Engineer

REST

🏛 リソースを中心に組み立てる、ウェブのアーキテクチャー様式

どのようなものか

RESTはRepresentational State Transferの略で、Roy Fieldingが2000年の博士論文の第5章で示したアーキテクチャー様式です。プロトコルでも標準でもありません。クライアント、サーバー、その間の仲介者がどうやり取りするかについての制約の集合で、Fieldingは各制約の利点と、その大半については代償も示しています。形容詞はRESTfulで、厳密には、すべての制約を守るシステムを指します。日常の用法では、以下の制約を守るかどうかにかかわらず、JSONをやり取りするほぼすべてのHTTP APIにまで、両方の言葉の意味が広がっています。

中心となる概念は、リソースです。Fieldingの言葉では、「名前を付けられる情報は、何でもリソースになり得ます」。文書、画像、ほかのリソースの集合、人、ある都市の現在の天気さえ含まれます。ネットワークを流れるのは、リソース自体では決してなく、その表現です。HTML文書やJPEG画像のように、バイト列と、そのバイト列を説明するメタデータを組み合わせたものです。クライアントは、リソースの現在の状態、あるいは意図する状態の表現を送受信することで、リソースを操作します。

Fieldingは、制約が何もないところから、一つずつ加えて様式を組み立てます。クライアント・サーバーは、ユーザーインターフェースとデータ保存を分離し、それぞれ独立して発展できるようにします。ステートレスは、各リクエストに理解に必要なすべてを含め、サーバーに保存した文脈に依存しないことを意味します。そのため、セッションの状態はすべてクライアント側に置きます。キャッシュは、各レスポンスについて、キャッシュ可能かどうかを明示的または暗黙的に示すことを求めます。クライアントは、後の同等のリクエストに、キャッシュ可能なレスポンスを再利用できます。後述する統一インターフェースは、すべての構成要素に共通の汎用インターフェースを与えます。階層化システムは、各構成要素が、直接やり取りする層だけを見ることを意味します。両側のインターフェースを変えずに、プロキシー、ゲートウェイ、ファイアウォールを間に置けま

す。コードオンデマンドは、サーバーが、クライアントで実行するスクリプトを送り、クライアントの機能を拡張できるようにします。Fieldingは、可視性を下げるため、これだけを任意の制約にしています。

彼は、ほかのネットワークベースの様式とRESTを区別する中心的な特徴を、統一インターフェースと呼び、四つの制約に分けています。リソースを識別すること（ウェブではURLを使います）、表現を通じてリソースを操作すること、メッセージを自己記述的にして、すべての仲介者が意味を理解できること、そして、ハイパーメディアをアプリケーション状態の駆動力とすることです。最後の制約は、人がブラウザーでリンクをたどるように、クライアントが受け取った表現から次の遷移を選ぶことを意味します。

概念を判断に生かす

「REST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

各制約は、特定の利点をもたらします。ステートレスなら、監視システムはリクエストの外を調べずに理解でき、部分的な障害からの復旧が容易になり、サーバーは応答直後にリソースを解放できます。キャッシュは、一部のやり取りを丸ごと不要にできます。階層化によって、仲介者はネットワークやマシンに負荷を分散でき、組織の境界に置いた一つのキャッシュで、内部の全員に応答できます。また、統一インターフェースは実装と提供するサービスを切り離すため、クライアントとサーバーは、相手も同時に変更しなくても、それぞれ変更できます。

Fieldingは代償についても明確です。ステートレスでは、サーバー側で文脈を共有すれば省けるデータも、リクエストごとに繰り返します。キャッシュは古いデータを返すことがあります。層を増やすたびに、処理負担と遅延が加わります。統一インターフェースは、特定のアプリケーションの必要に合わせた形ではなく、標準化した形式で情報を運ぶため、効率を下げます。Fieldingは、ウェブで一般的な、まとまったハイパーメディアデータの転送向けに設計し、それ以外のやり取りには最適ではないと明言しています。代替手法が成り立つのは、この点です。GraphQLサーバーは、単一のエンドポイントですべてのリクエストを受け、URL以外のものでエンティティーを識別し、返すフィールドをクライアント

が正確に選べるようにします。一方、gRPCは、サービス定義に宣言したメソッドを呼びます。どちらも、別の種類の通信に合わせるため、統一インターフェースの一部を手放します。

HTTP APIでは、統一インターフェースの多くが、すでに用意されています。HTTPは、意味が合意されたメソッド、ステータスコード、キャッシュや条件付きリクエスト用のヘッダーフィールドを定義しています。意味を共有しているため、途中のプロキシーやキャッシュは、そのAPI固有の事情を知らなくてもメッセージを扱えます。これらを定義どおりに使うAPIは、この仕組みを追加の実装なしで利用できます。たどるリンクの代わりに、クライアントが値を埋めるURLテンプレートを渡すAPIは、ハイパーメディアの制約を手放しています。クライアントはサーバーのURL構造に依存し、構造が変わると動作しなくなります。その結び付きをなくすために、統一インターフェースがあるのです。

実際の活用例

次のやり取りは注文を作成し、新しいリソースのアドレスを `Location` 、エンティティータグを `ETag` に入れて応答します。RFC 9110も、201レスポンスの `ETag` を例として挙げています。そのタグを後の条件付きリクエストの検証子として使えば、二つのクライアントが、気付かないまま互いの変更を上書きすることを防げます。

```
POST /orders HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"item": "notebook", "quantity": 2}

HTTP/1.1 201 Created
Location: /orders/7421
ETag: "v1"
Content-Type: application/json

{"id": 7421, "item": "notebook", "quantity": 2, "status": "open"}
```

