Engineer

JSON Schema

JSONデータの構造を記述する言語

どのようなものか

JSON Schemaは、JSONデータに求める構造を記述する言語で、それ自体もJSONで書きます。この言語による記述をスキーマと呼び、そこに規則を定めます。この値はオブジェクトで、これらのプロパティーが必要、この値は一以上の整数、別の値は最大200文字の文字列、といった規則です。バリデーターというプログラムが、スキーマと、検査するデータであるインスタンスを受け取り、インスタンスが適合するか、適合しなければどこに、どの理由で違反があるかを返します。プロジェクト名は、Sを大文字にしてJSON Schemaと書きます。この言語で書いた個々の文書は、英語では小文字のschemaです。

仕様は、json-schema.orgのJSON Schemaプロジェクトが保守しています。現在の版は2020-12で、その前は2019-09です。それ以前はdraft-04やdraft-07のような番号付きのドラフトで、古いツールや記事では、今もその名前を使います。仕様は二部に分かれます。中核の文書は、スキーマの処理方法、別のスキーマを参照する方法、メディアタイプ `application/schema+json` を定義します。もう一つの検証に関する文書は、値を検査するキーワードを定義します。現行の2020-12の文書は、どちらも2022年6月16日付です。

各キーワードが、一つの制約を加えます。 `type` は、string（文字列）、number（数値）、boolean（真偽値）、array（配列）、object（オブジェクト）、nullという六つの基本型か、小数部分がゼロの数だけを受け入れる `integer` を指定します。数値には `minimum` 、 `maximum` 、 `multipleOf` 、文字列には `minLength` 、 `maxLength` 、正規表現の `pattern` 、配列には `minItems` 、 `maxItems` 、 `uniqueItems` 、オブジェクトには `properties` 、 `required` 、 `additionalProperties` を使います。 `allOf` 、 `anyOf` 、 `oneOf` 、 `not` などのキーワードは、スキーマを組み合わせます。 `$ref` は、別の場所、多くの場合は同じ文書の `$defs` の下で定義したスキーマを指し、一度書いた構造を多くの箇所で使えるようにします。スキーマが触れないものは、すべて許可します。空のスキーマ `{}` は、あらゆるインスタンスを受け入れます。

スキーマには、言語のどの版で書いたかも示せます。`$schema` キーワードが持つのは、その版の有効なスキーマを記述する、メタスキーマのアドレスです。仕様のページは、このメタスキーマが自分自身を検証すると説明しています。2020-12では、アドレスは `https://json-schema.org/draft/2020-12/schema` です。

概念を判断に生かす

「JSON Schema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

JSONは、意味について何も定めません。形式が正しければ、読むプログラムが扱えるかどうかに関係なく、文書を解析できます。明文化した記述がなければ、文書を読む各プログラムが、有効なデータとは何かを、コード内の条件に分散した形で、それぞれに判断します。その判断は次第に食い違い、本番で要求が失敗するまで気付かないことがあります。スキーマは、人とプログラムの両方が読める形で規則を一か所に集めます。ほかのファイルと同じように、保存、版管理、レビュー、公開できます。

仕様が掲げる用途は、広いものです。JSON Schemaは、JSONデータの検証と文書化、ハイパーリンクによる移動、操作の制御を目的とし、検証用の語彙には、データを表示する画面へ、そのデータについてのヒントを与える役割もあります。実際には、一つの記述を使って、サービスの入口で不正な入力を拒否し、APIを文書化し、フォームを動かし、言語モデルが返すJSONに次の処理が期待するフィールドがあるかを検査できます。

ほかの仕様が基礎とする、共通の仕組みにもなっています。OpenAPI 3.1は、Schema ObjectをJSON Schema 2020-12の上位集合と定義するため、API記述内の要求本文と応答本文は、いくつかの追加を持つJSON Schemaです。AIエージェントがツールを見つけ、呼び出すプロトコルのMCPも、各ツールのパラメーターを、JSON Schemaである `inputSchema` で記述します。結果を `outputSchema` で記述することもでき、クライアントは、それに対して構造化された結果を検証するべきです。そのため、バリデーター、API文書、エージェントで共通の記述を利用できます。

実際の活用例

次のスキーマは、配送枠を予約するツールの入力を記述します。使う仕様の版を示し、四つのプロパティーとその規則を列挙し、どの三つが必須かを指定し、列挙していないプロパティーを拒否します。

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/tools/book-slot.json",
  "title": "Book a delivery slot",
  "type": "object",
  "properties": {
    "order": { "type": "string", "pattern": "^[0-9]{4,8}$" },
    "date": { "type": "string", "format": "date" },
    "pallets": { "type": "integer", "minimum": 1, "maximum": 30 },
    "note": { "type": ["string", "null"], "maxLength": 200 }
  },
  "required": ["order", "date", "pallets"],
  "additionalProperties": false
}
```

規則に違反するインスタンスです。

