Модули и браузерное окружение

Кирилл Талецкий

TeachMeSkills
21 августа 2023

Подключение JS к HTML

Подключение к HTML

- Тег <script />
  - Можно разместить в <head /> и в <body />
  - Можно написать скрипт внутри тега или загрузить через атрибут `src`

- **Скрипты блокирую отрисовку страницы!**
- **Поэтому скрипты “не видят” элементы, ниже себя**

Подключение к HTML

- **Скрипты блокирую отрисовку страницы!**
- **Поэтому скрипты “не видят” элементы, ниже себя**

```
<p>...содержимое перед скриптом...</p>

<script defer src=“https://cdn.jsdelivr.net/gh/ktaletski/host/hello.js"></script>

<!-- Это не отобразится, пока скрипт не загрузится -->
<p>...содержимое после скрипта...</p>
```

Асинхронное подключение к HTML

- async / defer — атрибуты, указывающие, что скрипт нужно загрузить параллельно с HTML, в фоне, не блокируя отрисовку

Асинхронное подключение к HTML

- defer — скрипт запустится только после того, когда весь HTML будет загружен доступен для скриптов (DOM построен)
    - Также обеспечивает правильный порядок запуска скриптов
    - Удобно использовать для серий больших скриптов

```
<p>...содержимое перед скриптом...</p>

<script defer src="https://cdn.jsdelivr.net/gh/ktaletski/host/script-1.js"></script>
<script defer src="https://cdn.jsdelivr.net/gh/ktaletski/host/script-2.js"></script>

<!-- Это не отобразится, пока скрипт не загрузится -->
<p>...содержимое после скрипта...</p>
```

Асинхронное подключение к HTML

- async — скрипт запустится сразу же, как только загрузится
  - **Не** обеспечивает правильный порядок запуска скриптов
  - Удобно использовать для скриптов не взаимодействующих с вёрсткой (аналитика и т.д.)

```
<p>...содержимое перед скриптом...</p>

<script async src="https://cdn.jsdelivr.net/gh/ktaletski/host/script-1.js"></script>
<script async src="https://cdn.jsdelivr.net/gh/ktaletski/host/script-2.js"></script>

<!-- Это не отобразится, пока скрипт не загрузится -->
<p>...содержимое после скрипта...</p>
```

Асинхронное подключение к HTML

- async / defer предназначены только для внешних скриптов:

  Атрибут async / defer будет проигнорирован, если в теге <script /> не будет атрибута `src`

Введение в модули

Модули

- По мере роста проекта, нам рано или поздно захочется разбивать скрипты на отдельные файлы

- Для этого в стандарт ES6 (2015) были добавлены модули
  - До этого, модальность достигалась за счёт сторонних библиотек
  - Серверный JS рантайм NodeJS ранее добавил свой формат импортов — CommonJS — вы его встретите в скриптах настраивающих окружение

Модули

- Модуль — это файл.
- Модули могут использовать функционал друг друга
  - `export` помечает переменные и функции, которые могут использовать другие модули
  - `import` объявляет, какую функциональность мы забираем из других модулей

Импорт и экспорт

Экспорт до объявления

```
// экспорт массива
export let months = ['Jan', 'Feb', 'Mar', 'Apr',
'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];

// экспорт константы
export const MODULES_BECAME_STANDARD_YEAR = 2015;

// экспорт функции
export function kek() {
  console.log('lol');
}
```

Экспорт отдельно от объявления

```
function sayHi(user) {
  alert(`Hello, ${user}!`);
}

function sayBye(user) {
  alert(`Bye, ${user}!`);
}

export { sayHi, sayBye }; // список экспортируемых переменных
```

Именованный импорт

```
import {sayHi, sayBye} from './say.js';

sayHi('John'); // Hello, John!
sayBye('John'); // Bye, John!
```

Импорт всего

```
import * as say from './say.js';

say.sayHi('John');
say.sayBye('John');
```

- **Важно**: старайтесь избегать подобных импортов, потому что они не позволяют сборщикам удалять неиспользуемый код

Переименованный импорт

```
import {sayHi as hi, sayBye as bye} from './say.js';

hi('John'); // Hello, John!
bye('John'); // Bye, John!
```

Экспорт по умолчанию

```
// user.js
export default function User(name) {
  this.sayHi = function () {
    console.log('I am a user ' + name)
  };
}
```

```
// main.js
import User from './user.js'; // не {User},
просто User

new User('John');
```

- Важно: неименованные экспорты не удобны при анализе и отладке кода

Сборщики проектов

Сборщики проектов

- Современный фронтенд код может быть очень сложным и разветвлённым
- Сегодня для сборки большого количества модулей в один используются сборщики
- Сборщики
  - Дают больший контроль над поиском, преобразованием и подключением файлов модулей
  - Позволяют подключать другие типы файлов (css модули, картинки, SVG), управлять их хранением и многое другое

Сборщики проектов

- Как работают сборщики (основы)
  - Заходят в основной JS файл — точку входа
  - Анализируют зависимости — проходят по всем импортам, импортам импортов и т.д.)
  - Собирают все модули в один огромный JS файл
  - Подключают этот JS файл к index.html

Сборщики проектов

- Сборщики также умеют
  - Удалять код, который нигде не импортируется (tree-shaking)
  - Трансформировать новый синтаксис в старый, поддерживаемый большим количеством браузеров (Babel, core-js)
  - Минифицировать код — удалить пробелы, заменить названия переменных на более короткие
  - Code Splitting — разбить JS файл на более маленькие кусочки (chunk) и подгружать их по мере необходимости

Сборщики проектов

- Мы будем рассматривать Webpack — он очень широко используется. Освоив Webpack, вы легко освоите остальные сборщики.

