hillel IT school

Lesson 11

План заняття

- Створення об'єктів;
- Object.freeze, Object.seal;
- Копіювання об'єктів та масивів;
- Додавання та видалення властивостей/методів в об'єкті;
- Object.defineProperty;
- Геттери та Сеттери в об'єктах;
- Object.hasOwnProperty;

Об’єкти

Об’єкт можна створити за допомогою фігурних дужок {…} з необов’язковим списком властивостей. Властивість – це пара “ключ: значення”, де ключ – це рядок або символ (також називається “ім’я властивості”), а значення може бути будь-яким.

Ми можемо уявити собі об’єкт як шафу з підписаними файлами. Кожен фрагмент даних зберігається у своєму файлі за допомогою ключа. Легко знайти файл за назвою або додати/видалити файл.

```
const user = {
  name: 'Sergey',
  age: 32,
  getBirthYear() {
    return new Date().getFullYear() - 32
  }
}
```

Об’єкти

Ключ може бути лише строкою!

Значення може бути будь-якого типу.

{ "key": any }

Щоб видалити властивість ми можемо використати оператор delete

delete user.name

Остання властивість у списку може закінчуватися комою.

Крапка вимагає, щоб ключ був правильним ідентифікатором змінної. Це означає: не містить пробілів, не починається з цифри та не містить спеціальних символів ($ та _ дозволені). Для усіх інших випадків існує альтернативний спосіб доступу до властивостей через квадратні дужки. Такий спосіб спрацює з будь-яким ім’ям властивості.

Об’єкти

Ми можемо використовувати квадратні дужки в літеральній нотації для створення обчислюваної властивості.

{ [variable + 'SomeKey']: value }

Немає ніяких обмежень щодо назв властивостей. Це можуть бути будь-які рядки або символи (спеціальний тип для ідентифікаторів, про які буде сказано пізніше).

Інші типи автоматично перетворюються на рядки

Оператор in

Помітною особливістю об’єктів у JavaScript, у порівнянні з багатьма іншими мовами, є можливість доступу до будь-якої властивості. Помилки не буде, якщо властивості не існує!

Читання відсутньої властивості просто повертає undefined. Тому ми можемо легко перевірити, чи існує властивість

object.key === undefined => перевірить чи є key у об’єкту. Поверне true якщо ключа не існує або він явно undefined або false якщо він існує;

"key" in object => перевірить чи є властивість у об’єкту. Поверне true якщо ключ є у обьекту або він має значення undefined або false якщо ключа нема;

For … in

Для перебору всіх властивостей об’єкта використовується цикл for..in

У циклі властивості з цілочисельними ключами сортуються за зростанням, інші розташовуються в порядку створення. Термін “цілочисельна властивість” означає рядок, який може бути перетворений в ціле число і назад без змін.

Копіювання об’єктів та посилання

Однією з принципових відмінностей об’єктів від примітивів є те, що об’єкти зберігаються та копіюються “за посиланням”, тоді як примітивні значення: рядки, числа, логічні значення тощо – завжди копіюються “за значенням”.

Пам’ятайте:

- Змінна зберігає не сам об’єкт, а його “адресу в пам’яті” – іншими словами “посилання” на нього.
- Коли копіюється змінна об’єкта, копіюється посилання, але сам об’єкт не дублюється.
- Два об’єкти рівні, лише якщо це той самий об’єкт.

Object.assign

Використовується для копіювання значень всіх власних властивостей з одного або більше вихідних об'єктів в цільовий об'єкт. Після копіювання він повертає цільовий об'єкт.

Синтксис:

Object.assign(target, ...sources)

Повертається цільовий об'єкт, що вийшов.

structuredClone()

Глобальний метод structuredClone() створює глибокий клон заданого значення за допомогою алгоритму структурованого клонування.

Цей метод також дозволяє переносити об’єкти, що підлягають передачі, у вихідному значенні, а не клонувати їх у новий об’єкт. Перенесені об’єкти від’єднуються від вихідного об’єкта та приєднуються до нового об’єкта; вони більше не доступні у вихідному об’єкті.

