Engineer

Ідемпотентний

безпечно запускати двічі, бо другий раз нічого не змінює

Що це таке

Операція ідемпотентна, коли кілька її виконань мають той самий ефект, що й одне, — так кажуть і про ідемпотентний запит чи ідемпотентне завдання, а саму властивість називають ідемпотентністю, англійською idempotence або idempotency. Слово походить із математики, де функцію називають ідемпотентною, якщо застосування до власного результату лишає його таким самим, тобто f(f(x)) дорівнює f(x): округлення вниз числа, яке вже округлили вниз, дає те саме число.

У програмах корисна перевірка — ще один запуск. Присвоєння значення ідемпотентне, бо другий запуск записує те значення, яке там уже є. Додавання до значення таким не є, бо другий запуск додає знову:

```
-- ідемпотентно: другий запуск задає те саме значення
UPDATE accounts SET plan = 'pro' WHERE id = 42;

-- не ідемпотентно: другий запуск додає ще десять
UPDATE accounts SET credits = credits + 10 WHERE id = 42;
```

HTTP дає цьому слову найточніше визначення. RFC 9110 називає метод запиту ідемпотентним, якщо запланований ефект на сервері від кількох однакових запитів такий самий, як від одного, і відносить до таких PUT, DELETE та безпечні методи GET, HEAD, OPTIONS і TRACE. У цьому визначенні важливі дві деталі. Властивість стосується ефекту, якого просив клієнт, тож сервер може записувати кожен запит у журнал окремо. І відповідь може відрізнятися: другий DELETE того самого ресурсу може отримати 404 там, де перший отримав 200, і метод однаково лишається ідемпотентним, бо ресурс зник в обох випадках. PATCH стоїть поза цим переліком. RFC 5789 каже, що він не є ні безпечним, ні ідемпотентним, але клієнт може надіслати

конкретний PATCH ідемпотентно, зробивши його умовним за допомогою сильного ETag у заголовку If-Match, тож повторну копію буде відхилено, щойно перша змінить ресурс.

Безпечність — суворіша з двох обіцянок. Безпечний метод у розумінні RFC 9110 по суті лише читає, тож кожен безпечний метод ідемпотентний, тоді як DELETE ідемпотентний, не будучи безпечним.

Застосуйте поняття до рішення

Застосуйте «Що означає ідемпотентний?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Цифри з'являються там, де потрібні, за розкладом, і ніхто не копіює таблицю наприкінці місяця.

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. Один інфраструктурний план для 14 підрозділів — 14 дослідницьких підрозділів, охоплених спільним інфраструктурним планом

Один інфраструктурний план для 14 підрозділів Зв'язатися з нами →

Чому це важливо

Мережі гублять відповіді. Коли клієнт надсилає запит, а з’єднання обривається до того, як надійде відповідь, клієнт не може знати, чи сервер так і не побачив запиту, чи побачив і зупинився, чи виконав його й загубив відповідь. Повторити запит — очевидний вихід, і саме ідемпотентність робить повтор нешкідливим. RFC 9110 будує на цьому свої правила повторних спроб. Ідемпотентний запит можна автоматично повторити після збою зв’язку. Неідемпотентний запит клієнту слід повторювати автоматично лише тоді, коли він знає, що на практиці цей запит ідемпотентний, або може встановити, що оригінал так і не було застосовано, а проксі не має права повторювати такий запит сам.

Черги ставлять те саме питання в масштабі. Багато систем повідомлень обіцяють доставку щонайменше один раз: споживач, який упав після того, як виконав роботу, але до того, як підтвердив повідомлення, отримає те саме повідомлення знову. Доставка рівно один раз через мережу — значно важча обіцянка, тож практична відповідь — споживач, для якого повторна доставка нічого не змінює. Керування конфігурацією теж залежить від цієї властивості. Інструмент, що приводить сервери до задекларованого стану, запускається знову і знову, після кожної зміни й часто за розкладом, і кожен запуск має залишити правильний сервер точно таким, яким його застав.

Справжнє рішення — як отримати ідемпотентність там, де операція за своєю природою її не має, і поширених відповідей три. Перша — проєктування: описати зміну як цільовий стан, а не як крок, тобто «переконайся, що цей рядок є у файлі» замість «додай цей рядок». Друга — ключ: клієнт додає до запиту унікальний ідентифікатор, сервер зберігає його під обмеженням унікальності, і повтор знаходить запис та отримує початковий результат замість другої дії. Третя — перевіряти, перш ніж діяти, як це роблять більшість модулів Ansible: вони порівнюють поточний стан із бажаним і завершуються без дій, коли ці два стани вже збігаються. Кожна відповідь має ціну. Проєктування через цільовий стан потребує чіткої моделі кінцевого стану, ключі потребують сховища й правила, як довго їх тримати, а перевірка наперед потребує надійного способу прочитати поточний стан.