- Есть и другие сборщики проектов
  - Gulp — хорош для того, чтобы собирать библиотеки (о них далее)
  - Vite — хороший современный и быстрый сборщик, пока не так уж часто встречается в продуктах
  - SWC — сборщик для Next.js фреймворка

Менеджеры пакетов

Менеджеры пакетов

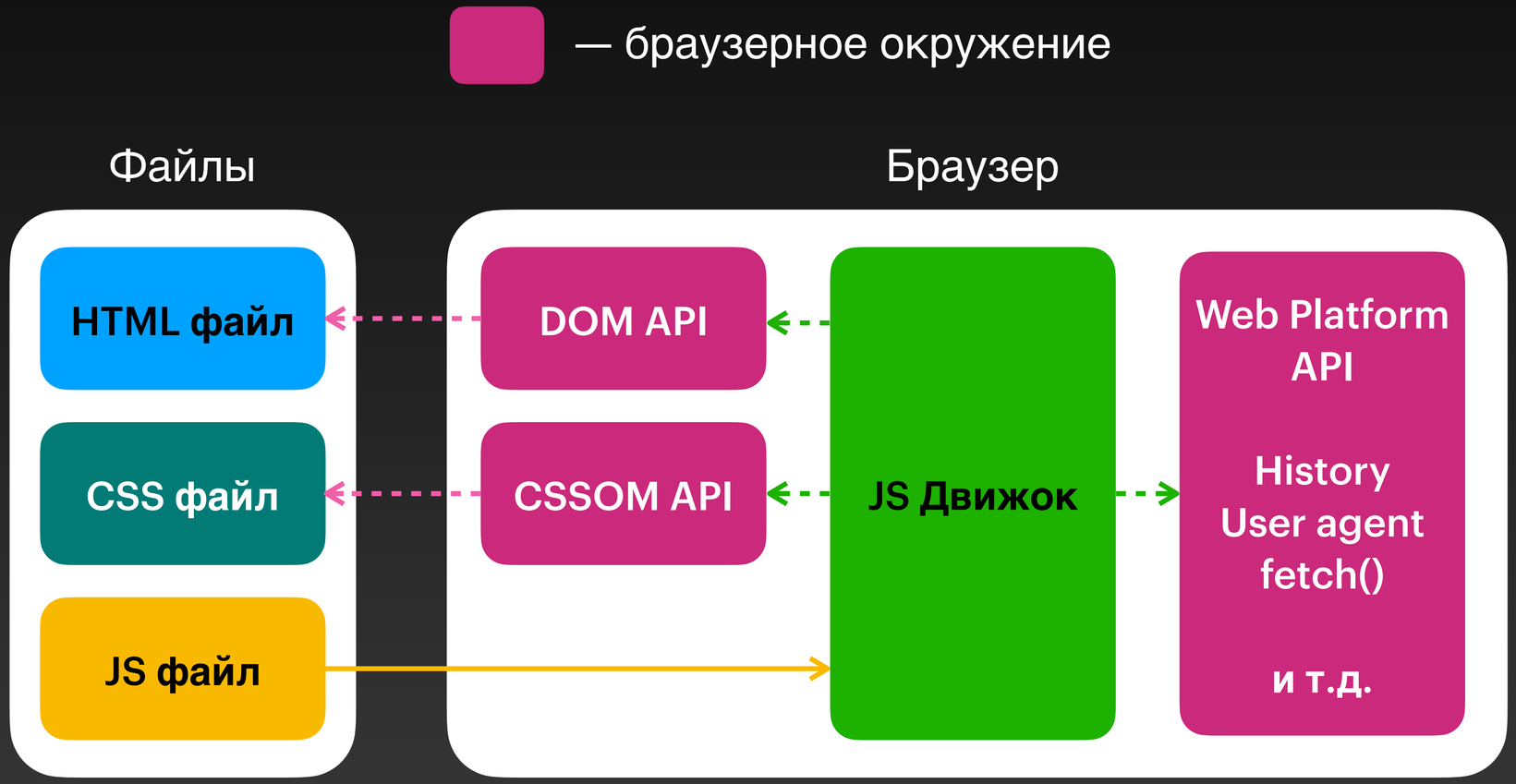
- NPM — Node Package Manager — исторически самый первый репозиторий JS библиотек и программа-менеджер пакетов

- Yarn — более поздний аналог от Facebook. В своё время был прорывным, но сейчас npm сравнялся с ним по удобству.

