Введение в DOM

Курс Frontend-разработки на Javascript + Vue/React

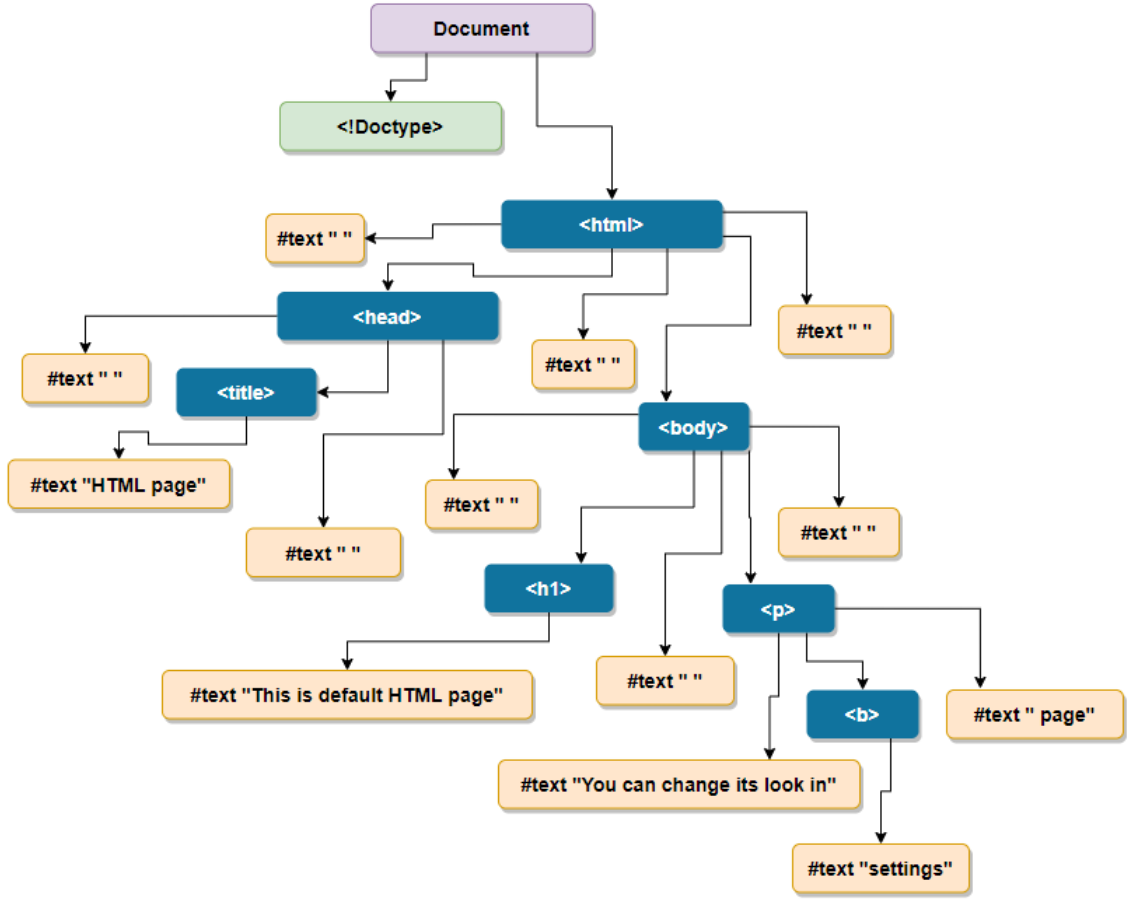
Дерево документа

- Веб-страницы хранятся в памяти браузера в виде дерева документа.
- При загрузке страницы браузер превращает HTML-разметку в дерево.
- Изменения, которые вносятся на страницу (например, скриптом), вносятся в дерево документа, а после этого браузер перерисовывает всю страницу в соответствии с новым деревом.

