hillel IT school

Lesson 23

План заняття

- Cookies;
- LocalStorage;
- SessionStorage;
- IndexedDB;

Cookies

Файли cookies – це невеликі рядки з даними, які зберігаються безпосередньо в браузері. Вони є частиною HTTP-протоколу, визначеного специфікацією RFC 6265.

Файли cookies зазвичай встановлюються вебсервером за допомогою HTTP-заголовка Set-Cookie. Потім браузер автоматично додаватиме їх при (майже) кожному запиті до відповідного домену використовуючи HTTP-заголовок Cookie.

Файли cookie HTTP в основному використовуються для керування сеансами користувачів, збереження налаштувань персоналізації користувачів та відстеження поведінки користувачів. Вони також є причиною всіх цих дратівливих форм згоди "на цій сторінці використовуються файли cookie", які ви бачите в Інтернеті.

Одним з найбільш поширених випадків використання є аутентифікація:

- При вході в систему, сервер використовує відповідь отриману з HTTP-заголовка Set-Cookie, щоб додати в файл cookie унікальний “ідентифікатор сесії” .
- Наступного разу, коли на той самий домен буде відправлено запит, браузер надішле файл cookie використовуючи HTTP-заголовок Cookie.
- Таким чином, сервер знає, хто зробив запит.

Документація

document.cookie

Значення document.cookie складається з пар name=value розділених ; . Кожна пара – це окремий файл cookie.

Запис в document.cookie оновлює лише вказані файли cookies, та не чіпатиме решту.

Технічно, ім’я та значення можуть містити будь-які символи. Щоб зберегти правильне форматування, вони повинні бути кодовані за допомогою вбудованої функції encodeURIComponent.

Є декілька обмежень:

- Пара name=value, після кодування encodeURIComponent, не повинна перевищувати 4KB. Тому ми не можемо зберігати великий об’єм даних у файлах cookie.
- Дозволена сумарна кількість файлів cookie на один домен приблизно 20+, точний ліміт залежить від браузера.
- НЕМАЄ ГАРНОГО ШЛЯХУ ДЛЯ ПАРСУ COOKIES!

У жодному разі не передавайте чутливі дані з куками.

Параметри cookie

Файли cookies мають ряд параметрів, деякі з них важливі і повинні бути задані.

Параметри перераховуються після пари key=value та розділяються ;

path

URL-префікс адреси повинен бути абсолютним. Для того щоб файли cookie були доступними зі сторінок за цією адресою. За замовчуванням, це поточна сторінка.

Якщо в файлі cookie задано path=/user, то він видимий на сторінках /user та /user/1, але не на /home.

Зазвичай, в якості path вказують кореневу сторінку: path=/ для того щоб зробити файли cookie доступними з будь-якої сторінки вебсайту.

domain

Домен визначає звідки будуть доступні файли cookie. Проте, на практиці, існують деякі обмеження – ми не можемо вказати будь-який домен.

Не існує способу зробити файли cookie доступними з іншого домену 2-го рівня, тому some.com ніколи не отримає файл cookie заданий на example.com.

За замовчуванням, файли cookie доступні лише на тому домені на якому були встановлені.

expires, max-age

За замовчуванням, якщо файл cookie не має одного з цих параметрів, то файл зникає при закриванні браузера. Такі файли cookies називаються сесійними (“session cookies”).

Щоб файли cookies могли “пережити” закривання браузера, можна встановити значення одного з параметрів expires або max-age.

expires=Tue, 19 Jan 2038 03:14:07 GMT

secure

Файл cookie повинен передаватися виключно по HTTPS-протоколу.

За замовчуванням, якщо ми створимо файл cookie на http://example.com, тоді він автоматично з’явиться на https://example.com та навпаки.

Тобто файли cookie базуються на домені, вони не залежать від протоколів.

samesite

Це ще один атрибут безпеки. Він створений щоб захищати від так званих XSRF-атак (міжсайтова підміна запиту).

Щоб зрозуміти як він працює та в яких випадках може бути корисним, давайте детальніше розглянемо поняття XSRF-атак.

samesite=strict (теж саме що samesite без заданого значення)

Файли cookie з параметром samesite=strict ніколи не відправляються якщо корстувач прийшов ззовні (з іншого сайту).

samesite=lax

Відправляється лише тоді, коли виконуються обидві умови:

1. Обраний HTTP-метод безпечний (наприклад GET, але не POST)
2. Операція виконує навігацію вищого рівня (змінює URL в адресному полі браузера)

local | sessionsStorage

Об’єкти веб-сховища localStorage та sessionStorage дозволяють зберігати дані в браузері у вигляди пар ключ/значення.

