

Лабораторная работа № 15

ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА (DOM)

Цель работы: изучить понятие DOM, научиться использовать JavaScript для управления объектной моделью документа

Теоретические сведения для выполнения работы

Основные понятия

Объектная модель документа (Document Object Model) — это прикладной программный интерфейс для HTML- и XML-документов, который представляет собой иерархическое дерево узлов, позволяя добавлять, удалять и изменять отдельные части страницы. Доступ к элементам для дальнейшего их изменения осуществляется с помощью языка программирования JavaScript, который является объектно-ориентированным и используется для создания сценариев динамического взаимодействия с веб-страницами.

Код JavaScript в HTML-документ добавляется с помощью тега **<script>**: **<script src="scriptFile.js">**. Браузер читает HTML-документ сверху вниз и, когда он встречает тег **<script>**, рассматривает текст программы как сценарий и выполняет его. Закончив его выполнение, возвращается обратно в HTML-документ и отображает оставшуюся часть документа. Если на веб-странице используется много сценариев, то могут возникнуть задержка загрузки веб-страницы. Поэтому считается лучше все ссылки на JavaScript-сценарии указывать после контента страницы перед закрывающим тегом **<body>**.

Вывод результата выполнения сценария или блока кода в консоль веб-страницы можно осуществлять с помощью метода **console.log()**. Можно набрать короткую последовательность команд на консоли, которая является частью комплекта средства разработки, входящего в состав браузера.

Значения в JavaScript могут иметь следующие типы: число, булево значение (*true* или *false*), специальные значения *null* и *undefined*, строка, символ, функция, объект. Значение, отличающееся от строки, числа, значения *symbol*, *true*, *false*, *null* или *undefined*, является объектом. Объекты имеют свойства и методы. Свойства

имеют имя и значения, а методы также могут иметь параметры, которые могут вернуть значение преобразования или конкретное значение объекта. Задать объекты можно в переменных, а затем применить к ним свойства или элементы. Объявление переменных осуществляется с помощью ключевых слов *let* или *const*. Устаревшими объявлениями переменных являются ключевого слова *var* и отсутствие ключевого слова.

Доступ к элементам DOM

Все элементы html-разметки соответствуют узлы в дереве. Всего существует 12 типов узлов, которые наследуются от одного базового типа. Элемент **<html>** является элементом документа (*document element*). Элемент документа — это корневой элемент в документе, содержащий все остальные элементы. В HTML-документах элементом документа всегда является **<html>**, а в XML-разметке им может быть заданный пользователем элемент.

Узлы в документе связаны с другими узлами в терминах традиционных семейных отношений. Элементы **<head>** и **<body>** являются одноуровневыми, имеющими общий родительский элемент **<html>**. Таким образом, у каждого узла есть свойство *childNodes*, которое содержит объект *NodeList*, используемый для хранения упорядоченного списка узлов, доступных по позиции. Объект *NodeList* является динамически обновляемым объектом, который отражает изменения DOM-структуры. Узлы в списке *childNodes* являются одноуровневыми и переходить от одного узла к другому можно с помощью *previousSibling* и *nextSibling*. Если дочерний узел единственный, то оба эти свойства равны *null*.

Свойства *firstChild* и *lastChild* родительского узла указывают на первый и последний узлы. Значение *someNode.firstChild* равно *someNode.childNodes[0]*, а значение *someNode.lastChild* равно *someNode.childNodes[someNode.childNodes.length-1]*.

Узел документа в JavaScript представлен с помощью типа *Document*, а в коде как объект *document* типа *HTMLDocument*. К свойствам *document* относятся *documentElement*, *firstChild*, *childNodes[0]*, которые указывают на элемент **<html>**. Также *document* имеет свойства *head*, *body* и *doctype*, указывающие на соответствующие элемент **<head>**, **<body>** и тег **<!DOCTYPE>**.

В качестве экземпляра типа *HTMLDocument* объект *document* имеет свойства, которые предоставляют информацию о загруженной веб-странице. Первой из них свойство *title*, которое содержит текст тега `<title>`. Следующие три свойства связаны с запросами веб-страниц. Свойство *URL* содержит полный URL-адрес страницы, свойство *domain* — доменное имя страницы, а *referrer* — URL-адрес страницы, с которой выполнен переход на текущую веб-страницу.

Создание нового элемента осуществляется с помощью метода *createElement()*, который принимает имя тега создаваемого элемента. Например, чтобы создать элемент `<div>` можно следующим образом: `let newDiv = document.createElement("div")`. Однако, чтобы его окончательно добавить в дерево элементов используются метод *appendChild()*: `document.body.appendChild(newDiv)`.

Для внесения изменений в текст или структуру внутри элемента используется свойство *innerHTML*, которое в режиме чтения возвращает HTML-код, представляющий все дочерние узлы элемента, в том числе комментарии и текстовые узлы. В режиме записи свойство *innerHTML* составляет из назначенной ему строки DOM-поддерево и заменяет им все дочерние узлы элемента. Все теги в ней преобразуются в элементы HTML-документа. Если строка не содержит таких элементов, в свойстве сохраняется обычный текст. Для работы с текстовым контентом элемента независимо используется свойство *innerText*.

Для доступа к конкретному элементу или множеству элементов для выполнения каких-либо действий с ними используются методы *getElementById()* и *getElementsByName()*. Метод *getElementById()* принимает идентификатор элемента, который нужно получить, и возвращает этот элемент или *null*, если его не существует. Если страница содержит несколько элементов с одинаковым идентификатором, то возвращается первый из них. Метод *getElementsByName()* принимает значение атрибута *name* элемента и возвращает объект *NodeList*, содержащий эти имена.

В HTML5 был добавлен метод *getElementsByClass()*, который принимает строку с одним или несколькими именами классов и возвращает объект *NodeList* с элементами, к которым применены эти классы. Свойство *className* используется для добавления, удаления и замены имен классов.

Для получения с помощью селекторов CSS доступа к элементам консорциумом W3C была разработана спецификация Selectors, в которой представлены методы ***querySelector()*** и ***querySelectorAll()***. Метод ***querySelector()*** принимает CSS-запрос по указанному селектору и возвращает первый соответствующий ему элемент или значение null, а метод ***querySelectorAll()*** возвращает соответствующие узлы в статическом экземпляре *NodeList*. Пример использования методов доступа к элементам представлен на рис. 15.1



а — код HTML-документа, б — результат в консоли браузера
Рис. 15.1 Использование методов доступа к элементам

События в DOM

Взаимодействие Javascript с HTML осуществляются с помощью событий (events), которые сигнализируют, что в документе или окне браузера что-то произошло. Например, когда пользователь нажимает клавишу на клавиатуре, перемещает указатель мыши, щелкает кнопкой мыши или касается сенсорного экрана, веб-браузер генерирует событие.

События соответствуют определенным действиям, которые выполняет пользователь. Функция, выполняемая в ответ на событие, называется обработчиком события или слушателем события. Эти функции регистрируются с помощью метода ***addEventListener()***, который принимает имя обрабатываемого события, функцию-обработчик и логическое значение, указывающее

нужно ли вызывать событие при перехвате или всплытии. Основные события представлены в таблице 15.1

Таблица 15.1

Основные события

Имя события	Описание события
blur	элемент теряет фокус
change	содержимое поля формы изменяется
click	щелчок мышью на объекте
error	произошла ошибка при загрузке документа или изображения
focus	элемент получает фокус
keydown	клавиша на клавиатуре нажата
keypress	клавиша на клавиатуре нажата и удерживается
keyup	клавиша на клавиатуре отпускается
load	страница или изображение загружены
mousedown	кнопка мыши нажата
mousemove	мышь перемещена
mouseout	мышь отодвигается от элемента
mouseover	мышь перемещается над элементом
mouseup	кнопка мыши отпущена
submit	в форме нажата кнопка отправки

Кроме метода *addEventListener()*, добавив к имени события приставку «on» и получив тем самым обработчик события, его можно указать для конкретного элемента в качестве атрибута (*<input onclick="myFunction();">*) или в виде метода объекта (*document.body.onclick="MyFunction()"*). Недостатком этих способов является вызов за один раз только одного события.

С помощью событий и использования свойства можно изменять стили элемента, например, *document.body.style.border*.

Работа с формами

Веб-форма представляет в JavaScript типом *HTMLFormElement*. Все формы на странице содержатся в коллекции *forms*. Каждая форма в ней доступна по числовому индексу (*document.forms[0]*) или по имени (*document.forms["form2"]*).

