GraphQL Resolvers

GraphQL resolvers are the collection of functions that are used to populate the data for a single field in your schema. Resolvers specify a way how Apollo Server processes GraphQL operations. When we use the Apollo server for communication, it needs to know how to populate data for every field in your schema because it has to respond to the request for that specific data. Here, we use resolver to make this communication possible.

In simple words, we can say that resolvers are used to handle GraphQL queries. Apollo Server automatically defines a default resolver if the user doesn't define a resolver for a particular field.

Every resolver function accepts the following four positional arguments in a [GraphQL](https://www.javatpoint.com/graphql) schema:

Syntax:

1. fieldName:(root, args, context, info) => { result }

**We can specify a resolver function in different ways:**

For example:

1. //A resolver function without parameters and a returning string
2. hello:() => {
3. **return** "Welcome to GraphQL..."
4. }
5. //A resolver function without parameters and returning list
6. employees:() => db.employees.list()
7. //A resolver function with arguments and returning object
8. employeeById:(root,args,context,info) => {
9. **return** db.employees.get(args.id);
10. }

Parameter Explanation

* **root/parent:** This is used to specify the object that contains the result returned from the resolver on the parent field. When the rootValue function is passed to Apollo Server's constructor, this value is returned.
* **args:** It is used to specify an object that contains all GraphQL arguments passed into the field in the query. For example: If you execute a query { user(id: "3") }, it shows that args object passed to the user resolver is { "id": "3" }.
* **context:** It is used to specify an object that is shared by all resolvers in a particular query. It is used to share authentication information, dataloader instances, and anything else to make it easy tracking across resolvers.
* **info:** It is used to contain all information about the operation's execution state or execution state of the query, including the field name, path to the field from the root, and more. The core fields of info are found in the GraphQL.js source code. These fields can be extended with additional functionality by other modules, like apollo-cache-control.

GraphQL Resolvers Result format/ Returned values

The GraphQL returned values, or result format is treated differently by Apollo Server according to its type:

* **Scalar / object:** A resolver can return a single value or an object, according to how the resolver is defined. You can pass this result or return value down to any nested resolvers via the root argument.
* **Array:** It returns an array if and only if your schema shows that the associated field of resolver contains a list. After that, the Apollo server executes nested resolvers for every item of the array.
* **null / undefined:** The null or undefined field specifies that the value for the field could not be found. In the case, your schema shows that the resolver's field is nullable, the operation result has a null value at the field's position. If it shows that the resolver's field is not nullable, Apollo Server sets the field's parent to null. This process repeats the resolver chain continuously until it gets a nullable field. This shows that a response can never include a null value for a non-nullable field.
* **Promise:** GraphQL resolvers can return promises. Resolvers perform asynchronous actions like fetching from a database or backend API. So, loading from a database is an asynchronous operation, this returns a Promise.

Example

Let's create a simple example to see how the resolver works. In this example, we shall create a schema to query the details of an employee by id from the server. The employee data will be stored in a flat-file, and we shall read data from the flat file by using a node module called **notarealdb**

Follow the previous steps defined in earlier pages like GraphQL first example or GraphQL schema file i.e. download and install required dependencies for the project and then create a schema as defined earlier.

Create Resolver

Create a file **resolvers.js** in the main project folder and use the following code within it:

1. **const** db = require('./db')
2. **const** Query = {
3. //resolver function for greeting
4. hello:() => {
5. **return** "Welcome to GraphQL..."
6. },
7. //resolver function for employeebyId
8. employeeById:(root,args,context,info) => {
9. //args will contain parameter passed in query
10. **return** db.employees.get(args.id);
11. }
12. }
13. module.exports = {Query}

Run the Server

Now, run the server as we did in previous pages and open the browser and enter the url, http://localhost:4000/graphiql.

**Use the following query in the editor:**

1. {
2. hello
3. employeeById(id:"E1001")
4. {
5. id
6. firstName
7. lastName
8. }
9. }