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 № 23**

**Инструкционно-технологическая карта**

Тема: Устройство DOM-дерева

Цель: Научиться использовать DOM-дерева для поиска информации на страницах.

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.

# Теоретические сведения

Основой HTML-документа являются теги.

В соответствии с объектной моделью документа («Document Object Model», коротко DOM), каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

Все эти объекты доступны при помощи JavaScript, их можно использовать для изменения страницы.

Например, **document.body** – объект для тега **<body>**.

Если запустить этот код, то **<body>** станет красным на 3 секунды:

document.body.style.background = 'red'; // сделать фон красным

setTimeout(() => document.body.style.background = '', 3000); // вернуть назад

Это был лишь небольшой пример того, что может DOM.

Начало можно положить с такого, простого, документа:

<!DOCTYPE HTML>

<html>

<head>

  <title>О лосях</title>

</head>

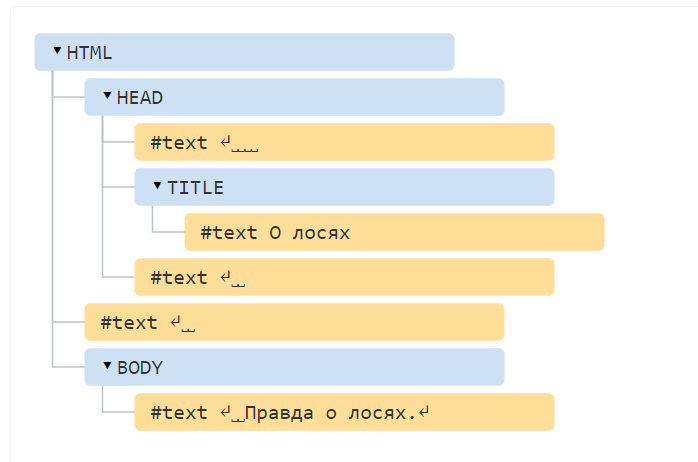
<body>

  Правда о лосях.

</body>

</html>

DOM – это представление HTML-документа в виде дерева тегов. Вот как оно выглядит:

На рисунке узлы-элементы можно кликать, и их дети будут скрываться и раскрываться.

Каждый узел этого дерева – это объект.

Теги являются узлами-элементами (или просто элементами). Они образуют структуру дерева: **<html>** – это корневой узел, **<head>** и **<body>** его дочерние узлы и т.д.

Текст внутри элементов образует текстовые узлы, обозначенные как **#text**. Текстовый узел содержит в себе только строку текста. У него не может быть потомков, т.е. он находится всегда на самом нижнем уровне.

Например, в теге **<title>** есть текстовый узел **«О лосях»**.

Обратите внимание на специальные символы в текстовых узлах: перевод строки: (в JavaScript он обозначается как \n); пробел: ␣.

Пробелы и переводы строки – это полноправные символы, как буквы и цифры. Они образуют текстовые узлы и становятся частью дерева DOM. Так, в примере выше в теге **<head>** есть несколько пробелов перед **<title>**, которые образуют текстовый узел **#text** (он содержит в себе только перенос строки и несколько пробелов).

Существует всего два исключения из этого правила:

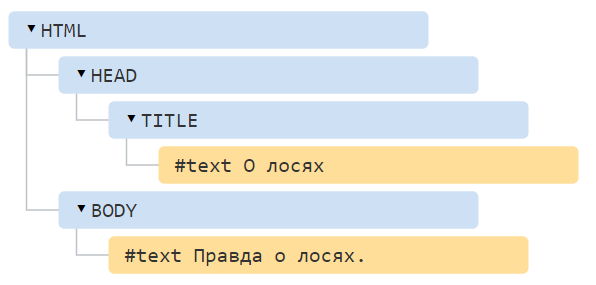
1. По историческим причинам пробелы и перевод строки перед тегом **<head>** игнорируются
2. Если записывать что-либо после закрывающего тега **</body>**, браузер автоматически перемещает эту запись в конец **body**, поскольку спецификация HTML требует, чтобы всё содержимое было внутри **<body>**. Поэтому после закрывающего тега **</body>** не может быть никаких пробелов.

В остальных случаях всё просто – если в документе есть пробелы (или любые другие символы), они становятся текстовыми узлами дерева DOM, и если их удалить, то в DOM их тоже не будет.

