Engineer

Серверний рендеринг

🖨 формування HTML сторінки на сервері для кожного запиту

Що це таке

Серверний рендеринг (англійською server-side rendering, скорочено SSR), який ще називають рендерингом на стороні сервера, означає, що HTML сторінки формується на сервері в ту мить, коли хтось її запитує. Браузер отримує готовий документ, де вміст уже на місці, і може показати його ще до того, як виконається бодай один скрипт. У JavaScript-фреймворку ті самі компоненти потім виконуються вдруге, уже в браузері, який підхоплює надісланий HTML і робить його інтерактивним.

Цей термін відповідає на питання про час: коли і де створюється HTML? Поширених відповідей три. Генератор статичних сайтів створює його один раз, під час збирання сайту, і віддає всім той самий файл. Серверний рендеринг створює його на сервері для кожного запиту, тож сторінка може залежати від того, хто питає і що саме кажуть дані в цю мить. Клієнтський рендеринг, який часто називають CSR, надсилає майже порожній документ і пакет JavaScript, що будує сторінку просто в браузері. Справжні застосунки часто поєднують усі три підходи, маршрут за маршрутом.

Next.js описує це так само. Його документація наводить dynamic rendering як іншу назву SSR і пояснює, що HTML сторінки генерується на кожен запит. У Pages Router сторінка обирає цей режим, експортуючи асинхронну функцію `getServerSideProps`, яку сервер викликає на кожен запит; споріднена `getStaticProps` робить ту саму роботу один раз, під час збирання. Механізм під капотом дає React. API з `react-dom/server` перетворюють дерево компонентів на HTML на сервері: або потоком, через `renderToPipeableStream` для Node.js і `renderToReadableStream` для Web Streams, або одним шматком через старішу `renderToString`, яку React відносить до legacy-API з обмеженою функціональністю.

Друга половина відбувається в браузері й називається гідратацією. Next.js описує перше завантаження так: HTML дає швидкий неінтерактивний попередній перегляд, а потім JavaScript гідратує компоненти. Саму гідратацію Next.js визначає як спосіб, у який React прив'язує обробники подій до DOM, щоб статичний HTML став інтерактивним. Довідка React до `hydrateRoot` не приховує компромісу: знімок у HTML створює враження, що застосунок завантажується швидше, і якусь мить сторінка виглядає готовою, хоча ще такою не є.

Застосуйте поняття до рішення

Застосуйте «Що таке серверний рендеринг?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Інтерфейси, що швидко завантажуються, працюють на телефоні й поводяться однаково в усіх браузерах ваших клієнтів.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Керувала full-stack-розробкою GIS-карт — full-stack-картографія — від бази даних до браузера

Керувала full-stack-розробкою GIS-карт Зв'язатися з нами →

Чому це важливо

Вміст є вже в першій відповіді. Читач на повільному з'єднанні бачить текст і зображення, поки JavaScript ще в дорозі, а краулер, який не виконує JavaScript, бачить ту саму сторінку, що й людина, а не порожню оболонку. Браузерові не треба спершу виконати скрипт, а потім завантажити дані, щоб намалювати справжній вміст.

Сторінка також може залежати від запиту, і саме це відрізняє її від статичного збирання. Next.js радить `getServerSideProps` для даних, які стосуються конкретного користувача або стають відомі лише під час запиту, як-от заголовок authorization чи місцезнаходження відвідувача, а все інше відсилає до генерації під час збирання. Дані отримують поруч із тим місцем, де вони зберігаються. Код у функції виконується лише на сервері, її імпорти не потрапляють до клієнтського бандла, і вона може звертатися до бази даних чи CMS напряму. Один похід на сервер може замінити запити, які браузер інакше надсилав би один за одним, щойно завантажаться його скрипти.

Ціна тут — дзеркальне відображення того, що заощаджує статичний сайт. Потрібен запущений сервер, і він виконує рендеринг заново для кожного запиту. Відповідь може початися лише тоді, коли дані готові, тож без

потокової передачі найповільніший запит до даних задає темп усій сторінці. До того ж рендеринг на сервері не прибирає JavaScript: браузер усе одно завантажує кожен компонент, який має гідратувати, і доки гідратація не завершиться, кнопка може бути намальована на екрані, але нічого не робити.

Тому вибір робиться для кожної сторінки окремо, а не для всього проєкту. Сторінка, однакова для всіх читачів, мало виграє від повторного рендерингу на кожен запит, а сторінка, що змінюється разом із читачем або з даними, які оновлюються щохвилини, якраз і виправдовує для SSR окремий сервер. Цей сайт стоїть на статичному кінці шкали. Набір технологій сайту ставить статику на перше місце там, де це доречно, тож ці сторінки не потребують виконання застосунку під час запиту, а збирає сайт Hugo.

Як це виглядає на практиці

Сторінку нижче написано для цієї статті, і вона використовує Pages Router. Вона показує замовлення клієнта, що увійшов у систему, відправляє відвідувача без сесії на сторінку входу й просить усі кеші не зберігати результат. `readSession` і `db` позначають те, що застосунок використовує для сесій і даних.

```
// pages/orders.js
export async function getServerSideProps({ req, res }) {
  const session = await readSession(req.cookies.session);
  if (!session) {
    return { redirect: { destination: "/sign-in", permanent: false } };
  }

  const rows = await db.orders.listFor(session.userId);
  res.setHeader("Cache-Control", "no-store");

  return {
    props: {
      orders: rows.map((o) => ({
        id: o.id,
        number: o.number,
        status: o.status,
      })),
    },
  };
}

export default function Orders({ orders }) {
  return (
```

```
    <ul>
      {orders.map((o) => (
        <li key={o.id}>
          {o.number}: {o.status}
        </li>
      ))}
    </ul>
  );
}
```

Next.js виконує цю функцію на кожен запит до `/orders`. `req.cookies` — це заголовок Cookie запиту, уже розібраний. `redirect` із `permanent: false` — тимчасове перенаправлення, а не постійне. Усе, що повертається в `props`, має пережити `JSON.stringify` і потрапляє також у браузер як частина початкового HTML, щоб React міг гідратувати список тими самими даними, які використав сервер. Тому функція зводить кожен рядок до трьох полів, які показує сторінка, а не передає рядки бази даних як є. Коли читач згодом приходить за посиланням `next/link`, Next.js просить сервер виконати функцію ще раз і отримує props у вигляді JSON, без повного завантаження сторінки.

