Engineer

イベントソーシング

状態を変更イベントの順序付きログとして保存する方式

どのようなものか

イベントソーシングは、何が起きたかの記録そのものをデータとする保存方式です。多くのアプリは、注文や口座の現在の状態を一行に置き、変更のたびに上書きします。イベントソーシングでは、代わりに変更を保存します。各変更をイベントとしてログの末尾へ加え、現在の状態が必要になったら、そのログから求めます。2005年にこのパターンを説明したMartin Fowlerは、一文で表しました。「アプリケーションの状態へのすべての変更を、一連のイベントとして記録する」。

イベントが記録するのは、すでに起きたことです。Greg YoungのCQRS Documentsは、`CustomerRelocated` や `CargoShipped` のように、過去形の動詞でイベントへ名前を付けるよう求めます。名前が、事業側が自分たちの仕事を語る言葉の一部になるためです。過去の事実を、起きなかったことにはできません。そのため、イベントは編集も削除もしません。Youngは、ログを会計の元帳に例えます。誤った記入を消すのではなく、新しい記入で訂正します。削除にも同じ規則を使います。客が、一度入れた靴下二足をカートから取り除くと、ログへ「取り除いた」というイベントを加えます。Youngは、このような記録を逆操作と呼びます。カートは、靴下を一度も入れなかったのと同じ状態へ戻りますが、ログには、入れた事実も残ります。

イベントは、状態を表す対象一つごとに、一つのストリームへ保存します。ドメイン駆動設計を土台にするYoungは、その対象を集約と呼びます。各イベントには、ストリームのIDと、そのストリーム内で一意の連続するバージョン番号があります。そのため、順序が曖昧になりません。注文を読み込むには、その注文のイベントをバージョン順に読み、一つずつ適用します。Youngは、保存機構自体がする仕事の少なさを指摘します。最も単純なものは、一つのストリームの全イベントを順に読むことと、新しいイベントを末尾へ加えることの、二つの操作を提供します。

ログが、正本になります。Fowlerによると、正式な記録は、ログにも現在の状態にもできます。イベントソーシングでは、ログです。現在残高のテーブル、検索索引、レポートなど、それ以外は、ログから作った複製です。捨てて、作り直せます。

概念を判断に生かす

「イベントソーシングとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 データの保存先、処理のルール、他のシステムとの接続方法など、利用者の目に見えない部分をつくります。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 ETL/ELTパイプライン6本を設計・運用 — 6 3種類のDBエンジンにまたがるETLパイプライン

ETL/ELTパイプライン6本を設計・運用 お問い合わせ →

なぜ重要なのか

Fowlerは、完全なログが可能にする、三つのことを挙げます。完全な再構築では、アプリの状態を捨て、空のアプリへ全イベントを再生して、作り直せます。時点を指定する問い合わせでは、任意の過去の瞬間の状態が、その時点までイベントを再生した結果になります。イベントの再生では、誤ったイベントが判明したら、それと以後のイベントを逆操作で戻し、訂正した順序を再生できます。同じログは、誰も書き残すことを意識せずに得られた監査証跡でもあります。
Fowlerは、日常の用途にサポートとデバッグを挙げます。サポート担当者は、利用者が何をしたかを正確に見られ、開発者は、本番のイベントをテスト環境で再生できます。

Youngは、オンライン小売店の二つのチームで、利点を示します。どちらも、客がカートから取り除く商品について、レポートを求められます。現在の状態しか保存しないチームは、今から削除を記録し始め、その日以降のデータしか出せません。イベントを保存するチームは、同じレポートの処理を書き、ログを最初から処理します。初めてのレポートでも、何年もさかのぼれます。システムを作ったときに誰も考えなかった問いにも、答えられます。最終状態ではなく、起きたことを残していたためです。