Практика: создание проекта

Браузерное окружение

Браузерное окружение



Браузерное окружение

window

DOM

document

documentElement
head
body
getElementById

CSSOM

FontFace()
CSSRule()
elem.matchMedia()

BOM

navigator
screen
location
history
XMLHttpRequest

JavaScript

Object()
Array()
Function()

Document Object Model

Document Object Model

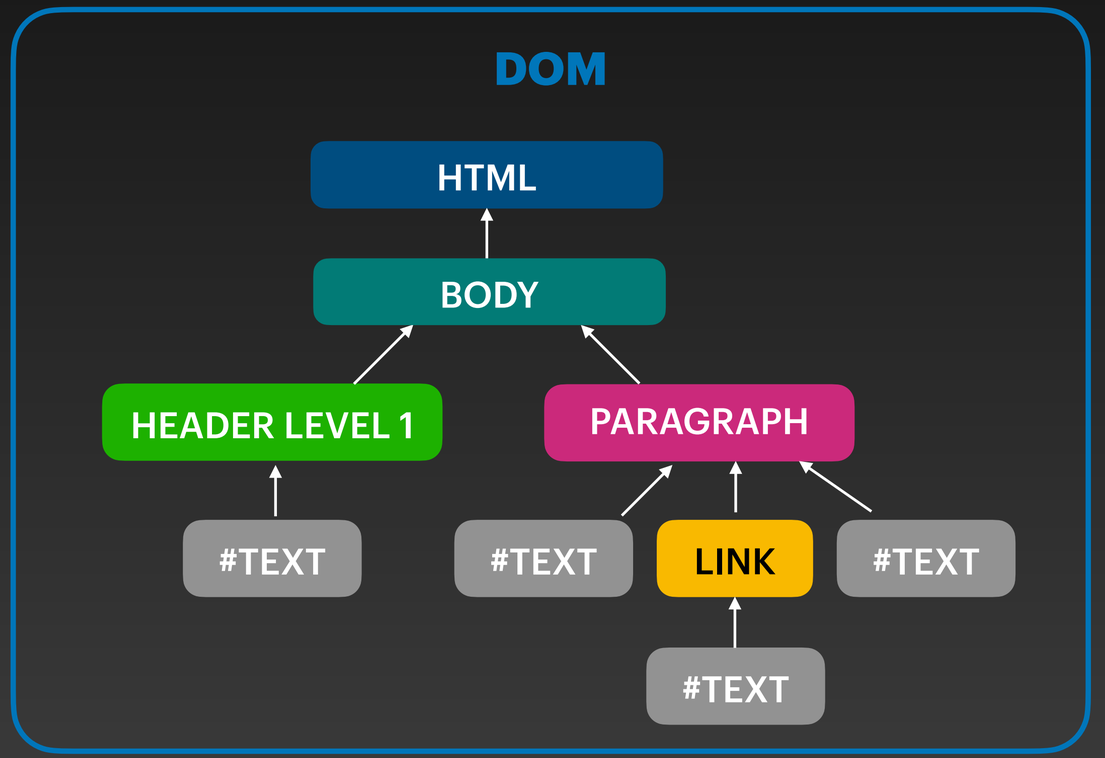
- Document Object Model или кратко DOM— представление html документа в виде вложенных объектов

- Каждый тег (, <img /> и т.д.) или текст является объектом, который содержит информацию о стилях и атрибутах, а также методы для их изменения.

- DOM — это дерево. Вложенные теги являются детскими узлами родительского тега.

Document Object Model

```
<body>
  <h1>Hello world!</h1>
  <p>
    This is some paragraph text. It is
    long and it has a <a>link</a> inside
    of it
  </p>
</body>
```



Document Object Model

- Важно — всё, что есть в HTML будет отражено в DOM
  - Комментарии
  - Переносы строк и пробелы (в том числе и отступы)

- DOM всегда пытается исправить некорректные HTML документы
  - Добавляет пропущенные закрывающие теги
  - Добавляет пропущенные обязательные теги (html, head, body)

Document Object Model

- Виды DOM узлов:
  - document — всегда один, является корневым (входная точка)
  - теги
  - текст
  - комментарии

