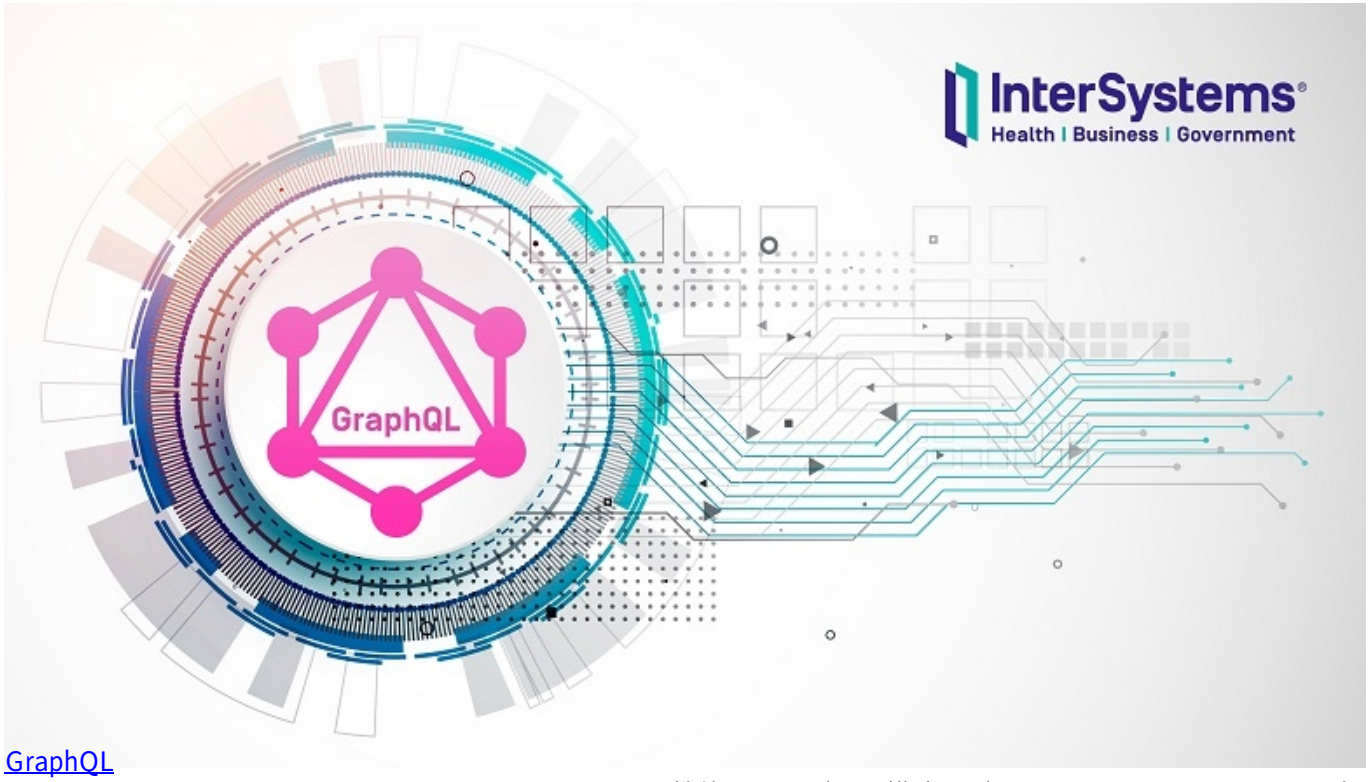
記事
Toshihiko Minamoto · 2020年10月13日 8m read

InterSystemsデータプラットフォームのGraphQL



GraphQL
は、クライアントとサーバー間のミドルウェア層として機能する、データ構造とデータアクセスのメソッドを選択するための標準です。
GraphQLについて聞いたことがない方は、こことこことここ
にある、有用なオンラインリソースをご覧ください。