イベントは、ほかのシステムが変更を知る方法でもあります。Youngによると、多くのシステムは、状態を保存してから、キューへイベントを発行します。間でクラッシュすると両者が食い違うため、厳密には、その二つの間に二相コミットが必要です。代案は、イベントストア自体をキューにすることです。各イベントのシーケンス番号で、別のプロセスが順に発行でき、書き込みは一回だけになり

ます。追記だけの保存なので、規模も広げやすくなります。Youngは、既存データを上書きする保存より、必要なロックがはるかに少なく、イベントストアを分割するときに使うキーも、集約IDだけでよいと述べます。

代償もあり、Fowlerは明記します。すべての変更をイベントにするプログラミングのやり方は、誰もが扱いやすいものではありません。最初からそれを前提にしなかったシステムへ、後で加えるのも、非常に困難です。保存機構が答える問いも、「このストリームを返して」という一種類だけです。Youngの例では、イベントストアに、Gregという名前の利用者が誰かを聞いても、分かりません。その問いには、イベントから作る、別の読み取りモデルが必要です。そのため、書くモデルと読むモデルを分けるCQRSと、イベントソーシングを、一組として扱います。

実際の活用例

動作する最小構成のイベントストアは、Youngの示す、集約ID、イベントデータ、バージョンという形に近い、一つのPostgreSQLテーブルです。ここでは、売りに出す一商品の在庫を保持します。各イベントが持つのは、新しい合計ではなく、販売可能数の変化です。Fowlerが、最も素直に逆操作できるとする形です。「十を足す」は、十を引けば戻せます。「110にする」では、置き換えた元の値を失っています。

```
CREATE TABLE stock_events (
    item_id  text    NOT NULL,
    version  integer NOT NULL,
    event    text    NOT NULL,
    order_id text,
    units    integer NOT NULL,  -- 販売可能数の変化
    PRIMARY KEY (item_id, version)
);

INSERT INTO stock_events VALUES
    ('item-881', 1, 'ItemStocked',   NULL,   100),
    ('item-881', 2, 'UnitsHeld',     'A-17',  -2),
    ('item-881', 3, 'HoldExpired',   'A-17',   2),
    ('item-881', 4, 'UnitsHeld',     'B-02',  -1),
    ('item-881', 5, 'HoldConfirmed', 'B-02',   0);

-- 現在の状態：ストリーム全体を集計する
SELECT sum(units) AS on_sale FROM stock_events
WHERE item_id = 'item-881';                    -- 99
```

```
-- 時点を指定する問い合わせ：バージョン2の直後の商品状態
SELECT sum(units) AS on_sale FROM stock_events
WHERE item_id = 'item-881' AND version <= 2;   -- 98
```

主キーが、Youngのいう楽観的な同時実行の検査を担います。二つの購入処理が、どちらもバージョン5でストリームを読み、一個を確保しようとすると、それぞれバージョン6を末尾へ加えようとします。

```
INSERT INTO stock_events VALUES ('item-881', 6, 'UnitsHeld', 'C-40', -1);
INSERT INTO stock_events VALUES ('item-881', 6, 'UnitsHeld', 'D-11', -1);
-- 二つ目は失敗する：(item-881, 6)が既に存在する
```

PostgreSQLの文書では、複数の列にまたがる主キーは、その組み合わせを一意にし、重複する挿入はエラーになると説明します。負けた購入処理はストリームを読み直し、まだ一個売れるかを確かめ、バージョン7で再試行します。どちらも、存在しない在庫を売っていません。ログには、両方の判断が、実際の順序で残ります。

ノート最後のチケットを、厳密に一回だけ売るは、同じ在庫の問題を検討し、この構成の限界を示します。最初の答えは、一行へ個数を置き、一つの条件付き `UPDATE` で変える方法です。競合の激しいカウンターでは、バージョン列が競合を再試行へ変えるため、楽観的ロックを勧めません。最もリクエストが集中する商品には、すべてのリクエストを一つのKafkaパーティションへ置き、一つのコンシューマーが読み、個数をメモリー内に保持し、判断をバッチで確定します。ノートは、これをLMAX型と呼びます。出典の一つであるFowlerのLMAXアーキテクチャーの記事は、最も純粋なイベントソーシングを説明します。データベースをまったく持たない取引エンジンで、状態全体を、入力イベントから導けます。ジャーナル処理が入力イベントを永続保存するため、再生できます。一方、ノートの購入処理は、別種のイベントを使います。各段階がアウトボックステーブルへメッセージを書き、変更データキャプチャーが発行します。正本は、テーブルのままです。そのイベントは、変更の知らせです。イベントソーシングでは、イベント自体が正本になります。

