.

* *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.

Например:

// получить все элементы div в документе

let divs = document.getElementsByTagName('div');

Давайте найдём все **input** в таблице:

<table id="table">

    <tr>

        <td>Ваш возраст:</td>

        <td>

            <label>

                <input type="radio" name="age" value="young" checked> младше 18

            </label>

            <label>

                <input type="radio" name="age" value="mature"> от 18 до 50

            </label>

            <label>

                <input type="radio" name="age" value="senior"> старше 60

            </label>

        </td>

    </tr>

</table>

<script>

    let inputs = table.getElementsByTagName('input');

    for (let input of inputs) {

        alert(input.value + ': ' + input.checked);

    }

</script>

Ищем элементы с классом **.article**:

<form name="my-form">

    <div class="article">Article</div>

    <div class="long article">Long article</div>

</form>

<script>

    // поиск по имени атрибута

    let form = document.getElementsByName('my-form')[0];

    // поиск по классу внутри form

    let articles = form.getElementsByClassName('article');

    alert(articles.length); // 2, обнаруживаем два элемента с классом article

</script>

## Живые коллекции

Все методы «**getElementsBy\***» возвращают живую коллекцию. Такие коллекции всегда отражают текущее состояние документа и автоматически обновляются при его изменении.

В приведённом ниже примере есть два скрипта.

1. Первый создаёт ссылку на коллекцию **<div>**. На этот момент её длина равна **1**.
2. Второй скрипт запускается после того, как браузер встречает ещё один **<div>**, теперь её длина – **2**.

<div>First div</div>

<script>

    let divs = document.getElementsByTagName('div');

    alert(divs.length); // 1

</script>

<div>Second div</div>

<script>

    alert(divs.length); // 2

</script>

Напротив, **querySelectorAll** возвращает *статическую* коллекцию. Это похоже на фиксированный массив элементов.

Если использовать его в примере выше, то оба скрипта вернут длину коллекции, равную **1**:

<div>First div</div>

<script>

    let divs = document.querySelectorAll('div');

    alert(divs.length); // 1

</script>

<div>Second div</div>

<script>

    alert(divs.length); // 1

</script>

Теперь легко видим разницу. Длина статической коллекции не изменилась после появления нового **div** в документе.

## Итого

Есть 6 основных методов поиска элементов в DOM:

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Ищет по... | Ищет внутри элемента? | Возвращает живую коллекцию? |
| querySelector | CSS-selector | ✔ | - |
| querySelectorAll | CSS-selector | ✔ | - |
| getElementById | id | - | - |
| getElementsByName | name | - | ✔ |
| getElementsByTagName | tag or '\*' | ✔ | ✔ |
| getElementsByClassName | class | ✔ | ✔ |

Безусловно, наиболее часто используемыми в настоящее время являются методы **querySelector** и **querySelectorAll**, но и методы **getElement(s)By\*** могут быть полезны в отдельных случаях, а также встречаются в старом коде.

Кроме того:

* Есть метод elem.matches(css), который проверяет, удовлетворяет ли элемент CSS-селектору.
* Метод elem.closest(css) ищет ближайшего по иерархии предка, соответствующему данному CSS-селектору. Сам элемент также включён в поиск.

И, напоследок, упомянем ещё один метод, который проверяет наличие отношений между предком и потомком:

* *elemA.contains(elemB)* вернёт **true**, если **elemB** находится внутри **elemA** (elemB потомок elemA) или когда **elemA==elemB**.

# Контрольные вопросы

1. Назовите 6 основных методов поиска элементов в DOM.
2. Расскажите, что такое «Живые коллекции»?
3. Расскажите про методы поиска элементов.

# Задания

В документе, представленном ниже, найдите следующие элементы:

1. Таблицу с id="age-table".
2. Все элементы label внутри этой таблицы (их три).
3. Первый td в этой таблице (со словом «Age»).
4. Форму form с именем name="search".
5. Первый input в этой форме.
6. Последний input в этой форме.

<!DOCTYPE HTML>

<html>

<body>

    <form name="search">

        <label>Search the site:

            <input type="text" name="search">

        </label>

        <input type="submit" value="Search!">

    </form>

    <hr>

    <form name="search-person">

        Search the visitors:

        <table id="age-table">

            <tr>

                <td>Age:</td>

                <td id="age-list">

                    <label>

                        <input type="radio" name="age" value="young">less than 18</label>

                    <label>

                        <input type="radio" name="age" value="mature">18-50</label>

                    <label>

                        <input type="radio" name="age" value="senior">more than 50</label>

                </td>

            </tr>

            <tr>

                <td>Additionally:</td>

                <td>

                    <input type="text" name="info[0]">

                    <input type="text" name="info[1]">

                    <input type="text" name="info[2]">

                </td>

            </tr>

        </table>

        <input type="sub7mit" value="Search!">

    </form>

</body>

</html>

# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич