# Лабораторная работа № 7 Объектная модель документа (DOM)

**Цель работы:** изучить возможности взаимодействия JavaScript с элементами страницы и объектами DOM.

**Краткие теоретические сведения:**

Одой из ключевых задач JavaScript является взаимодействие с пользователем и манипуляция элементами веб-страницы. Для JavaScript веб-страница доступна в виде объектной модели документа (document object model) или сокращенно DOM.

В соответствии с объектной моделью документа, каждый HTML-тег является объектом. Вложенные теги являются дочерними объектами родительского элемента. Текст, который находится внутри тега, также является объектом.

Все эти объекты представляют собой узлы в иерархической структуре DOM и доступны при помощи JavaScript для изменения страницы.

Существует [12 типов узлов](https://dom.spec.whatwg.org/" \l "node). Но на практике в основном используются 4 из них:

* **Element**: html-элемент;
* **Document**: корневой узел html-документа;
* **Comment**: элемент комментария;
* **Text**: текст элемента.

У DOM-узлов есть свойства и методы, которые позволяют выбирать любой из элементов, изменять, перемещать их на странице и многое другое.

**Объект document**

Для работы со структурой DOM в JavaScript предназначен объект document, который определен в глобальном объекте window. Объект document предоставляет ряд свойств и методов для управления элементами страницы.

Для поиска элементов на странице применяются следующие методы:

* **getElementById(value)**: выбирает элемент, у которого атрибут id равен value;
* **getElementsByTagName(value)**: выбирает все элементы, у которых тег равен value;
* **getElementsByClassName(value)**: выбирает все элементы, которые имеют класс value;
* **querySelector(value)**: выбирает первый элемент, который соответствует css-селектору value;
* **querySelectorAll(value)**: выбирает все элементы, которые соответствуют css-селектору value.

Кроме методов объект document позволяет обратиться к определенным элементам веб-страницы через свойства:

**documentElement**: предоставляет доступ к корневому элементу <html>;

**body**: предоставляет доступ к элементу <body> на веб-странице;

**images**: содержит коллекцию всех объектов изображений (элементов img);

**links**: содержит коллекцию ссылок - элементов <a> и <area>, у которых определен атрибут href;

**anchors**: предоставляет доступ к коллекции элементов <a>, у которых определен атрибут name;

**forms**: содержит коллекцию всех форм на веб-странице.

Эти свойства не предоставляют доступ ко всем элементам, однако позволяют получить наиболее часто используемые элементы на веб-странице.

**Объект Node. Навигация по DOM**

Каждый отдельный узел, будь то html-элемент, его атрибут или текст, в структуре DOM представлен объектом Node. Этот объект предоставляет ряд свойств, с помощью которых мы можем получить информацию о данном узле:

* **childNodes**: содержит коллекцию дочерних узлов;
* **firstChild**: возвращает первый дочерний узел текущего узла;
* **lastChild**: возвращает последний дочерний узел текущего узла;
* **previousSibling**: возвращает предыдущий элемент, который находится на одном уровне с текущим;
* **nextSibling**: возвращает следующий элемент, который находится на одном уровне с текущим;
* **ownerDocument**: возвращает корневой узел документа;
* **parentNode**: возвращает элемент, который содержит текущий узел;
* **nodeName**: возвращает имя узла;
* **nodeType**: возвращает тип узла в виде числа;
* **nodeValue**: возвращает или устанавливает значение узла в виде простого текста;

**Создание, добавление и удаление элементов веб-станицы**

Для создания элементов объект document имеет следующие методы:

* **createElement(elementName)**: создает элемент html, тег которого передается в качестве параметра. Возвращает созданный элемент.
* **createTextNode(text)**: создает и возвращает текстовый узел. В качестве параметра передается текст узла.

Для добавления элементов можно использовать один из методов объекта Node:

* **appendChild(newNode)**: добавляет новый узел newNode в конец коллекции дочерних узлов;
* **insertBefore(newNode, referenceNode)**: добавляет новый узел newNode перед узлом referenceNode.

Иногда элементы бывают довольно сложными по составу, и гораздо проще их скопировать, чем с помощью отдельных вызовов создавать из содержимое. Для копирования уже имеющихся узлов у объекта Node можно использовать метод **cloneNode().** В метод cloneNode() в качестве параметра передается логическое значение: если передается true, то элемент будет копироваться со всеми дочерними узлами; если передается false - то копируется без дочерних узлов.