Навигация по DOM элементам

Навигация по DOM

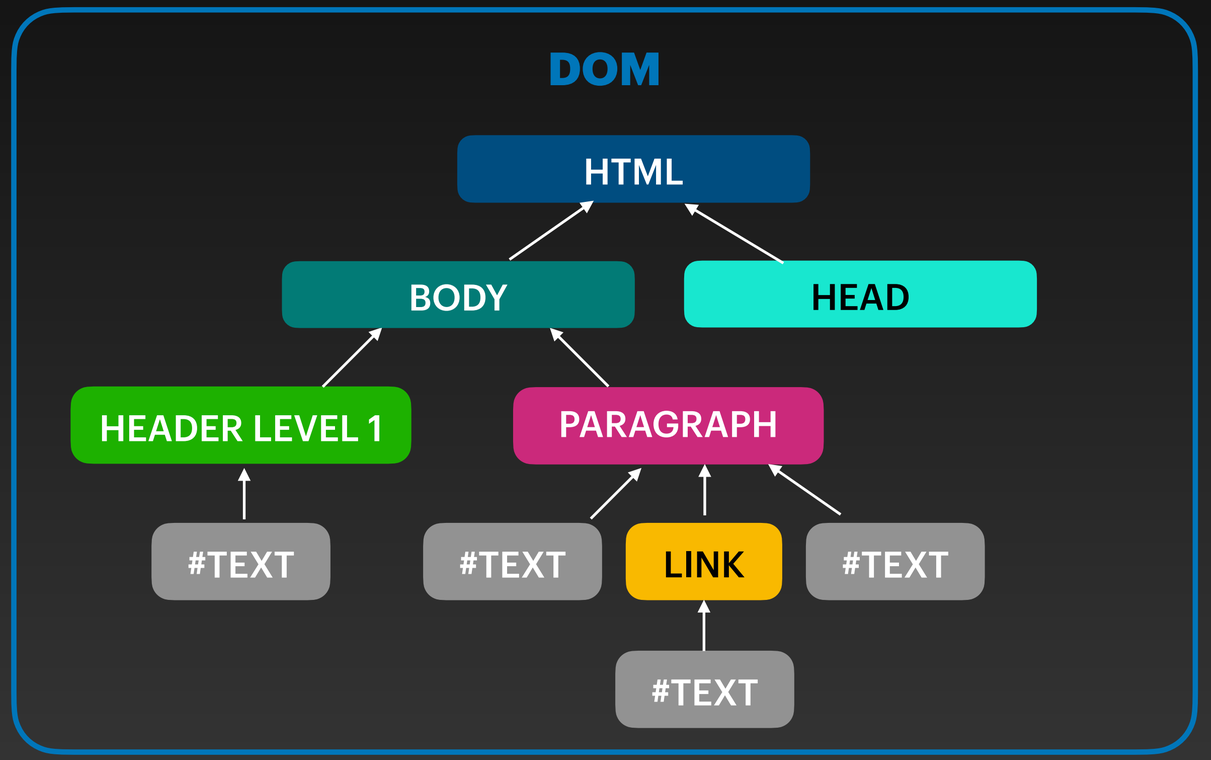
HTML

HEAD

BODY

```
document.documentElement

document.head

document.body
```

```
<html>
  <head>Hello World!</head>
  <body>
    <h1>Hello World!</h1>
    <p>
    This is some paragraph text. It is long and it
    has a <a>link</a> inside it
    </p>
  </body>
</html>
```



В каком случае document.body будет равен null?

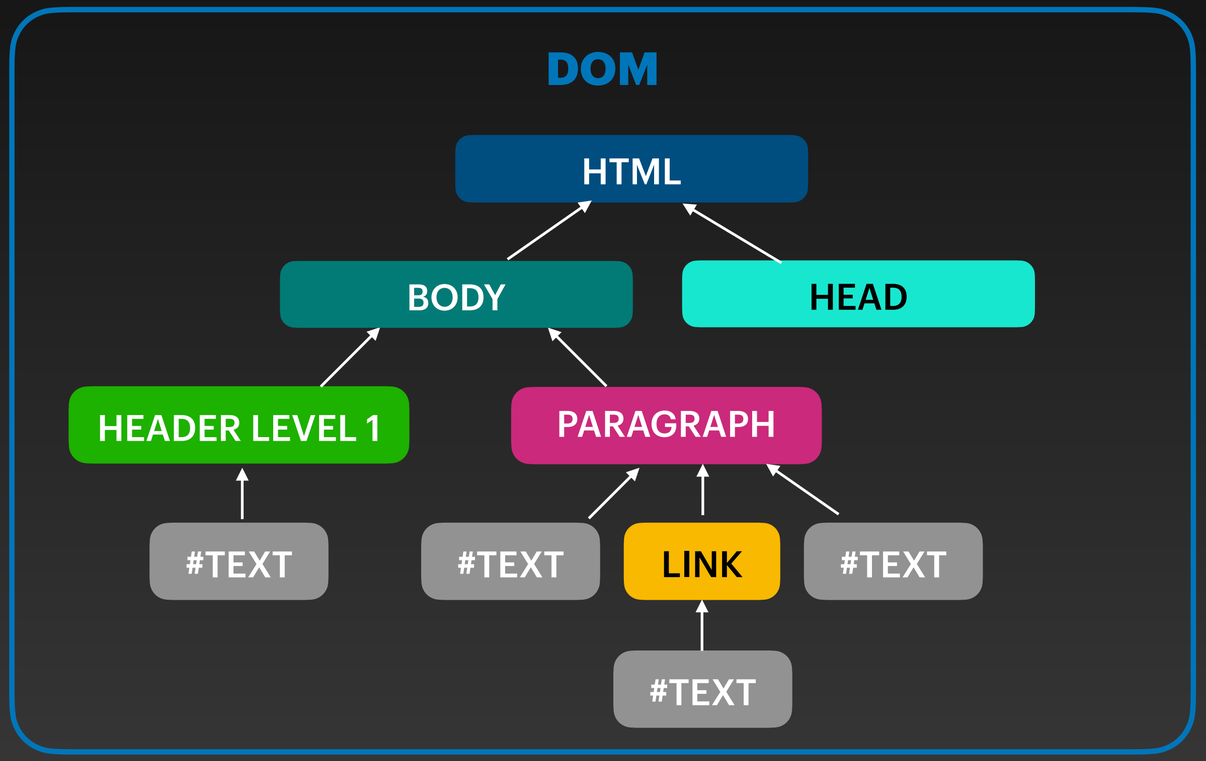
Навигация по DOM

Навигация по детям

- **Дети** — элементы, которые непосредственно вложены в элемент
- **Потомки** — все элементы внутри данного

BODY

Дети	Потомки
HEADER LEVEL 1	HEADER LEVEL 1
PARAGRAPH	PARAGRAPH
	LINK
	#TEXT x4



Навигация по DOM

Навигация по детям

```
/**
 * Все дети, включая текстовые узлы
 */
elem.childNodes
```

