メッセージキューを使おう

somnicat

October 19, 2026

自己紹介

名前: **somnicat** (@somnicattus)



職業: Web アプリケーションエンジニア

使う技術:

- フルスタック TypeScript (Vue/Node.js)
- 分散メッセージング (Amazon SQS, SNS)



今日話すこと

- 分散メッセージングとは
- いろいろな分散メッセージング
- メッセージキューの便利なところ
- 使うときの注意

分散メッセージングとは

分散メッセージングとは

サービス同士がメッセージを介してやり取りする

- 送る側 (Producer) はメッセージを**ブローカー**に置く
- 受け取る側 (Consumer) はブローカーからメッセージを受け取る
- 送る側と受け取る側は、お互いを**直接知らない**
  - どちらもブローカーのことは知っている

> 送る側は「注文が確定した」というメッセージをブローカーに置くだけ
> 受け取る側はブローカーに届いた「注文確定」メッセージを順番に処理するだけ

直接呼び出しとの違い

```
★直接呼び出し（HTTP など）
   注文サービス ──呼ぶ──▶ メール送信サービス
                ◀──結果──

★分散メッセージング
   注文サービス ──置く──▶ [ ブローカー ] ◀──取りに行く── メール送信サービス
```

- 直接呼び出しでは、相手が**今**動いていて、**すぐ**返事をくれることが前提
- メッセージングでは、送る側はメッセージを置きに行くだけ (**非同期**)

いろいろな分散メッセージング

ちょっと知ってる人向け

本題ではないのでささっと紹介だけ

Queue (キュー)

１つのメッセージを１つの 受け取り手 が処理する

- メッセージは受け取り手が処理するまで**キューに溜まる**
- 処理が終わったメッセージは**削除される**
- 受け取り手を複数並べるとメッセージの処理を**分担できる**
  - 他の受け取り手が処理中のメッセージは見えない

> 例: Amazon SQS, Azure Queue Storage
> 「この仕事を誰か１人にやってほしい」

Pub/Sub (パブサブ)

1 つのメッセージを **複数の購読者全員** に配る

- 送る側は**トピック**にメッセージを発行 (publish) する
- トピックを購読 (subscribe) している全員に**同じメッセージが届く**
- 購読を追加したり削除したりして後から **受け取る側を変更できる**
  - 購読を始めるまでに届いたメッセージは消えてしまって見れない

> 例: Amazon SNS, Google Cloud Pub/Sub
> 「このできごとを関係者全員に知らせたい」

Streaming Log (ストリーミングログ)

メッセージの履歴を保持して何度でも読めるようにする

- メッセージは処理されても消えず、**保持期間まで残る**
- 受け取り手は「どこまで読んだか (オフセット)」を自分で管理する
- 受け取り手はメッセージを**任意の時点からから読み直せる**（リプレイ）
  - 期間内ならメッセージの履歴が残るので過去分も含めて見れる

> 例: Apache Kafka, Amazon Kinesis
> 「できごとの履歴を、あとから何度でも読みたい」

3 つの比較

	Queue	Pub/Sub	Streaming Log
代表的なサービス	Amazon SQS	Pub/Sub (Google)	Apache Kafka
誰に届くか	誰か 1 人	購読した全員	全員
メッセージの保持	処理されるまで	基本は残らない	保持期間まで
向いている用途	仕事の受け渡し	イベントの通知	履歴・集計
コスト・複雑さ	安い・シンプル	安い・少し複雑	高い・複雑

> Queue は**仕事を確実にやってもらう**ための仕組み
>
> Pub/Sub や Streaming Log は**できごとの通知や記録**の仕組み

メッセージキューの便利なところ

メッセージキューの便利なところ ①

受け取る側が止まっていてもメッセージを送れる

- メッセージはブローカーが用意した**キューに溜まる**ので、受け取り手が落ちていても送る側は困らない
- 受け取り手が復帰したら、溜まったメッセージを順に処理するだけ
- デプロイやメンテナンスで受け取り手を止めやすくなる

> メール送信サービスが止まっていても、注文は受け付けられる

メッセージキューの便利なところ ②

受け取る側の**都合で頻度を調整**して受け取れる

- 受け取り手は自分のペースでメッセージを**取りに行く**
- アクセスが急増しても、増えた分はキューが**受け止める**
- 受け取り手は処理できる量以上を抱え込まない

> セール開始直後の注文の山も、普段どおりのペースで順に処理できる
> 送る側のピークに合わせて事前に受け取り手を用意しなくていい

メッセージキューの便利なところ ③

受け取る側を水平スケールすると負荷分散できる

- 同じキューから複数の受け取り手がメッセージを取りに行ける
- １つのメッセージは**どれか１つの受け取り手**にだけ渡る
- キューに溜まった数を見て、受け取り手の台数を増減できる

> スパイクが発生してスケールが間に合わなくてもキューで受け止められる
> キューがしばらく空なら受け取り手の台数を 0 にしてもいい

メッセージキューの便利なところ ④

受け取る側の処理に失敗してもリトライできる

- メッセージは**処理が成功してから削除**する
- 処理失敗で削除されなかったメッセージは、一定時間後にまた受け取る
- 何度やっても失敗するメッセージは**デッドレターキュー**に退避する

> 誰かが受け取って処理を始めたメッセージは同時に受け取れない
> タイムアウトの時間が過ぎたら失敗と判断して、また誰かが受け取ってやり直す

使うときの注意

使うときの注意点（一部）

メッセージは 2 回以上届くことがある

- 処理が成功してもメッセージ削除が失敗してリトライされることがある
- 同じメッセージを 2 回処理しても結果が変わらないように (**冪等に**) 作る

メッセージは順序どおりに届かないことがある

- 標準的なキューはメッセージが届く順序を保証しない
  - 順序の保証が必要なら FIFO キューを使う

> 他にも細かい設計上の注意点がいくつかある（省略）

ブローカーを自前でホスティングするのは大変

ブローカーが落ちると、全部のやり取りが止まる

- 止められないので、複数台での**冗長化**が必須になる
- メッセージを失わないように、ディスクや**レプリケーション**を管理する
- 溜まる量に合わせた**容量の見積もり**とスケール
- バージョンアップ・監視・障害対応も**全部自分たち**でやる

> RabbitMQ や Kafka のクラスタ運用は、それだけで専任の仕事になりがち
> SQS / SNS などのマネージドサービスなら、運用はクラウドに任せられる

まとめ

- サービス同士が**ブローカーを介して**非同期にやり取りする
- Queue / Pub/Sub / Streaming Log は役割が違う
- メッセージキューの便利なところ
  - 受け取る側が**止まっていても**送れる
  - 受け取る側の**都合で頻度を調整**できる
  - 受け取る側を**水平スケール**できる
  - 処理に**失敗してもリトライ**できる
- ブローカーの運用は大変なので、**マネージドサービス**に任せるのが吉

Thank You!