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

InterSystemsデ-タプラットフォ-ムのGraphQL

GraphQL
は、ｸﾗｲｱﾝﾄとｻｰﾊﾞｰ間のミドルウェア層として機能する、デ-タ構造とデ-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-スをご覧ください。

この記事では、InterSystemsﾃｸﾉﾛｼﾞｰに基いて、ﾌﾟﾛｼﾞｪｸﾄでGraphQLを使用する方法を説明します。

InterSystemsﾌﾟﾗｯﾄﾌｫｰﾑは現在、ｸﾗｲｱﾝﾄ/ｻｰﾊﾞｰｱﾌﾟﾘｹｰｼｮﾝを作成る方法をいくつかｻﾎﾟｰﾄしています。

- 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-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-タ取得用のｻｰﾊﾞｰﾘｸｴｽﾄ。

- **ミュ-テ-ション** - RESTのPOST(PUT、DELETE)リクエストに類似する、サ-バ-側デ-タ変更を担う。ミュ-テ-ション クエリはただちにデ-タを返すことができます。ミュ-テ-ションを実行した直後に、サ-バ-に更新済みのデ-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-タを出力する同じクエリタイプ。唯一の違いは、ミュ-テ-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-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-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-タの構造とボリュ-ムがクライアントアプリケ-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-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-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
  }
}
```

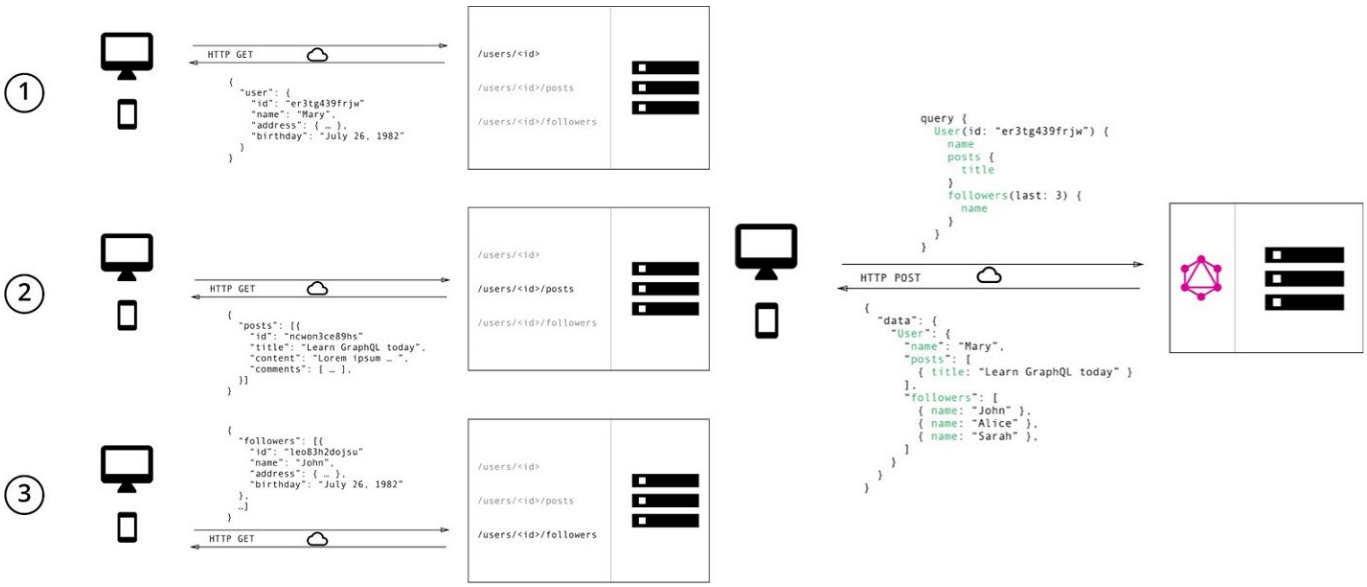
JSON形式のレスポンス:

```
{
  "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-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-バ-に接続し、クエリの構造、フィ-ルド、パラメ-タ-を変更して異なるデ-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