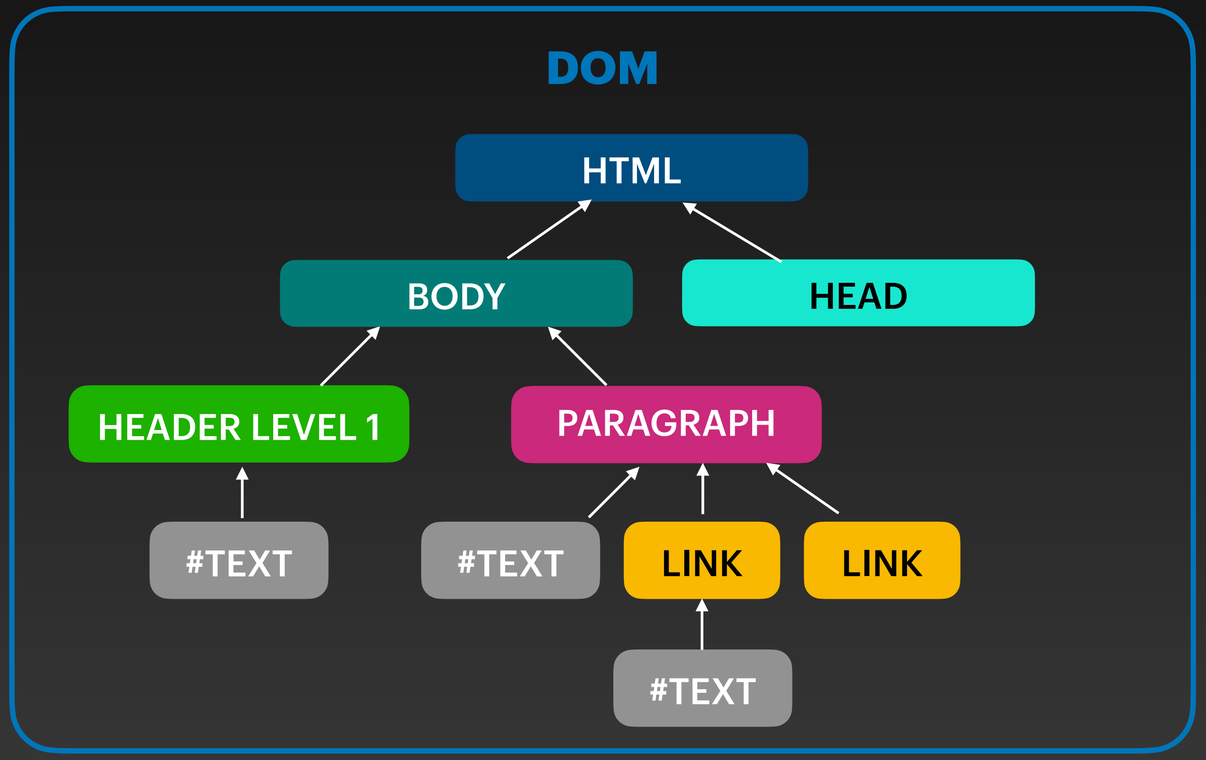
#TEXT LINK LINK

```
/**
 * Первый детский узел
 */
elem.firstChild
```

#TEXT

```
/**
 * Последний детский узел
 */
elem.lastChild
```

LINK



DOM-коллекции

- Коллекции DOM элементов, такие как childNodes — псевдомассивы
  - У них есть length и числовые индексы
  - Их можно обойти через цикл for ... of
  - У них нет методов массивов!

DOM-коллекции

- DOM коллекции только для чтения
- DOM коллекции живые (за редким исключением)
  - Коллекция будет меняться если будет меняться DOM
- DOM коллекции содержат некоторые вспомогательные свойства, поэтому не рекомендуется обходить их через for ... in