Відмінності від cookie:

- На відміну від файлів cookies, об’єкти веб-сховища не надсилаються на сервер із кожним запитом. Завдяки цьому ми можемо зберігати набагато більше даних. Більшість браузерів дозволяють принаймні 5 мегабайтів даних (або більше), користувач може навіть змінити цей об’єм.
- Крім того, на відміну від файлів cookies, сервер не може маніпулювати об’єктами сховища через HTTP-заголовки. Все зроблено на JavaScript.
- Сховище прив’язане до оригінального сайту (домен/протокол/порт). Таким чином, що різні протоколи або субдомени мають різні об’єкти зберігання, і не можуть отримати доступ до даних один одного

local | sessionStorage methods

Обидва об’єкти сховища забезпечують однакові методи та властивості:

- setItem(key, value) – зберегти пару ключ/значення.
- getItem(key) – отримати значення за ключем.
- removeItem(key) – видалити дані за ключем.
- clear() – видалити все.
- key(index) – отримати ключ на заданій позиції.
- length – кількість збережених елементів.

localStorage

Основними особливостями localStorage є:

- Спільний доступ з усіх вкладок і вікон для одного і того ж самого сайту.
- Термін дії даних не закінчується. Дані залишаються після перезавантаження браузера і навіть перезавантаження ОС.

Ми повинні бути на тому ж самому сайті (домен/порт/протокол), шлях URL-адреси може відрізнятись

localStorage доступний для одного сайту в усіх відкритих вікнах, тому якщо ми встановимо дані в одному вікні, зміна стане видимою в іншому.

Важливо!

І ключ, і значення мають бути рядками.

sessionStorage

Властивості та методи ті самі, але можливості більш обмежені:

- sessionStorage існує лише на поточній вкладці браузера. Інша вкладка з тією ж сторінкою матиме інше сховище. Але він використовується між iframes на одній вкладці (за умови, що це один сайт).
- Дані зберігаються після оновлення сторінки, але не закриття/відкриття вкладки.

Подія storage

Коли дані оновлюються в localStorage або sessionStorage, запускається подія storage із властивостями:

- key – ключ, який було змінено (null, якщо викликається .clear()).
- oldValue – старе значення (null, якщо це новий ключ).
- newValue – нове значення (null, якщо дані видалено).
- url – URL-адреса документа, де відбулося оновлення.
- storageArea – об’єкт localStorage або sessionStorage, де відбулося оновлення.

Важливо!

Подія запускається на всіх об’єктах window, де доступне сховище, крім того, що його викликало.

IndexedDB

IndexedDB — це база даних, вбудована в браузер, набагато потужніша, ніж localStorage.

- Зберігає майже будь-які значення за ключами, використовує кілька типів ключів.
- Підтримує транзакції для надійності.
- Підтримує запити за диапазоном ключів та індекси.
- Може зберігати набагато більші обсяги даних, ніж localStorage.

Специфікація. Базове виконання базується на подіях.

Ми також можемо використовувати async/await за допомогою обгортки на основі промісів, наприклад https://github.com/jakearchibald/idb.

Технічно дані зазвичай зберігаються в домашньому каталозі відвідувача разом з налаштуваннями браузера, розширеннями тощо.

Відкрити базу даних

Синтаксис:

let openRequest = indexedDB.open(name, version);

- name – рядок, ім’я бази даних.
- version – версія є цілим числом, типово 1.

Виклик повертає об’єкт openRequest, ми повинні прослухати події на ньому:

- success: база даних готова, в openRequest.result є “об’єкт бази даних”, ми повинні використовувати його для подальших викликів.
- error: не вдалося відкрити.
- upgradeneeded: база даних готова, але її версія застаріла (див. нижче).

IndexedDB

IndexedDB має вбудований механізм «схему контролю версій», який відсутній у серверних базах даних.

Якщо версія локальної бази даних менша за вказану в open, тоді запускається спеціальна подія upgradeneeded, і ми можемо порівнювати версії та оновлювати структури даних за потребою.

Подія upgradeneeded також запускається, коли база даних ще не існує (технічно її версія дорівнює 0), тому ми можемо виконати ініціалізацію.

