Engineer

Idempotente

Varias ejecuciones tienen el mismo efecto previsto que una

Qué es

Una operación es idempotente cuando ejecutarla varias veces tiene el mismo efecto que hacerlo una sola vez; esa propiedad se llama idempotencia. El término procede de las matemáticas: una función es idempotente si aplicarla a su propio resultado no cambia nada, de modo que f(f(x)) es igual a f(x). Redondear hacia abajo un número ya redondeado hacia abajo da el mismo número.

En software, la prueba útil es una segunda ejecución. Asignar un valor es idempotente, porque la segunda ejecución escribe el valor que ya existe. Sumar a un valor no lo es, porque vuelve a sumar:

```
-- idempotent: a second run sets the same value
UPDATE accounts SET plan = 'pro' WHERE id = 42;

-- not idempotent: a second run adds ten more
UPDATE accounts SET credits = credits + 10 WHERE id = 42;
```

HTTP da al término su definición más precisa. RFC 9110 llama idempotente a un método de petición cuando varias peticiones idénticas están concebidas para producir el mismo efecto en el servidor que una sola, y enumera PUT, DELETE y los métodos seguros GET, HEAD, OPTIONS y TRACE. Dos detalles importan. La propiedad cubre el efecto solicitado por el cliente, por lo que el servidor puede registrar cada petición por separado. Además, la respuesta puede diferir: un segundo DELETE del mismo recurso puede responder 404 donde el primero respondió 200 y seguir siendo idempotente, porque el recurso ya no existe en ambos casos. PATCH queda fuera de la lista. RFC 5789 indica que no es seguro ni idempotente, aunque el cliente puede enviar un PATCH concreto de forma

idempotente haciéndolo condicional, con un ETag fuerte en una cabecera If-Match, de modo que se rechace una copia repetida después de que la primera haya cambiado el recurso.

Ser seguro es la más estricta de las dos promesas. Un método seguro en el sentido de RFC 9110 es esencialmente de solo lectura: todo método seguro es idempotente, mientras que DELETE es idempotente sin ser seguro.

Utilice el concepto en una decisión

Aplique «¿Qué significa idempotente?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Las cifras llegan donde hacen falta según un calendario, sin que nadie tenga que copiar una hoja de cálculo al terminar el mes.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Un plan de infraestructura para 14 departamentos — 14 departamentos de investigación con un único plan de infraestructura

Un plan de infraestructura para 14 departamentos Póngase en contacto →

Por qué importa

Las redes pierden respuestas. Cuando el cliente envía una petición y la conexión se corta antes de que llegue la respuesta, no puede saber si el servidor nunca la vio, la vio y se detuvo, o la ejecutó y perdió la respuesta. Repetirla es la recuperación evidente; la idempotencia hace inocua esa repetición. RFC 9110 basa precisamente en ello sus reglas de reintento. Una petición idempotente puede repetirse automáticamente tras un fallo de comunicación. El cliente no debería reintentar automáticamente una petición no idempotente, salvo si sabe que en la práctica sí lo es o puede determinar que la original nunca se aplicó; un proxy no debe reintentar una petición no idempotente por iniciativa propia en ningún caso.

Las colas plantean la misma cuestión a escala. Muchos sistemas de mensajes prometen entrega al menos una vez: un consumidor que falla después de hacer su trabajo, pero antes de confirmar el mensaje, vuelve a recibirlo. Prometer entrega exactamente una vez a través de una red es mucho más difícil; la respuesta práctica es un consumidor para el que una segunda entrega no cambie nada. La gestión de configuración también depende de esa propiedad. Una herramienta que lleva los servidores a un estado declarado se ejecuta una y otra vez, tras cada cambio y a menudo de forma programada; cada ejecución debe dejar un servidor correcto exactamente como lo encontró.

La decisión real es cómo obtener idempotencia cuando una operación no la tiene por naturaleza. Hay tres respuestas habituales. La primera es el diseño: expresar el cambio como estado objetivo, no como paso; «asegurar que esta línea esté en el archivo», en vez de «añadir esta línea al final». La segunda es una clave: el cliente adjunta un identificador único a la petición, el servidor lo registra con una restricción de unicidad y una repetición encuentra el registro y recibe el resultado original, en lugar de provocar una segunda acción. La tercera es comprobar antes de actuar, como hace la mayoría de los módulos de Ansible: comparan el estado actual con el deseado y terminan sin actuar si coinciden. Cada opción tiene un coste. Diseñar por estado objetivo exige un modelo claro del estado final; las claves necesitan almacenamiento y una regla de conservación, y comprobar primero requiere una forma fiable de leer el estado actual.