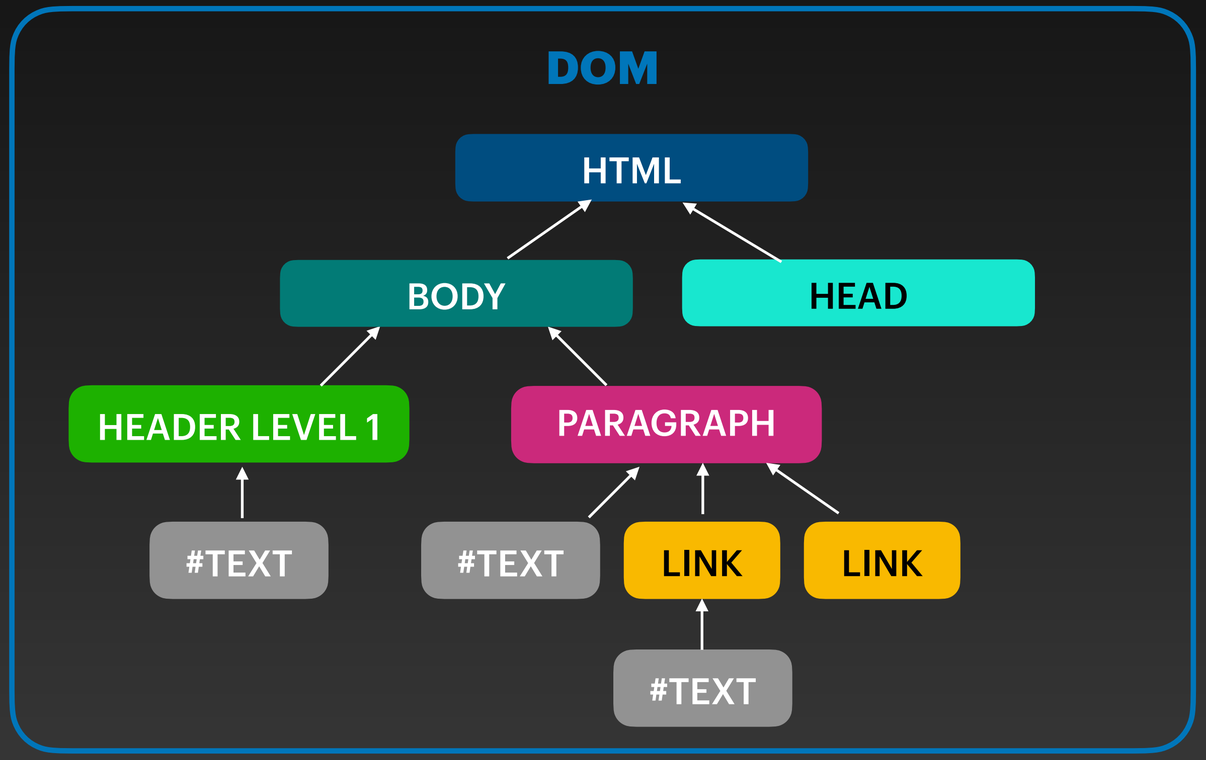
Навигация по DOM

Соседи (siblings) и родитель

```
let elem = document.body;

elem = elem.firstChild;

elem = elem.nextSibling;

elem = elem.previousSibling;

elem = elem.parentNode;
```

BODY

?



Навигация по DOM

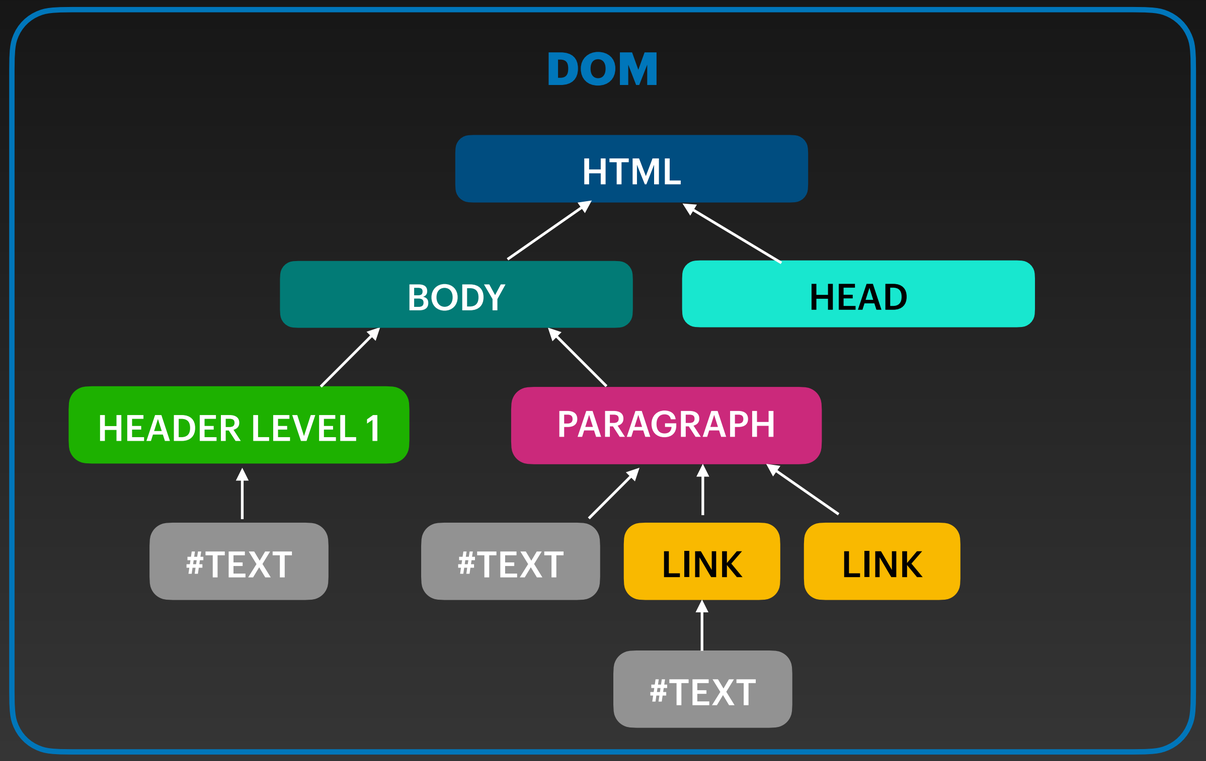
Соседи (siblings) и родитель

```
let elem = document.body;

elem = elem.firstChild;

elem = elem.nextSibling;

elem = elem.previousSibling;

elem = elem.parentNode;
```

BODY

HEADER LEVEL 1

?



