**Engineer**

# Кеш

збережена копія, яку браузер може взяти замість запиту

## Що це таке

Кеш — це сховище, що тримає копії даних близько до тих, хто їх читає, щоб повторне читання отримувало відповідь із копії, а не з повільнішого місця, де дані насправді зберігаються. У множині це кеші; один збережений елемент називають записом кешу, а дані в ньому — кешованою копією.

Кеш ніколи не є джерелом істини. Файл, рядок чи відповідь сервера, які він копіює, лишаються авторитетними, а кожен запис можна викинути й отримати знову. Цим кеш відрізняється від репліки, яку реплікація навмисно тримає повною й синхронною, і від буфера, що тримає дані в дорозі, які, можливо, існують лише в ньому. Page cache у Linux — і те, й інше: він зберігає прочитані дані файлів, а також першим отримує записи як «брудні» сторінки, доки ті не потраплять на диск.

Кеші вишиковуються шарами між користувачами й даними. Page cache ядра лежить під кожним файлом, який читає програма. PostgreSQL тримає над ним власний пул сторінок, `shared_buffers`. Браузер зберігає HTTP-відповіді в приватному кеші для однієї людини; CDN чи зворотний проксі тримає спільний кеш для багатьох. Скрипт сторінки може зберігати відповіді в Cache Storage через service worker, окремо від HTTP-кешу браузера. А застосунок може тримати обчислені значення у сховищі ключ-значення, як-от Redis.

Словник однаковий на кожному шарі. Читання, яке знаходить свій запис, — це влучання (hit); інакше це промах (miss), і запит іде далі, до джерела. Частка влучань серед усіх читань — коефіцієнт влучань (hit ratio). Кожен запис шукають за ключем, який для HTTP-кешу складається щонайменше з методу запиту та URL. Заповнене сховище викидає записи, щоб звільнити місце, — це витіснення (eviction), а політика витіснення вирішує, що саме піде; найпоширеніша — least recently used, або LRU.

Застосуйте поняття до рішення

Застосуйте «Що таке кеш?» 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

Кеш зазвичай найдешевша швидкість, яку може купити система, без нового заліза й без нової моделі даних. Рахунок приходить пам’яттю, копіями в місцях, до яких важко дістатися, і копіями, що перестали бути правдивими. Більшість цього визначають три рішення щодо самого сховища.

Перше — де воно стоїть. Що ближче копія до читача, то більше вона заощаджує: копія в браузері оминає і мережу, і сервер. Водночас вона далі від власника. Екземпляр Redis можна спорожнити однією командою; копія на телефоні відвідувача лишається там за тими правилами, які з нею надіслали.

Друге — розмір, бо пам’ять, віддана кешу, забрана в чогось іншого. Документація PostgreSQL називає типове значення `shared_buffers` — зазвичай 128 МБ. Для виділеного сервера з 1 ГБ пам’яті або більше вона радить почати з 25%, а оскільки PostgreSQL покладається ще й на кеш операційної системи, понад 40%, за її словами, навряд чи працюватиме краще за менший обсяг. Два кеші ділять одну машину.

Третє — хто ділить сховище. RFC 9111 розрізняє приватний кеш для одного користувача та спільний кеш, що зберігає відповіді для повторного використання кількома. Особиста відповідь у спільному сховищі — це шлях, яким сторінка однієї людини потрапляє до іншої, тому спільний кеш не має права повторно віддавати відповідь на запит із заголовком `Authorization`, якщо сама відповідь цього явно не дозволяє.

З усіх трьох випливає одне правило: кеш має бути безпечно спорожнити. Якщо при цьому губляться дані, кеш став базою даних без гарантій бази даних. Скільки запис лишається досить свіжим, щоб його віддавати, і що

прибирає його раніше, — це питання кешування, тобто практики; власне завдання сховища — тримати копії, знаходити їх за ключем і звільняти місце, коли його бракує.

## Як це виглядає на практиці

Три сторінки цього сайту зустрічаються з кешем на трьох різних шарах.

Перший — page cache ядра. Хост на 464 МБ, де працюють git-сервер і вебсервер, має dead man's switch, що б'є на сполох мовчанням: що п'ятнадцять хвилин, із випадковим зсувом, скрипт вимірює доступну пам'ять і вільне місце на диску та пінгує зовнішній сервіс лише тоді, коли обидва показники вищі за свої пороги. Доступна пам'ять — правильна міра, бо на справному Linux-хості ядро пускає незайняту пам'ять під кеш, і вільної лишається майже нуль. Скрипт читав правильний стовпчик, але шість імен змінних навколо нього казали «free». Хто повірив би цим іменам і звірив їх із порогом у десять відсотків, побачив би дванадцять мегабайтів вільних на 464 МБ, вирішив би, що перевірка зламана, і змінив би в ній один символ. Вийшла б сигналізація, що постійно за порогом і яку протягом тижня заглушили б. Імена відтоді виправлено.

