Engineer

Ліниве завантаження

завантаження ресурсу чи даних лише тоді, коли вони потрібні

Що це таке

Ліниве завантаження (англійською lazy loading) — це підхід, за якого щось завантажують лише тоді, коли воно справді знадобилося, а не все одразу. На вебсторінці йдеться здебільшого про зображення й вбудовані фрейми нижче за текстом: їх отримують, коли читач догортає до них. Та сама ідея стосується коду, який має запуститися тільки після натискання, і пов'язаних записів, які програма читає з бази даних.

Стандарт HTML вбудовує це просто в розмітку. Зображення та iframe мають атрибут `loading` із двома значеннями. `eager` , значення за замовчуванням, завантажує ресурс одразу. `lazy` , за словами стандарту, «використовується, щоб відкласти отримання ресурсу, доки не виконаються певні умови». Той самий атрибут стандарт визначає також для `audio` і `video` . Відсутнє або написане з помилкою значення вважається `eager` , тож друкарська помилка просто лишає сторінку завантажуватися так, як завжди.

Умова полягає в тому, що елемент наближається до області перегляду (viewport). Кожен документ має для цього власний intersection observer, і коли той повідомляє, що елемент із `lazy` от-от з'явиться в полі зору, браузер відновлює його завантаження. Спостерігач зазирає трохи далі за видиму область, щоб зображення встигло прийти, поки читач ще гортає до нього, а ширину цього запасу стандарт залишає на розсуд браузера.

Сторінки можуть користуватися цим механізмом і напряму. Специфікація W3C Intersection Observer дає скрипту змогу асинхронно дізнаватися, коли елемент починає або перестає перетинатися з viewport чи з іншим елементом, і серед завдань, заради яких її написано, називає власні реалізації відкладеного завантаження вмісту й даних. Вона покриває те, чого

не покриває атрибут: фонове зображення, задане в CSS, мапу, якій потрібен власний скрипт, список, що підвантажує наступну порцію, коли читач доходить до кінця.

Для коду одиницею є модуль. Документація Next.js подає ліниве завантаження як спосіб зменшити обсяг JavaScript, потрібного, щоб показати маршрут: клієнтські компоненти й бібліотеки не входять до першого бандла і завантажуються тоді, коли стають потрібні, а прикладом слугує модальне вікно, яке завантажується лише після того, як хтось натисне кнопку, щоб його відкрити. `next/dynamic` поєднує `lazy()` і `Suspense` з React, а звичайний виклик `import()` робить те саме для бібліотеки.

Сам термін старший за все це. У каталозі шаблонів корпоративних застосунків Мартіна Фаулера Lazy Load залишає позначку там, де мали б бути пов’язані дані, і завантажує їх лише тоді, коли щось ними скористається. Тож читання одного об’єкта з бази даних не тягне за собою велику мережу інших об’єктів, на які ніхто й не подивиться. ORM працює саме так, коли завантажує пов’язаний запис у мить, коли код уперше до нього звертається. Django робить так за замовчуванням: поле, якого не було в початковому запиті, читається з бази даних під час першого звернення, і документація описує це, не вживаючи самої назви.

Ліниве завантаження — це не кешування, яке зберігає копію вже отриманого, щоб скористатися нею знову. Воно протилежне попередньому завантаженню: `<link rel="preload">` просить браузер отримати ресурс заздалегідь, бо сторінці він, найпевніше, знадобиться.

Застосуйте поняття до рішення

Застосуйте «Що таке ліниве завантаження?» 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

Сторінка платить за кожен байт, який запитує, незалежно від того, чи хтось його побачить. Довга стаття з сорока фотографіями, футер із вбудованою мапою, сторінка товару з відео в самому низу — усі вони витрачають трафік, заряд батареї й час з'єднання на вміст, до якого частина читачів так і не догортає. З лінивим завантаженням ці запити надсилаються лише для тих, хто дійшов так далеко, а те, що потрібне першому екрану, не мусить ділити з ними з'єднання.

Наскільки рано треба почати, вирішує браузер. Стандарт радить, щоб запас ураховував швидкість прокручування та якість мережі, а на повільних з'єднаннях і з'єднаннях із високою затримкою був ширшим, тож зображення вже на місці, коли до них догортають. Він також вимагає, щоб цей запас не розкривав відомостей, за якими читача можна було б розпізнати за цифровим відбитком браузера.

