Engineer

OpenAPI

HTTP APIを機械可読な形で記述する形式

どのようなものか

OpenAPIは、HTTP APIを呼ぶ側が知るべきすべて、つまりアドレス、操作、入力、出力、エラー、セキュリティーを、人もプログラムも読める、一つのJSONまたはYAMLファイルに記す標準形式です。標準自体をOpenAPI Specificationと呼び、OASと略すことが多く、それに従ったファイルがOpenAPI記述です。元はSwagger Specificationという形式で、Swagger UIなど、周辺のツールには以前の名前が残っています。

仕様は、「HTTP APIのための、標準的で、プログラミング言語に依存しないインターフェース記述」を定めると説明します。目標は、人やプログラムが、ソースコード、追加のドキュメント、ネットワーク通信のキャプチャーなしで、サービスの機能を理解できることです。入口となる文書名には、`openapi.json` か `openapi.yaml` を推奨し、大きな記述は、参照でつないだ複数のファイルに分けることもよくあります。

Swagger 2.0は2014年9月に公開され、2015年12月31日にOpenAPI Initiativeへ寄贈されました。新しい名前での3.0.0は2017年7月、3.1.0は2021年2月、3.2.0は2025年9月に続きました。

記述の最上位には、いくつかの要素があります。`info` はAPIの名前とバージョン、`servers` は動かす場所を示します。`paths` は各アドレスを持ち、その下に、HTTPメソッドごとの操作と、パラメーター、リクエストボディー、想定する応答を置きます。`components` は、データのスキーマ、パラメーター、応答、呼び出し側の認証方法を定めるセキュリティースキームなど、再利用するものをまとめます。3.1以降は、`webhooks` で、APIが利用側へ送るリクエストも記述できます。最小限の断片は、次のようになります。

```
openapi: 3.1.0
info:
    title: Orders
    version: 1.4.0
paths:
    /orders/{id}:
        get:
            operationId: getOrder
            parameters:
                - name: id
                  in: path
                  required: true
                  schema:
                      type: string
            responses:
                "200":
                    description: One order
```

データの形は、スキーマ言語で書きます。3.1以降のSchema Objectは、JSON Schema Draft 2020-12の上位集合です。3.0は、以前の草案の一部を基に、独自の追加を加えたものでした。JSON Schemaが一つのデータを記述するのに対し、OpenAPIはHTTP API全体を記述し、各部分にはJSON Schemaを借りて使います。隣接する仕様は、別の種類のインターフェースを記述します。AsyncAPIはメッセージ駆動のシステムを扱います。gRPCのサービスはProtocol Buffersの `.proto` ファイルで宣言し、GraphQLサーバーは自身のスキーマを公開します。

概念を判断に生かす

「OpenAPI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

標準形式の記述は、多くのツールが読める一つのファイルであり、それぞれが、チームの手作業を減らします。参照ドキュメントを生成できます。複数の言語のクライアントライブラリーも生成でき、フロントエンドはURLを組み立てる代わりに、型付きの関数を呼べます。モックサーバーも記述を基に応答するため、バ

ックエンドができる前にフロントエンドを作れます。ゲートウェイはルートを取り込み、テストスイートは記述に照らして通信を検証し、APIを呼ぶAIエージェントは、何が使えるかを知るために読みます。

その広がりは、リスクでもあります。すべてのツールが、そのファイルを信じます。間違った記述は、ないより悪いものです。生成したクライアントはコンパイルでき、ドキュメントには権威があるように見える一方、問題が表れるのは、他者の製品の実行時だからです。

そのため、大きな判断は、記述をどこから作るかです。設計先行では、人が記述を書き、契約としてレビューして、それに合うサーバーを実装します。コードを書く前にAPIの形を議論でき、公開APIや、並行して作業するチームに向きます。代償は、記述とサーバーという二つのものに、食い違いが起こり得ることです。コード先行では、サーバー自身の型と注釈から記述を生成します。サーバーにないルートは記述できませんが、その型で表せる範囲を超える精度は得られず、設計の議論は文書でなくコードレビューで行います。どちらも、ビルド時に検査して初めて、記述の正しさを保てます。

3.0から3.1への移行では、3.1がJSON Schema 2020-12を採用したため、スキーマの書き方が変わりました。ジェネレーター、バリデーター、ゲートウェイが対応するバージョン範囲は、それぞれ違います。そのため、チームが書けるバージョンは、処理の連鎖の中で最も古いツールに制約されます。

