Engineer

JSON

</> データの受け渡しで主に使われるテキスト形式

どのようなものか

JSONは、入れ子のデータをプレーンテキストで表す記法です。波括弧、角括弧、二重引用符、コンマを使い、ほぼどの言語のプログラムでも、別のプログラムが書いた内容をそのまま読み戻せます。JavaScriptがリテラル値を書く方法から構文を取り入れたため、JavaScript Object Notationの頭文字を取っています。ただし、JSON文書は特定の言語に属さず、読むためにJavaScriptも必要ありません。通常、ファイルの拡張子は `.json` で、HTTPでは、メディアタイプ `application/json` を付けます。

形式全体は、一ページに収まります。構成要素は、文字列、数値、 `true` と `false` の真偽値、 `null` という四つの基本型と、二つの構造型です。オブジェクトは、名前と値の組からなる、順序のない集合です。配列は、順序のある一覧です。どの要素も、ほかの要素の中に入れ子にできます。日付、バイナリーデータ、コメント、文書内の部分同士の参照は、すべて形式の外の取り決めに任せています。

Douglas Crockfordが、2006年7月にRFC 4627で、初めてIETFに登録しました。現在の定義は、2017年12月のRFC 8259です。文法ではEcmaの標準ECMA-404と一致し、二つのシステムが交換するときに必要な規則を加えています。閉じたエコシステムの外へ出るテキストは、UTF-8でなければならず、送信側は先頭にバイトオーダーマークを付けてはなりません。

XMLは、より古く、表現力に富みます。属性や名前空間があり、スキーマ言語にも長い歴史があります。YAMLは、バージョン1.2でJSONの厳密な上位集合になるよう改訂され、手で編集するファイルに求められるコメントや、記号を減らした書き方を加えました。Protocol BuffersやCBORなどのバイナリー形式は、読みやすさと引き換えに、データを小さくし、処理を速くします。

概念を判断に生かす

「JSON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を選ぶことは、実際には二つのシステム間の契約を選ぶことです。通信で渡すものの構造と、場合によっては、保存するものの構造を決めます。

通信では、既にあらゆるものが扱えるため、既定の選択になります。ブラウザーは、ライブラリーなしで解析でき、主要な言語には成熟したパーサーがあり、ログを読む技術者も、要求の内容を確認できます。負担は、別のところにあります。各レコードでフィールド名を繰り返し、数値を十進数のテキストで送り、再び解析する必要があります。形式自体は、どのフィールドが必須か、どの型を持つかを定めません。Protocol Buffersなど、スキーマを先に定めるバイナリー形式は、より小さく、解析も速く、コンパイラーが契約を検査しますが、一つのメッセージを人が読むにも道具が必要です。公開APIやブラウザー向けのAPIでは、JSONは妥当な選択です。データ量や遅延が支配的な内部サービス間では、逆の判断になることもあります。

不足する契約の記述は、通常、スキーマで補います。JSON Schemaは、オブジェクトに必要なプロパティー、その型、追加のプロパティーを認めるかを宣言します。OpenAPIは、その上にHTTP API全体の記述を組み立てます。スキーマは別の文書であり、コードとの一致を強制する仕組みが働いている間だけ、正確さを保ちます。

保存時に問うのは、構造をJSON列に入れるか、通常の列に分けるかです。PostgreSQLには、二つの型があります。`json` は、入力テキストを完全にコピーして保ち、関数で扱うたびに再解析します。`jsonb` は、分解したバイナリー表現を保存します。書き込みは少し遅くても、問い合わせははるかに速く、インデックスも作れます。文書は、大半のアプリケーションに `jsonb` を勧めています。JSON列が向くのは、行ごとに構造が変わるデータや、利用者が保存した設定のように、クライアント側が管理するデータです。データベースが結合、制約、合計に使うフィールドには、独立した列を用意するべきです。

実際の活用例

ある基盤では、PostgreSQLの関数自身が、応答全体をJSONとして組み立て、Go APIが、データベースからブラウザーへ、そのJSONを加工せず転送します。ハンドラーは `json.RawMessage` として受け取り、デコードも再エンコードもせず渡します。小さな `QueryJSON` ヘルパーが、それを既定の方法にしました。階層化したAPIのDTO、マッパー、応答用の構造体が不要になりました。関数の出力が変わると、ハンドラーのコードを編集せず、新しい構造がクライアントに届きます。