Здесь пробельных текстовых узлов нет:

<!DOCTYPE HTML>

<html><head><title>О лосях</title></head><body>Правда о лосях.</body></html>



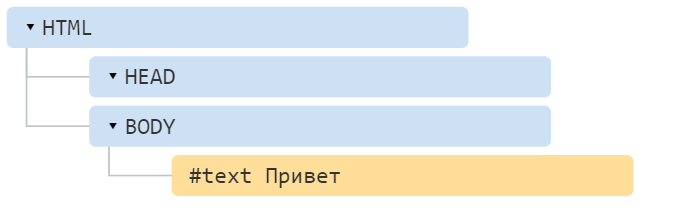
В дальнейших иллюстрациях DOM для краткости можно пропустить пробельные текстовые узлы там, где они не имеют значения. Обычно они не влияют на то, как отображается документ.

## Автоисправление

Если браузер сталкивается с некорректно написанным HTML-кодом, он автоматически корректирует его при построении DOM.

Например, в начале документа всегда должен быть тег **<html>**. Даже если его нет в документе – он будет в дереве DOM, браузер его создаст. То же самое касается и тега **<body>**.

Например, если HTML-файл состоит из единственного слова **«Привет»**, браузер обернёт его в теги **<html>** и **<body>**, добавит необходимый тег **<head>**, и DOM будет выглядеть так:



При генерации DOM браузер самостоятельно обрабатывает ошибки в документе, закрывает теги и так далее.

Есть такой документ с незакрытыми тегами:

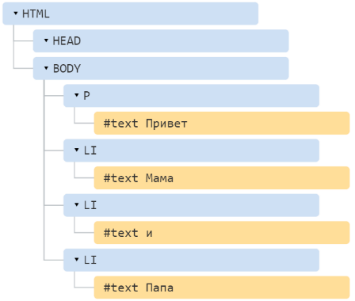
<p>Привет

    <li>Мама

    <li>и

    <li>Папа

Но DOM будет нормальным, потому что браузер сам закроет теги и восстановит отсутствующие детали:

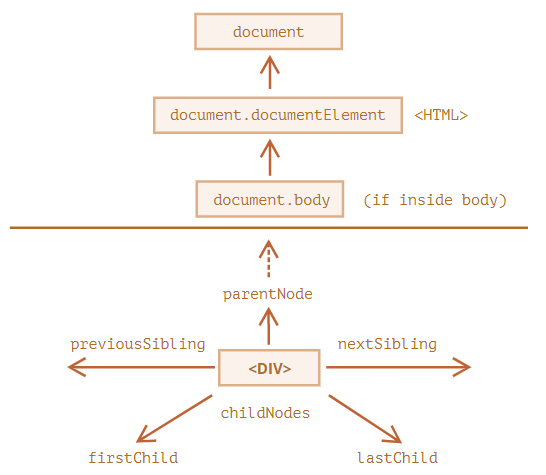


## Навигация по DOM-элементам

DOM позволяет делать что угодно с элементами и их содержимым, но для начала нужно получить соответствующий DOM-объект.

Все операции с DOM начинаются с объекта **document**. Это главная «точка входа» в DOM. Из него можно получить доступ к любому узлу.

Так выглядят основные ссылки, по которым можно переходить между узлами DOM:



Сверху: **documentElement** и **body**

Самые верхние элементы дерева доступны как свойства объекта **document**:

**<html> = document.documentElement.** Самый верхний узел документа: **document.documentElement**. В DOM он соответствует тегу **<html>**.

**<body> = document.body**. Другой часто используемый DOM-узел – узел тега **<body>: document.body**.

**<head> = document.head.** Тег <head> доступен как **document.head**.

## Дети: childNodes, firstChild, lastChild

Здесь и далее будет использоваться два принципиально разных термина:

Дочерние узлы (или дети) – элементы, которые являются непосредственными детьми узла. Другими словами, элементы, которые лежат непосредственно внутри данного. Например, **<head>** и **<body>** являются детьми элемента **<html>.**

Потомки – все элементы, которые лежат внутри данного, включая детей, их детей и т.д.

В примере ниже детьми тега **<body>** являются теги **<div>** и **<ul>** (и несколько пустых текстовых узлов):

<html>

<body>

    <div>Начало</div>

    <ul>

        <li>

            <b>Информация</b>

        </li>

    </ul>

</body>

</html>

А потомки **<body>** – это и прямые дети **<div>**, **<ul>** и вложенные в них: **<li>** (потомок **<ul>**) и **<b>** (потомок **<li>**) – в общем, все элементы поддерева.

Коллекция **childNodes** содержит список всех детей, включая текстовые узлы.

