hillel IT school

Lesson 17

План заняття

- Деструктуроване присвоєння;
- Що таке DOM?
- Навігація по DOM;
- Отримання елементів за допомогою DOM-методів;

Spread operator

Spread syntax дозволяє розширити доступні для ітерації елементи (наприклад, масиви або рядки) у місцях:

- для функцій: де очікувана кількість аргументів для викликів функцій дорівнює нулю або більше нуля;
- для елементів (літералів масиву);
- для виразів об'єктів: у місцях, де кількість пар "ключ-значення" має дорівнювати нулю або більше (для об'єктних літералів);

Синтаксис:

- Для фунцій => myFunction(...iterableObj);
- Для литералов массива или строк => [...iterableObj, '4', 'five', 6];
- Для литералов объекта => { ...obj };

Spread operator arr

hillel IT school

const arr = [1, 2, 3];

const copyArr = [...arr];

Spread syntax насправді переходить лише на один рівень глибше при копіюванні масиву. Таким чином, він може не підходити для копіювання багаторозмірних масивів

Rest parameters

Синтаксис залишкових параметрів функції дозволяє представляти безліч аргументів у вигляді масиву. Якщо останній іменований аргумент функції має префікс ..., він автоматично стає масивом з елементами від 0 до theArgs.length-1 у відповідності до актуальної кількості аргументів, переданих у функцію.

Існує три основні відмінності залишкових параметрів від об'єкта arguments:

1. залишкові параметри включають лише ті, яким не задано окреме ім'я, у той час як об'єкт arguments містить усі аргументи, що передаються у функцію;
2. об'єкт arguments не є масивом, в той час як залишкові параметри є екземпляром Array і методи sort, map, forEach або pop можуть безпосередньо використовуватися;
3. об'єкт arguments має додаткову функціональність, специфічну лише йому (наприклад, властивість callee).

Деструктуроване присвоєння

Деструктуроване присвоєння – це спеціальний синтаксис, що дозволяє нам “розпаковувати” масиви чи об’єкти в купу змінних, оскільки іноді це зручніше.

Деструктурування також чудово працює зі складними функціями, які мають багато параметрів, типових значень тощо.

Це називається “деструктуроване присвоєння”, оскільки воно “деструктурує” шляхом копіювання елементів у змінні. Але сам масив не змінюється.

Деструктурування масиву

Небажані елементи масиву можна викинути за допомогою додаткової коми

Деструктурування

Працює з будь-якими типами даних, що перебираються у правій стороні

Залишкові параметри ‘…’

Зазвичай, якщо масив довший від списку зліва, “зайві” елементи опускаються. Якщо ми хочемо також зібрати все наступне – ми можемо додати ще один параметр, який отримує “решту”, використовуючи три крапки "..."

default

Якщо ми хочемо, щоб “типове” значення замінило б відсутнє, ми можемо надати його за допомогою =. Початкові значення можуть бути більш складними виразами або навіть викликами функцій. Вони визначаються, лише якщо значення не надано.

Деструктурування об’єктів

Ми повинні мати існуючий об’єкт праворуч, який ми хочемо розділити на змінні. Ліва частина містить об’єктоподібний “шаблон” для відповідних властивостей. У найпростішому випадку це список імен змінних у {...}. Шаблон з лівого боку може бути більш складним і визначати зіставлення властивостей та змінних.

Залишок об’єкту “...”

Ми можемо використовувати шаблон залишкового оператору, так само, як ми робили з масивами. Він не підтримується деякими старішими браузерами (IE, використовуйте Babel для поліфілу), але працює в сучасних.

```
let name, surname;

{ name, surname } = user // Error
```

JavaScript розглядає {...} в основному потоці коду (а не всередині іншого виразу) як блок коду. Отже, тут JavaScript припускає, що у нас є блок коду, тому і виникає помилка. Натомість ми хочемо деструктурування.

