**Two Ways to Script and Modularize GraphQL Schema**

* [**GraphQL**](http://tjcccc.github.io/all.html?tag=GraphQL)
* [**Schema**](http://tjcccc.github.io/all.html?tag=Schema)

Mar 07, 2019

**Two ways to write and modularize GraphQL Schema**

I learned about GraphQL last week. I think the difficulty of getting started is to write Schema, follow YouTube tutorial videos and documents, and learn two ways, one is written in the Schema syntax of GraphQL, and the other is written in the GraphQLSchema constructor.

**Using GraphQL Schema Language**

Built-in Schema syntax is written in the template string, by the buildSchema()creation.

For example, there are two data types: Userand Post, you can query User and Posts, you can create User and Post:

**const** express **=** require('express');

**const** graphqlHTTP **=** require('express-graphql');

**const** { buildSchema } **=** require('graphql');

**var** rootSchema **=** buildSchema(`

type User {

id: String

name: String

}

type Post {

id: String

title: String

content: String

}

input PostInput {

title: String,

content: String

}

type RootQuery {

user(id: String): User,

posts: [Posts]

}

type RootMutation {

createUser(name: String): User,

createPost(newPost: PostInput): Post

}

`);

**const** rootResolvers **=** {

user: (id) **=>** {

**return** user;

},

posts: () **=>** {

**return** posts;

},

createUser: (name) **=>** {

**return** user;

}

createPost: (args) **=>** {

**return** post;

}

};

**const** app **=** express();

app.use('/graphql', graphqlHTTP({

schema: rootSchema,

rootValue: rootResolvers,

graphiql: **true**,

}));

app.listen(4000);

rootResolversIn userand postsis a method of inquiry, and the method name to schemathe RootQuerysame, the former is the realization, which is similar to a registration statement. createUserAnd createPostthe data is changed, it should be placed RootMutationin the definition. For createPostparameter input defines a new type PostInput, also written in the schemamiddle, with a inputdefinition, if the input parameters and more, can use this method.

rootResolvers The specific content of the method is omitted.

Look again:

app.use('/graphql', graphqlHTTP({

schema: rootSchema,

rootValue: rootResolvers,

graphiql: **true**,

}));

The definition of good rootSchemaand rootResolversfilling into, GraphQL service is ready for use.

**Modularize for Schema Language**

The above examples are simple and straightforward, but they are not suitable for general serious projects. Because data can't be so small, are all the definitions and methods written in one file? Must be modular.

First rootSchemaand rootResolversto be divided into two files now. Then continue to subdivide. rootSchemaCan be divided userSchema, and postSchematwo kinds. With postSchemaan example:

module.exports.postSchema **=** `

type Post {

id: String

title: String

content: String

}

input PostInput {

title: String

content: String

}`;

module.exports.postQuery **=** `posts: [Post]`;

module.exports.postMutation **=** `createPost(newPost: PostInput): Post`

Because schemaactual Query, Mutation consists of three definitions Type,, so that each schemamodule is preferably three defined content.

Well all the definitions in the rootSchemaintegration:

**const** { userSchema, userQuery, userMutation } **=** require('./user.schema');

**const** { postSchema, postQuery, postMutation } **=** require('./post.schema');

**const** schemas **=** buildSchema(

``.concat(

userSchema,

postSchema,

`

type RootQuery {

${userQuery},

${postQuery}

}

type RootMutation {

${userMutation},

${postMutation}

}

schema {

query: RootQuery

mutation: RootMutation

}

`

)

);

module.exports **=** schemas;

Because it is a string, so it can be concatconnected. A stitching is like a lot.

Next is rootResolversmodular. This is relatively simple, is to define each type of Query and Mutation methods, and then integrate:

**const** userResolver **=** require('./user.resolver');

**const** postResolver **=** require('./post.resolver');

**const** resolvers **=** {

...userResolver,

...postResolver

}

module.exports **=** resolvers;

Finally, the original way of writing becomes:

**const** express **=** require('express');

**const** graphqlHTTP **=** require('express-graphql');

**const** rootSchema **=** require('./schemas/index');

**const** rootResolvers **=** require('./resolvers/index');

**const** app **=** express();

app.use('/graphql', graphqlHTTP({

schema: rootSchema,

rootValue: rootResolvers,

graphiql: **true**,

}));

app.listen(4000);

**Using GraphQLSchema Constructor**

Using the Schema syntax sometimes feels too low, can it be more object-oriented, and more programmatically? Yes, with a GraphQLSchema()constructed schema.