Є й вбудоване правило приватності. Браузери застосовують ліниве завантаження лише тоді, коли скрипти ввімкнено. Інакше, пояснює стандарт, сайт міг би розкидати зображення по своїй розмітці й приблизно визначати, як далеко прогорнув кожен читач, за тим, які зображення і коли запитують у сервера. Коли скрипти вимкнено, `loading="lazy"` непомітно стає звичайним завантаженням.

Для коду Next.js описує виграш через перше завантаження: менше JavaScript потрібно, перш ніж маршрут можна показати. Бібліотеці діаграм, що використовується на одній вкладці, чи текстовому редактору, схованому за кнопкою редагування, нема чого сповільнювати решту сторінок.

В ORM виграш полягає в тому, що так і не довелося прочитати. Фаулер наголошує: завантажувати об'єкт разом з усім, що з ним пов'язано, зручно, доки це не означає завантаження величезної кількості об'єктів, коли потрібно лише кілька, і Lazy Load розриває цей ланцюжок.

Як це виглядає на практиці

Для зображень і фреймів усю роботу виконує атрибут. Зображення й мапа нижче розташовані далеко внизу сторінки, тож обидва можуть зачекати:

```
<img
  src="/photos/harbour-800.jpg"
  srcset="/photos/harbour-800.jpg 800w, /photos/harbour-1600.jpg 1600w"
  sizes="auto, 100vw"
  width="1600"
  height="900"
  alt="Рибальські човни в гавані на світанку"
  loading="lazy"
/>

<iframe
  src="/embeds/route-map.html"
  title="Мапа пішого маршруту"
  width="640"
  height="360"
  loading="lazy"
></iframe>
```

`width` і `height` задають пропорції зображення ще до того, як прийде хоч щось із нього: правила відображення HTML перетворюють ці два числа на співвідношення сторін, тож браузер може заздалегідь залишити рамку потрібної форми. `sizes="auto"` дозволено лише для зображення з `lazy`. Це значення просить браузер вибирати з `srcset` за шириною, яку зображення справді отримує в макеті, а для лінивого зображення ця ширина вже відома, коли настає час його завантажувати. `100vw` після нього — запасний варіант на випадок, якщо ширину визначити не вдасться.

Інколи для лінивого завантаження потрібен скрипт. Тут бібліотеку мапи завантажують лише тоді, коли її контейнер от-от з’явиться на екрані:

```
const box = document.querySelector("#route-map");

const observer = new IntersectionObserver(
  async (entries) => {
    if (!entries.some((entry) => entry.isIntersecting)) return;
    observer.disconnect();
    const { drawMap } = await import("./route-map.js");
    drawMap(box);
  },
  { rootMargin: "200px 0px" },
);

observer.observe(box);
```

`rootMargin` розширює viewport на 200 пікселів згори й знизу для цієї перевірки, тож модуль починає завантажуватися трохи раніше, ніж блок стане видимим. Колбек виконується асинхронно й може повідомити про кілька елементів одразу, а `isIntersecting` перевіряє тому, що спостерігач повідомляє й про елементи поза областю. Оскільки `disconnect()` викликається першим, колбек виконує свою роботу один раз, скільки б разів читач не прогортав повз. До того `route-map.js` взагалі не є частиною сторінки.

Шрифти можна розбити на частини, щоб браузер завантажував лише ті, які використовує сторінка, і скрипт для цього не потрібен. Нотатка на цьому сайті про ваду підкреслення в Safari фіксує, як тоді шрифт сайту Ubuntu подавався шістьма файлами: три накреслення для латинського тексту й три для кириличного, і кожен файл указує діапазон символів, які охоплює. Браузер вибирає латинські накреслення для тексту латинкою, а кириличні — для тексту кирилицею. Тому назва мови, написана кирилицею, може потребувати кириличного накреслення навіть на сторінці, решта якої написана англійською чи данською. Нотатка не називає це лінивим завантаженням, але той самий вибір за потребою не додає невикористовуване накреслення до запитів сторінки на шрифти, окрім випадків, коли його явно завантажують заздалегідь через preload.

