hillel IT school

Lesson 10

План заняття

- Замикання;
- Лексична область видимості;
- Глобальний об'єкт window (глобальний контекст);
- Функціональний контекст;
- Функції в об'єктах (методи);
- Методи підміни контексту;

Блоки коду

Якщо змінна оголошена всередині блоку коду {...}, вона видно лише всередині цього блоку. За допомогою блоків {...} ми можемо ізолювати частину коду, що виконує своє власне завдання, зі змінними, що належать тільки їй.

Для if, for, while і т.д. змінні, оголошені в блоці коду {...}, також видно лише всередині.

```
{
  let a = 'example' // can use inside this block
}

for (let i = 0; i < 3; i++) {
  // змінна i видно тільки всередині for
  alert(i); // 0, потім 1, потім 2
}
```

Вложені функції

Функція називається вкладеною, коли вона створюється всередині іншої функції.

```
let x = 10
function a() {
    let y = 5
    function b() {
        let z = 1
        console.log(x + y + z)
    }
    b()
}
a() // 16
```

Лексична область

У JavaScript у кожної виконуваної функції, блоку коду {...} і скрипта є пов'язаний із ними внутрішній (прихований) об'єкт, званий лексичним оточенням LexicalEnvironment.

Об'єкт лексичного оточення складається із двох частин:

1. Environment Record (сховище змінних) – об'єкт, у якому як властивості зберігаються всі локальні змінні (і навіть інша інформація, така як значення this).
2. Посилання на зовнішнє лексичне оточення – тобто те, що відповідає коду зовні (зовні поточних фігурних дужок).

"Змінна" - це просто властивість спеціального внутрішнього об'єкта: Environment Record. "Отримати або змінити змінну" означає "отримати або змінити властивість цього об'єкта".

У глобального лексичного оточення немає зовнішнього оточення, тож вона вказує на null.

Лексична область

Процес зміни лексичного оточення:

1. При запуску скрипта лексичне оточення попередньо заповнюється всіма оголошеними змінними.
2. Спочатку вони перебувають у стані «Uninitialized». Це особливий внутрішній стан, яке означає, що двигун знає про змінну, але на неї не можна посилатися, доки вона не буде оголошена за допомогою let. Це майже те саме, якби змінна не існувала.
3. З'являється визначення змінної let x. У неї ще немає присвоєного значення, тому надається undefined. З цього моменту ми можемо використати змінну.
4. Змінної x надається значення.
5. Змінна x змінює значення.

Лексична область

"Лексичне оточення" - це об'єкт специфікації: він існує тільки "теоретично" у специфікації мови для опису того, як усе працює. Ми не можемо отримати цей об'єкт у нашому коді та маніпулювати їм безпосередньо.

JavaScript-двигуни також можуть оптимізувати його, відкидати змінні для економії пам'яті, що не використовуються, і виконувати інші внутрішні дії, але при цьому видима поведінка залишається такою, як описано.

Лексична область

Функція – це значення, як і змінна.

Різниця полягає в тому, що Function Declaration миттєво повністю ініціалізується.

Коли створюється лексичне оточення, Function Declaration відразу стає функцією, готової до використання (на відміну let, який досі оголошення може бути використаний).

Саме тому ми можемо викликати функцію, оголошену як Function Declaration, до її оголошення.

Коли функція запускається, на початку її виклику автоматично створюється нове лексичне оточення для зберігання локальних змінних та параметрів виклику.

Коли код хоче отримати доступ до змінної – спочатку відбувається пошук у внутрішньому лексичному оточенні, потім у зовнішньому, потім у наступному тощо, до глобального.

Заминання

Замикання – це функція, яка запам'ятовує свої зовнішні змінні та може отримати доступ до них. У деяких мовах це неможливо, або функція має бути написана спеціальним чином, щоб вийшло замикання. Але, у JavaScript, всі функції спочатку є замикання.

Тобто вони автоматично запам'ятовують, де були створені, за допомогою прихованої властивості Environment, і всі вони можуть отримати доступ до зовнішніх змінних.

Усі функції пам'ятають лексичне оточення, де вони були створені. Технічно всі функції мають приховану властивість [Environment], яка зберігає посилання на лексичне оточення, в якому була створена функція. Посилання на [[Environment]] встановлюється один раз і назавжди під час створення функції

Сборка сміття

Зазвичай лексичне оточення видаляється з пам'яті разом із усіма змінними після завершення виклику функції. Це зв’язано з тим, що у нього немає посилань. Як і будь-який об'єкт JavaScript, воно зберігається в пам'яті лише доти, доки до нього можна звернутися.