Синтаксис:

structuredClone(value)

Конструктор, оператор "new"

Звичайний синтаксис {...} дозволяє створити тільки один об’єкт. Проте часто нам потрібно створити багато однотипних об’єктів, таких як, наприклад, користувачі чи елементи меню тощо.

Це можна зробити за допомогою функції-конструктора та оператора "new".

Функція-конструктор

Технічно, функції-конструктори – це звичайні функції. Однак є дві загальні домовленості:

- Ім’я функції-конструктора повинно починатися з великої літери.
- Функції-конструктори повинні виконуватися лише з оператором "new".

Коли функція виконується з new, відбуваються наступні кроки:

- Створюється новий порожній об’єкт, якому присвоюється this.
- Виконується тіло функції. Зазвичай воно модифікує this, додає до нього нові властивості.
- Повертається значення this.

Функція-конструктор

Технічно, функції-конструктори – це звичайні функції. Однак є дві загальні домовленості:

- Ім’я функції-конструктора повинно починатися з великої літери.
- Функції-конструктори повинні виконуватися лише з оператором "new".

Коли функція виконується з new, відбуваються наступні кроки:

- Створюється новий порожній об’єкт, якому присвоюється this.
- Виконується тіло функції. Зазвичай воно модифікує this, додає до нього нові властивості.
- Повертається значення this.

Технічно будь-яка функція може бути використана як конструктор. Вона може бути запущена через new, і буде виконано наведений вище алгоритм. “Ім’я з великої літери” це загальна домовленість, яка допомагає чітко зрозуміти, що функцію слід запускати з new.

Перевірка виклику

Використовуючи спеціальну властивість new.target всередині функції, ми можемо перевірити чи була ця функція викликана за допомогою оператора new чи без нього.

Якщо функція була викликана за допомогою new, то в new.target буде сама функція, в іншому разі отримаємо undefined

Така можливість може бути використана всередині функції для того, щоб дізнатися чи функція була викликана за допомогою оператора new, “у режимі конструктора”, чи без нього, “у звичайному режимі”.

constructor return

Зазвичай конструктори не мають інструкції return. Їх завдання – записати усе необхідне у this, яке автоматично стане результатом.

Але якщо є інструкція return, то застосовується просте правило:

- Якщо return викликається з об’єктом, тоді замість this буде повернено цей об’єкт.
- Якщо return викликається з примітивом, примітив ігнорується.
- Інакше кажучи, return з об’єктом повертає цей об’єкт, у всіх інших випадках повертається this

Створення методів у конструкторі

Звичайно, ми можемо додати до this не лише властивості, але й методи.

```
function User(name) {
  this.name = name;
  this.showName = () => console.log(this.name);
}
```

Опціональний ланцюжок '?.'

Опціональний ланцюжок ?. – це безпечний спосіб доступу до вкладених властивостей об’єктів, навіть якщо проміжних властивостей не існує.

Опціональний ланцюжок ?. припиняє обчислення, якщо значення перед ?. є undefined або null, і повертає undefined.

Іншими словами, value?.prop:

- працює як value.prop, якщо value існує,
- інакше (коли value є undefined/null) воно повертає undefined.

Не зловживайте опціональним ланцюжком. Нам слід використовувати ?. тільки в тих ситуаціях коли ми припускаємо що значення може не існувати.

Опціональний ланцюжок '?.'

Ми можемо використовувати ?. з ?.[] та з ?.(), наприклад:

- obj?.[key]
- obj.showName?.()

Символ

“Символ” являє собою унікальний ідентифікатор.

- Створити символ можна за допомогою Symbol()
- Символи не перетворюються автоматично в рядок
- Символи дозволяють нам створювати “приховані” властивості об’єкта, до яких жодна інша частина програми не може випадково отримати доступ або перезаписати їх.
- Символи ігноруються циклом for…in. Object.keys(user) також ігнорує їх. Це частина загального принципу “приховування символьних властивостей”.
- Object.assign копіює властивості рядка та символу

Глобальні символи

Зазвичай усі символи унікальні, навіть якщо вони мають однакову назву. Але іноді ми хочемо, щоб однойменні символи були однаковими сутностями. Наприклад, різні частини нашого додатка хочуть отримати доступ до символу "id", що означає абсолютно однакову властивість.