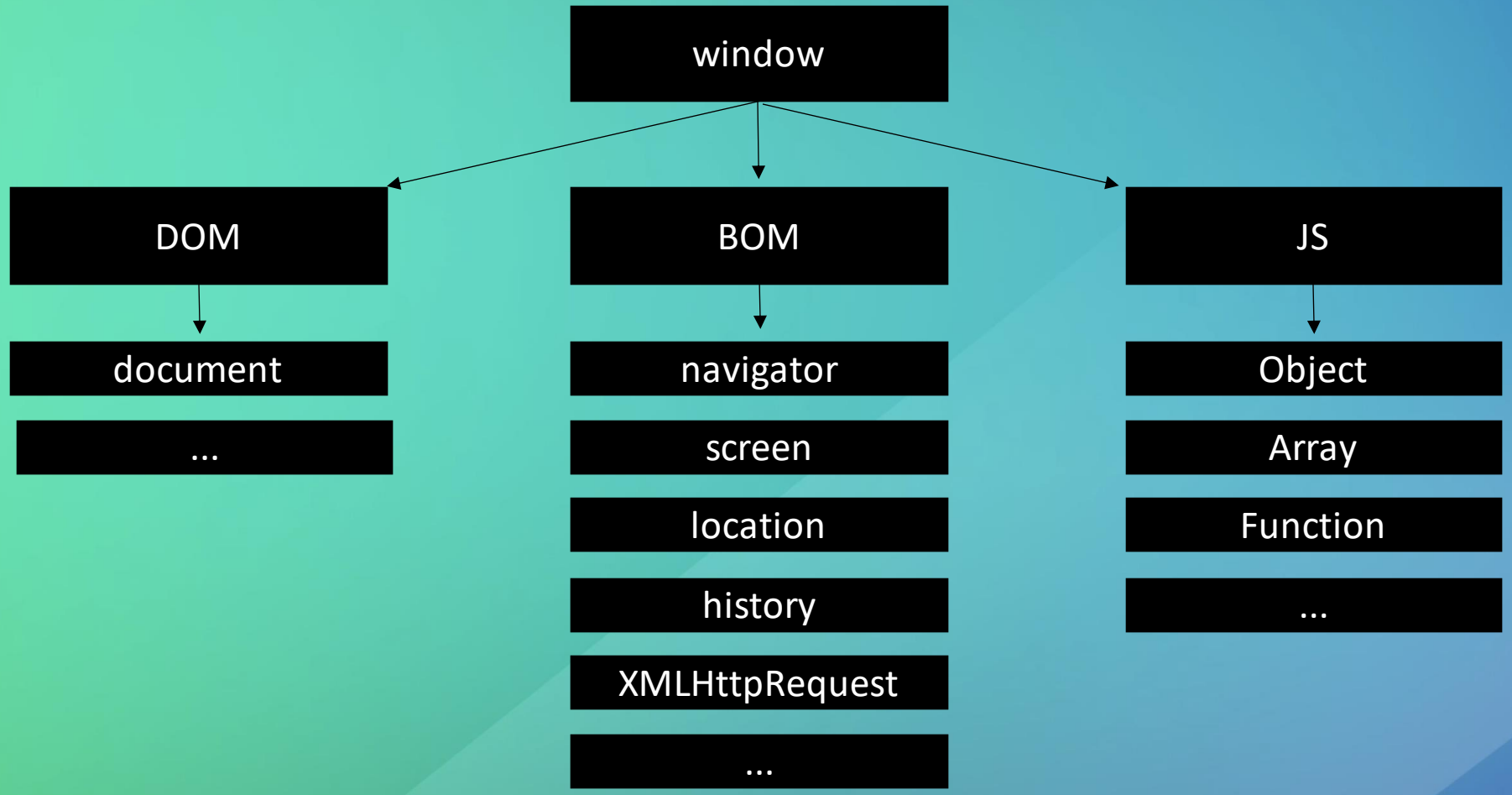
Вкладене деструктурування

Якщо об’єкт або масив містять інші вкладені об’єкти та масиви, ми можемо використовувати більш складні шаблони з лівого боку для вилучення більш глибоких частин.

Параметри функції

Ми можемо передати параметри як об’єкт, і функція негайно деструктурує їх на змінні

Загальна схема брузера



window

Кореневий об’єкт. Він має дві ролі:

- По-перше, це глобальний об’єкт для коду JavaScript, як описано в розділі Глобальний об’єкт.
- По-друге, він являє собою “вікно браузера” та надає способи для керування ним.

Об'єкт window є вікном, що містить DOM документ; властивість document вказує на DOM document (en-US), завантажений у цьому вікні. Вікно поточного документа можна отримати за допомогою властивості document.defaultView.

Об'єкт window реалізує інтерфейс Window, який успадковується від інтерфейсу AbstractView.

Інтерфейс Window успадковує властивості з інтерфейсу EventTarget і реалізує властивості WindowOrWorkerGlobalScope і міксин WindowEventHandlers.

DOM (Document Object Model)

Представляє весь контент сторінки як об’єкти, які можуть бути змінені.

Об’єкт document – це головна “точка входу” до сторінки. Ми можемо змінити або створити що-небудь на сторінці, використовуючи цей об’єкт.

