УДК 004.4'22

M.A. Leshok
I.V.Bogach

ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ SWR У ДОДАТКАХ NEXT.JS[1]

Вінницький національний технічний університет

Анотація

У статті аналізується робота технології SWR у веб-додатках з використанням технології NEXT.JS, переваги цієї технології, структура коду, використання та практичні приклади

Ключові слова: swr, reactjs, nextjs, javascript.

Abstract

The article analyzes SWR technology in web applications using NEXT.JS technology, the advantages of this technology, code structure, use and practical examples.

Keywords: swr, reactjs, nextjs, javascript.

Вступ

На сьогоднішній день технологія SWR все більше набирає популярності серед розробників ефективних веб-додатків, особливо з використанням Next.JS.

Ця назва походить від stale-while-revalidate, основна ідея якого полягає в кешуванні даних, що робить його все більш популярним у розробці веб-додатків. Ця стратегія інвалідності HTTP-кешу була популяризована протоколом HTTP RFC 5861. Його головна перевага в тому, що він завантажує закешовані дані відразу ж і оновлює їх "на льоту", і тільки потім відправляє fetch запит (повторна валідація) і отримує актуальні дані. Цей патерн дозволяє обробляти вже оновлений контент, що є відмінним компромісом між UX (користувацьким досвідом) та ефективністю веб-програми.

Основні переваги SWR

Основні переваги SWR:

• Легкість: швидка, легка та багаторазова вибірка даних

• Вбудований кеш та дедуплікація запитів

• Не залежить від протоколу даних

• Орієнтований на "JAM" стек (JavaScript, API, Markup)

• Не залежить від серверної частини, тобто не вимагає, щоб фронтенд частина вашої програми була заснована на таких технологіях та мовах програмування, як rails, node.js, python або будь-яких ще. Його написано на node.js.

• Має строгу типізацію з використанням TypeScript

• Легко інтегрується в React та React Native програми

• Працює в реальному часі

Більш того, за допомогою SWR ми можемо робити швидку навігацію сторінками, задавати параметри та інтервал ревалідації даних, оновлювати дані на основі відновлення мережі, мутувати дані та багато іншого.

Базове використання SWR

Для звичайних RESTful API з даними JSON спочатку потрібно створити функцію, яка буде обгорткою для своєї функції запиту:

```
const fetcher = (...args) => fetch(...args).then(res => res.json())
```

Потім можна імпортувати хук useSWR і почати використовувати його всередині будь-яких функціональних компонентів:

```
import useSWR from 'swr'
function Profile () {
const { data, error } = useSWR('/api/user/123', fetcher) if (error) return <div>failed to load</div>
```

```
if (!data) return <div>loading...</div>
return <div>hello {data.name}!</div>
}
```

Зазвичай існує 3 можливі стани запиту: «завантаження», «готовий» або «помилка». Ви можете використовувати значення data та error, щоб визначити поточний стан запиту та повернути відповідний UI та стан.

При створенні веб-програми вам може знадобитися повторно використовувати дані в багатьох місцях інтерфейсу користувача. Створювати перевикористовувачі даних, що перевикористовуються, поверх SWR можна наступним чином:

```
function useUser (id) {
const { data, error } = useSWR(`/api/user/${id}`, fetcher) return {
user: data,
isLoading: !error && !data, isError: error
}
}
```

Далі ви можете використати цю функцію у ваших компонентах:

```
function Avatar ({ id }) {
const { user, isLoading, isError } = useUser(id)

if (isLoading) return <Spinner /> if (isError) return <Error /> return <img src={user.avatar} />
}
```

Використовуючи цей шаблон, можна забути про обов'язкове отримання даних: запустіть запит, оновіть стан завантаження і поверніть остаточний результат. Натомість ваш код більш декларативний: вам просто потрібно вказати, які дані використовуються компонентом.

Практичний приклад використання SWR

Припустимо, що нам сайт показує меню навігації та контент, які залежать від «user». Зазвичай ми отримуємо дані один раз, використовуючи useEffect в компоненті верхнього рівня, і передаємо їх дочірнім компонентам через props (зверніть увагу, що ми поки що не обробляємо стан помилки):

```
// компонент сторінки Page

function Page () {
const [user, setUser] = useState(null)

// отримання даних запитом fetch

useEffect(() => {
fetch('/api/user')
.then(res => res.json())
.then(data => setUser(data))
}, [])

// глобальний стан завантаження
if (!user) return <Spinner/>

return <div>
<Navbar user={user} />
<Content user={user} />
```

```
</div>
}

// дочірні компоненти

function Navbar ({ user }) { return <div>
...
<Avatar user={user} />
</div>
}

function Content ({ user }) {
return <h1>Welcome back, {user.name}</h1>
}

function Avatar ({ user }) {
return <img src={user.avatar} alt={user.name} />
}
```

Зазвичай нам потрібно зберегти вибірку всіх даних у компоненті верхнього рівня та додати властивості для кожного компонента у глибині дерева. Код буде складніше підтримувати, якщо ми додамо на сторінку більше залежностей даних.

Хоча ми можемо уникнути передачі властивостей (props) з допомогою Context, все ще існує проблема з динамічним контентом: компоненти всередині контенту сторінки може бути динамічними, і компонент верхнього рівня може знати, які дані знадобляться його дочірнім компонентам.