Однак якщо існує вкладена функція, яка все ще доступна після завершення функції, вона має властивість [[Environment]], що посилається на лексичне оточення.

Лексичне оточення залишається доступним навіть після завершення роботи функції.

Об'єкт лексичного оточення зникає, коли стає недоступним (як будь-який інший об'єкт). Іншими словами, він існує тільки доти, доки на нього посилається хоча б одна вкладена функція.

Сборка сміття

Двигуни JavaScript намагаються оптимізувати доступність. Вони аналізують використання змінних та, якщо легко за кодом зрозуміти, що зовнішня змінна не використовується – вона видаляється.

Одним із важливих побічних ефектів у V8 (Chrome, Edge, Opera) є те, що така змінна стає недоступною при налагодженні.

Сборка сміття

Основною концепцією управління пам'яттю JavaScript є принцип досяжності.

Якщо спростити, то «досяжні» значення – це доступні або використовуються. Вони гарантовано перебувають у пам'яті. Існує базова множина досяжних значень, які не можуть бути видалені.

Наприклад:

- Функція, що виконується в даний момент, її локальні змінні і параметри.
- Інші функції в поточному ланцюжку вкладених дзвінків, їх локальні змінні та параметри.
- Глобальні змінні.
- (деякі інші внутрішні значення)

Ці значення ми називатимемо корінням.

Будь-яке інше значення вважається досяжним, якщо воно доступне з кореня за посиланням або ланцюжком посилань.

Сборка сміття

Наприклад, якщо глобальної змінної є об'єкт, і він має властивість, у якому зберігається посилання інший об'єкт, цей об'єкт вважається досяжним. І ті, на які він посилається, також досяжні. Далі ви познайомитеся із докладними прикладами на цю тему.

```
let user = {
  name: "John"
};
user = null
```

Сборка сміття

Наприклад, якщо глобальної змінної є об'єкт, і він має властивість, у якому зберігається посилання інший об'єкт, цей об'єкт вважається досяжним. І ті, на які він посилається, також досяжні. Далі ви познайомитеся із докладними прикладами на цю тему.

```
let user = {
  name: "John"
};
user = null
```

Прив’язка контексту

При передачі методів об'єкта як колбеки, наприклад для setTimeout або forEach, виникає відома проблема - втрата цього.

Найпростіший варіант рішення – це обернути виклик в анонімну функцію, створивши замикання. Але це може призвести до багів.

Рішення:

- .bind()
- .call()
- .apply()

Bind

.bind() створює нову функцію, яка при викликі встановлює як контекст виконання цього наданого значення. До методу також передається набір аргументів, які будуть встановлені перед переданими у прив'язану функцію аргументами під час її виклику.

Синтаксис:

fun.bind(thisArg[, arg1[, arg2[, ...]]])

thisArg - значення, що передається як це в цільову функцію при виклику прив'язаної функції. Значення ігнорується, якщо функція конструюється за допомогою оператора new.

arg - аргументи цільової функції передаються перед аргументами прив'язаної функції при виклику цільової функції.

Bind - часткове застосування

Часткове застосування - можливість у ряді мов програмування зафіксувати частину аргументів багатомісної функції та створити іншу функцію, меншу арність.

Користь від цього полягає в тому, що можна створити незалежну функцію зі зрозумілою назвою (double, triple). Ми можемо використовувати її і не передавати щоразу перший аргумент, т.к. він зафіксований за допомогою bind.

В інших випадках часткове застосування корисне, коли ми маємо дуже спільну функцію і для зручності ми хочемо створити її більш спеціалізований варіант.

Call & Apply

Метод call() викликає функцію із зазначеним значенням цього та індивідуально наданими аргументами.

Синттаксис:

fun.call(thisArg[, arg1[, arg2[, ...]]])

Метод apply() викликає функцію із зазначеним значенням цього і аргументами, наданими у вигляді масиву (або масивоподібного об'єкта.

fun.apply(thisArg, [argsArray])

Window

Об'єкт window є вікном, що містить DOM документ; властивість document вказує на DOM document, завантажений у цьому вікні.

Цей розділ містить опис усіх методів, властивостей та подій, доступних через об'єкт wondowDOM. Об'єкт window реалізує інтерфейс Window, який успадковується від інтерфейсу AbstractView. Деякі додаткові глобальні функції, простори імен об'єктів, інтерфейси та конструктори, як правило, не пов'язані з вікном, але доступні в ньому, перераховані в JavaScript посилання та посилання.

Має купу властивостей та методів:

- Window.console
- Wondow.closed
- Window.document
- Window.hisory
- Window.innerHeight.....

Дякую за увагу