Сховище об’єктів

Щоб зберегти щось у IndexedDB, нам потріне сховище об’єктів.

Сховище об’єктів є основною концепцією IndexedDB. Аналоги в інших базах даних називаються «таблицями» або «колекціями». Саме там зберігаються дані. База даних може мати кілька сховищ: одне для користувачів, інше для товарів тощо.

Ми можемо зберігати практично будь-які значення, включаючи складні об’єкти.

IndexedDB використовує стандартний алгоритм серіалізації щоб клонувати та зберігати об’єкт. Це як JSON.stringify, але потужніший, здатний зберігати набагато більше типів даних.

Для кожного значення в сховищі має бути унікальний “ключ”.

Синтаксис створення:

db.createObjectStore(name[, keyOptions]);

Зауважте, що операція синхронна, та не потребує await.

- name – назва сховища, наприклад. "books" для книжок,
- keyOptions є необов’язковим об’єктом з однією з двох властивостей:

keyPath – шлях до властивості об’єкта, який IndexedDB використовуватиме як ключ, напр. id.

autoIncrement – якщо true, тоді ключ для щойно збереженого об’єкта генерується автоматично у вигляді постійно зростаючого числа.

Сховище об’єктів

Сховище об’єктів можна створити/змінити лише під час оновлення версії БД в обробнику upgradeneeded.

Це технічне обмеження. За межами обробника ми зможемо додавати/вилучати/оновлювати дані, але сховища об’єктів можна створювати/вилучати/змінювати лише під час оновлення версії.

Щоб виконати оновлення версії бази даних, існує два основних підходи:

- Ми можемо реалізувати функції оновлення для кожної версії: від 1 до 2, від 2 до 3, від 3 до 4 тощо. Потім у upgradeneeded ми можемо порівняти версії (наприклад, стару 2, тепер 4) та запустити крок оновлення для кожної версії покроково, для кожної проміжної версії (2 до 3, потім від 3 до 4).
- Або ми можемо просто перевірити базу даних: отримати список існуючих сховищ об’єктів як db.objectStoreNames. Цим об’єктом є DOMStringList який надає метод contains(name) для перевірки існування. А потім ми можемо виконувати оновлення залежно від того, що існує, а що ні.

Щоб видалити використовуємо

db.deleteObjectStore('books')

Транзакції

Транзакція — це група операцій, які повинні або всі завершитися успішно, або всі невдало.

Наприклад, коли людина щось купує, нам потрібно:

- Знайти товар.
- Додати товар до списку.

Усі операції з даними повинні виконуватися в рамках транзакції в IndexedDB.

Почати транзакцію:

db.transaction(store[, type]);

- store це назва сховища, до якого має отримати доступ транзакція, напр. "cars". Це може бути масив імен сховищ, якщо ми збираємося отримати доступ до кількох сховищ.
- type – тип транзакції, один з:

readonly – може лише читати дані, типово.

readwrite – може лише читати та записувати дані, але не створювати/видаляти/змінювати сховища об’єктів.

Транзакції

Ось чотири основні кроки:

- Створіть транзакцію, вказавши всі сховища, в які вона збирається отримати доступ (1).
- Отримайте об’єкт магазину за допомогою transaction.objectStore(name) (2).
- Виконайте запит до сховища об’єктів books.add(book) (3).
- …Обробіть запит успішно/помилка (4), потім можливо робити інші запити, якщо потрібно.

Сховища об’єктів підтримують два методи збереження значення:

- put(value, [key]) Додайте value до сховища. Key надається лише в тому випадку, якщо в сховищі об’єктів не було параметра keyPath або autoIncrement. keyPath чи autoIncrement опція. Якщо вже є значення з таким самим ключем, воно буде замінено.
- add(value, [key]) Те саме що put, але якщо вже є значення з тим самим ключем, запит не вдається, і генерується помилка з назвою "ConstraintError".

Автозавершення транзакцій

Коли всі запити транзакції завершено, а черга мікрозадач порожня, вона завершується автоматично.

Принцип автозавершення транзакцій має важливий побічний ефект. Ми не можемо вставити асинхронну операцію, як-от fetch, setTimeout, у середину транзакції. IndexedDB не буде чекати транзакцію, поки вона не виконана.

Щоб вручну припинити транзакцію, викличте: transaction.abort();

Дякую за увагу