Пример ниже последовательно выведет детей **document.body**:

<html>

<body>

    <div>Начало</div>

    <ul>

        <li>Информация</li>

    </ul>

    <div>Конец</div>

    <script>

        for (let i = 0; i < document.body.childNodes.length; i++) {

            alert(document.body.childNodes[i]); // Text, DIV, Text, UL, ..., SCRIPT

        }

    </script>

    ...какой-то HTML-код...

</body>

</html>

Обратим внимание на маленькую деталь. Если запустить пример выше, то последним будет выведен элемент **<script>**. На самом деле, в документе есть ещё «какой-то HTML-код», но на момент выполнения скрипта браузер ещё до него не дошёл, поэтому скрипт не видит его.

Свойства **firstChild** и **lastChild** обеспечивают быстрый доступ к первому и последнему дочернему элементу.

Они, по сути, являются всего лишь сокращениями. Если у тега есть дочерние узлы, условие ниже всегда верно:

elem.childNodes[0] === elem.firstChild

elem.childNodes[elem.childNodes.length - 1] === elem.lastChild

Для проверки наличия дочерних узлов существует также специальная функция **elem.hasChildNodes()**.

## DOM-коллекции

Как было описано ранее, **childNodes** похож на массив. На самом деле это не массив, а коллекция – особый перебираемый объект-псевдомассив.

И есть два важных следствия из этого:

1. Для перебора коллекции можно использовать **for..of**:

for (let node of document.body.childNodes) {

    alert(node); // покажет все узлы из коллекции

}

Это работает, потому что коллекция является перебираемым объектом (есть требуемый для этого метод **Symbol.iterator**).

2. Методы массивов не будут работать, потому что коллекция – это не массив:

alert(document.body.childNodes.filter); // undefined (у коллекции нет метода filter!)

Первый пункт – это хорошо. Второй – бывает неудобен, но можно пережить. Если хочется использовать именно методы массива, то можно создать настоящий массив из коллекции, используя **Array.from**:

alert( Array.from(document.body.childNodes).filter ); // сделан массив

## Соседи и родитель

Соседи – это узлы, у которых один и тот же родитель.

Например, здесь **<head>** и **<body>** соседи:

<html>

  <head>...</head><body>...</body>

</html>

* говорят, что **<body>** – «следующий» или «правый» сосед **<head>**
* также можно сказать, что **<head>** «предыдущий» или «левый» сосед **<body>**.

Следующий узел того же родителя (следующий сосед) – в свойстве **nextSibling**, а предыдущий – в **previousSibling**.

Родитель доступен через **parentNode**.

Например:

// родителем <body> является <html>

alert( document.body.parentNode === document.documentElement ); // выведет true

// после <head> идёт <body>

alert( document.head.nextSibling ); // HTMLBodyElement

// перед <body> находится <head>

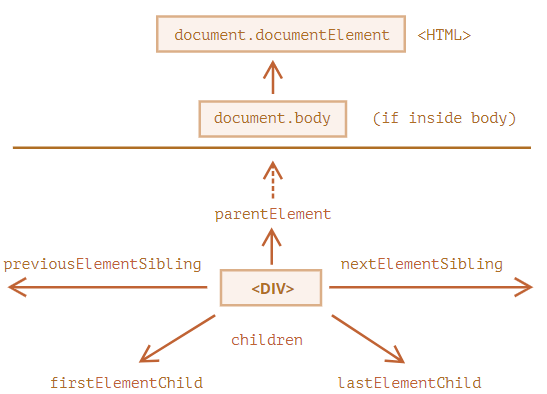
alert( document.body.previousSibling ); // HTMLHeadElement

## Навигация только по элементам

Навигационные свойства, описанные выше, относятся ко всем узлам в документе. В частности, в **childNodes** находятся и текстовые узлы и узлы-элементы и узлы-комментарии, если они есть.

Но для большинства задач текстовые узлы и узлы-комментарии нам не нужны. Необходимо манипулировать узлами-элементами, которые представляют собой теги и формируют структуру страницы.

Поэтому можно рассмотреть дополнительный набор ссылок, которые учитывают только узлы-элементы:



Эти ссылки похожи на те, что раньше, только в ряде мест стоит слово Element:

* **children** – коллекция детей, которые являются элементами.
* **firstElementChild**, **lastElementChild** – первый и последний дочерний элемент.
* **previousElementSibling**, **nextElementSibling** – соседи-элементы.
* **parentElement** – родитель-элемент.