Written in a file like this:

**const** express **=** require('express');

**const** graphqlHTTP **=** require('express-graphql');

**const** graphql **=** require('graphql');

**const** UserType **=** **new** graphql.GraphQLObjectType({

name: 'User',

fields: {

id: { type: graphql.GraphQLString },

name: { type: graphql.GraphQLString },

}

});

**const** PostType **=** **new** graphql.GraphQLObjectType({

name: 'Post',

fields: {

id: { type: graphql.GraphQLString },

title: { type: graphql.GraphQLString },

content: { type: graphql.GraphQLString }

}

});

**const** PostInput **=** **new** GraphQLInputObjectType({

name: 'PostInput',

fields: {

title: { type: GraphQLString },

content: { type: GraphQLString }

}

});

**const** queryType **=** **new** graphql.GraphQLObjectType({

name: 'Query',

fields: {

users: {

type: **new** GraphQLList(UserType),

args: **null**,

resolve: (\_, {id}) **=>** {

**return** user;

}

},

createUser: {

type: {

name: { type: GraphQLString }

},

resolve: (\_, args) **=>** {

**return** newUser;

}

},

posts: {

type: **new** GraphQLList(PostType),

args: **null**,

resolve: **async** () **=>** {

**return** posts;

}

},

createPost: {

type: PostType,

args: {

newPost: { type: PostInput }

},

resolve: (\_, args) **=>** {

**return** newPost;

}

}

}

});

**var** schema **=** **new** graphql.GraphQLSchema({query: queryType});

**var** app **=** express();

app.use('/graphql', graphqlHTTP({

schema: schema,

graphiql: **true**,

}));

app.listen(4000);

console.log('Running a GraphQL API server at localhost:4000/graphql');

Note that it is the attribute type is defined { type: graphql.GraphQLString }in this way, there is an array type new GraphQLList(YourType).

**Modularize for GraphQLSchema**

The next step is to refactor in a modular way. The way is very simple, Type is placed in a folder, each Model is a file, Query is also divided by Model, and then integrated with an Index.

Example of Type:

*// post.type.js*

**const** PostType **=** **new** GraphQLObjectType({

name: 'Post',

fields: () **=>** ({

id: { type: GraphQLString },

title: { type: GraphQLString },

content: { type: GraphQLString }

})

});

module.exports.PostType **=** PostType;

Input parameter types available GraphQLInputObjectType()definitions:

**const** PostInput **=** **new** GraphQLInputObjectType({

name: 'PostInput',

fields: {

title: { type: GraphQLString },

content: { type: GraphQLString }

}

});

module.exports.PostInput **=** PostInput;

Query example (specific definition omitted):

**const** { PostType, PostInput } **=** require('../types/post.query');

**const** postQueries **=** {

posts: {

type: **new** GraphQLList(PostType),

args: **null**,

resolve: **async** () **=>** {

**return** posts;

}

},

createPost: {

type: PostType,

args: {

newPost: { type: PostInput }

},

resolve: (\_, args) **=>** {

**return** newPost;

}

}

}

module.exports.postQueries **=** postQueries;

The integration Query is as follows:

*// queries/index.js*

**const** { GraphQLObjectType } **=** require('graphql');

**const** { userQueries } **=** require('../queries/user.query');

**const** { postQueries } **=** require('../queries/post.query');

**const** QueryType **=** **new** GraphQLObjectType({

name: 'QueryType',

fields: {

...userQueries,

...postQueries

}

});

module.exports **=** QueryType;

Then used to construct the schema:

*// schema.constructor.js*

**const** { GraphQLSchema } **=** require('graphql');

**const** QueryType **=** require('./queries/index');

module.exports **=** **new** GraphQLSchema({

query: QueryType

});

At last:

**const** express **=** require('express');

**const** graphqlHTTP **=** require('express-graphql');

**const** schemaConstructor **=** require('./schema.constructor');

**const** app **=** express();

app.use('/graphql', graphqlHTTP({

schema: schemaConstructor

graphiql: **true**,

}));

app.listen(4000);

Note that at this time has no need rootValueof