Для удаления элемента вызывается метод **removeChild()** объекта Node. Этот метод удаляет один из дочерних узлов.

Для замены элемента применяется метод **replaceChild(newNode, oldNode)** объекта Node. Этот метод в качестве первого параметра принимает новый элемент, который заменяет старый элемент oldNode, передаваемый в качестве второго параметра.

**Объект Element. Управление элементами**

Кроме методов и свойств объекта Node в JavaScript мы можем использовать свойства и методы объектов Element. Важно не путать эти два объекта: Node и Element. Node представляет все узлы веб-станицы, в то время как объект Element представляет непосредственно только html-элементы. То есть объекты Element - это фактически те же самые узлы – объекты Node, у которых тип узла (свойство nodeType) равно 1.

Основные свойства объекта Element:

* **nodeType** позволяет узнать тип DOM-узла. Его значение – числовое: 1 для элементов, 3 для текстовых узлов, и т.д.
* **tagName** возвращает название тега (записывается в верхнем регистре, за исключением XML-режима).
* **innerHTML** устанавливает или получает внутреннее HTML-содержимое элемента.
* **outerHTML** устанавливает или получает полный HTML-код элемента. Запись в elem.outerHTML не меняет elem. Вместо этого она заменяет его во внешнем контексте.
* **innerText, textContent** устанавливает или получает текст внутри элемента за вычетом всех <тегов>.
* **hidden** скрывает или отображает элемент. Когда значение установлено в true, делает то же самое, что и CSS display:none.
* **offsetWidth** определяет ширину элемента в пикселях. В ширину включается граница элемента.
* **offsetHeight** определяет высоту элемента в пикселях. В высоту включается граница элемента.
* **clientWidth** определяет ширину элемента в пикселях без учета границы.
* **сlientHeight** определяют высоту элемента в пикселях без учета границы.

Среди методов объекта Element можно отметить методы управления атрибутами:

* **hasAttribute(name)** проверяет на наличие атрибута.
* **getAttribute(attr)** возвращает значение атрибута attr.
* **setAttribute(attr, value)** устанавливает для атрибута attr значение value. Если атрибута нет, то он добавляется.
* **removeAttribute(attr)** удаляет атрибут attr и его значение.

**Изменение стиля элементов**

Для работы со стилевыми свойствами элементов в JavaScript применяются, главным образом, два подхода:

* изменение свойства style;
* изменение значения атрибута class.

Свойство **style** представляет сложный объект для управления стилем и напрямую сопоставляется с атрибутом style html-элемента. Этот объект содержит набор свойств CSS.

Свойства объекта style совпадает со свойством css. Однако ряд свойств css в названиях имеют дефис, например, font-family. В JavaScript для этих свойств дефис не употребляется. Только первая буква, которая идет после дефиса, переводится в верхний регистр: fontFamily.

С помощью свойства **className** можно установить атрибут class элемента html. Благодаря использованию классов не придется настраивать каждое отдельное свойство css с помощью свойства style. Но при этом надо учитывать, что прежнее значение атрибута class удаляется. Поэтому, если нам надо добавить класс, надо объединить его название со старым классом в виде конкатенации строк.

Для управления множеством классов гораздо удобнее использовать свойство **classList**. Это свойство представляет объект, реализующий следующие методы:

* add(className): добавляет класс className.
* remove(className): удаляет класс className.
* toggle(className): переключает у элемента класс на className. Если класса нет, то он добавляется, если есть, то удаляется.

**События**

Для взаимодействия с пользователем в JavaScript определен механизм событий. Например, когда пользователь нажимает кнопку, то возникает событие нажатия кнопки.

В JavaScript есть следующие типы событий:

* события мыши (перемещение курсора, нажатие мыши и т.д.);
* события клавиатуры (нажатие или отпускание клавиши клавиатуры);
* события жизненного цикла элементов (например, событие загрузки веб-станицы);
* события элементов форм (нажатие кнопки на форме, выбор элемента в выпадающем списке и т.д.);
* события, возникающие при изменении элементов DOM;
* события, возникающие при касании на сенсорных экранах;
* события, возникающие при возникновении ошибок.

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

Есть несколько способов назначить событию обработчик.

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>.

