Бібліотека graphql-java як реалізація специфікації GraphQL.

Способи задання GraphQL схеми. Витягування даних в graphql-java: DataFetchers та DataFetchingEnvironment. Виконання запитів в graphql-java

Стельмащук Віталій Володимирович

Львівський національний університет імені Івана Франка
Кафедра інформаційних систем

17 лютого 2025 р.

Outline

Application of graphql-java

GraphQL schema in graphql-java

Fetching data

Execution of GraphQL

Examples

Application of graphql-java

https://www.graphql-java.com/documentation/getting-started

What graphql-java is/does:

- Java implementation of GraphQL specification.
- Provides the core GraphQL engine to define schemas, queries, mutations, and subscriptions.
- Works as a standalone library or can be integrated into frameworks like Spring Boot's Spring for GraphQL.

What graphql-java is/does not:

- HTTP transfer
- JSON encoding
- Data pagination
- Data authorisation
- Caching data
- Database access

GraphQL schema definition

There are 2 possible ways to define GraphQL schema in graphql-java:

- Programmatically as Java code

```
GraphQLObjectType queryType = newObject()
        .name("Query")
        .field(newFieldDefinition()
                        .name("hello")
                        .type(GraphQLString))
        .build();
```

- Via a special graphql DSL called SDL (Schema definition language) **(Recommended way)**

```
type Query {
        hello: String
}
```

GraphQL schema using SDL. E-commerce example

```
type Query {
        user(id: ID!): User
        products: [Product!]
}
type User {
        id: ID!
        name: String!
        email: String!
        cart: [CartItem!]
}
type CartItem {
        product: Product!
        quantity: Int!
}
type Product {
        id: ID!
        name: String!
        price: Float!
}
```

GraphQL schema using SDL

- Static schema definition file `e-commerce.graphqls` contains the field and type definitions
- `SchemaParser` parses this file into `TypeDefinitionRegistry` object
- `RuntimeWiring` object has to be built to define dynamic behaviour of the fields and therefore make the schema executable
- Create executable schema `GraphQLSchema` object using `SchemaGenerator` class out of `TypeDefinitionRegistry` and RuntimeWiring objects:
- Create `GraphQL` object as an entry point for the created GraphQL engine

GraphQL schema using Java code

- Define several `GraphQLObjectType` objects:

```
1   GraphQLObjectType queryType = newObject()
2                   .name("Query")
3                   .field(newFieldDefinition()
4                                   .name("hello")
5                                   .type(GraphQLString))
6                   .build();
```

- Define dynamic behaviour using `GraphQLCodeRegistry`:
- Define `GraphQLSchema` object using defined types and `GraphQLCodeRegistry` object
- Create `GraphQL` object as an entry point for the created GraphQL engine

Data Fetchers

- Each field has a `graphql.schema.DataFetcher` associated with it
- Sometimes data fetchers are called "resolvers"in other GraphQL implementations
- `graphql.schema.PropertyDataFetcher` is the default data fetcher for fields in graphql-java and will use standard patterns for fetching object field values:
  - `Map` approach
  - POJOs
- In most cases the data fetcher is more complex: get data from database, REST API, etc.
- Interface `graphql.schema.DataFetcher` allows us to define our own implementation of data fetcher

DataFetcher interface. DataFetchingEnvironment context object

```
1  public interface DataFetcher<T> {
2          T get(DataFetchingEnvironment environment) throws
           ↪  Exception;
3  }
```

`DataFetchingEnvironment` context object contains information about:

- what field is being fetched
- what arguments have been supplied to the field
- field's type
- its parent type
- the query root object
- etc.

Queries execution

- The overloaded method `execute` of `GraphQL` class
- `ExecutionInput` and `ExecutionResult`
- `ExecutionResult` has to be turned into a JSON payload via JSON serialisation library like Jackson or GSON.
- `ExecutionResult.toSpecification` method should be called before sending this object to Jackson or GSON to ensure the result will conform to the GraphQL specification.

Execution strategies

- A class derived from `graphql.execution.ExecutionStrategy` is used to run a query or mutation.
- Various strategies provided in graphql-java and you can even write your own.
- Queries are being executed using `graphql.execution.AsyncExecutionStrategy` (fully parallel execution).
- According to GraphQL specification, **mutations MUST be executed serially and in the order in which the query fields occur.** Therefore, `graphql.execution.AsyncSerialExecutionStrategy` is used by default for mutations.
- For GraphQL subscriptions `graphql.execution.SubscriptionExecutionStrategy` is used as it has support for the reactive-streams APIs.

Examples

- "Hello world"example
- "E-commerce"example

Thank you for your attention!

Questions?