Другий — кеш застосунку перед базою даних. Нотатка про оптимізацію бази перед шардуванням ставить його перед товаром, який от-от розкуплять. Коли всі одиниці зарезервовано, база може сказати, коли спливе найраніший резерв — через десять хвилин після того, як його зробили, — і нотатка кешує «повністю зарезервовано» саме з таким терміном. До того часу відвідувачам відмовляє кеш, а не база. Резерв, звільнений раніше через відхилену картку чи покинутий кошик, видаляє запис, тож одиниця одразу знову в продажу. Кеш може відмовити; продає лише база, через умовний `UPDATE`. Найгірше, що може зробити застарілий запис, — втратити продаж на кілька секунд; подвійний продаж тієї самої одиниці виключено.

Третій — пристрій самого відвідувача, де цей сайт не створює керованого скриптами сховища. Його сторінка про файли cookie каже, що він не використовує ані IndexedDB, ані Cache Storage і не встановлює service worker. Браузер усе одно може зберігати відповіді у своєму звичайному HTTP-кеші, окремому від офлайн-копії, якою керує сайт.

# Де виникають проблеми

**Після перезапуску джерело отримує всі читання одночасно.** Холодний кеш промахується на всьому, тож деплой, перезапуск кешу чи failover спрямовують усе навантаження читання на базу даних. Система, яка трималася лише завдяки тому, що більшість читань були влучаннями, може впасти, перш ніж кеш знову наповниться. Розраховуйте джерело на шлях промахів, прогрівайте кеш до того, як він прийме трафік, і повертайте трафік поетапно. Розширення `pg_prewarm` у PostgreSQL робить це для буферного кешу й уміє записати, що містили shared buffers, та прочитати це знову після перезапуску.

**Кеш росте, доки машині не забракне пам'яті.** На 64-бітних системах Redis типово не обмежує пам'ять, тож екземпляр, який слугує кешем, росте разом зі своїм простором ключів, доки не втрутиться операційна система. Задавайте ліміт і політику разом. За `noeviction` заповнений екземпляр відповідає помилкою на команди, які кешували б нові дані: це пасує сховищу, що мусить зберегти все, і дивує те, що мало б забувати. Документація Redis називає `allkeys-lru` добрим типовим вибором, якщо немає причин віддати перевагу іншій політиці:

```
maxmemory 256mb
maxmemory-policy allkeys-lru
```

**Відвідувач отримує сторінку не тією мовою.** Ключ пропускає вхідне значення, від якого залежить результат. Сторінку, відрендерену данською, зберігають лише під її URL, і наступний читач отримує данську, хоч би що просив. Будуйте ключ з усіх вхідних даних, що змінюють результат, як-от мова, валюта чи роль. Для HTTP-кешів заголовок відповіді `Vary` називає заголовки запиту, які входять до ключа, а RFC 9111 дозволяє кешу повторно віддати збережену відповідь, лише коли ці заголовки збігаються, або після її повторної перевірки.

**Встановлений застосунок і далі виконує старий код.** Записи в Cache Storage не мають власного терміну дії й переживають оновлення скрипту service worker. Змінені файли під тією самою назвою кешу й надалі віддаються зі старих копій, а кеш, який перейменовують на кожному випуску без прибирання, щоразу лишає на пристрої ще один. Додавайте до назви кожного кешу версію, як радить специфікація Service Workers, і видаляйте всі старші, щойно новий worker бере справу на себе.

# Що почитати далі

- RFC 9111: HTTP Caching — стандарт IETF про приватні й спільні HTTP-кеші, їхні ключі і про те, що спільному кешу чіпати заборонено.
- Linux kernel: memory management concepts — як page cache тримає дані файлів у пам’яті й чому ці сторінки ядро може звільнити за потреби.
- PostgreSQL: Resource Consumption — `shared_buffers`, його типове значення й поради щодо розміру з урахуванням кешу операційної системи.
- Service Workers — специфікація W3C про Cache Storage і про те, чим він відрізняється від HTTP-кешу браузера.
- Redis: Key eviction — `maxmemory` і те, що робить заповнений кеш за кожної політики витіснення.

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 кеш?» 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/cache/**