Engineer

Кешування

⏭ збереження відповіді, щоб не повторювати роботу

Що це таке

Кешування — це практика зберігати результат певної роботи, як-от HTTP-відповідь, результат запиту чи відрендерену сторінку, і віддавати його наступним запитам за правилами, які визначають, скільки йому можна довіряти й що виводить його з ужитку. Дієслово — кешувати, а збережений у такий спосіб результат називають кешованим. Коли кешуванням займається одна функція всередині програми, запам'ятовуючи власні повернені значення, це зазвичай називають мемоізацією.

Сховище, що тримає копії, — це кеш. Кешування — це політика навколо нього, і в цієї політики три питання. Скільки часу копія свіжа, тобто придатна, щоб її віддавати, нікого не питаючи? Коли вона застаріла, як її перевіряють? І коли оригінал змінюється, що прибирає або замінює копію?

HTTP відповідає на всі три в заголовках. RFC 9111 називає збережену відповідь свіжою, поки її вік менший за термін свіжості, який джерело задає директивою `max-age` у `Cache-Control` або датою `Expires`. Після цього відповідь застаріла, і кеш зазвичай мусить перевірити її, перш ніж віддати знову. Він надсилає умовний запит із валідатором, який зберіг, найчастіше з `ETag` відповіді в заголовку `If-None-Match`, і якщо нічого не змінилося, джерело відповідає 304 Not Modified без тіла. Перевірка все одно коштує одного обміну запитом і відповіддю, але заощаджує передавання й роботу зі збирання тіла заново.

```
HTTP/1.1 200 OK
Cache-Control: max-age=3600, stale-while-revalidate=60
ETag: "v42"
```

Усередині застосунку ті самі питання набувають інших форм. За схемою cache-aside код спершу читає кеш, а в разі промаху читає базу даних і сам заповнює кеш. За схемою write-through кожен запис оновлює кеш разом із базою, тож читання знаходить поточне значення. Термін дії кожного запису, його time to live, або TTL, грає роль `max-age`. Кешувати вміє навіть база даних: матеріалізоване подання в PostgreSQL зберігає результат запиту як таблицю, і цей результат змінюється лише тоді, коли хтось виконає `REFRESH MATERIALIZED VIEW`.

Застосуйте поняття до рішення

Застосуйте «Що таке кешування?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Пошук запитів і відсутніх індексів у системі, що рік за роком важчає, та виправлення тих, що справді мають значення.

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. 30 днів даних про електроенергію за 180 мс — 180 ms на агрегацію місяця показань лічильників

30 днів даних про електроенергію за 180 мс Зв'язатися з нами →

Чому це важливо

Кожне правило кешування обмінює свіжість на швидкість, і правильний обмін залежить від типу даних. Логотип може бути річної давності. Ціна товару може бути хвилинної давності. Баланс рахунку чи кількість вільних місць варто підтвердити в джерелі, перш ніж хтось ухвалить на їхній підставі рішення. Найчастіше кешування підводить тоді, коли одне правило застосовують до всіх трьох.

Для статичних файлів відповідь найчистіша. Коли під час збирання в назву кожного файлу вписують хеш його вмісту, змінений файл отримує нову URL-адресу, тож стару можна кешувати рік. RFC 8246 додає директиву `immutable`, яка повідомляє браузеру, що відповідь лишатиметься тією самою, поки вона свіжа, тож навіть перезавантаження сторінки може обійтися без повторної перевірки. HTML-сторінка, що називає ці файли, мусить жити недовго або перевірятися щоразу, інакше відвідувачі й далі завантажуватимуть старі назви.

Дві директиви легко сплутати. `no-cache` дозволяє кешу зберегти відповідь, але забороняє віддавати її знову, не звірившись спершу з джерелом; `no-store` забороняє зберігати її взагалі. Для сторінки, що часто змінюється, `no-cache`

разом з `ETag` дає актуальний вміст і все одно заощаджує передавання, коли нічого не змінилося.

Інвалідація — складніша половина. RFC 9111 вимагає, щоб кеш робив недійсними збережені відповіді для URL, коли небезпечний запит до нього, як-от `POST`, `PUT` чи `DELETE`, завершується успішно, і той самий розділ застерігає, що це сягає лише кешів, крізь які проходить цей запит. Браузер, CDN і кеш застосунку — три окремі місця, і якщо прибрати копію з одного, у двох інших вона лишається. Тому більшість систем спираються на закінчення терміну дії як на запобіжник, а явну інвалідацію використовують, щоб ставати актуальними швидше.

Застарілі дані теж бувають корисними. RFC 5861 визначає `stale-while-revalidate`, що дозволяє кешу віддавати застарілу відповідь протягом указаної кількості секунд, поки він у фоні отримує свіжу, і `stale-if-error`, що дозволяє й далі віддавати застарілу копію, коли джерело відповідає помилкою. Обидві директиви перетворюють жорстке закінчення терміну на м’яке: відвідувач одразу отримує відповідь, а оновлення надходить слідом.