Например, чтобы назначить обработчик события click на элементе input, можно использовать атрибут onclick, вот так:

<input value="Нажми меня" onclick="alert('Клик!')" type="button">

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick.

Атрибут HTML-тега – не самое удобное место для написания большого количества кода, поэтому лучше создать отдельную JavaScript-функцию и вызвать в обработчике.

Можно назначать обработчик, используя свойство DOM-элемента on<событие>.

К примеру, elem.onclick:

<input id="elem" type="button" value="Нажми меня!">

<script>

elem.onclick = function() {

alert('Спасибо');

};

</script>

Фундаментальный недостаток описанных выше способов назначения обработчика – невозможность повесить несколько обработчиков на одно событие.

Разработчики стандартов предложили альтернативный способ назначения обработчиков при помощи специальных методов addEventListener и removeEventListener.

Синтаксис добавления обработчика:

element.addEventListener(event, handler[, options]),

где event – имя события, например "click", handler – ссылка на функцию-обработчик, options – дополнительный объект со свойствами:

* once: если true, тогда обработчик будет автоматически удалён после выполнения.
* capture: фаза, на которой должен сработать обработчик, подробнее об этом будет рассказано в главе Всплытие и погружение. Так исторически сложилось, что options может быть false/true, это тоже самое, что {capture: false/true}.
* passive: если true, то указывает, что обработчик никогда не вызовет preventDefault().

Для удаления обработчика следует использовать removeEventListener с аналогичным синтаксисом.

**Объект Event**

При обработке события браузер автоматически передает в функцию обработчика в качестве параметра объект Event, который инкапсулирует всю информацию о событии. И с помощью его свойств мы можем получить эту информацию:

* **bubbles**: возвращает true, если событие является восходящим;
* **cancelable**: возвращает true, если можно отменить стандартную обработку события;
* **currentTarget**: определяет элемент, к которому прикреплен обработчик события;
* **defaultPrevented**: возвращает true, если был вызван у объекта Event метод preventDefault();
* **eventPhase**: определяет стадию обработки события;
* **target**: указывает на элемент, на котором было вызвано событие;
* **timeStamp**: хранит время возникновения события;
* **type**: указывает на имя события.

С помощью метода **preventDefault()** объекта Event мы можем остановить дальнейшее выполнение события.

**События мыши**

Одну из наиболее часто используемых событий составляют события мыши:

* **click**: возникает при нажатии указателем мыши на элемент;
* **mousedown**: возникает при нахождении указателя мыши на элементе, когда кнопка мыши находится в нажатом состоянии;
* **mouseup**: возникает при нахождении указателя мыши на элементе во время отпускания кнопки мыши;
* **mouseover**: возникает при вхождении указателя мыши в границы элемента;
* **mousemove**: возникает при прохождении указателя мыши над элементом;
* **mouseout**: возникает, когда указатель мыши выходит за пределы элемента.

Объект Event является общим для всех событий. Однако для разных типов событий существуют также свои объекты событий, которые добавляют ряд своих свойств. Так, для работы с событиями указателя мыши определен объект MouseEvent, который добавляет следующие свойства:

* **altKey**: возвращает true, если была нажата клавиша Alt во время генерации события;
* **button**: указывает, какая кнопка мыши была нажата;
* **clientX**: определяет координату Х окна браузера, на которой находился указатель мыши во время генерации события;
* **clientY**: определяет координату Y окна браузера, на которой находился указатель мыши во время генерации события;
* **ctrlKey**: возвращает true, если была нажата клавиша Ctrl во время генерации события;
* **metaKey**: возвращает true, если была нажата во время генерации события метаклавиша клавиатуры;
* **relatedTarget**: определяет вторичный источник возникновения события;
* **screenX**: определяет координату Х относительно верхнего левого угла экрана монитора, на которой находился указатель мыши во время генерации события;
* **screenY**: определяет координату Y относительно верхнего левого угла экрана монитора, на которой находился указатель мыши во время генерации события;
* **shiftKey**: возвращает true, если была нажата клавиша Shift во время генерации события

**События клавиатуры**

Другим распространенным типом событий являются события клавиатуры:

* **keydown**: возникает при нажатии клавиши клавиатуры и длится, пока нажата клавиша;
* **keyup**: возникает при отпускании клавиши клавиатуры;
* **keypress**: возникает при нажатии клавиши клавиатуры, но после события keydown и до события keyup. Надо учитывать, что данное событие генерируется только для тех клавиш, которые формируют вывод в виде символов, например, при печати символов. Нажатия на остальные клавиши, например, на Alt, не учитываются.