実際の活用例

初めは、一つのAPIをFiberで動かし、エンドポイントごとに手で検証を書き、全体の形を記述する単一の文書はありませんでした。HTTP層を、Fiber上のHuma v2へ移しました。各エンドポイントを、入力と出力の構造体で宣言し、その型からOpenAPI記述を生成します。ドキュメントは、Wikiで古くなるのでなく、コードに追随します。二つのWebSocketエンドポイントは、Humaのリクエストと応答のモデル外なので、素のFiberに残ります。記述が重要になるのは規模のためです。649パスに760操作あると、実際には、この記述だけがAPI全体を読み解く方法になります。操作が五十だったとき以上に、記述が完全かどうかが重要で、CIの検査がルートの一致率を100%に保ちます。

別のプロジェクトでは、記述は、データベースから外へ伸びる連鎖の中間にあります。データベースのスキーマと関数がGoの型を形作り、Humaが記述を出し、フロントエンドが使うものを、そこから生成します。44,076行の型付きTypeScriptクライアント、フロントエンドのテスト用の61のモックハンドラー、各フォームが適用する制限です。フロントエンドとバックエンドのずれは、食い

違ったまま公開した後、ブラウザーの422として現れがちですが、この連鎖で対処します。CIの契約検査は、ずれがあればビルドを失敗させます。検査対象には、「欠落」と「null」を混同しやすいボディーの任意フィールドと、クエリーパラメーターの列挙値を含めます。代償は、すべての間に生成処理が一段入り、連鎖全体の信頼性が、最も保護の弱い部分に制約されることです。

AIエージェントが利用できる状態についての自社ページも、同じ考えをエージェントに適用します。エージェントが使えるアプリケーションは、記述済みのAPIを提供します。OpenAPI文書、エージェントから直接呼ばせるべき場合のMCPサーバー、パスワードを渡さず利用者の代わりに動くためのOAuthメタデータです。ページは、勧告をそのまま守るのでなく、一つずつ分類しています。自社サイトにはAPIがないため、APIカタログとOAuthメタデータは、そこで採用しなかった五つのうちの二つです。

問題が起きるところ

サーバーが行わないことまで、記述が約束します。 手で保守する記述は、追加したクエリーパラメーター、名前を変えたフィールド、サーバーだけが知るエラーコードなどに、追い付かなくなります。生成クライアントは、その架空の仕様に合わせてコンパイルされます。コードから記述を生成するか、別に保ち、テストスイートで、実際のリクエストと応答を記述に照らして検査します。サーバーが登録するルートと、記述が並べるパスも比較します。

クライアントを再生成するたびに、半分のメソッド名が変わります。 `operationId` がなければ、ジェネレーターは、パスとHTTPメソッドから名前を作ります。ルートを移すと、すべての呼び出し側が使う関数名が変わります。仕様は、各 `operationId` が一意であることを求め、大文字と小文字を区別します。すべての操作に付け、公開関数名と同じ注意を払って選び、一つでも欠けたら、リンターでビルドを失敗させます。

更新後、nullを許すフィールドが、nullを拒否します。 OpenAPI 3.0は、`nullable: true` で指定していました。3.1では、そのキーワードがなくなり、型はリストになります。バリデーターは、知らないキーワードを通常無視するため、3.0のスキーマを3.1と付け替えただけでは、作者が許したつもりのnullを拒否します。 `exclusiveMinimum` が数値になる変更も含む、OpenAPI Initiativeの移行ガイドに従って変換します。

```
# OpenAPI 3.0
type: string
nullable: true
```

```
---
# OpenAPI 3.1
type: [string, "null"]
```

例が、自分のスキーマに合いません。 仕様は、例はスキーマに合うべきとし、検査をツールに任せます。読者がドキュメントからコピーすると422を受け、モックサーバーが返した例を基に、実際のAPIが拒否するデータでフロントエンドを作ってしまいます。CIで、すべての例を、本番の通信と同じ厳しさで検証します。

公開した記述が、管理APIを案内します。 全ルートから生成した一つの記述を公開アドレスで配ると、内部の操作とそのスキーマを、誰でも読めます。利用対象ごとに、内部の操作を除外した記述を生成して公開します。仕様は、まさにこの種のアクセス制御のために、オブジェクトを空のままにしたり、削除したりすることを認めます。

参考資料

- OpenAPI仕様 — すべてのオブジェクトとフィールドの定義、および各版の履歴を含む、標準の最新版です。
- JSON Schema Draft 2020-12 — OpenAPI 3.1以降が、各リクエストと応答のボディーに使うスキーマ言語です。
- OpenAPI 3.0から3.1への移行 — `nullable` など、スキーマの変更をまとめた、OpenAPI Initiative自身の一覧です。
- Huma：OpenAPI生成 — 上のプロジェクトで使うGoフレームワークが、ハンドラーの型から記述を作る方法です。

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

「OpenAPI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/openapi/