Поидиться на властивості document можна тут

Специфікація DOM описує структуру документа та надає об’єкти, щоб керувати ним. Є також інші інструменти окрім браузерів, які також використовують DOM.

Наприклад, скрипти сервера, які завантажують HTML-сторінки та обробляють їх також можуть використовувати DOM. При цьому вони можуть підтримувати лише частину специфікації.

Також є окрема специфікація, CSS Object Model (CSSOM) для правил CSS та таблиць стилів, що пояснює, як стилі повинні бути представлені у вигляді об’єктів, як читати та писати їх.

BOM (Browser Object Model)

Модель об’єкта браузера (Browser Object Model, BOM) – це додаткові об’єкти, надані браузером (хост-середовищем) для роботи з усім, крім документа.

Наприклад:

- Об’єкт navigator забезпечує інформацію про браузер та операційну систему. Існує багато його властивостей, але два найбільш широко відомих: navigator.userAgent – інформація про поточний браузер, та navigator.platform – про платформу (може допомогти визначити на якій платформі відкрито браузер – Windows/Linux/Mac тощо).
- Об’єкт location дозволяє нам прочитати поточну URL-адресу і може перенаправити веббраузер на нову адресу.

BOM – це частина загальної HTML-специфікації.

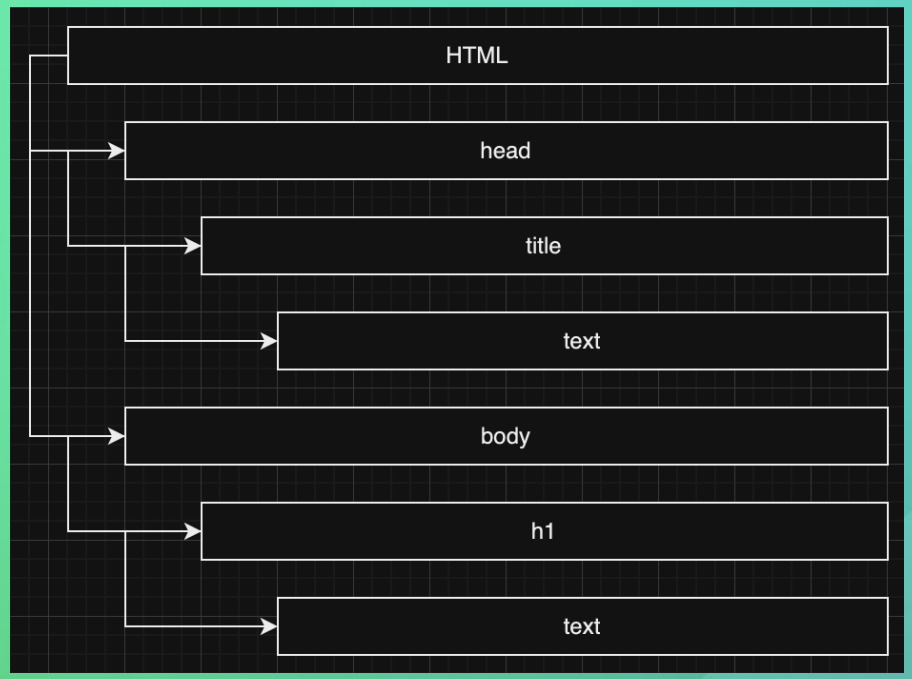
Специфікація HTML – це не тільки про “мову HTML” (теги, атрибути). Вона також охоплює купу об’єктів, методів та специфічних для браузера розширень DOM. Це “HTML в широкому сенсі”. Крім того, деякі частини мають додаткові специфікації, перераховані на https://spec.whatwg.org.

Занурюємось у DOM

Відповідно до об’єктної моделі документа (“Document Object Model”, DOM), кожен HTML-тег є об’єктом. Вкладені теги є “дітьми” атьківського елементу. Текст, який знаходиться всередині тегу, також є об’єктом.

Всі ці об’єкти доступні за допомогою JavaScript, і ми можемо використовувати їх для зміни сторінки.

DOM – це представлення HTML-документа в вигляді дерева тегів:



Кожен вузол дерева є об’єктом.

Теги є вузлами-елементами (або просто елементами), вони утворюють структуру дерева: <html> – кореневий, <head> та <body> – його дочірні вузли тощо.

Текст всередині елементів утворює текстові вузли, позначені як text. Текстовий вузол містить лише рядок. У нього не може буди нащадків, тобто він завжди знаходиться на найнижчому рівні.