Для работы с событиями клавиатуры определен объект KeyboardEvent, который добавляет к свойствам объекта Event ряд специфичных для клавиатуры свойств:

**altKey**: возвращает true, если была нажата клавиша Alt во время генерации события;

**charCode**: возвращает символ Unicode для нажатой клавиши (используется для события keypress);

**keyCode**: возвращает числовое представление нажатой клавиши клавиатуры;

**ctrlKey**: возвращает true, если была нажата клавиша Ctrl во время генерации события;

**metaKey**: возвращает true, если была нажата во время генерации события метаклавиша клавиатуры;

**shiftKey**: возвращает true, если была нажата клавиша Shift во время генерации события.

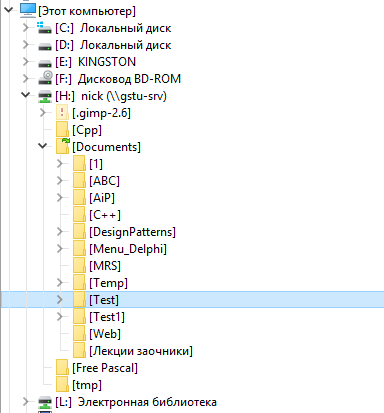
**Практическая часть:**

**Задание 1.** На основании задания 4 предыдущей лабораторной работы сформировать таблицу для отображения элементов массива. Каждое свойство объекта выводится в отдельную ячейку. Таблица должна формироваться динамически с использованием JavaScript. Предусмотреть форматирование таблицы.

**Задание 2.** Создать HTML-документ со списком ссылок. Ссылки на внешние источники (которые начинаются с http:// или https://) необходимо подчеркнуть пунктиром. Искать такие ссылки в списке и устанавливать им дополнительные стили необходимо с помощью JS.



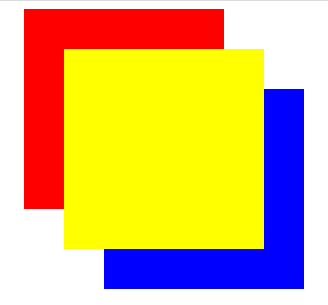
**Задание 3.** Создать HTML-документ с деревом вложенных директорий (см. рисунок ниже). Размер дерева и содержимое – произвольные. Каждый узел является элементом списка. При клике на элемент списка, он должен сворачиваться или разворачиваться. При наведении на элемент, шрифт должен становится жирным (с помощью свойства classList).



**Задание 4.** Написать сценарий, который позволяет менять фоновое изображение документа выбором из таблицы цветов.



**Задание 4.** Создать HTML-документ, в котором присутствуют три перекрывающихся (но не полностью) блока <div> с различным цветом фона и разными значениями z-индекса. Написать сценарий, в котором при клике мыши на блоке, соответствующий блок будет отображаться поверх остальных.



**Задание 5.** Создать HTML-документ со списком книг (автор и название). При щелчке на элементу спиcка, цвет текста должен меняться на оранжевый. При этом, если к какому-то элементу было применено выделение цветом, это выделение должно сбрасываться. При повторном щелчке на элементе необходимо возвращать прежний цвет. Если при клике мышкой была нажата клавиша Ctrl, то элемент добавляется/удаляется из выделенных. Если при клике мышкой была нажата клавиша Shift, то к выделению добавляются все элементы в промежутке от предыдущего кликнутого до текущего. (Как в проводнике Window или любом файловом менеджере). Для снятия выделения по умолчанию (синий цвет) используйте свойство user-select: none для контейнера списка.

**Требования по содержанию отчета:**

Отчет должен включать следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задание.
4. Листинг прогрммного кода.
5. Результаты выполнения сценария.
6. Выводы

**Контрольные вопросы для защиты:**

1. Как изменить правила стилей для заданного элемента?
2. Как программно получить доступ к элементу?
3. Какие методы и свойства объекта document Вы знаете?
4. Что такое DOM?
5. Что такое коллекция?
6. Как добавить атрибут к элементу?
7. Как создать (удалить) узел в DOM?
8. Какие события Вы знаете?
9. Опишите события клавиатуры.
10. Опишите события мыши.