**Питання Fetch**

1. Fetch API

надає спрощений, але гнучкий і потужний інструмент для доступу до мережевих ресурсів у порівнянні зі стандартним XMLHttpRequest.

1. функція fetch()

Функція fetch має такий синтаксис:

const fetchPromise = fetch(resource [, init])

В якості необхідного параметра - функція приймає параметри ресурсу, до якого буде доступ функція. В якості необов'язкового параметра функція може приймати об'єкт з додатковими параметрами запиту.resourceinit

Функція fetch() повертає обіцянку, яка отримує відповідь після завершення запиту на мережний ресурс.

1. fs.readFile()

використовується для читання файлів. Першим параметром функції є адреса файлу (в даному випадку передбачається, що індекс.html файл знаходиться в тій же папці, що і сервер.js файл). Другий параметр - це функція, яка викликається після прочитання файлу і отримання його вмісту через його дані другого параметра. Тоді читабельний також можна розморозити за допомогою функції .response.end(data)

1. Отримати данні з fetch

Метод response.text() також повертає обіцянку. А щоб отримати фактичний текст відповіді, підключіть другий метод then(), в якому в функції зворотного виклику отримуємо текст відповіді:

.then(responseText => console.log(responseText));

1. fetch з використанням async/await

Оскільки функція fetch() повертає Promise, замість того, щоб нанизувати методи then(), ми можемо використовувати оператори async/await для отримання відповіді.

1. Параметри запиту

Другий аргумент - це параметри запиту. Список параметрів:{options}

* method - метод запиту (GET, POST, PUT, DELETE, HEAD);
* headers - ЗАГОЛОВКИ HTTP;
* body - корпус запиту (використовується для методу: POST / PUT);
* cache - режим кешу (за замовчуванням, перезавантаження, без кешу);
* mode - режим запиту (cors, no-cors, однакового походження);
* redirect - вказує, як поводитися з переспрямуваннями (стежити, помилятися, вручну);
* referrer - реферер запитів;
* signal - Перерваний підпис, переривання запиту;
* credentials - відправка файлів cookie разом із запитом - mit, того ж походження.

1. Параметри відповіді

Метод Promise повертає об'єкт класу, який має такі властивості:fetch()Response

* status - код відповіді;
* statusText - текстове 📜 повідомлення, що відповідає коду відповіді;
* ok - Логічне значення, що вказує на успіх коду відповіді (true: 200-299)
* headers - об'єкт з заголовками відповідей, в якому ключ - ім'я заголовка, а значення ключа - значення відповідного заголовка;
* url - URL-адреса, на яку був відправлений запит;
* body - дані відповіді у форматі ReadableStream
* bodyUsed - Логічне значення, яке вказує на те, що дані були прочитані.

1. Обробка відповіді

Передані дані знаходяться в області . Для зміни формату можна скористатися наступними способами:ReadableStream

* text() - перетворює відповідь на рядок;
* json() - перетворює відповідь у форматі JSON;
* blob() - перетворює відповідь на об'єкт BLOB;;
* formData() - перетворює відповідь на екземпляр FormData;
* arrayBuffer() - Перетворює відповідь на об'єкт ArrayBuffer.

1. Обробка помилок

Дізнатися, чи не вдалася помилка, можна за допомогою властивостей: "status" і "ok".fetch()

fetch('https://jsonplaceholder.typicode.com/users')

.then(response => {

if (!response.ok) {

console.log('Что-то пошло не так... Статус: ' + response.status)

} else {

return response.json()

}

})

.then(data => console.log(data))

Використання .catch()

fetch('https://jsonplaceholder.typicode.com/users')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.log(error))

1. список заборонених заголовків HTTP

* Accept-Charset, Accept-Encoding
* Access-Control-Request-Headers
* Access-Control-Request-Method
* Connection
* Content-Length
* Cookie, Cookie2
* Date
* DNT
* Expect
* Host
* Keep-Alive
* Origin
* Referer
* TE
* Trailer
* Transfer-Encoding
* Upgrade
* Via
* Proxy-\*
* Sec-\*

1. Завантаження зображення

Ми можемо надсилати двійкові дані за допомогою , використовуючи об'єкти або .fetchBlobBufferSource

1. Відмінність Fetch API від XMLHttpRequest (XHR)

перший використовує програми, які значно полегшують роботу з запитами та відповідями. Код простіший і чистіший.FetchXMLHttpRequestFetch

1. Основний синтаксис:

let promise = fetch(url, [options])

1. Заголовки відповідей

Заголовки відповідей зберігаються в об'єкті Map-like response.headers.