Занурюємось у DOM

Якщо браузер зтикається з невалідним HTML-кодом, він автоматично виправляє його при створенні DOM.

Наприклад, напочатку документу завжди повинен бути тег <html>. Навіть якщо він не існує в документі, він буде існувати в дереві DOM, тому що браузер створить його. Те ж саме стосується <body>.

Об’єкт document, який представляє весь документ, формально також є вузлом DOM.

Існує 12 типів вузлів. На практиці ми зазвичай працюємо з 4-ма з них:

- document – “пункт входу” в DOM.
- вузли-елементи – HTML-теги, будівельні блоки дерев.
- текстові вузли – містять текст.
- коментарі – іноді ми можемо записати туди інформацію, вона не буде показана, але JS може читати її з DOM.

console

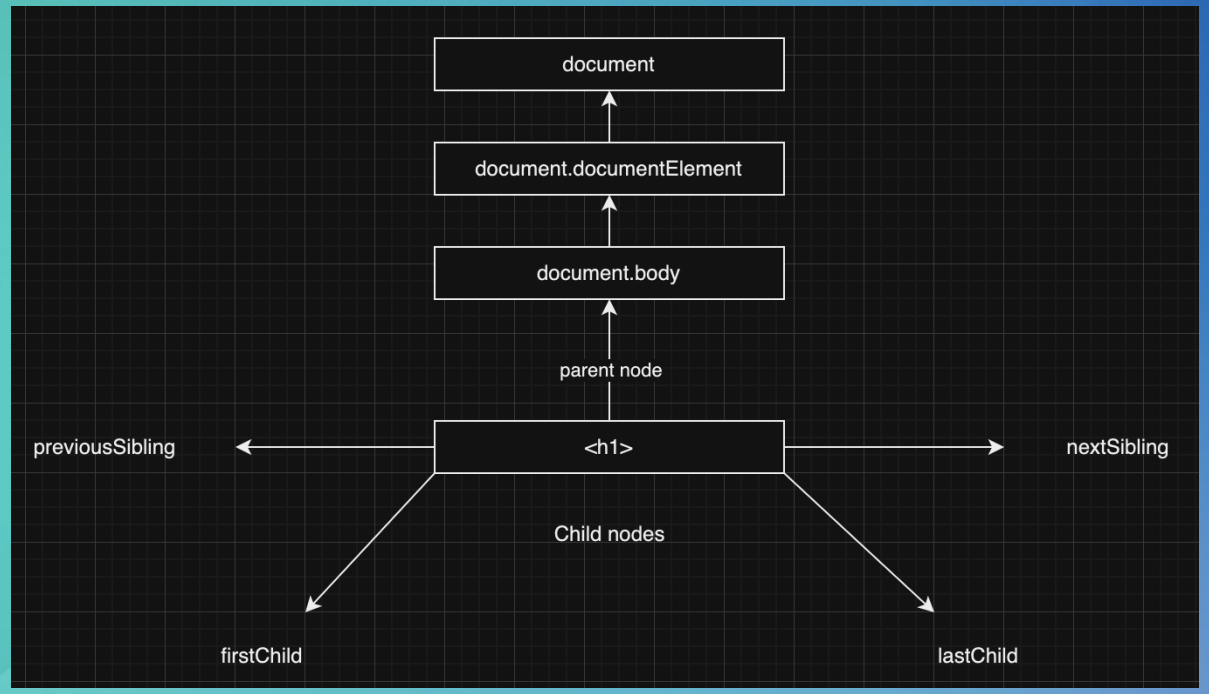
- останній вибраний елемент доступний як $0
- раніше вибраний – $1, тощо.

Навігація по DOM

<html> = document.documentElement
<body> = document.body
<head> = documennt.head

- Дочірні вузли (або діти) – елементи, які є безпосередніми дітьми. Іншими словами, вони вкладені саме в цей вузол. Наприклад, <h1> є дочірніми елементами <body>.
- Нащадки – всі елементи, які вкладені в даний, включаючи дітей, їхніх дітей тощо.

Колекція childNodes містить список усіх дочірніх вузлів, включаючи текстові вузли.
Властивості firstChild і lastChild надають швидкий доступ до першого та останнього дочірнього вузла.



DOM

Пробіли та переходи на нові рядки є абсолютно діючими символами, як букви та цифри. Вони утворюють текстові вузли і стають частиною DOM.