問題が起きるところ

ログの再生で、副作用も繰り返します。 確認メールを送ったり、カードへ請求したりする処理は、ログを再生すると、もう一度行います。Fowlerは、外部システムには、再生と本当の業務を区別できないと指摘します。読み取りモデルを作り直すだけで、全顧客へ、昨年のメールを送り直しかねません。すべての外部シ

ステムを、アプリが再生中であることを把握するゲートウェイの背後へ置き、再生中の外向きの呼び出しを捨てます。ドメインロジックは、それを知る必要がなくなります。問い合わせにも、同じ配慮が必要です。Fowlerの例は、通貨換算です。再生したイベントには、今日ではなく、そのイベントの日の為替レートが必要です。そのため、ゲートウェイは、受け取った答えを記録し、再生時に返すべきです。

混み合うストリームで、再試行が増えます。 楽観的な同時実行制御は、競合が少ないことを前提にします。多数の書き手が、同じストリームへ同時に加えると、競合のたびに読み直しと再試行が必要です。トラフィックが急増すると、再試行自体が負荷になります。上のノートが、バージョン列について警告するのは、この点です。同時に二つの書き手がいることが少なくなるよう、ストリームを分けます。製品群全体に一つではなく、注文ごとに一つです。本当に競合が激しいストリームには、LMAXのように書き手を一つだけ割り当て、競合の相手をなくします。

長いストリームは、読み込みが遅くなります。 一つのオブジェクトを読むため、何千ものイベントを再生すると、リクエストのたびに時間がかかります。Fowlerは、再構築の遅さを、現在状態の複製を保持する理由に挙げます。Youngの答えは、更新していくスナップショットです。既知のバージョンでのストリームの状態をシリアライズし、専用テーブルに置き、別プロセスがバックグラウンドで書きます。読み込みには、最新のスナップショットと、そのバージョンより後のイベントを使います。スナップショットは、イベントログ自体の外へ置きます。Youngは、混み合うストリームで、スナップショットを作る処理が、同時実行の検査に繰り返し失敗する例を述べます。最初はスナップショットなしで作ることも勧めます。読み込みを速めるためなら、後でいつでも加えられるためです。

読み手が、書き手に追い付きません。 読み取りモデルは、イベントを書いた後で作るため、ログより遅れます。Fowlerは、ログを読むシステムは、その内容とずれやすいと警告します。フォームを送っても、その変更のない一覧が見えれば、利用者はもう一度送ります。各読み取りモデルに、最後に適用したイベントを記録させ、遅れを測り、最初から作り直せるようにします。利用者自身のコマンドの結果は、まだ追い付いていないかもしれない読み取りモデルではなく、そのコマンドへの応答から見せます。

参考資料

- イベントソーシング — Martin Fowlerによる、完全な再構築、時点を指定する問い合わせ、イベントの再生、ほかのシステムへ再生を届かせないゲートウェイの説明です。
- Greg YoungのCQRS Documents — 過去形の事実としてのイベント、削除の代わりの逆操作、二つのテーブルからなるイベントストア、更新していくスナップショット、キューとしてのイベントストアです。
- LMAXアーキテクチャー — ジャーナルへ記録した入力イベントから、全状態を導く、メモリー内の取引エンジンです。
- PostgreSQL：制約 — 複数の列に対する一意制約と主キー制約、楽観的な同時実行の検査をデータベース側で担う部分です。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「イベントソーシングとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/event-sourcing/