Як це виглядає на практиці

Власні сервери компанії налаштовано через Ansible, і в цьому проєкті ідемпотентність є контрактом: хост у бажаному стані звітує про нуль змін. Запуск автоматизації без жодних аргументів приводить кожен хост до його задекларованого стану, а сценарій Ansible, який не може звітувати про нуль змін на хості, що вже налаштований правильно, вважають незавершеним. Так ідемпотентність стає тим, що вимірює кожен запуск, а не тим, на що сподіваються.

Та сама ідея, застосована до даних, стоїть за тестом, що двічі запускає дев'ятнадцятикроковий конвеєр збірки. Сторінка називає різновид вади, заради якого він існує: генератор, що працює першого разу й псує другого, через кроки, які вставляють без перевірки або залежать від порядку виводу попереднього кроку. Тест збирає базу даних із порожнього файлу, робить знімок, ще раз запускає весь конвеєр поверх результату й вимагає, щоб другий прохід нічого не змінив, аж до кількості рядків, вмісту та ідентифікаторів. Тест дає свідчення про повторний запуск для тих вхідних даних і стану бази, які він перевіряє.

Для продажу обмеженого запасу нотатка про продаж останнього квитка рівно один раз поєднує ключ з умовними записами. Кожне оформлення замовлення несе ключ ідемпотентності, а таблиця резервів має `UNIQUE (tenant_id, idempotency_key)`, тож повторне натискання отримує резерв, який уже має, а не ще одну одиницю. Звільнення, що спрацювало двічі, наприклад від тайм-ауту і від кнопки скасування, повертає одиницю один раз, бо повертає запас лише тоді, коли його власне оновлення змінило рівно один

рядок. Далі кожен споживач повідомлень ідемпотентний, з ідентифікатором замовлення як ключем: повідомлення приходять щонайменше один раз, а завдяки ідемпотентності кожне фактично обробляється один раз.

Де виникають проблеми

Повторна спроба списує гроші двічі. POST, що створює платіж, замовлення чи лист, сам собою не ідемпотентний, і клієнт або проксі, що повторює його після тайм-ауту, створює ще один. Варто дати таким запитам ключ ідемпотентності, який сервер зберігає під обмеженням унікальності, і повертати збережений результат, коли той самий ключ надходить знову.

Порожній результат сприймають як помилку. У PostgreSQL `INSERT ... ON CONFLICT DO NOTHING` пропускає рядок, який порушив би обмеження унікальності, а `RETURNING` повертає лише ті рядки, які справді було вставлено чи оновлено. Тому повторна вставка того самого рядка повертає порожній результат, і це правильний наслідок другого запуску. Код, що вважає такий результат збоєм, перетворює нешкідливу повторну спробу на сторінку з помилкою; натомість він має знайти рядок, який уже існує.

Командне завдання змінює хост за кожного запуску. Документація Ansible каже, що більшість модулів перевіряють, чи бажаного стану вже досягнуто, а також що так поводяться не всі playbook і не всі модулі. Завдання, що виконує команду, — типовий виняток, бо команда довільна, і Ansible не може знати, що вона робить. Параметр `creates` модуля command пропускає крок, коли вказаний файл уже існує, `removes` виконує його лише доти, доки файл ще є, а `changed_when` визначає, який результат вважати зміною. Командне завдання без жодного з них виконується знову під час кожного застосування конфігурації, хоч би в якому стані був хост.

Лічильник рахує й повторну спробу. За доставки щонайменше один раз споживач, що збільшує лічильник чи надсилає сповіщення на кожне повідомлення, зробить це двічі для повідомлення, доставленого двічі. Слід зберігати бізнес-ключ повідомлення разом з ефектом в одній транзакції й пропускати кожен ключ, який уже збережено.

Другий запуск псує те, що збудував перший. Крок може бути правильним на порожній базі й хибним на заповненій, бо вставляє дублікат, перенумеровує ідентифікатори чи покладається на порядок, який залишив

перший запуск. Тест, що починає з нуля й запускається один раз, однаково пройде. Варто запускати всю задачу двічі й вимагати, щоб другий запуск нічого не змінив, як це робить тест конвеєра вище.

Що почитати далі

- RFC 9110, section 9.2.2: Idempotent Methods — визначення HTTP, ідемпотентні методи й правила автоматичних повторних спроб.
- RFC 5789: PATCH Method for HTTP — чому PATCH не є ні безпечним, ні ідемпотентним і як умовний запит робить його безпечним для повторення.
- Ansible: Introduction to playbooks — бажаний стан, ідемпотентність і режим перевірки словами самого Ansible.
- Ansible: command module — параметри `creates` і `removes`, що роблять команду безпечною для повторного запуску.
- PostgreSQL: INSERT — `ON CONFLICT` і те, що повертає `RETURNING`, коли рядок уже існує.

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 означає ідемпотентний?» 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/idempotent/