Є лише два винятки з цього правила:

1. Пробіли та переходи на нові рядки до <head> ігноруються з історичних причин.
2. Якщо ми запишемо щось після закриваючого тегу </body>, браузер автоматично перемістить цей запис в кінець body, оскільки специфікація HTML вимагає, щоб весь вміст був всередині <body>. Отже, після <body> не може бути ніяких пробілів.

childNodes

Це колекція – спеціальний ітеративний об’єкт-псевдомасив.

Є два важливих наслідки з цього:

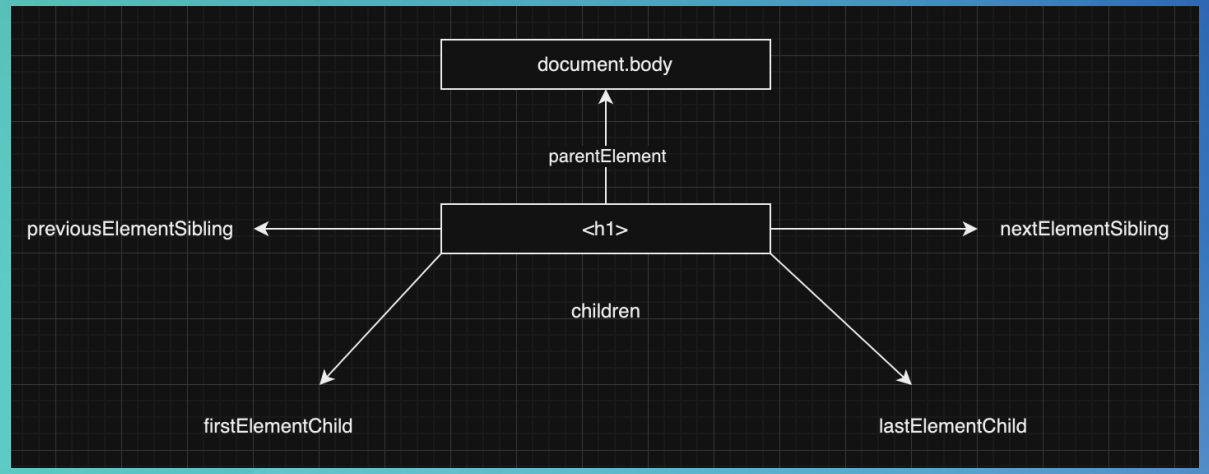
1. Ми можемо використовувати for..of, щоб перебирати його;
2. Методи масиву не працюватимуть, бо колекція це не масив.

Запам’ятайте!

- Колекції DOM доступні лише для зчитування;
- Майже всі колекції DOM, за незначними винятками, завжди відображають поточний стан DOM;
- Не використовуйте for..in для перебору колекцій.

Навігація по DOM2

- children – колекція дітей, які є елементами.
- firstElementChild, lastElementChild – перший і останній дочірні елементи.
- previousElementSibling, nextElementSibling – сусідні елементи.
- parentElement – батьківський елемент.



Властивість parentElement повертає батьківський елемент “element”, тоді як parentNode повертає батьківський “будь-який вузол”. Ці властивості зазвичай однакові: обидві вони отримують батьківський елемент.

За винятком document.documentElement

Пошук елементів

- document.getElementById або просто id - шукає елемент за його ідентифікатором.

Будь ласка, не використовуйте id-іменовані глобальні змінні для доступу до елементів!

- element.querySelectorAll(css) - повертає всі елементи всередині elem, що відповідають заданому CSS-селектору. Псевдокласи в CSS-селекторі, такі як :hover і :active, також підтримуються.
- element.querySelector(css) повертає перший елемент, що відповідає даному CSS-селектору.
- Метод elem.matches(css) нічого не шукає, він просто перевіряє, чи відповідає elem заданому CSS-селектору. Він повертає true або false.
- elem.closest(css) - шукає найближчого предка, який відповідає CSS-селектору. Сам elem також включається в пошук.

Легасі:

- elem.getElementsByTagName(tag) шукає елементи з заданим тегом і повертає колекцію цих елементів. Параметр tag також може бути зірочкою "*" для “будь-яких тегів”.
- elem.getElementsByClassName(className) повертає елементи, які мають вказаний CSS-клас.
- document.getElementsByName(name) повертає елементи з заданим атрибутом name для всього документа. Використовується дуже рідко.

Пошук елементів

Метод	Шукає, використовуючи...	Чи може викликатися на елементі?	Чи повертає живу колекцію?
querySelector	CSS-селектор	так	ні
querySelectorAll	CSS-селектор	так	ні
getElementById	id	ні	ні
getElementsByName	ім’я	ні	так
getElementsByTagName	тег або '*'	так	так
getElementsByClassName	клас	так	так

Дякую за увагу