Як це виглядає на практиці

На клієнтському кінці системи правила кешування задає платформа, значна частина користувачів якої перебуває в морі: моряки та польовий персонал, що працюють із телефонів і часто офлайн. Там кешування дало змогу встановити вебзастосунок і працювати з ним у морі. Service worker налаштовано через next-pwa, а runtime-кешування Workbox віддає шрифти, зображення, аудіо, відео та CDN-ресурси за стратегією cache first: із пристрою, коли там є копія, і з мережі лише в разі промаху. Це файли, які лишаються однаковими від одного запиту до наступного, і сторінка доповнює їх HTTP-заголовками cache-control. Оскільки service workers потребують HTTPS, встановлення й офлайн-роботу перевіряли через HTTPS на реальних пристроях. Результат, про який повідомляє сторінка, — застосунок, що встановлюється, працює офлайн і завантажує повторні відвідування з кешу, а не з мережі. Правило cache first повністю пропускає перевірку свіжості, поки запис існує, і саме тому пасує файлам, чий вміст прив’язаний до їхньої адреси.

На протилежному кінці системи стоїть база даних PostgreSQL, яка стала в 10 разів швидшою і зберігає облікові записи, орендарів, географічні дані та щоденні показники споживання електроенергії. Кешування було одним

заходом із кількох. Поряд із цільовими індексами, GiST для просторових запитів і композитними для фільтрів за орендарем і датою, та партиціонуванням таблиці електроенергії за часовими діапазонами деякі результати запитів тримали як матеріалізовані подання, а кешування запитів і пул з'єднань налаштували, щоб зменшити накладні витрати. Сторінка повідомляє, що час відповіді покращився в 10 разів, критичні операції виконуються за мілісекунди, а використання ресурсів знизилося на 40%. Цей результат вона приписує роботі загалом, тож внесок кешування окремо не виміряно.

Де виникають проблеми

Після розгортання частина відвідувачів досі отримує стару версію. Відповідь прийшла без явної свіжості, тож кеші оцінили її самі. RFC 9111 дозволяє їм брати частку часу, що минув від `Last-Modified`, і називає 10% типовим значенням, тож файл, востаннє змінений місяць тому, можна повторно віддавати близько трьох днів, не звертаючись до джерела. Надсилайте явний `Cache-Control` із кожною відповіддю, додавайте хеш вмісту до назв статичних файлів і тримайте HTML, що на них посилається, короткоживучим.

Очищення спорожнює CDN, а браузери й далі тримають стару копію. Очищення (purge) прибирає копію з CDN і більше нічого; кожен браузер, що зберіг відповідь, тримає її, доки не мине її власний термін. Давайте публічним відповідям короткий `max-age` для браузерів і довший `s-maxage`, який застосовують лише спільні кеші, щоб найдовше жила саме та копія, яку можна очистити. Коли копія в браузері мусить змінитися негайно, змініть URL.

Сторінка, якої ненадовго не було, так і лишається відсутньою. RFC 9110 робить відповідь 404 евристично кешованою, тож кеш може зберегти таку відповідь, що прийшла без даних про свіжість, на термін, який оцінить сам. Посилання, за яким перейшли за мить до публікації сторінки, може й далі повертати 404, коли сторінка вже існує. Задавайте відповідям із помилками явний короткий термін і очищайте URL, щойно сторінка з'являється.

Старе значення записується назад після оновлення. За схемою cache-aside читач промахується й читає рядок; перш ніж він збереже значення, інший процес оновлює рядок і видаляє запис у кеші; тоді читач зберігає прочитане, і старе значення лишається, доки щось його не прибере. Видаляйте запис

після коміту транзакції й задавайте кожному запису TTL, щоб запис, видалення якого запізнилося чи загубилося, зник сам після закінчення терміну. У Redis звичайний `SET` відкидає будь-який попередній термін дії, тож задавайте TTL у тій самій команді:

```
SET price:42 "19.90" EX 300
```

Що почитати далі

- RFC 9111, Section 3: Storing Responses in Caches — коли відповідь можна зберегти, а далі правила свіжості, перевірки й інвалідації.
- RFC 9110, Section 13: Conditional Requests — `ETag`, `If-None-Match` і відповідь 304, завдяки яким повторна перевірка коштує мало.
- RFC 5861: HTTP Cache-Control Extensions for Stale Content — дві директиви, що дозволяють кешу свідомо віддавати застарілі відповіді.
- RFC 8246: HTTP Immutable Responses — директива `immutable` для файлів, чия URL-адреса змінюється разом із вмістом.
- Workbox: caching strategies — cache first, network first і stale-while-revalidate у service worker, із прикладами коду.
- PostgreSQL: Materialized Views — як тримати результат запиту всередині бази даних і оновлювати його на вимогу.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке кешування?» 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/caching/