Изменим один из примеров выше: заменим **childNodes** на **children**. Теперь цикл выводит только элементы:

<html>

<body>

    <div>Начало</div>

    <ul>

        <li>Информация</li>

    </ul>

    <div>Конец</div>

    <script>

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

            alert(elem); // DIV, UL, DIV, SCRIPT

        }

    </script>

    ...

</body>

</html>

## Таблицы

До сих пор происходило описание основных навигационных ссылок.

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа.

Таблицы – отличный пример таких элементов.

Элемент **<table>**, в дополнение к свойствам, о которых речь шла выше, поддерживает следующие:

* table.rows – коллекция строк <tr> таблицы.
* table.caption/tHead/tFoot – ссылки на элементы таблицы <caption>, <thead>, <tfoot>.
* **table.tBodies** – коллекция элементов таблицы <tbody> (по спецификации их может быть больше одного).

**<thead>**, **<tfoot>**, **<tbody>** предоставляют свойство rows:

* **tbody.rows** – коллекция строк **<tr>** секции.

#### <tr>:

* **tr.cells** – коллекция **<td>** и **<th>** ячеек, находящихся внутри строки <tr>.
* **tr.sectionRowIndex** – номер строки **<tr>** в текущей секции **<thead>/<tbody>/<tfoot>**.
* **tr.rowIndex** – номер строки **<tr>** в таблице (включая все строки таблицы).

#### <td> and <th>:

* **td.cellIndex** – номер ячейки в строке <tr>.

Пример использования:

<table id="table">

    <tr>

        <td>один</td>

        <td>два</td>

    </tr>

    <tr>

        <td>три</td>

        <td>четыре</td>

    </tr>

</table>

<script>

    // выводит содержимое первой строки, второй ячейки

    alert(table.rows[0].cells[1].innerHTML) // "два"

</script>

## Ещё немного ссылок: таблицы

До сих пор происходило описание основных навигационных ссылок.

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа.

Таблицы – отличный пример таких элементов.

Элемент **<table>**, в дополнение к свойствам, о которых речь шла выше, поддерживает следующие:

* table.rows – коллекция строк <tr> таблицы.
* table.caption/tHead/tFoot – ссылки на элементы таблицы <caption>, <thead>, <tfoot>.
* table.tBodies – коллекция элементов таблицы <tbody> (по спецификации их может быть больше одного).

<thead>, <tfoot>, <tbody> предоставляют свойство rows:

tbody.rows – коллекция строк <tr> секции.

<tr>:

tr.cells – коллекция <td> и <th> ячеек, находящихся внутри строки <tr>.

tr.sectionRowIndex – номер строки <tr> в текущей секции <thead>/<tbody>/<tfoot>.

tr.rowIndex – номер строки <tr> в таблице (включая все строки таблицы).

<td> and <th>:

td.cellIndex – номер ячейки в строке <tr>.

Пример использования:

<table id="table">

    <tr>

        <td>один</td>

        <td>два</td>

    </tr>

    <tr>

        <td>три</td>

        <td>четыре</td>

    </tr>

</table>

<script>

    // выводит содержимое первой строки, второй ячейки

    alert(table.rows[0].cells[1].innerHTML) // "два"

</script>

# Контрольные вопросы

1. Какие существуют два основных набора ссылок для навигации по DOM дереву?
2. Кто такие потомки и родители в DOM дереве?
3. Какая функция существует для проверки наличия дочерних узлов?

# Задания

#### Задание 1.

Для страницы:

<html>

<body>

    <div>Пользователи:</div>

    <ul>

        <li>Джон</li>

        <li>Пит</li>

    </ul>

</body>

</html>

Написать код, как получить…

* элемент <div>?
* <ul>?
* второй <li> (с именем Пит)?

#### Задание 2.

Если elem – произвольный узел DOM-элемента…

Правда, что elem.lastChild.nextSibling всегда равен null?

Правда, что elem.children[0].previousSibling всегда равен null ?

**Привести примеры**.

#### Задание 3.

Выделить ячейки по диагонали

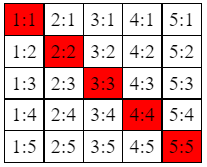
Написать код, который выделит красным цветом все ячейки в таблице по диагонали.

Нужно получить из таблицы **<table>** все диагональные **<td>** и выделить их, используя код:

// в переменной td находится DOM-элемент для тега <td>

    td.style.backgroundColor = 'red';

Должно получиться так:



# 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