Навигация по DOM

Соседи (siblings) и родитель

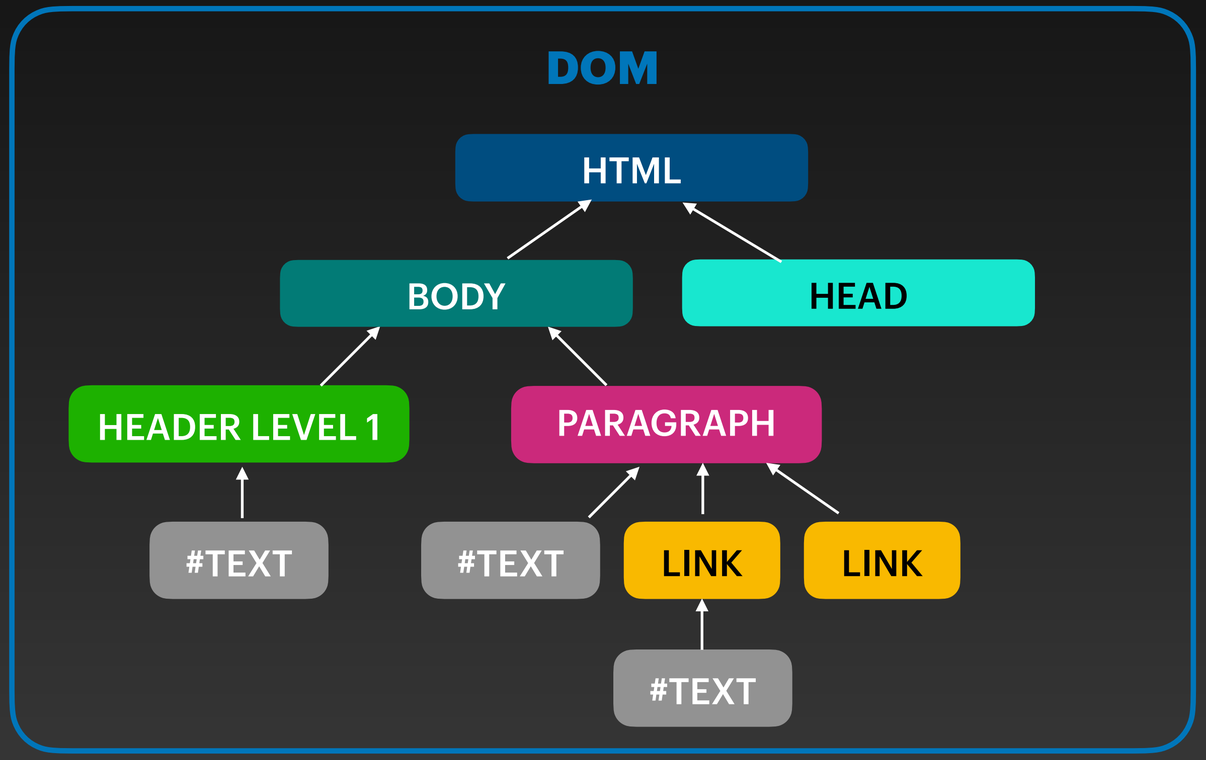
```
let elem = document.body;

elem = elem.firstChild;

elem = elem.nextSibling;

elem = elem.previousSibling;

elem = elem.parentNode;
```

BODY

HEADER LEVEL 1

PARAGRAPH

?



Навигация по DOM

Соседи (siblings) и родитель

```
let elem = document.body;
elem = elem.firstChild;
elem = elem.nextSibling;
elem = elem.previousSibling;
elem = elem.parentNode;
```