Для цього існує глобальний реєстр символів. Ми можемо створити в ньому символи та отримати до них доступ пізніше, і це гарантує, що повторні звернення з тим самим іменем нам повернуть абсолютно однаковий символ.

Для того, щоб знайти (створити, якщо його немає) символ у реєстрі, використовуйте Symbol.for(key).

Цей виклик перевіряє глобальний реєстр, і якщо є символ з іменем key, тоді повертає його, інакше створює новий символ Symbol(key) і зберігає його в реєстрі за вказаним key.

Symbol.keyFor(sym) працює навпаки: приймає глобальний символ і повертає його ім’я.

Системні символи

Існує багато “системних” символів, які JavaScript використовує внутрішньо, і ми можемо використовувати їх для налаштування різних аспектів наших об’єктів.

Вони вказані в таблиці специфікації Well-known symbols:

- Symbol.hasInstance
- Symbol.isConcatSpreadable
- Symbol.iterator
- Symbol.toPrimitive
- …та інші.

Object.defineProperty

Основний метод управління властивостями – Object.defineProperty

Він дозволяє оголосити властивість об'єкта і, що найголовніше, тонко налаштувати його особливі аспекти, які ніяк не змінити інакше.

Синтаксис:

Object.defineProperty(obj, prop, descriptor)

1. Obj - Об'єкт, у якому визначається властивість.
2. Prop - Ім'я властивості, що визначається або змінюється.
3. Descriptor - Дескриптор визначуваної або змінної властивості.

Object.defineProperty

Дескриптор – об'єкт, який визначає поведінка якості.

У ньому можуть бути такі поля:

- value – значення властивості, за умовчанням undefined
- writable – значення якості можна змінювати, якщо true. За промовчанням false.
- configurable – якщо true, то властивість можна видаляти, і навіть змінювати його надалі з допомогою нових викликів defineProperty. За промовчанням false.
- enumerable – якщо true, то властивість проглядається у циклі for..in та методі Object.keys(). За промовчанням false.
- get – функція, яка повертає значення якості. За замовчуванням undefined.
- set – функція, яка записує значення якості. За замовчуванням undefined.

Object.defineProperty

1. Для того, щоб зробити властивість незмінною, змінимо її прапори writable та configurable
2. Для того щоб виключити метод або властивість зі списку ітерації, поставивши йому прапор enumerable: false

Властивість як функція

Дескриптор дозволяє встановити властивість, яка насправді працює як функція. Для цього в ньому потрібно вказати цю функцію get.

Також можна вказати функцію, яка використовується для запису значення за допомогою дескриптора set.

Щоб уникнути конфлікту, заборонено одночасно вказувати значення значення та функції get/set. Або значення, чи функції щодо його читання-запису, одне з двох. Також заборонено і немає сенсу вказувати writable за наявності get/set-функцій.

Object.defineProperties

Дозволяє оголосити кілька властивостей відразу.

Синтаксис:

Object.defineProperties(obj, descriptors)

Object.keys(obj), Object.getOwnPropertyNames(obj)

Повертають масив – перелік властивостей об'єкта.

- Object.keys повертає тільки enumerable-властивості.
- Object.getOwnPropertyNames – повертає все

Object.getOwnPropertyDescriptor(obj, prop)

Повертає дескриптор властивості obj[prop].

Отриманий дескриптор можна змінити та використовувати defineProperty для збереження змін

Object.hasOwnProperty()

Цей метод може використовуватися визначення того, чи містить об'єкт зазначене властивість як власної властивості об'єкта; на відміну від оператора in цей метод не перевіряє існування властивостей в ланцюжку прототипів об'єкта.

Дякую за увагу