SWR чудово вирішує проблему. За допомогою щойно створеного хука useUser код можна відредагувати так:

```
// компонент сторінки Page

function Page () {
return <div>
<Navbar />
<Content />
</div>
}

// дочірні компоненти

function Navbar () { return <div>
...
<Avatar />
</div>
}

function Content () {
const { user, isLoading } = useUser()

if (isLoading) return <Spinner />

return <h1>Welcome back, {user.name}</h1>
}
```

```
function Avatar () {
const { user, isLoading } = useUser() if (isLoading) return <Spinner />
return <img src={user.avatar} alt={user.name} />
}
```

Тепер дані прив'язані до компонентів, яким необхідні дані, і всі компоненти незалежні один від одного. Всім батьківським компонентам не потрібно нічого знати про дані або передачі даних. Вони просто відображаються на UI. Код став набагато чистішим і простішим у підтримці.

Найпрекраснішим є те, що в API буде надіслано лише 1 запит, тому що вони використовують один і той же ключ SWR, а запит автоматично виводиться, кешується та передається.

Крім того, програма тепер має можливість оновлювати дані при фокусі користувача або повторному підключенні до мережі! Це означає, що коли ноутбук користувача виходить з режиму сну або він перемикається між вкладками браузера, дані оновлюються автоматично.

Використання SWR у додатках Next.JS

Якщо сторінка містить дані, що часто оновлюються, і не потрібно попередньо обробляти дані, SWR у Next.JS не вимагає спеціального налаштування: необхідно імпортувати useSWR і використати хук всередині будь-яких компонентів, які використовують дані.

Тобто, спочатку необхідно показати сторінку без даних. Можна відобразити стан завантаження для відсутніх даних.

Потім отримати дані на стороні клієнта та відобразити їх, коли запит буде виконано.

Цей підхід добре працює, наприклад, для сторінок, які потребують авторизації. Оскільки така сторінка є приватною, орієнтованою на конкретного користувача, пошукова оптимізація не має значення і сторінку не потрібно попередньо обробляти. Дані часто оновлюються, що потребує вибірки даних під час запиту.

Якщо сторінка повинна бути попередньо відображена, Next.js підтримує 2 форми попереднього відмалювання: статична генерація (SSG) та рендеринг на стороні сервера (SSR).

Разом з SWR можна попередньо обробити сторінку для SEO, а також мати такі функції, як кешування, повторна перевірка, відстеження фокусу, повторна вибірка інтервалу на стороні клієнта.

Ви можете передати попередньо вибрані дані як початкове значення в опцію initialData:

```
export async function getStaticProps() {

// getStaticProps виконується на боці серверу

const posts = await fetcher('/api/posts')

return { props: { posts } }
}

function Posts (props) {

// тут функція fetcher буде виконуватись на боці клієнту

const { data } = useSWR('/api/posts', fetcher, { initialData: props.posts })
// ...
}
```

Сторінку все ще попередньо оброблено. Це означає, що вона оптимізована для SEO, може бути кешована та доступна дуже швидко. Але після гідратації вона повністю виконується від SWR на стороні клієнта. Це означає, що дані можуть бути динамічними та оновлюватися з часом та при

взаємодії з користувачем.

У наведеному вище прикладі функція fetcher використовується для завантаження даних як з клієнта, так і сервера, і вона повинна підтримувати обидва середовища. Але це не вимога. Ви можете використовувати різні способи завантаження даних із сервера або клієнта.

Висновки

Використання SWR у додатках, заснованих на Next.JS или React.JS, значно спрощує та пришвидшує розробку. Головною перевагою є те, що запит кешується та надсилається на сервер лише один раз, що позитивно впливає на ефективність та продуктивність додатку.

Більше того, за допомогою SWR ми спрощуємо архітектуру додатку та не ускладнюємо керування станом. Все більше розробників починають використовувати дану технологію, а також покращують якість коду за допомогою open-source.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Документація SWR. [Электронный ресурс], 2021. Режим доступу: https://swr.vercel.app/
2. Репозиторий GitHub / SWR. [Электронный ресурс], 2021. Режим доступу: https://github.com/vercel/swr/
3. Документация React.JS. [Электронный ресурс], 2021. Режим доступу: https://reactjs.org/
4. Репозиторий GitHub / React.JS. [Электронный ресурс], 2021. Режим доступу: https://github.com/facebook/react/
5. Документация Next.JS. [Электронный ресурс], 2021. Режим доступу: https://nextjs.org/
6. Репозиторий GitHub / SWR. [Электронный ресурс], 2021. Режим доступа: https://github.com/vercel/next.js

Лешок Максим Андрійович – *студент групи 1ІСТ-19Б, кафедра автоматизації та інтелектуальних інформаційних технологій, Факультет комп'ютерних систем і автоматики, Вінницький національний технічний університет, м.Вінниця, e-mail: max6leshok@gmail.com*

Богач Ілона Віталіївна *– к.т.н., доцент кафедри Автоматизації та інтелектуальних інформаційних технологій, Вінницький національний технічний університет, м.Вінниця, e-mail: ilona.bogach@gmail.com*

Leshok Maksym Andreevich – *student of 1IST-19B group, Department of Automatization and Intellectual Informational Technologies, Faculty of Computer Systems and Automatics, Vinnytsia National Technical University, Vinnytsia, e-mail: max6leshok@gmail.com*

Bogach Ilona Vitaliivna *- Associate Professor of Automation and Intelligent Information Technologies, Vinnytsia National Technical University, Vinnytsia, e-mail: ilona.bogach@gmail.com*