Дерево документа

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML page</title>
    </head>

    <body>
        <h1>This is default HTML page</h1>
        <p>You can change its look in <b>settings</b> page</p>
    </body>
</html>
```



Понятие DOM

- DOM (Document Object Model) – набор возможностей в некотором окружении, который позволяет работать с некоторым документом как с деревом.
- В браузере DOM реализован как набор возможностей в глобальной области видимости и доступный в Javascript
- При помощи DOM скрипт может во время своей работы общаться с деревом документа, получать из него информацию и вносить изменения
- Работа с DOM в Javascript начинается с объекта document в глобальной области видимости.

Типы узлов в документе

- В браузерном DOM-дереве существует всего более десятка различных типов элементов, из которых может состоять дерево. Среди них самые популярные:

Тип узла	Константа и значение	Описание
Элемент	`document.ELEMENT_NODE (1)`	Элемент (тег вместе со своими атрибутами и содержимым)
Текстовый узел	`document.TEXT_NODE (3)`	Текстовый узел – участок текста от любого открывающего/закрывающего тега до ближайшего открывающего/закрывающего.
Атрибут	`document.ATTRIBUTE_NODE (2)`	Атрибут тега. При работе с DOM в скриптах не применяется
Комментарий	`document.COMMENT_NODE (8)`	Комментарий в разметке страницы
Блок DOCTYPE	`document.DOCUMENT_TYPE_NODE (10)`	Блок <!doctype html>
CDATA-секция	`document.CDATA_SECTION_NODE (4)`	CDATA-секция для XML-сайтов. В обычном вебе не применяется.
Фрагмент	`document.DOCUMENT_FRAGMENT_NODE (11)`	Отдельное дерево документа, которое хранится в памяти и применяется для ёмких операций над деревом до вставки в страницу (Shadow DOM)
Документ	`document.DOCUMENT_NODE (9)`	Весь документ (выше уровня тега HTML)

DOM-объект

- DOM-объект – объект языка Javascript специального типа. Этот объект обеспечивает двустороннюю связь JS-кода и одного узла в дереве страницы, на которой запущен скрипт.
    - Внесение изменений в DOM-объект в скрипте ведёт к изменению дерева документа и перерисовке страницы.
    - Наоборот, если дерево документа было изменено каким-либо образом, то DOM-объект обновится до актуального состояния в дереве.
    - Работа со страницей в Javascript сводится к работе с DOM-объектами дерева этой страницы.

Основные DOM-объекты

```
document

document.head

document.body

document.documentElement
```

- document – стартовый объект для работы с деревом. Он указывает на корневой узел документа.
- document.head – DOM-объект, указывающий на тег <head> страницы.
- document.body – DOM-объект, указывающий на тег body в странице
- document.documentElement – указывает на тег <html> страницы.

Навигация по дереву документа

```
const panel = document.getElementById("panel");

const panel = document.querySelector("body .wrapper #panel");
```

- Метод document.getElementById вернет DOM-объект того элемента, у которого атрибут id равен (в нашем случае) panel.

- Метод `document.querySelector` вернёт первый элемент в дереве, который подпадает под CSS-селектор, указанный в методе.

- Если методы навигации не нашли подходящего элемента, то они вернут null.

Построение дерева документа

- Дерево документа строится при загрузке документа постепенно.

- Выполнение скрипта на странице останавливает построение дерева. Это значит, что элементы, которые в HTML-коде записаны ниже скрипта, не попадут в дерево, с которым скрипт сможет работать. Они попадут в дерево только когда скрипт закончит свою работу.

- По этой причине скрипты желательно размещать в самом конце тега <body>

Примеры

- Построение дерева документа для некоторой страницы, начиная с тега body

Управление содержимым элемента

- Если в скрипте хранится DOM-объект, то с его помощью можно читать и перезаписывать содержимое этого узла.

- Внесенные изменения сразу же отразятся на странице.

Управление содержимым элемента

Свойство	При чтении	При записи
`elem.innerHTML`	Строка с HTML-кодом внутренностей этого элемента	Сотрет содержимое элемента и задаст ему новый HTML-код
`elem.innerText`	Строка со всем текстом внутри элемента (без тегов и атрибутов)	Сотрет содержимое элемента и добавит в него один текстовый узел с текстом, который записывается
`elem.outerHTML`	Строка с HTML-кодом элемента (тег, атрибуты и внутренности)	Удалит элемент и на его место вставит HTML-код, который записывается
`elem.outerText`	То же, что innerText	Удалит элемент из дерева и на его место вставит один текстовый узел с текстом, который записывается
`elem.textContent`	То же, что и innerText. Но вернёт также текст внутри тегов style, script	То же, что и innerText

Примеры

1. Вывод результатов вычислений на страницу
2. Генерация HTML-кода для некоторого объекта
3. Вывод массива объектов в HTML-список

Множественная выборка элементов

- В DOM есть методы, которые позволяют выбрать сразу несколько элементов за раз.
- Такие методы возвращают не DOM-объект, а коллекцию типа NodeList.
- NodeList – индексируемая коллекция. У неё есть длина и индексы, как у массивов, но нет методов обхода массивов.
- Множественную выборку можно запускать на любом DOM-объекте (элементе). В этом случае поиск будет осуществляться только внутри этого элемента.

Множественная выборка элементов

Метод	Тип поиска
`getElementsByClassName(c)`	Вернет коллекцию элементов со страницы (из элемента), у которых есть класс c.
`getElementsByTagName(t)`	Вернет коллекцию элементов со страницы (из элемента), у которых название тега равно t
`getElementsByName(n)`	Вернет коллекцию элементов, у которых атрибут name равен n
`querySelectorAll(s)`	Вернет коллекцию элементов, которые подпадают под CSS-селектор в скобках.

Примеры