Элементы формы содержатся в коллекции *elements*. Элементы хранятся в коллекции в том порядке, в котором они расположены в разметке, и индексируются по позиции и имени. Для эле-

ментов типа *radio* и *checkbox* используются операторы цикла и условия для определения выбора значения. Пример представлен на рис. 15.2

```
<p>
<label><input type="checkbox" class="checkbox" value="checkbox 1">checkbox 1</label>
<label><input type="checkbox" class="checkbox" value="checkbox 2">checkbox 2</label>
<label><input type="checkbox" class="checkbox" value="checkbox 3">checkbox 3</label>
</p>
```

a

```
1 document.querySelector('button').addEventListener('click', function(){
2     let check = document.querySelectorAll('.checkbox');
3     for (let i = 0; i < check.length; i++) {
4         if (check[i].checked) {
5             let list=document.createElement("li")
6             list.innerHTML=check[i].value
7             document.getElementById('footer').append(list);
8         }
9     }
```

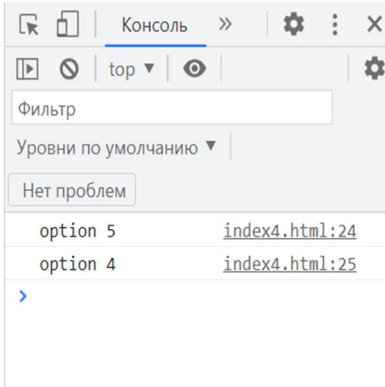
б

a — HTML-разметка, б — JavaScript-код
Рис. 15.2 Пример вывода значений checkbox

Согласно рис. 15.2 в строках 5–7 создается новый элемент, который будет добавлен в элемент footer в переменной list. В строке 6 используя свойство *innerHTML* добавляется значение выбранного *checkbox*.

Чтобы получить текст списка формы можно использовать свойства *text*. Для доступа к элементам списка используется коллекция *options*. Пример представлен на рис. 15.3

```
11 <form>
12   <select>
13     <option value="option 1">option 5</option>
14     <option value="option 2">option 2</option>
15     <option value="option 4">option 3</option>
16   </select>
17 </form>
18 <button>from Select</button>
19 <script>
20   document.querySelector('button').addEventListener('click', function(){
21     let selectbox = document.forms[0].elements[0]
22     let text = selectbox.options[0].text;
23     let value = selectbox.options[2].value;
24     console.log(text)
25     console.log(value)
26   })
```



The screenshot shows the browser's developer console with the following output:

Text	File	Line
option 5	index4.html	24
option 4	index4.html	25

Рис. 15.3 Пример получения значения списка формы

С помощью свойства *selected* можно узнать, какие элементы списка выбраны. Чтобы получить все выбранные элементы, можно перебрать набор элементов в цикле, проверяя их свойство *selected*.

Методом *checkValidity()* можно проверить допустимо ли значение конкретного поля формы. Он доступен для всех элементов и возвращает true или false.

Задания к лабораторной работе № 15

Задание 1 Создайте фотогалерею из четырех фотографий. При наведении на первой фотографии необходимо, чтобы появлялось вместо него описание, состоящее из нескольких строк, при щелчке мыши на второй фотографии должна появляться граница толщиной 10px сплошного красного фото. При наведении на последнем фото оно должно быть заменено на другое, при отводе курсора возвращаться к исходному.

Задание 2 Создайте копию формы задания 4 лабораторной работы № 1, убрав теги таблицы и добавив к ней раскрывающиеся списки для выбора факультета, группы и курса. Значения заполненной формы должны выводиться в **<footer>** по нажатию кнопки.

Контрольные вопросы

1. Дайте понятие DOM
2. Перечислите все способы доступа к элементам HTML-документа
3. Для чего используется *childNodes*?
4. Что такое событие?
5. Для чего предназначен метод *addEventListener()*?
6. Чем отличается использование метод *addEventListener()* от атрибута обработчика события?
7. Что такое *forms*?
8. Что такое *elements*?
9. Что такое *innerHTML* и для чего он необходим?
10. Как получить доступ к элементам **<select>**?
11. Для чего используется циклы?
12. Что такое метод *checkValidity()*?
13. Какие события вы знаете?