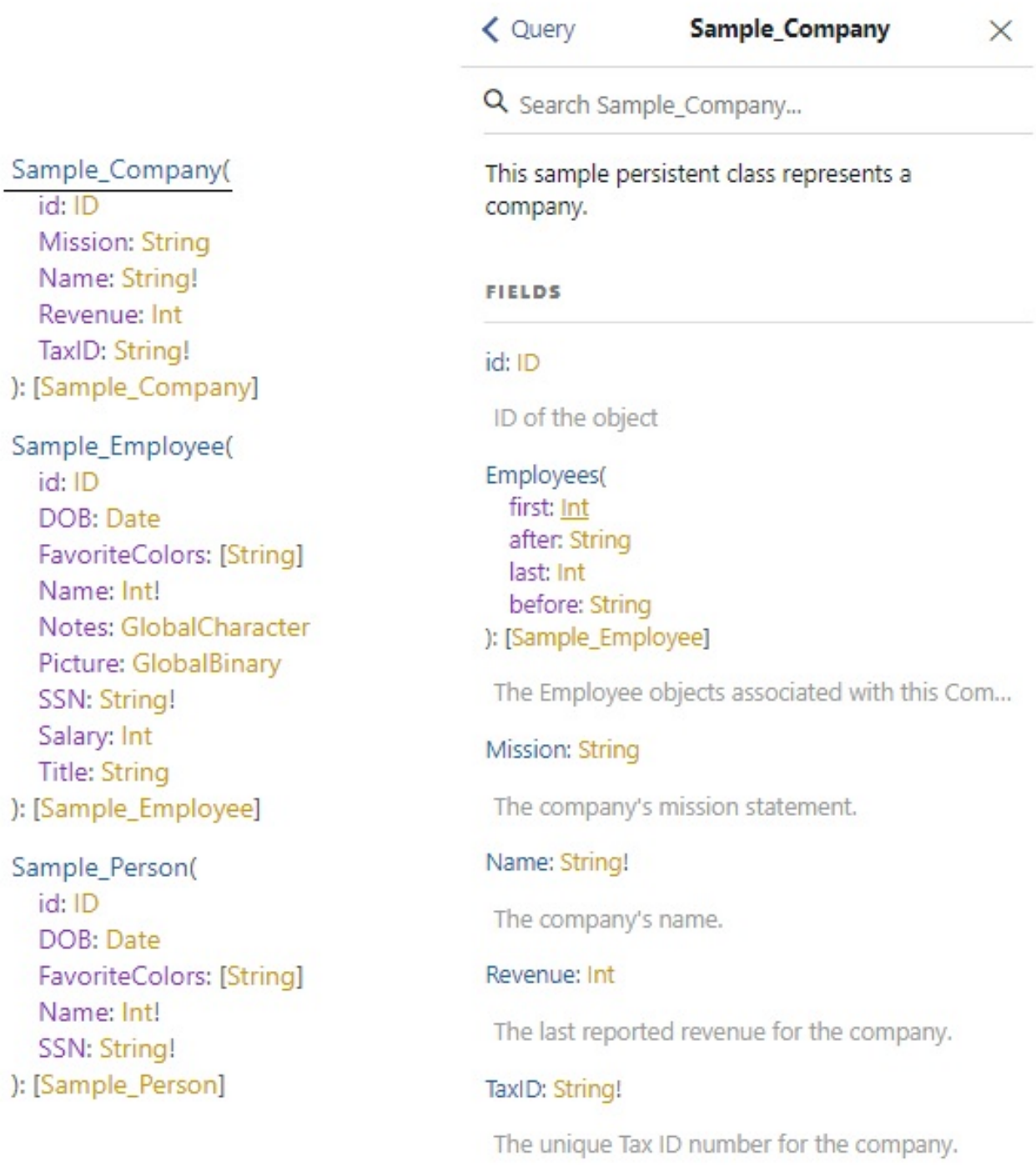
ブラウザで、 http://localhost:57772/graphiql/index.html（localhost — サーバー、57772 — ポート）に移動します。

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ましょうか。 **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。

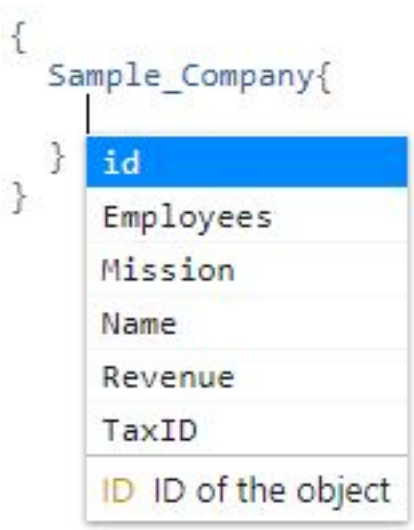
スキーマには次の項目が含まれます。

- 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

Sample_Company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ﾌﾞﾙできる自動ｺｰﾄﾞ補完機能をｻﾎﾟｰﾄしています。