この記事では、InterSystemsテクノロジーに基づいて、プロジェクトでGraphQLを使用する方法を説明します。

InterSystemsプラットフォームは現在、クライアント/サーバーアプリケーションを作成する方法をいくつかサポートしています。

- REST
- WebSocket
- SOAP

では、GraphQLのメリットは何でしょうか。
たとえばRESTと比較した場合、どのようなメリットが提供されるのでしょうか。

GraphQLのリクエストにはいくつかの種類があります。

- **クエリ** - RESTを使用してデータをフェッチするために推奨されるGETリクエストに類似する、データ取得用のサーバーリクエスト。
- **ミューテーション** -
RESTのPOST（PUT、DELETE）リクエストに類似する、サーバー側データ変更を担う。 ミューテーションとクエリはともにデータを返すことができます。ミューテーションを実行した直後に、サーバーに更新

済みのデータをリクエストする場合に便利です。
- **サブスクリプション** - データを出力する同じクエリタイプ。 唯一の違いは、ミューテーションによってサブスクリプションがアクティブ化される間に、クライアント側でレンダリングされたページでクエリが発行されることです。

GraphQLの主な機能とメリット

どのデータが返されるかはクライアント次第

GraphQLの主な機能の1つに、返されるデータの構造とボリュームがクライアントアプリケーションによって定義されることがあります。 クライアントアプリケーションは、JSON
形式によく似た宣言的なグラフのような構造を使用して、受信するデータを指定します。
レスポンス構造は、クエリの構造に対応しています。

簡単なGraphQLクエリは次のようになります。

```
{
  Sample_Company {
    Name
  }
}
```

JSON形式のレスポンス:

```
{
  "data": {
    "Sample_Company": [
      {
        "Name": "CompuSoft Associates"
      },
      {
        "Name": "SynerTel Associates"
      },
      {
        "Name": "RoboGlomerate Media Inc."
      },
      {
        "Name": "QuantaTron Partners"
      }
    ]
  }
}
```

単一エンドポイント

GraphQLを使ってデータを操作する場合、必ず単一の**エンドポイント**
であるGQLサーバーに接続し、クエリの構造、フィールド、パラメーターを変更して異なるデータを取得します。 RESTは、それとは対照的に、複数のエンドポイントを使用します。

簡単な例を使用して、RESTとGraphQLを比較してみましょう。

ユーザーのコンテンツを読み込む必要があるとします。
RESTを使用している場合、サーバーに3つのクエリを送信する必要があります。

1. IDでユーザーのデータを取得する
2. ユーザーのIDを使用してその投稿を読み込む
3. ユーザーのIDを使用してフォロワー/サブスクライバーのリストを取得する

以下は、上記のクエリに対応するRESTマップです。

```
<Route Url="/user/:id" Method="GET" Call="GetUserByID"/>
<Route Url="/user/:id/posts" Method="GET" Call="GetUserPostsByID"/>
<Route Url="/user/:id/followers" Method="GET" Call="GetUserFolloversByID"/>
```

新しいデータセットを取得するには、このRESTマップを新しいエンドポイントで更新する必要があります。

GraphQLはこれを1つのクエリで処理します。 リクエストの本文に次のように指定してください。

```
{
operationName: null,    //a query can have a name  ( query TestName(...){...} )
query: "query {
       User(id: "ertg439frjw") {
        name
        posts {
           title
        }
        followers(last: 3) {
           name
        }
    }
}",
variables: null      // initialization of the variables used in the query
}
```

このクエリに対応するRESTマップ:

```
<Route Url="/graphql" Method="POST" Call="GraphQL"/>
```

これがサーバーの唯一のエンドポイントです。

GraphQLとGraphiQLのインストール

GraphQLを使用し始めるには、いくつかの手順を完了する必要があります。

1. GitHubから最新リリースをダウンロードし、必要なネームスペースにインポートします。
2. システム管理ポータルに移動し、InterSystemsデータプラットフォーム製品（Caché、Ensemble、またはIRIS）に基づいて、新しいWebアプリケーションを作成します。
   - 名前 - **/**
   - ネームスペース - **SAMPLESなど**
   - ハンドラクラス - **GraphQL.REST.Main**
3. GraphiQL — GraphQLクエリをテストするためのシェルです。 最新ビルド、または独自のソースのビルドをダウンロードします。
4. 新しいWebアプリケーションを作成します。
   - 名前 - **/graphiql**
   - ネームスペース - **SAMPLESなど**