Фронтенд на Next.js 16 з поточної роботи засновниці в NextMariner показує, яку проблему це розв'язує. Сторінка портфоліо про цей фронтенд починається з дашборду для автентифікованих користувачів, який мав бути відчутно швидким і водночас інтерактивним. Якщо отримувати всі дані в браузері, перше завантаження затягується, і з'являється патерн N+1: кожен компонент прокидається й надсилає власний запит, і одна сторінка перетворюється на каскад мережевих звернень. У дашборді й на великих сторінках-каталогах фабрика `createPrefetchedServerPage` тепер отримує дані сторінки на сервері й передає їх клієнтові вже готовими до використання, тож компоненти одразу мають свої дані, а не йдуть по них кожен окремо. Сторінки, перенесені на фабрику, завантажуються за одне звернення замість цілої зливи звернень. Два інші режими проєкт залишає там, де вони доречні: публічні сторінки генеруються статично, бо не залежать від користувача, а справді інтерактивні частини й далі рендеряться на клієнті. Стратегію інвалідації кешу задокументовано для кожного запиту, тож дані лишаються актуальними, а застосунок не завантажує вдруге те, що вже має.

Де виникають проблеми

Сервер і браузер розходяться. Гідратація очікує, що перший рендер у браузері дасть той самий результат, що й на сервері. Компонент, який перевіряє `typeof window`, викликає `window.matchMedia` під час рендерингу або виводить поточний час, рендерить одне на сервері й інше в браузері. Серед причин React також називає різні дані на двох боках і зайві пробіли всередині кореневого елемента. У кращому разі розбіжність сповільнює гідратацію, а в гіршому обробники подій опиняються не на тих елементах, тож довідка React до `hydrateRoot` каже прямо: «You should treat mismatches as bugs and fix them.» Вихід — рендерити те саме на обох боках. Там, де браузер справді мусить показати щось інше, спершу рендерять серверну версію, а після гідратації перемикаються в ефекті (effect), і це коштує ще одного рендеру. `suppressHydrationWarning` доречний лише для неминучої різниці на кшталт позначки часу, і діє він тільки на один рівень углиб.

Секрети потрапляють у props. Усе, що повертається як props, записується в початковий HTML, щоб браузер міг гідратувати сторінку, і документація Next.js застерігає не передавати так нічого чутливого. Функція, яка повертає повний запис користувача, віддає хеш його пароля й внутрішні прапорці кожному, хто відкриє вихідний код сторінки. Повертати варто лише ті поля, які сторінка показує, як це робить приклад вище.

Спільний кеш віддає сторінку одного читача наступному. Документація Next.js ілюструє кешування сторінок, відрендерених на сервері, заголовком `Cache-Control: public, s-maxage=10, stale-while-revalidate=59`. Перенесений на персоналізовану сторінку, такий заголовок запрошує CDN або проксі перед сервером зберегти відповідь і віддати її тому, хто запитає наступним. RFC 9111 визначає `public` як дозвіл кешу зберегти відповідь, яку він інакше відкинув би, а `s-maxage` — як час життя, призначений для спільних кешів. Кожна з цих директив до того ж знімає правило, за яким спільний кеш не може повторно використати відповідь на запит із заголовком `Authorization`. Сторінці, побудованій із cookie чи облікових даних, потрібен `private`, що забороняє спільним кешам її зберігати, або `no-store`, що забороняє це всім кешам.

Найповільніший запит затримує всю сторінку. Якщо сторінку рендерити одним шматком, вона нічого не надсилає, доки не повернеться кожен запит за нею, тож один повільний запит затримує шапку, навігацію і все інше. Потоковий рендерер React ділить сторінку на межах `<Suspense>`. Він надсилає оболонку, тобто все поза межами, щойно вона готова, із fallback-

ами на місцях, а потім потоком досилає кожну частину, що лишилася, з невеликим вбудованим скриптом, який її вставляє. Із раннього надсилання випливає одна заковика: коли потік почався, код статусу вже не можна змінити, тож ситуацію з відсутнім записом чи невдалою перевіркою треба обробити до того, як піде оболонка, і саме в `onShellError` встановлюють статус помилки.

Що почитати далі

- Server-side Rendering (SSR) — Next.js про HTML, що генерується на кожен запит, і про те, чим `getServerSideProps` відрізняється від `getStaticProps`.
- getServerSideProps — коли її використовувати, що потрапляє до клієнта в початковому HTML і як кешувати з `Cache-Control`.
- Server React DOM APIs — потокові та legacy-API, які React пропонує для рендерингу компонентів у HTML на сервері.
- hydrateRoot — як React підхоплює відрендерений на сервері HTML і типові причини розбіжностей під час гідратації.
- renderToPipeableStream — оболонка, межі `<Suspense>` і застереження щодо коду статусу під час потокової передачі.
- RFC 9111: HTTP Caching — директиви `public`, `private`, `s-maxage` і `no-store` та правила для спільних кешів.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке серверний рендеринг?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/server-side-rendering/