1. Використання методів Map

Це насправді не Map, але ми можемо використовувати ті ж методи, що і з Map, щоб отримати назву за її назвою або ітерувати через заголовки в циклі:

let response = await fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits');

// получить один заголовок

alert(response.headers.get('Content-Type'));

// application/json; charset=utf-8

// перебрать все заголовки

for (let [key, value] of response.headers) {

alert(`${key} = ${value}`);

}

1. Функція submit() без async/await

function submit() { canvasElem.toBlob(function(blob) {

fetch('/article/fetch/post/image', { method: 'POST', body: blob })

.then(response => response.json())

.then(result => alert(JSON.stringify(result, null, 2))) },

'image/png'); }

1. referrer, referrerPolicy

Ці параметри визначають, як fetch встановлює заголовок HTTP REFERER.

Зазвичай цей заголовок розміщується автоматично і містить URL-адресу сторінки, з якої надійшов запит. У більшості випадків це абсолютно неважливо, в деяких випадках заради більшої безпеки має сенс видалити або скоротити його.

1. Можна іннорувати referrer?

Щоб не відправляти referrer, потрібно вказати порожній рядок зі значенням:

fetch('/page', { referrer: "" // не ставить заголовок Referer });

1. Параметр ReferenceerPolicy

Існує три типи запитів:

* Запит на те ж джерело.
* Запит на інше джерело.
* Запит від HTTPS до HTTP (від захищеного протоколу до небезпечного).

1. Відмінності referrer і referrerPolicy

На відміну від параметра реферера, який дозволяє вказати точне значення Referer, параметр referrerPolicy повідомляє браузеру загальні правила про те, що робити для кожного типу запиту.

1. Можливі значення описані в специфікації політики реферера:

* "no-referrer-when-downgrade" - це значення за замовчуванням: реферер завжди надсилається, якщо ми не надсилаємо запит з HTTPS до HTTP (від більш безпечного протоколу до менш безпечного).
* "без рефері" - ніколи не надсилайте рефері.
* "origin" – надсилайте рефереру тільки поточне джерело, а не повну URL-адресу сторінки, наприклад, надсилайте тільки http://site. com замість http://сайту. com/path.
* "origin-when-cross-origin" – надішліть повного Референта для запитів у межах поточного джерела, але для запитів до іншого джерела надішліть лише саме джерело (як зазначено вище).
* "same-origin" – відправляйте повного Референта для запитів в межах поточного джерела, а для запитів іншому джерелу, не надсилайте його взагалі.
* "strict-origin" – надсилайте тільки вихідне значення, не надсилайте референта на HTTPS→HTTP запити.
* «strict-origin-when-cross-origin» – для запитів в межах поточного джерела відправляйте повного Референта, для запитів в інше джерело надсилайте тільки вихідне значення, у випадку HTTPS→HTTP запитів, нічого не надсилайте.
* "dangerous-URL" - Завжди надсилайте повну URL-адресу рефереру, навіть при запиті HTTPS→HTTP.

1. Політика referer

Політика referer - не тільки для отримання

Політика установки referer, описана в специфікації політики реферера, не існує тільки для отримання, вона є більш глобальною.

Зокрема, ви можете встановити політику за замовчуванням для всієї сторінки за допомогою заголовка HTTP Політики реферера або на рівні посилання <a rel ="noreferrer", >.

1. Mode

Параметр Mode - це захист від випадкового надсилання запиту іншому джерелу:

"cors" - це за замовчуванням, дозволяє робити такі запити, як описано в Fetch: запити на інші сайти,

«однакове походження» – запити на інше джерело заборонені,

"no-cors" - допускаються тільки прості запити до іншого джерела.

1. Використання Mode

Ця опція може стати в нагоді, якщо URL-адреса для отримання надходить від третьої сторони, і нам потрібен якийсь "глобальний перемикач" для запитів до інших джерел.