Файли, які кожній сторінці потрібні одразу, вимагають протилежного рішення. Запис у портфоліо про скорочення пакета стилів пояснює, чому з таблиці стилів прибрали шрифт, вбудований туди як base64. Його вставили, щоб пришвидшити перше відображення сторінки, але окремий файл із preload дає ту саму перевагу, тоді як вбудована копія додавала свою вагу до кожної сторінки — потрібна вона там була чи ні. На тому етапі натомість три накреслення шрифту завантажувалися заздалегідь через preload, і в записі сказано, що третє потрапило до цього переліку завдяки даним реальних відвідувань, а не прагненню мати повний комплект. Ліниве завантаження й попереднє — два боки одного судження: те, що потрібне першому екрану, приходить рано, а решта чекає своєї черги.

Де виникають проблеми

Перше зображення приходить останнім. Зображення з `lazy` чекає не лише на мережу. Його завантаження відновлюється тільки тоді, коли спостерігач браузера повідомить, що воно поблизу viewport, а спостерігачі оновлюються під час підготовки кожного кадру, уже після перерахунку стилів і макета. Для фотографії далеко внизу сторінки це очікування нічого не

коштує. Для великого зображення вгорі воно затримує саме ту картинку, яку бачить кожен відвідувач. Не ставте `loading` взагалі або ставте `eager` для всього, що може опинитися на першому екрані телефона чи комп'ютера, а `lazy` залишайте для того, що нижче.

Сторінка стрибає, коли приходять зображення. Зображення без `width` і `height` не має відомої форми, доки не прийде його файл, а тоді зсуває вниз усе, що під ним. З лінивим завантаженням це стається саме тоді, коли читач гортає, часто посеред речення. Задавайте розміри кожному зображенню в розмітці або в CSS. Щодо `sizes="auto"` стандарт HTML висловлюється прямо: без указаних розмірів таке зображення, найімовірніше, буде намальоване розміром 300 на 150 пікселів, тому стандарт наполегливо радить їх задавати.

Зображення, яким для існування потрібен скрипт. Саморобне рішення кладе справжню адресу в атрибут `data-src` і доручає скрипту скопіювати її в `src`, коли зображення наближається. Якщо скрипт не завантажився, видав помилку або був заблокований, зображення так і не з'явиться, а все, що читає розмітку без виконання скриптів, бачить зображення без джерела. Вбудований атрибут ламається в протилежний бік: коли скрипти вимкнено, стандарт велить браузеру завантажити зображення одразу. Використовуйте `loading="lazy"` для зображень і фреймів, а скрипти залишайте для того, чого атрибут не покриває.

Відкладати те, що потрібне кожному відвідуванню. Ліниве завантаження окупається, коли значна частина відкладеного так і не знадобиться. Коли знадобиться все, воно лише розбиває одне отримання на багато. Цикл, що читає пов'язаний запис для кожного елемента списку, надсилає окремий запит на кожен: документація Django в режимі за замовчуванням рахує один запит на список плюс по одному на елемент і називає це проблемою N+1 запитів. Сторінка, де кожен компонент запитує власні дані, щойно з'являється, робить те саме через мережу. Завантажуйте те, що точно знадобиться, разом зі списком, одним запитом до бази даних чи одним зверненням через мережу, а ліниве завантаження залишайте для того, що, найімовірніше, пропустять.

Що почитати далі

- HTML Standard: Lazy loading attributes — ключові слова `lazy` і `eager`, власний intersection observer браузера та правило, що вимикає ліниве завантаження без скриптів.

- Intersection Observer — робоча чернетка W3C, на якій тримається спостерігач: корені, відступи, пороги й те, як колбеки стають у чергу.
- How to lazy load Client Components and libraries — `next/dynamic`, `React.lazy()` із `Suspense`, відмова від рендерингу на сервері й завантаження бібліотеки через `import()`.
- Lazy Load — шаблон Мартіна Фаулера з Patterns of Enterprise Application Architecture і чотири його різновиди.
- Fetch modes — як Django отримує поля, яких не було в початковому запиті, і режими, що групують ці звернення або блокують їх.

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 ліниве завантаження?» 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/lazy-loading/