```
{ "order": "4471", "date": "27/10/2026", "pallets": 0, "pallet": 2 }
```

このインスタンスは、二点で違反します。 `pallets` が最小値1未満であり、 `pallet` は `pallets` の打ち間違いで、スキーマに列挙していないプロパティーです。 `note` が欠けても問題ありません。 `note` は `required` にないためです。 `null` として送っても、型がnullを認めるため、合格します。日付は、理解の助けになる例です。 `date` 形式は年を先に求めますが、 `27/10/2026` は日を先に書いています。それでも、既定設定のバリデーターは受け入れます。仕様では、 `format` を、検査を有効にしたときだけ検査する注釈としているためです。日付を検査できるバリデーターで、その機能を有効にすると、同じインスタンスは三点で違反します。

こうした記述を持つツールは、このサイトの既存ソフトウェアをAIエージェント用のツールにするページが扱うものです。配送枠の予約や注文のリリースなど、エージェントが使えるようにする操作を、型付きの入力、エージェントが対処できるエラー、再試行でも二重予約しない冪等な呼び出しを持つツールとして提供すると説明しています。記述は、コードから生成するOpenAPI文書で、MCPサーバーも同じ操作を提供します。同ページは、エージェントは古くなった記述も信

じてしまい、手書きの記述は実装と食い違う可能性があると論じています。そのため、そこでの記述は、コードを起点に、データベース、API、記述、クライアントへと生成します。各段階で、二つが食い違えばビルドを失敗させる検査を行います。

問題が起きるところ

プロパティーを列挙しても、必須にはしていません。 `properties` が検査するのは、存在するプロパティーだけです。そのため、すべて省いたオブジェクトも合格します。存在を求めるのは別のキーワードで、 `required` を書かなければ、何も必須にはなりません。毎回、 `required` を `properties` と並べて書き、各必須プロパティーを取り除いたテストケースを用意します。

打ち間違いが追加プロパティーとして通ります。 `additionalProperties` がなければ、列挙しないプロパティーも許可するため、 `pallet` が通り、本来の `pallets` が欠けても気付かないことがあります。 `false` にすると、それ以外を拒否しますが、このキーワードが見るのは、自分と同じスキーマオブジェクト内の `properties` と `patternProperties` だけです。 `allOf` や `$ref` で別のスキーマを基礎にし、この方法で閉じると、継承したはずのプロパティーまで拒否します。2020-12には、この場合に使う `unevaluatedProperties` があります。 `allOf` 、 `$ref` 、そのほか同じ対象にスキーマを適用するキーワードを通して検査したプロパティーも数えるためです。単独のスキーマには `additionalProperties: false` 、組み合わせたスキーマには `unevaluatedProperties: false` を使います。

`format` **が何も検査しません。** 既定では、 `format` は注釈であり、検証条件ではありません。仕様は、形式を検査できるバリデーターにも、明示的に求められるまで、その検査を無効に保つよう求めています。有効にしても、バリデーターは、どんな形式も検査せず通してよいことになっています。OpenAPI 3.1も同じ規則に従い、検査の程度は実装によって異なるとも述べています。使うバリデーターで形式の検査を有効にし、失敗するはずのテストケースで確かめます。重要な形式なのに検査できなければ、 `pattern` か、コードによる検査を追加します。

パターンが値の一部分に一致します。 `pattern` は先頭や末尾に固定されていないため、 `[0-9]{4}` は `order 4471b` を受け入れます。どこかに四桁の数字があればよいためです。仕様自体の例では、 `es` が `expression` に一致します。値全体を一致させる必要があれば、先頭に `^` 、末尾に `$` を付けます。パターンはECMA-262の正規表現の構文に従います。仕様は、どこでも同じ動きを求めるパターンに、限られた構文の使用を勧めています。そのため、単純な文字クラス、量指定子、アンカーにとどめます。

欠落、null、空を同じものと見なします。 存在しないプロパティー、`null` を設定したプロパティー、空文字列は、三つの異なるインスタンスです。`"type": "string"` は `null` を拒否し、`required` にないプロパティーは、省けます。値なしを表すために `null` を送るクライアントは、省略を想定したスキーマに対して、そのような要求を送るたびに失敗します。三者の意味をプロパティーごとに決め、明記します。nullに意味があれば `["string", "null"]`、任意のプロパティーなら `required` に含めず、空文字列を値と認めなければ `minLength: 1` を使います。

`$schema` がないか、二つの版が混ざります。 スキーマのルートに `$schema` がなければ、仕様は動作を実装に委ね、各バリデーターは、それぞれが既定とする版を適用します。各版には、重要な違いがあります。2019-09では、`items` に配列を与えるとタプルを記述しました。2020-12では、その役目は `prefixItems` に移り、`items` は `additionalItems` の役目を引き継ぎました。すべてのルートに `$schema` を入れ、版を移すときはリリースノートに従い、CIで各スキーマをメタスキーマに対して検証します。

参考資料

- JSON Schema仕様 — 現在の版、メタスキーマ、過去の各ドラフトのリリースノートをまとめた、プロジェクトの索引です。
- JSON Schema：JSON文書を記述するメディアタイプ — Core 2020-12です。スキーマの処理、`$schema`、`$ref`、`$defs`、スキーマを組み合わせるキーワードを扱います。
- JSON Schema検証：JSONの構造を検証する語彙 — 値の型ごとのキーワード、`required`、`format` の規則です。
- JSON Schema 2020-12リリースノート — 2019-09からの変更です。`prefixItems`、`$dynamicRef` と、それぞれが置き換えた仕組みを対比します。
- OpenAPI仕様v3.1.1 — JSON Schema 2020-12の上位集合としてのSchema Objectと、既定で注釈である `format` です。
- Model Context Protocol：ツール — ツールが、パラメーターと構造化した出力をJSON Schemaで宣言する方法です。

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

「JSON Schema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/json-schema/