Cómo se aplica en la práctica

Los servidores de la empresa se configuran con Ansible. En ese proyecto, la idempotencia es un contrato: un host que ya ha convergido informa de cero cambios. El aprovisionamiento sin argumentos lleva cada host a su estado declarado; un play que no pueda informar de cero cambios en un host ya correcto se considera inacabado. Así, cada ejecución mide la idempotencia en lugar de confiar en que exista.

La misma idea, aplicada a los datos, sustenta una prueba que ejecuta dos veces la cadena de construcción de diecinueve pasos. La página nombra el fallo que pretende detectar: un generador que funciona la primera vez y corrompe los datos la segunda, mediante pasos que insertan sin comprobar o dependen del orden de la salida de un paso anterior. La prueba construye la base de datos desde un archivo vacío, toma una instantánea, vuelve a ejecutar toda la cadena sobre el resultado y exige que la segunda pasada no cambie nada, incluidos los recuentos, contenidos e identificadores de las filas. La prueba aporta evidencia sobre las ejecuciones repetidas con las entradas y el estado de la base de datos que comprueba.

Para vender existencias limitadas, la nota sobre vender la última entrada exactamente una vez combina una clave con escrituras condicionales. Cada proceso de compra lleva una clave de idempotencia y la tabla de reservas tiene `UNIQUE (tenant_id, idempotency_key)` : un clic reintentado devuelve la reserva existente, no una segunda unidad. Una liberación que se ejecuta dos veces, por ejemplo por un tiempo de espera agotado y un botón de cancelación, devuelve la unidad una sola vez, porque solo repone existencias cuando su propia actualización cambió exactamente una fila. Más adelante, cada consumidor de

mensajes es idempotente y usa el pedido como clave: los mensajes llegan al menos una vez y la idempotencia hace que su efecto se produzca una sola vez en la práctica.

Dónde surgen los problemas

Una petición reintentada cobra dos veces. Un POST que crea un pago, un pedido o un correo no es idempotente de forma predeterminada; un cliente o proxy que lo reintenta tras un tiempo de espera agotado crea otro. Asigne a estas peticiones una clave de idempotencia que el servidor almacene con una restricción de unicidad y devuelva el resultado guardado si recibe la misma clave.

Un resultado vacío se interpreta como error. En PostgreSQL, `INSERT ... ON CONFLICT DO NOTHING` omite una fila que infringiría una restricción de unicidad y `RETURNING` solo devuelve filas realmente insertadas o actualizadas. Repetir la inserción de la misma fila no devuelve nada: es el resultado correcto de una segunda ejecución. Tratarlo como fallo convierte un reintento inocuo en una página de error; el código debería consultar la fila existente.

Una tarea de comando cambia el host en cada ejecución. La documentación de Ansible dice que la mayoría de los módulos comprueban si ya se alcanzó el estado deseado y que «no todos los playbooks ni todos los módulos se comportan así». Una tarea que ejecuta un comando es la excepción habitual: el comando es arbitrario y Ansible no puede saber qué hace. La opción `creates` del módulo command omite el paso si ya existe un archivo dado; `removes` lo ejecuta solo mientras el archivo siga presente, y `changed_when` define qué resultado cuenta como cambio. Sin ninguna de ellas, la tarea se ejecuta otra vez en cada convergencia, sea cual sea el estado del host.

Un contador cuenta el reintento. Con entrega al menos una vez, un consumidor que incrementa un contador o envía una notificación por mensaje lo hace dos veces si recibe dos veces un mensaje. Guarde la clave de negocio del mensaje junto con el efecto, en la misma transacción, y omita las claves ya almacenadas.

La segunda ejecución corrompe lo construido por la primera. Un paso puede ser correcto en una base de datos vacía e incorrecto en una llena, al insertar un duplicado, renumerar identificadores o depender de un orden dejado por la primera ejecución. Una prueba que parte de cero y ejecuta una sola vez pasará igualmente. Ejecute todo el trabajo dos veces y exija que la segunda no cambie nada, como hace la prueba anterior.

Lecturas adicionales

- RFC 9110, sección 9.2.2: métodos idempotentes — Definición HTTP, métodos idempotentes y reglas de reintento automático.
- RFC 5789: método PATCH para HTTP — Por qué PATCH no es seguro ni idempotente y cómo una petición condicional permite repetirlo sin efectos adicionales.
- Ansible: introducción a los playbooks — Estado deseado, idempotencia y modo de comprobación explicados por Ansible.
- Ansible: módulo command — Opciones `creates` y `removes` para poder repetir un comando sin efectos adicionales.
- PostgreSQL: INSERT — `ON CONFLICT` y lo que devuelve `RETURNING` si ya existe una fila.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué significa idempotente?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por **D.**

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/idempotent/