1. Облікові дані

Параметр облікові дані визначає, чи слід отримувати надсилати файли cookie та заголовки авторизації HTTP разом із запитом.

* «same-origin» - це за замовчуванням, не відправляти запити в інше джерело,
* "include" - завжди надсилати, але вам потрібен заголовок Access-Control-Allow-Credentials у відповіді від сервера, щоб JavaScript міг отримати доступ до відповіді сервера,
* «omit» – не відправляти ні за яких обставин, навіть для запитів, зроблених в рамках поточного джерела.

1. Cache

За замовчуванням fetch робить запити за допомогою стандартного кешування HTTP. Тобто враховуються заголовки "Термін дії", "Кеш-контроль", "If-Modified-Since" і так далі. Так само, як і звичайні HTTP-запити.

1. Налаштування кешу

Налаштування кешу дозволяє ігнорувати кеш HTTP або налаштовувати його використання:

* "default" – fetch буде використовувати стандартні правила та заголовки кешування HTTP,
* "no-store" - повністю ігнорувати кеш HTTP, цей режим стає режимом за замовчуванням, якщо є заголовки, такі як If-Modified-Since, If-None-Match, If-Unmodified-Since, If-Match або If-Range,
* «перезавантажити» – не брати результат з кешу HTTP (навіть якщо він присутній), а зберігати відповідь в кеші (якщо це дозволено заголовками відповідей);
* "no-cache" – якщо є кешована відповідь – створіть умовний запит, інакше – звичайний запит. Збережіть відповідь у кеші HTTP,
* «force-cache» – використовуйте відповідь з кешу HTTP, навіть якщо вона застаріла. Якщо в кеші HTTP немає відповіді, зробіть звичайний HTTP-запит, дійте як зазвичай,
* "only-if-cached" – використовуйте відповідь з кешу HTTP, навіть якщо вона застаріла. Якщо в кеші HTTP немає відповіді, кинуто помилку. Це працює лише тоді, коли в режимі встановлено значення "однакове походження".

1. Redirect

Зазвичай отримання прозоро слідує за HTTP-редиректам, такими як 301, 302 тощо.

Це можна змінити за допомогою опції переспрямування:

" follow " – за замовчуванням дотримуйтесь HTTP-редиректам,

"error" – помилка в разі перенаправлення HTTP,

" manual " - не дотримуйтесь перенаправлення HTTP перенаправлення, але встановіть адресу перенаправлення на response.url, і response.redirected буде встановлено на true, щоб ми могли зробити перенаправлення на нову адресу вручну.

1. Integrity

Опція цілісності дозволяє заздалегідь перевірити, чи відповідає відповідь відомій контрольній сумі.

Як описано в специфікації, підтримуваними хеш-функціями є SHA-256, SHA-384 і SHA-512. Залежно від браузера, можуть бути й інші.

Наприклад, ми завантажуємо файл, і ми точно знаємо, що його контрольна сума SHA-256 є "abcdef" (звичайно, справжня контрольна сума буде довшою).

1. Keepalive

Параметр keepalive вказує на те, що запит може «пережити» сторінку, яка його надіслала.

Наприклад, ми збираємо статистичні дані про те, як відвідувач поводиться на нашій сторінці (що він натискає, частини сторінки, яку вони переглядають), щоб проаналізувати та покращити інтерфейс.

Коли відвідувач залишає нашу сторінку, ми хочемо зберігати зібрані дані на нашому сервері.

1. Обмеження keepalive

* Ми не можемо надсилати мегабайти: межа тіла для запиту з keepalive становить 64kb.
* Якщо ми зберемо більше даних, то зможемо відправляти їх регулярно, «партіями», то на момент останнього запиту в onunload їх буде мало.
* Це обмеження застосовується до всіх запитів з keepalive. Тобто ми не можемо обійти його, відправивши 100 запитів одночасно – кожен по 64 КБ.
* Ми не зможемо обробити відповідь з сервера, якщо запит буде зроблений з невичерпним навантаженням: на той момент документ вже вивантажений, його функції не будуть працювати.

Зазвичай сервер відправляє порожню відповідь на такі запити, тому це не проблема.