保存したJSONは、それを書いたコードより長く残るため、別の問題も生じます。ある基盤は、通知設定などを、既定値の上に利用者の保存値を重ねる形で保持し、当初は一段だけマージしていました。後から既定値に追加したキーは、それより前に保存した全行に欠けていました。そのため、クライアントのコードが未定義のフィールドを読み、異常終了しました。異常終了は、最も長く使っている利用者に起きていました。修正は、既定値をカタログとして扱う深いマージです。保存したデータの内容にかかわらず、入れ子のキーも含め、カタログの全キーを結果に含めます。移行処理が、既存の行にも一括で適用し、フロントエンドは防御用の検査を外しました。

JSONは、単にデータが届く形式でもあります。一行に一つのJSONイベントを持つJSONLで届いた、過去の電力データでは、生のイベントをゾーンごとの時間別消費量に変える作業があり、Python、JQ、SQLの三種類を実装しました。最も速かったのはPythonで、一日分は34 ms、三十日分は180 msでした。

問題が起きるところ

文法が簡潔なため、有効なテキストがどのようなものかは、パーサー間で一致します。問題は、文法が定めない部分すべてから生じます。

大きなIDが、一桁違う値で戻ります。 文法は数値の上限を定めませんが、JavaScriptを含む大半のパーサーは、64ビット浮動小数点として読みます。整数を正確に保持できるのは、$2^{53} - 1$までです。RFC 8259は、$-(2^{53} - 1)$から$2^{53} - 1$を、実装間で一致する範囲として挙げています。それを超えると、警告なしで識別子が変わります。

```
JSON.parse('{"id": 9007199254740993}').id;
// 9007199254740992
```

64ビットの識別子は文字列で送り、金額も、文字列か、最小単位の整数で送ります。浮動小数点では送りません。

二つのサービスが、同じオブジェクトを違う形で読みます。 RFC 8259は、オブジェクト内の名前は一意であるべきとし、それ以外を定めていません。同じ名前が繰り返されると、多くのパーサーは最後の値を採用し、別のものはエラーを出し、すべての組を返すものもあります。ゲートウェイが最初の `role` を検査し、バックエンドが二つ目を使えば、検査は何も守っていません。システムの入口で、重複する名前を拒否します。RFC 7493が定義するI-JSONは、それを禁じるJSONのプロファイルであり、契約でその名前を指定できます。

手で編集したファイルを解析できなくなります。 コメントと末尾のコンマは文法にないため、最初に設定ファイルへ注釈を加えたり、最後の項目の後にコンマを残したりした人が、ファイルを壊します。一部のツールは、規則を緩めた派生形式を受け入れます。そのため、同じファイルが、一つの読み手では有効でも、次の読み手では壊れます。JSONは機械が書くデータに使い、手で保守するファイルには、YAMLやTOMLなど、編集のための形式を使います。

文字化けするか、あるパーサーが受け入れるものを別のパーサーが拒否します。 RFC 8259は、システム間でUTF-8を要求し、先頭のバイトオーダーマークを禁じています。Latin-1で書く生成側や、マークを付けるエディターは、パーサー次第で扱いの変わるテキストを生みます。文法は、対になる側が欠けたサロゲートペアの片側である `"\uDEAD"` のようなエスケープも認めます。有効なUnicode文字列では保持できないものですが、ライブラリーによって扱いが異なります。生成元でUTF-8にエンコードし、システムの境界で不正な符号列を拒否します。

小さな要求一つが、パーサーを占有します。 数キロバイトの入れ子の括弧でも、再帰パーサーの処理が数千段の深さに達することがあります。大きすぎる本文は、スキーマが見る前に、全体を解析してしまいます。RFC 8259は、実装がテキストのサイズと入れ子の深さの両方を制限することを認めています。知らない相手からJSONを受け入れるAPIは、両方を使うべきです。読む前に要求サイズを制限し、解析中に深さを制限し、その後で初めて構造を検査します。

参考資料

- RFC 8259：JSONデータ交換形式 — 現在のIETFの定義です。数値、重複する名前、エンコーディングに関する注意を含みます。
- ECMA-404：JSONデータ交換構文 — 同じ文法についてのEcmaの標準で、RFC 8259が規範として参照します。

- RFC 7493：I-JSONメッセージ形式 — UTF-8を要求し、重複する名前を禁じる、より厳しいプロファイルです。
- JSON Schema仕様 — 文書に必要なプロパティーを宣言する語彙です。
- PostgreSQL：JSON型 — `json` と `jsonb` の違いと、それぞれが元のテキストから保持するものです。
- Go：json.RawMessage — デコードを遅らせたり、JSONを加工せず渡したりできる、Goの型です。

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/