hillel IT school

Lesson 16

План заняття

- Map;
- Set;
- WeakMap;
- WeakSet;

Map

Map – це колекція ключ/значення, як і Object. Але основна відмінність полягає в тому, що Map дозволяє мати ключі будь-якого типу.

Методи та властивості:

- new Map() – створює колекцію.
- map.set(key, value) – зберігає значення value за ключем key.
- map.get(key) – повертає значення за ключем; повертає undefined якщо key немає в колекції.
- map.has(key) – повертає true якщо key існує, інакше false.
- map.delete(key) – видаляє елемент (пару ключ/значення) за ключем.
- map.clear() – видаляє всі елементи колекції.
- map.size – повертає поточну кількість елементів.

Map

Порівнюючи ключі, об’єкт Map використовує алгоритм SameValueZero. Це майже таке ж порівняння, що і ===, з тією лише різницею, що NaN вважається рівним NaN. Таким чином NaN може також бути використаний як ключ.

Цей алгоритм не може бути замінений або модифікований.

Перебір Map

Для перебору колекції Map є 3 метода:

1. map.keys() – повертає об’єкт-ітератор для ключів,
2. map.values() – повертає об’єкт-ітератор для значень,
3. map.entries() – повертає об’єкт-ітератор зі значеннями виду [ключ, значення], цей варіант типово використовується з for..of.

На відміну від звичайних об’єктів Object, в Map перебір відбувається в тому ж порядку, в якому відбувалося додавання елементів.

Крім цього, Map має вбудований метод forEach()

Map з Object

При створенні Map ми можемо вказати масив (або інший об’єкт-ітератор) з парами ключ-значення для ініціалізації

Object з Map

Ми можемо використати Object.fromEntries метод, який виконає зворотну дію: трансформує отриманий масив пар [ключ, значення] в об’єкт.

Set

Об’єкт Set – це особливий тип колекції: “множина” значень (без ключів), де кожне значення може з’являтися тільки раз.

Основні методи:

- new Set([iterable]) – створює Set, якщо аргументом виступає об’єкт-ітератор, тоді значення копіюються в Set.
- set.add(value) – додає нове значення до Set, повертає Set.
- set.delete(value) – видаляє значення з Set, повертає true, якщо value наявне в множині значень на момент виклику методу, інакше false.
- set.has(value) – повертає true, якщо value присутнє в множині Set, інакше false.
- set.clear() – видаляє всі значення множини Set.
- set.size – повертає кількість елементів в множині.

Set

Set має ті ж вбудовані методи, що і Map:

- set.keys() – повертає об’єкт-ітератор для значень,
- set.values() – те ж саме, що set.keys(), для сумісності з Map,
- set.entries() – повертає об’єкт-ітератор для пар виду [значення, значення], присутній для сумісності з Map.

WeakMap та WeakSet

Рушій JavaScript зберігає значення в пам’яті, поки воно є “доступним” і потенційно може бути використаним.

```
let obj = { key: "value" };
// Об’єкт можна отримати, obj -- це посилання на нього
// перезапишемо посилання
obj = null;
// об’єкт буде видалено з пам’яті
```

Зазвичай властивості об’єкта або елементів масиву або іншої структури даних вважаються доступними та зберігаються в пам’яті, поки та структура даних є в пам’яті.

WeakMap та WeakSet

Якщо ми покладемо об’єкт в масив, то, поки масив живий, об’єкт буде живим, навіть якщо немає інших посилань на цей об’єкт.

```
let obj = { key: "value" };

let array = [ obj ];

obj = null; // перезапишемо посилання

// об’єкт, на який раніше посилалася змінна obj, зберігається всередині масиву

// тому він не буде видалений збирачем сміття

// ми можемо отримати його як array[0]
```

Подібно до цього, якщо ми використовуємо об’єкт як ключ у звичайному Map, то в той час, коли Map існує, цей об’єкт також існує. Він займає пам’ять і не може бути видалений збирачем сміття.

WeakMap

WeakMap не перешкоджає збиранню сміття серед об’єктів, що є ключами. Відмінність між Map та WeakMap – це те, що ключі повинні бути об’єктами, а не примітивними значеннями. Якщо ми використовуємо об’єкт як ключ, і немає інших посилань на цей об’єкт – його буде видалено з пам’яті (і з мапи) автоматично.

WeakMap має лише такі методи:

- weakMap.set(key, value)
- weakMap.get(key)
- weakMap.delete(key)
- weakMap.has(key)

WeakMap

Якщо об’єкт втратив всі інші посилання (наприклад, obj), то він буде автоматично видалений збирачем сміття. Але технічно немає точних вказівок коли відбувається видалення.

Рушій JavaScript вирішує це. Він може вибрати очищення пам’яті негайно або почекати, і зробити очищення пізніше, коли трапиться більше видалень. Отже, технічно, поточна кількість елементів WeakMap невідома. Рушій, можливо, очистив його чи ні, або зробив це частково. З цієї причини методи, які дають доступ до всіх ключів/значень не підтримуються.

Приклади

Основна область застосування для WeakMap – це зберігання додаткових даних.

Якщо ми працюємо з об’єктом, що “належить” до іншого коду, можливо навіть сторонньої бібліотеки, і хотіли б зберегти деякі дані, пов’язані з ним, що повинні існувати лише поки об’єкт живий – тоді WeakMap це саме те, що потрібно.

Ми покладемо дані в WeakMap, використовуючи об’єкт як ключ, і коли об’єкт буде видалено збирачем сміття, то ці дані також автоматично зникнуть.

WeakSet

WeakSet поводитися аналогічно:

- Це аналог Set, але ми можемо додати лише об’єкти до WeakSet (не примітиви).
- Об’єкт існує в наборі, коли він доступний з де-небудь ще.
- Так само як Set, він підтримує add, has і delete, але не підтримує size, keys() та ітерацію.
- Будучи “слабким”, він також служить для зберігання додаткових даних. Але не для довільних даних, а для фактів “так/ні”.

Найбільш помітним обмеженням WeakMap та WeakSet є відсутність ітерацій та нездатність отримати весь поточний вміст. Це може виявитися незручним, але не перешкоджає WeakMap/WeakSet виконувати свою основну роботу – бути «додатковим» сховищем даних для об’єктів, які зберігаються/управляються в іншому місці.

Дякую за увагу