```
const list = document.getElementById("list");
const items = list.querySelectorAll("li.item");

for (let i = 0; i < items.length; i++) {
    items[i].innerHTML = `Item #${i}`;
}

// OR as following
Array.from(items).forEach((item, i) => {
    item.innerHTML = `Item #${i}`;
});
```

- NodeList можно обойти при помощи обычного цикла с итератором.
- Также можно использовать преобразование в массив, чтобы обойти набор элементов при помощи методов обхода.

Задача (самостоятельно)

- Создать на странице несколько тегов div. В каждом – создать несколько тегов p.
- Пройти по странице и проставить в теги <p> номер этого тега в виде X.Y, где X – номер div-а, а Y номер тега <p> внутри этого div.

Работа с атрибутами

- В Javascript работа с атрибутами элемента сводится к использованию методов управления атрибутами:

Метод	Назначение
`e.setAttribute(name, value)`	Задаст атрибуту name значение из переменной value в элементе, что хранится в elem
`e.getAttribute(name)`	Вернёт значение атрибута с названием из переменной name в элементе elem
`e.hasAttribute(name)`	Вернёт true, если у элемента в elem есть атрибут с названием из переменной name, false в противном случае
`e.removeAttribute(name)`	Удалит атрибут name с его значением из элемента elem

Примеры

```
const paragraphs = document.getElementsByTagName('p');
Array.from(paragraphs).forEach((p, i) => {
    p.setAttribute('title', `Paragraph #${i}`);
});
```

Вставка дополнительного содержимого

- Использование innerHTML для последовательного заполнения содержимого элемента опасно по некоторым причинам:
  - Использование innerHTML += полностью перезаписывает содержимое элемента. При этом удаляется и заново создается часть дерева внутри элемента. Это приводит к колоссальным затратам производительности.
  - При перезаписи HTML-содержимого с внутренних элементов будут удалены все обработчики событий

- Решений у проблемы несколько:
  - Собирать HTML-код в переменную и записывать в конце цикла
  - Использовать метод вставки дополнительного содержимого
  - Создавать точечно DOM-узлы и добавлять их в элемент

Вставка дополнительного содержимого

- Вставка дополнительного HTML-содержимого осуществляется вызовом метода insertAdjacentHTML(position, content)

- В качестве position указывается строка, показывающая, куда относительно элемента будет вставлено содержимое:

```
<!-- beforebegin -->
<p>
<!-- afterbegin -->
foo
<!-- beforeend -->
</p>
<!-- afterend -->
```

```
const elem = document.getElementById('list');
for(let i = 0; i < 10; i++) {
  elem.insertAdjacentHTML('beforeend', `<li>Element #${i}</li>`);
}
```

Создание новых элементов

Метод	
`document.createElement(name)`	Создаст новый узел для дерева документа и вернёт в качестве результата. Созданный узел будет находиться отдельно от дерева документа и не будет виден на странице.
`elem.appendChild(newElement)`	Вставляет узел из newElement в самый конец элемента в elem. Если newElement уже был в дереве, он будет удален из старого места и помещен в новое.
`elem.prepend(newElement)`	Вставляет узел из newElement в самое начало внутренностей элемента в elem.
`elem.cloneNode()`	Делает копию элемента. Такую копию потом можно вставить в другое место без опасения, что он будет удален из старого места.

