Engineer

GraphQL

クライアントが取得するフィールドを選ぶクエリ言語

どのようなものか

GraphQLは、API向けのクエリ言語と、そのクエリをサーバーで実行するランタイムを合わせたものです。クライアントは欲しいフィールドを正確に指定した文書を送り、サーバーは同じ形のデータを返します。仕様は、クライアントとサーバーからなるアプリケーションのための言語と実行エンジンと説明します。2012年に生まれ、2015年にオープン標準としての作業が始まり、2019年からはJoint Development Foundation傘下のGraphQL Specification Projectの成果物になったと記録しています。最新は2025年9月版です。

出発点は、サービスが公開するスキーマです。名前付きのフィールドを持つ型の集合で、各フィールドにも型があります。組み込みのスカラー型は `Int` 、 `Float` 、 `String` 、 `Boolean` 、 `ID` で、UUID用など、独自のスカラーも追加できます。 `!` を付けなければ、すべての型はnullを許します。 `String!` は決してnullになりません。リストは角括弧で書き、 `[LineItem!]!` なら、必ず存在し、nullの要素を持たないリストです。スキーマ自体にも問い合わせられます。クライアントはイントロスペクションのフィールド `__schema` と `__type` で、サービスの提供するものを尋ねられ、開発ツールやクライアントライブラリーも、これを基に作っています。

操作は三種類あります。クエリは、仕様の言葉で「読み取り専用の取得」、ミューテーションは、書き込みに続く取得、サブスクリプションは、一連の出来事に応じて時間とともにデータを届ける、長く続くリクエストです。サーバーでは、各フィールドに、その値を作る、一般にリゾルバーと呼ぶ関数があります。サービスは、まずスキーマに照らして操作を検証し、その後で、選ばれたフィールドのリゾルバーだけを呼びます。ミューテーションの最上位フィールドは、並列ではなく順番に実行するため、同じリクエストの二つの書き込みが競合しません。

仕様は、転送方法もシリアライズ形式も定めません。最も一般的なのはHTTPで、GraphQL APIは普通、末尾が `/graphql` の一つのURLで提供し、最も一般的な形式のJSONで答えます。クライアントは、JSON本文に `query` の文字列と、必要なら `variables` を入れてPOSTを送ります。サーバーはクエリならGETも受け付けられますが、ミューテーションは必ずPOSTです。結果は `data` 、問題があれば `errors` へ入ります。この分野のGraphQL Foundationのガイドは、まだ草案のGraphQL-over-HTTP仕様に従います。

概念を判断に生かす

「GraphQL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

大半のHTTP APIでは、各エンドポイントが返すものをサーバーが決めます。注文の状態と顧客名が欲しい画面は、求めたより多い、注文全体と顧客全体を受け取るか、リソースごとにリクエストし、応答を自分でつなぐ必要があります。仕様は、この二つを過剰取得と取得不足と呼びます。GraphQLは関係を逆にします。クライアントはフィールド単位で求め、そのフィールドだけを受け取ります。注文から明細、顧客へと、一つのリクエストで関係もたどれます。リクエストが、受け取るデータと同じ形だからです。

型システムは、二つの面で役立ちます。リゾルバーを一つでも動かす前に、すべての操作をスキーマで検証するため、フィールド名の誤記や型の違う引数があれば、何も実行せず、操作全体を拒否します。ツールも、コードを書いているうちに同じ誤りを見つけられます。また、クライアントは選んだフィールドしか受け取らないため、フィールドを追加しても、既存のクライアントに負担をかけません。そのため、バージョン番号なしでAPIを発展させられます。古いものの隣に新しいフィールドを加え、古いものは理由付きの `@deprecated` で印を付けます。既存のクライアントを壊さないよう、有効なまま保ち、使うクライアントがなくなってから消します。

こうした仕組みには実際に代償があり、その大半はHTTP本来の仕組みを外れることから生まれます。個々の対象をURLで識別せず、すべて一つのアドレスへ、普通はPOSTで送ります。一方、既定でキャッシュ可能と扱うのは、一般にGET

です。REST APIが中継サーバーから自然に得るキャッシュを、意図して作り直す必要があります。データとエラーが一緒に返っても、HTTPでは成功になるため、状態コードだけでは全体を語れません。クライアントは、サーバーが見たことのないリクエストを構成します。一件で大量の仕事を求められるため、その負荷からサーバーを守るのはサービスの仕事で、GraphQLが代わりにするものではありません。フィールドごとのリゾルバーは簡単に書けますが、単純に実装すると、背後のデータベースへの問い合わせが多くなります。

実際の活用例

明細と顧客を持つ注文について、スキーマのごく一部を示します。

```
type Query {
  order(id: ID!): Order
}

type Order {
  id: ID!
  status: String!
  items: [LineItem!]!
  customer: Customer
}

type LineItem {
  name: String!
  quantity: Int!
}

type Customer {
  name: String!
}
```

注文の状態、明細、顧客名を表示する画面は、この操作を `/graphql` へPOSTの本文で送り、変数に `{"id": "7421"}` を渡します。

```
query OrderSummary($id: ID!) {
  order(id: $id) {
    status
    items {
      name
      quantity
    }
    customer {
      name
```

```
    }
  }
}
```