    - CSPファイルへの物理パス - **C:\InterSystems\GraphiQL**

結果を見てみましょう

ブラウザで、http://localhost:57772/graphiql/index.html（localhost — サーバー、57772 — ポート）に移動します。

クエリと**レスポンス**のネームスペースについて明確に説明できたでしょうか。 **スキーマ**
は、ネームスペースに格納されているすべてのクラスに対して生成されるドキュメントです。

スキーマには次の項目が含まれます。

- クラス
- プロパティ、引数、それらの型
- コメントから生成される上記すべての説明

SampleCompany クラスのスキーマを詳しく見てみましょう。

GraphiQLは、**Ctrl + Space**キーを押してアクティブ化できる自動コード補完機能もサポートしています。

クエリ

クエリは、単純なクエリや、複数のデータのセットで使用する複雑なクエリであったりあします。
以下は、**SamplePerson** と**SampleCompany** という異なるクラス間でのサンプルクエリです。

フィルタリング

現在のところ、厳密な等価のみがサポートされています。

ページネーション

ページネーションは、必要な結果を得るために組み合わせて子よできる4つの関数を通じてサポートされています。

- **after: n** – nより大きいidを持つすべてのレコード
- **before: n** – nより小さいidを持つすべてのレコード
- **first: n** – 最初のn件のレコード
- **last: n** – 最後のn件のレコード

可視領域

ほとんどの場合、アプリケーションのビジネスロジックは、特定クライアントのみに特定のネームスペースクラスへのアクセスを与えています（ロールベースの許可）。
それに基づき、クライアントのクラスの可視性を制限する必要がある場合があります。

- ネームスペースのすべてのクラス（**GraphQL.Scope.All**）
- スーパークラスから継承されたクラス（**GraphQL.Scope.Superclass**）
- 特定のパッケージに属するクラス（**GraphQL.Scope.Package**）

可視性制限のメソッ
ドを変更するには、Studioを開き、必要なネームスペースに切り替えて、**GraphQL.Settings**クラスを開きます。
これには、**GraphQL.Scope.All**というデフォルト値を使用する**SCOPECLASS**
パラメーターがあります。これがネームスペースのクラスの可視性制限の説明が含まれるクラスです。

クラスの可視性制限を変更するには、上記に
示されるいずれかの値（**GraphQL.Scope.Package**または**GraphQL.Scope.Superclass**
）に設定する必要があります。

GraphQL.Scope.Packageを選択した場合、そのクラスに移動して、**Package**
パラメーターの値を必要なパッケージの名前（**Sample**など）に変更する必要があります。
これにより、このパッケージのすべての格納済みクラスを完全に利用できるようになります。

GraphQL.Scope.Superclassを選択した場合は、もう一度必要なクラスで、このクラスから継承します。

現在のサポート状況

クエリ:

- 基本
- 埋め込みオブジェクト
    - 多対1の関係のみ
- 単純な型のリスト
- オブジェクトのリスト

現在開発中

クエリ:

- 埋め込みオブジェクト
    - すべての型のリレーションのサポート
- フィルタリング
    - 不等価のサポート

今後の予定

- ミューテーション
- エイリアス
- ディレクティブ
- フラグメント

→ プロジェクトリポジトリへのリンク → デモサーバーへのリンク

ぜひプルリクエストを発行してください。 今後のプロジェクト情報にご期待ください！

#API #IRIS Analytics Architect #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/intersystems%E3%83%87%E3%83%BC%E3%82%BF%E3%83%97%E3%83%A9%E3%83%83%E3%83%88%E3%83%95%E3%82%A9%E3%83%BC%E3%83%A0%E3%81%AEgraphql