```
const p = document.createElement('p');
p.innerText = 'Paragraph, generated from JS!';
p.setAttribute('title', 'Custom title');
document.body.appendChild(p);
```

Создание новых элементов

- Создать новые элементы можно просто добавив их HTML-код в innerHTML у родительского элемента.

- В своих скриптах следует использовать способ с разметкой – это работает лучше, чем с узлами дерева напрямую.

- Тем не менее, вставка через HTML-код не даёт такой гибкости, как в случае с узлами – таким элементам невозможно быстро навесить обработчики событий и свойства, недоступные через HTML-разметку.

Примеры

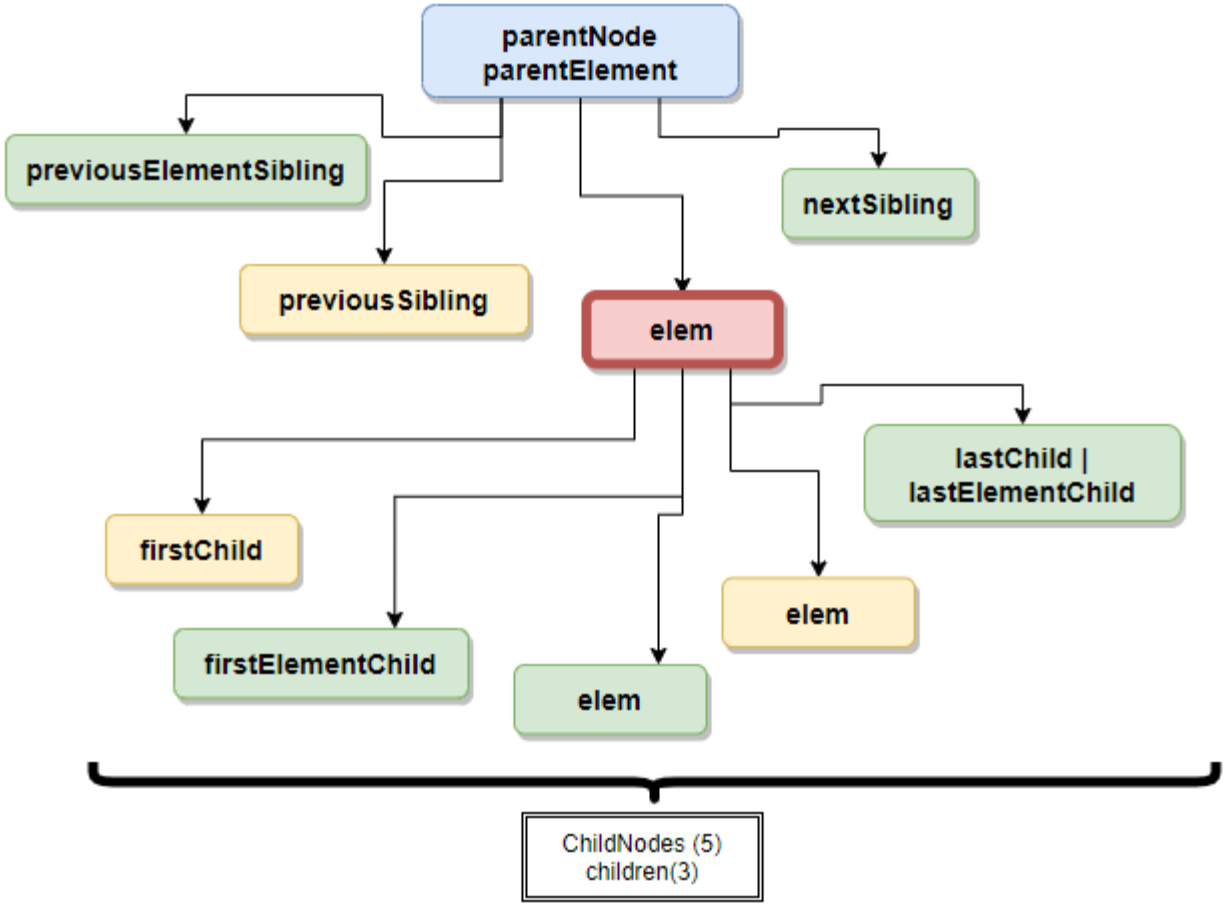
- Создание списка пользователей из массива объектов через создание DOM-узлов.
- Генерация таблицы

Самостоятельно

- Создать массив из объектов, в каждом объекте – название вида спорта и массив имен спортсменов, которые участвуют в этом виде спорта. Сгенерировать на страницу список видов спорта, в каждом списке – название и вложенный список имён.

Относительная навигация в дереве

- Имея в переменной некоторый DOM-объект, мы можем делать от него навигацию по детям, родителям и соседям. Для этого не нужно знать, где именно в дереве находится этот элемент.



Относительная навигация в дереве документа

Свойство навигации	Вернет	Описание
`elem.parentNode`	`DOM Node| null`	Вернёт объект родительского узла (любого типа), если таковой есть
`elem.parentElement`	`DOM Element | null`	Вернёт объект родительского узла – тега, если он есть
`elem.nextSibling`	`DOM Node | null`	Вернёт следующего соседа по дереву, если есть
`elem.nextElementSibling`	`DOM Element | null`	Вернёт следующего соседа – тег, если он есть. Текстовые узлы и прочее будут пропущены
`elem.previousSibling`	`DOM Node | null`	Вернёт предыдущего соседа по дереву, если такой есть
`elem.previousElementSibling`	`DOM Element | null`	Вернёт предыдущего соседа-тег, если есть
`elem.firstChild`	`DOM Node | null`	Вернёт первый узел внутри элемента, если такой есть
`elem.firstElementChild`	`DOM Element | null`	Вернёт первый тег внутри элемента, если есть
`elem.lastChild`	`DOM Node | null`	Вернёт последний узел внутри элемента, если есть
`elem.lastElementChild`	`DOM Element | null`	Вернёт последний тег внутри элемента, если такой есть