注文と明細は解決できても、`customer`の背後のサービスが停止しているとします。応答は、作れたものをすべて含みます。失敗したフィールドはnullで、`errors`の項目が、何が問題で、文書のどこで起き、応答のどの経路に影響するかを示します。8行目の5列目が`customer`です。

```
{
  "data": {
    "order": {
      "status": "open",
      "items": [{ "name": "notebook", "quantity": 2 }],
      "customer": null
    }
  },
  "errors": [
    {
      "message": "Customer service unavailable",
      "locations": [{ "line": 8, "column": 5 }],
      "path": ["order", "customer"]
    }
  ]
}
```

`customer`はnullを許す宣言なので、影響はそこで止まります。もし`Customer!`なら、エラーは親へ伝わり、注文全体がnullで返ります。また、`data`はnullではないため、エラーがあってもサーバーは2xxで答えます。HTTPには部分的な成功の状態コードがないためです。状態コードしか見ないクライアントは、これを完全な応答と扱います。

反対に、応答の形をすべてサーバーが決める設計を、このサイトの事例の一つで作りました。PostgresからAPIへJSONをそのまま通す実装では、PostgreSQLの関数が応答全体をJSONに組み立て、Goのハンドラーは変更せず転送します。そのため、関数の戻すものを変えれば、ハンドラーを一行も編集せずに、クライアントの受け取るものが変わります。そこではフロントエンドが、データベースの与える形へ直接依存します。GraphQLではクライアントがフィールドを指定し、サーバーは、スキーマが許す、どの組み合わせでも解決できる必要があります。

問題が起きるところ

一つのフィールドで、データベースを千回呼びます。 リゾルバーはフィールドごとの小さい関数で、単純な実装では、オブジェクトを一つずつ処理します。各注文の顧客も含む一覧なら、一覧用に一回、各注文にさらに一回のデータベース呼び出しが必要です。GraphQL FoundationのガイドはN+1問題と呼び、まとめて処理する方法を示します。短い時間枠に要求された読み込みを集め、FacebookのDataLoaderなどで、一つのリクエストとしてデータベースへ送ります。

一つのリクエストが、すべてを求めます。 操作を書くのはクライアントなので、一文書で友人の友人を何層も深くたどったり、重いフィールドを百個の別名で繰り返したり、多くの操作を一つのリクエストへ詰めたりできます。どれも有効なGraphQLで、どれもサーバーを占有しかねません。防御はサービス側で行います。リストのフィールドをページ分割し、実行前に深さや幅の上限を超える操作を拒否し、別名の数とバッチの大きさを制限します。さらに細かくするなら、各フィールドへ重みを与え、見積もった負荷が予算を超える操作を拒否します。クエリ複雑度解析と呼ぶ手法ですが、仕様自体は、その規則を定めません。自社のクライアントだけに提供するAPIなら、事前に保存してハッシュで探す操作だけを実行する、信頼済み文書の方式で、さらに制限できます。

200 OKが失敗を隠します。 上の例のように、エラーを含む応答も2xxになり得ます。古いメディア型 `application/json` について、ガイドはさらに、正しいリクエストを実行できなかった場合でも2xxを勧め、一部の古いサーバーは、検証エラーでも2xxを返します。クライアントと周囲の監視は、状態だけでなく `errors` を読む必要があります。新しいメディア型 `application/graphql-response+json` なら、正しいリクエストが実行できない場合は4xxか5xxとなり、状態が再び意味を持ちます。クライアントは `Accept` ヘッダーで、この形式を求めるべきです。

キャッシュが機能しなくなります。 どの操作も同じURLへのPOSTなので、CDNを含む、クライアントとサーバーの間のキャッシュからは、一つのアドレスと、既定ではキャッシュしないメソッドに見えます。クエリをGETで送れば役立ちますが、長い文書はブラウザーやCDNが許すURL長を超える場合があります。永続化クエリは両方を解決します。クライアントはハッシュを送り、サーバーは文書全体を引き出してから、検証、実行します。HTTPで応答をキャッシュするなら、機微な情報を含む応答はプライベートキャッシュに限って保存するか、まったくキャッシュしない必要があります。

フィールドが消えます。 クライアントは一つずつ選ぶため、フィールドの削除や改名は、それを選ぶすべてのクライアントを壊します。軽い影響では済みません。存在しないフィールドを指定した操作は、全体が検証に失敗し、データは何

も返りません。仕様が非推奨のフィールドも有効に保つのは、既存のクライアントを壊さないためです。まず代わりを追加し、新しいものを示す理由を付けて、古いものを `@deprecated` にします。どのクライアントも選ばなくなってから、古いフィールドを削除します。

参考資料

- GraphQL仕様の各版 — 2015年7月から2025年9月版までの全版と、作業中の草案です。言語、型システム、検証、実行、エラーを扱います。
- HTTPでの提供 — 単一エンドポイント、POSTとGET、二つの応答メディア型、それぞれの状態コードの規則です。
- 性能 — クエリのGET送信、永続化クエリ、N+1問題、バッチ処理です。
- セキュリティ — 信頼済み文書、深さ、幅、バッチの制限、クエリ複雑度解析、および本番でのイントロスペクションとエラーの詳細です。

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

「GraphQL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/graphql/