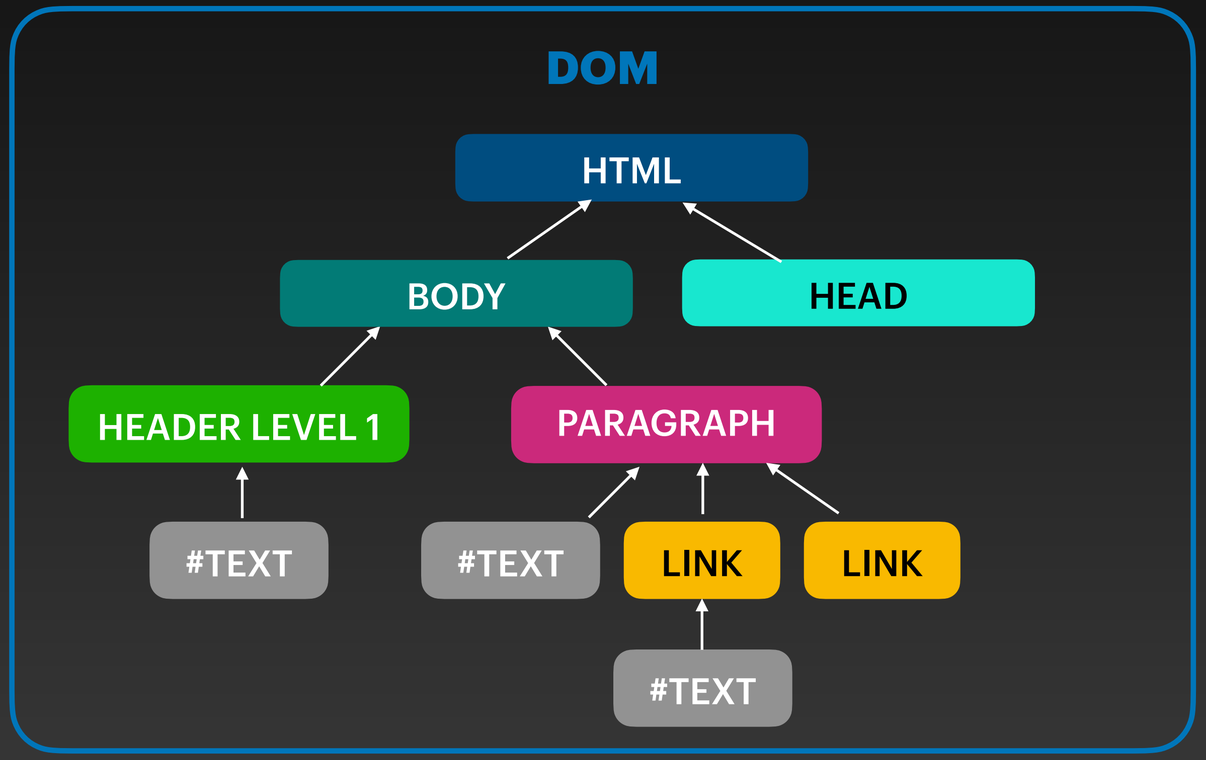
BODY

HEADER LEVEL 1

PARAGRAPH

HEADER LEVEL 1

?



Навигация по DOM

Соседи (siblings) и родитель

```
let elem = document.body;
elem = elem.firstChild;
elem = elem.nextSibling;
elem = elem.previousSibling;
elem = elem.parentNode;
```

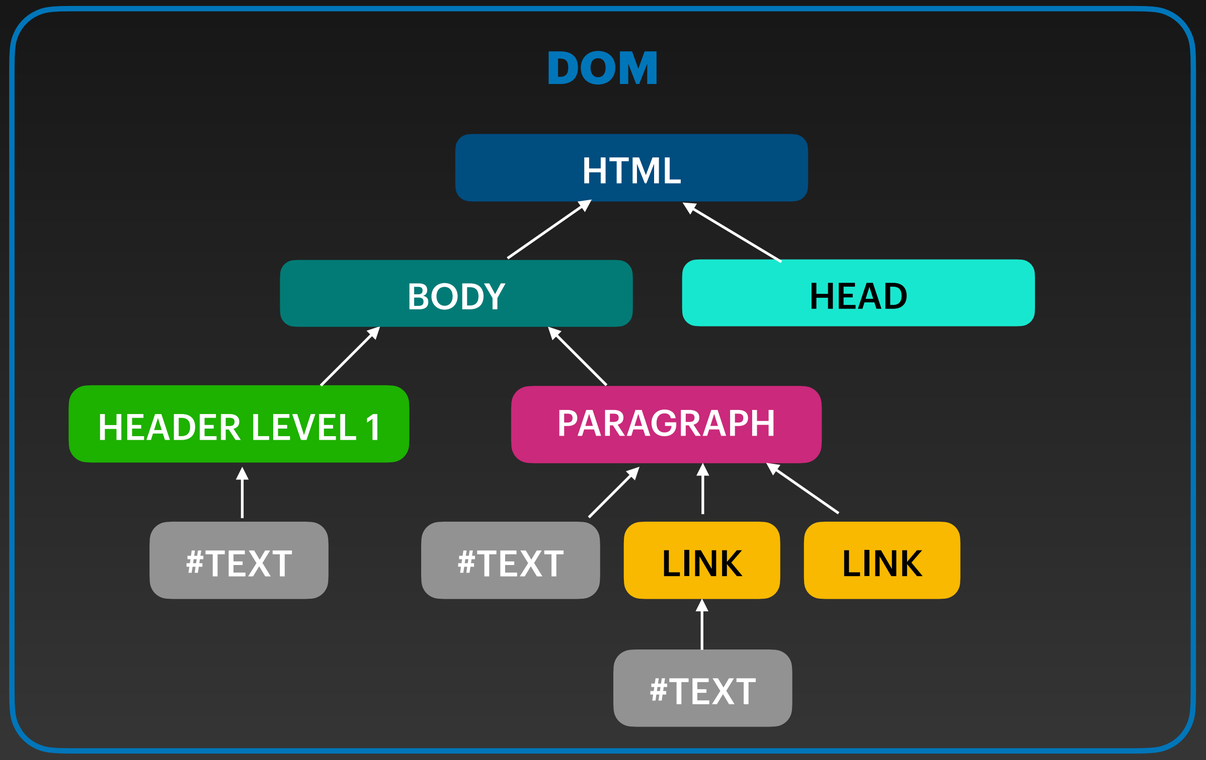
BODY

HEADER LEVEL 1

PARAGRAPH

HEADER LEVEL 1

BODY



Навигация по DOM
Элементы

- На практике мы часто хотим работать только с элементами, игнорируя узлы с текстом или комментариями.
- Для этого у элементов есть аналогичный набор свойств

Любые узлы

```
elem.childNodes;
elem.firstChild;
elem.lastChild;
elem.nextSibling;
elem.previousSibling;
elem.parentNode;
```

Только элементы

```
elem.children;
elem.firstElementChild;
elem.lastElementChild;
elem.nextElementSibling;
elem.previousElementSibling;
elem.parentElement;
```

Зачем нужен parentElement?