ｸｴﾘ

ｸｴﾘは、単純なｸｴﾘや、複数のﾃﾞ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、 **Sample_Person**と **Sample_Company**という異なるｸﾗｽ間でのｻﾝﾌﾟﾙｸｴﾘです。

```
{
  Sample_Person{
    Name
    DOB
    FavoriteColors
    Office {
      City
      State
      Street
      Zip
    }
  }

  Sample_Company{
    Mission
    Name
    Revenue
  }
}
```

```
{
  "data": {
    "Sample_Person": [
      {
        "Name": "Solomon,Natasha P.",
        "DOB": "1945-04-09",
        "FavoriteColors": [],
        "Office": {
          "City": "Vail",
          "State": "CA",
          "Street": "6531 Elm Avenue",
          "Zip": 21935
        }
      }
    ],
    "Sample_Company": [
      {
        "Mission": "Post-sale services for virtual seven-sigma database services for the Fortune 5.",
        "Name": "CompuSoft Associates",
        "Revenue": 539036191
      }
    ]
  }
}
```

ﾌｨﾙﾀﾘﾝｸﾞ

現在のところ、厳密な等価のみがｻﾎﾟｰﾄされています。

```
{
  Sample_Person(id: 116){
    id
    Name
  }
}
```

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使用できる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

```
{
  Sample_Employee(after: 120, before: 123){
    id
    Name
  }

  Sample_Person(first: 2){
    id
    Name
    Home {
      City
      State
      Street
      Zip
    }
  }

  Sample_Company(last: 2){
    id
    Name
  }
}
```


```
{
  "data": {
    "Sample_Employee": [
      {
        "id": 121,
        "Name": "Brown,Greta T."
      },
      {
        "id": 122,
        "Name": "Donaldson,Umberto Y."
      }
    ],
    "Sample_Person": [
      {
        "id": "1",
        "Name": "Solomon,Natasha P.",
        "Home": {
          "City": "Fargo",
          "State": "CO",
          "Street": "9257 Maple Place",
          "Zip": 16298
        }
      },
      {
        "id": "2",
        "Name": "Quince,Lydia S.",
        "Home": {
          "City": "Tampa",
          "State": "SC",
          "Street": "7517 Madison Blvd",
          "Zip": 36215
        }
      }
    ],
    "Sample_Company": [
      {
        "id": "20",
        "Name": "XenaMatix Media Inc."
      },
      {
        "id": "19",
        "Name": "SynerNet Group Ltd."
      }
    ]
  }
}
```


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。

ｸﾗｽの可視範囲を変更するには、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**
ﾊﾟﾗﾒｰﾀの値を必要なﾊﾟｯｹｰｼﾞの名前（ **Sample**など）に変更する必要があります。
これにより、このﾊﾟｯｹｰｼﾞ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。

```
Parameter Package = {"Sample"};
```

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。

```
/// This sample persistent class represents a person.
/// <p>Maintenance note: This class is used by some of the bindings samples.
Class Sample.Person Extends (%Persistent, %Populate, %XML.Adaptor, GraphQL.Scope.Superclass)
{
```

現在のｻﾎﾟｰﾄ状況

ｸｴﾘ:

- 基
- 埋め込みｵﾌﾞｼﾞｪｸﾄ
  - 1対1の関係のみ
- 単純な型のﾘｽﾄ
- ｵﾌﾞｼﾞｪｸﾄのﾘｽﾄ

現在開発中

ｸｴﾘ:

- 埋め込みｵﾌﾞｼﾞｪｸﾄ
  - すべての型のﾘﾚｰｼｮﾝのｻﾎﾟｰﾄ
- ﾌｨﾙﾀﾘﾝｸﾞ
  - 不等価のｻﾎﾟｰﾄ

今後の予定

- ミュｰﾃｰｼｮﾝ
- エイリアス
- ディｸﾚﾃｨﾌﾞ
- フラグメント

→ ﾌﾟﾛｼﾞｪｸﾄﾘﾎﾟｼﾞﾄﾘへの リンｸ → ﾃﾞﾓｻｰﾊﾞｰへの リンｸ

ぜひﾌﾟﾙﾘｸｴｽﾄを発行してください。今後のﾌﾟﾛｼﾞ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