どのメソッドにも、すべての仲介者が依拠できる意味があります。RFC 9110はGET、HEAD、OPTIONS、TRACEを安全と定義しています。クライアントは、サーバーの状態の変更を求めず、期待もしません。PUTとDELETEは、安全なメソッドとともに冪等であり、同じリクエストを複数回行っても、意図した効果は一回の場合と同じになります。POSTはどちらでもなく、RFC 5789はPATCHも同じ

としています。この二つの性質によって、クローラーは害を与えずリンクをたどれ、キャッシュは先読みでき、クライアントは応答前に接続が切れたPUTを繰り返せます。

REST APIは、汎用HTTPクライアントだけで操作できます。このサイトには、それで何が省け、何が代償になるかの両方の例があります。コードから二台目のサーバーを構築した際、ベンダーのコレクションではなく、汎用HTTPタスクで、両クラウド事業者のファイアウォールをREST APIから直接設定しました。依存関係とそのバージョンのずれをなくす代わりに、すべてのリクエストの形を手で書く必要があります。同じ実績は、二つ目の代償も挙げています。汎用HTTPタスクはチェックモードへの対応を宣言していないため、マシンを作るプレイをドライランすると、その中のすべてのタスクがスキップされます。

問題が起きるところ

GETが何かを変更する。 取得するだけで注文を取り消す `/orders/7421/cancel` のようなアドレスは、GETは安全であるという約束に反します。RFC 9110は、クローラーや先読みするキャッシュが、害を与える心配なく動けるように、この約束があると説明しています。最初にそのリンクを見つけたクローラーや先読みキャッシュが、注文を取り消してしまいます。代わりに、注文に対するPOSTや、状態に対するPATCHとして、動作をメソッドに持たせます。

すべての応答が200 OKになる。 どんな場合も200を返し、エラーかどうかも含め、実際の結果を本文に入れるAPIがあります。メッセージは自己記述的でなくなります。キャッシュ、プロキシー、監視、クライアントライブラリーは、すべてステータスコードを読み、成功と判断します。RFC 9457は、エラー用の標準本文として、 `application/problem+json` を定め、 `type` 、 `title` 、 `status` 、 `detail` などのメンバーを用意しています。 `status` メンバーは助言にすぎません。この形式を知らない汎用HTTPソフトウェアが正しく動くように、サーバーは、HTTPレスポンス自体でも同じステータスコードを送らなければなりません。

POSTの再試行で、注文が二つ作られる。 サーバーがPOSTを処理した後、クライアントが応答を読む前に接続が切れると、クライアントには注文が存在するか分かりません。RFC 9110は、実はそのリクエストが冪等だと分かっているか、最初の試行が適用されていないと判断できるのでなければ、非冪等なリクエストを自動再試行すべきではなく、プロキシーはその自動再試行をしてはならないと定めています。また、特にアイドル状態の持続的接続で、応答をまったく受信する前に接続が閉じた場合、あえてPOSTを再試行する、より危険な動作を取るク

ライアントもあると説明しています。対策の一つは、クライアントが識別子を選び、その場所へ注文をPUTすることです。PUTは冪等なので、最初のリクエストが注文を作って201を返し、繰り返しは同じ状態で置き換えるだけになります。

二つのクライアントが、互いの変更を上書きする。 両方が注文を読み、それぞれ別のフィールドを変え、全体をPUTで戻すと、二回目の書き込みが、気付かれないまま最初の変更を取り消します。RFC 9110は、これを更新の喪失と呼びます。対策は、注文を読んだときに各クライアントが受け取ったエンティティータグを、`If-Match` で送り返すことです。サーバーは、そのクライアントが最後に見た状態のままである場合にだけ、変更を適用します。違う場合は適用してはならず、412 Precondition Failedで知らせることができます。クライアントは、注文を読み直し、自分の変更を再び加え、もう一度試します。

```
PUT /orders/7421 HTTP/1.1
Host: api.example.com
If-Match: "v1"
Content-Type: application/json

{"item": "notebook", "quantity": 3, "status": "open"}

HTTP/1.1 412 Precondition Failed
```

サーバーが会話を記憶する。 一覧の現在のページや、メモリーに置いた入力途中のフォームなど、リクエスト間でクライアントごとの状態を保持するサーバーは、ステートレスの制約を捨てています。リクエストだけでは意味が分からなくなり、ロードバランサーは各クライアントを同じマシンに戻す必要があり、再起動でその状態を失います。Fieldingが挙げたステートレスの利点、すなわち可視性、部分的な障害からの復旧の容易さ、応答後にサーバーがリソースを解放できることが、そのまま失われます。状態は、URL内のページトークンのようにリクエストに含めるか、入力途中のものを下書きの注文など独立したリソースにして、どのサーバーからも読めるようにします。

参考資料

- Fielding、第5章：Representational State Transfer（REST） — 定義そのものです。各制約、それによって得られる性質、その代償を説明します。
- RFC 9110：HTTP Semantics — メソッド、安全性と冪等性、ステータスコード、`ETag` と `If-Match` による条件付きリクエストを説明します。
- RFC 5789：PATCH Method for HTTP — リソースの部分的な変更と、PATCHが安全でも冪等でもない理由です。

- RFC 9457：Problem Details for HTTP APIs — ステータスコードの意味を保つ、エラーレスポンス用の標準JSON本文です。

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

「REST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/rest/