`elem.childNodes`	`NodeList`	Вернёт коллекцию всех узлов, лежащих напрямую в элементе
`elem.children`	`HTMLCollection`	Вернёт коллекцию всех тегов, которые лежат напрямую в элементе

Примеры

- Относительная навигация в дереве документа по стартовому и конечному узлу.

Управление деревом документа

- Javascript при помощи DOM может модифицировать структуру дерева документа на уровне работы с узлами.

- На самом деле, работая с перезаписью HTML-кода в элементе, мы тоже изменяем структуру дерева документа. Однако этот подход в некоторых случаях может оказаться слишком громоздким или негибким.

- В DOM существуют операции, позволяющие произвести точечные изменения дерева документа.

Управление деревом документа

Метод	Описание
`elem.removeChild(childNode)`	Удалит элемент, который хранится в childNode из своего родителя (элемент elem)
`elem.remove()`	Удалит elem из дерева документа
`elem.appendChild(newNode)`	Добавит элемент из newNode в конец содержимого элемента в elem
`elem.prepend(newNode)`	Добавит элемент newNode в самое начало элемента elem
`elem.insertBefore(newNode, target)`	Вставит элемент newNode внутрь элемента в elem на позицию перед элементом target
`elem.replaceChild(newNode, target)`	Заменит внутри элемента elem элемент target на newNode.

- Обратите внимание: операции вставки элементов удалят вставляемый элемент из того места в дереве, где он находится в данный момент.

Примеры

- Создание и удаление элементов по таймеру
- Перестановка элементов по таймеру в списке

Дополнительные материалы

Timers, intervals

https://javascript.info/settimeout-setinterval

Введение в DOM

https://developer.mozilla.org/ru/docs/DOM/DOM_Reference/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5

Global search in DOM

https://learn.javascript.ru/searching-elements-dom

DOM Navigation

https://learn.javascript.ru/traversing-dom

NodeList

https://developer.mozilla.org/ru/docs/Web/API/NodeList

Работа с атрибутами

https://developer.mozilla.org/ru/docs/Web/API/Element/setAttribute

Insert Adjacent HTML

https://developer.mozilla.org/ru/docs/Web/API/Element/insertAdjacentHTML

insertBefore

https://developer.mozilla.org/ru/docs/Web/API/Node/insertBefore