GraphQL 实践

- timqian

Me

- Javascript Enthusiasts
- Nodejs backend at work
- Full stack at home
- github@timqian

Target audiance

- Some experience on building/using REST API;
- Little experience on building/using GraphQL;

Table of contents

What is an APP from the perspective of data?

TJ Holowaychuk 🙃
@tjholowaychuk

正在关注

my js state management library: {}

翻译推文

下午9:51 - 2018年1月28日

349 转推 **1,634** 喜欢

48 349 1.6千

Frontend's Job:

1. Get the `{}` from backend
2. Render the APP based on the `{}`
3. Update the `{}` based on user's input
4. Maybe store the update back to backend

How does frontend get the { } through REST API

```
GET /users/:id # get user info
GET /users/:id/blogs # get all blogs of user
```

```
// The object frontend used to render the page
{
  user: {
    id: 1,
    username: 'timqian',
    blogs: [{
      id: 1,
      title: 'hi world'
      content: 'hello world'
    }]
  }
}
```

How does frontend get the { } through GraphQl

```
# GraphQL query
query {
  user(id: 1) {
    username
    blogs {
      id
      title
      content
    }
  }
}
```

```
// returned object from GraphQL backend
{
  user: {
    username: 'timqian',
    blogs: [{
      id: 1,
      title: 'hi world'
      content: 'hello world'
    }]
  }
}
```

Want more info about the user and less about the blog?

```
query {
  user(id: 1) {
+   id
    username
    blogs {
-     id
      title
-     content
    }
  }
}
```

```
// returned object from GraphQL backend
{
  user: {
    id: 1,
    username: 'timqian',
    blogs: [{
      title: 'hi world'
    }]
  }
}
```

So *What* is GraphQL

A query language for your API. Frontend define what data it want.

How to implement (1)

```
# 1. Define schema
type Query {
  user(id: ID!): User!
  blogs: [Blog]
}

type User {
  id: ID!
  username: String!
  blogs: [Blog]
}

type Blog {
  id: ID!
  title: String!
  content: String!
  createdBy: User!
}
```

How to implement (2)

```
// 2. Define resolvers as a nested object that
// maps type and field names to resolver functions
const resolver = {
  Query: {
    user: (obj, args) => daos.User.get(args.id),
    blogs: (obj, args) => daos.Blog.getAll(),
  },
  User: {
    blogs: (obj, args) => daos.Blog.getByUser(obj.id),
  },
  Blog: {
    createdBy: (obj, args) => daos.User.get(obj.createdBy)
  },
}
```

How to implement (3)

```
// 3. Bind schema and resolver together using graphql-yoga
// This is just for example. You can also use `graphql-too
//`express-graphql` or `apollo-server` to do this
import { GraphQLServer } from 'graphql-yoga';

const server = new GraphQLServer({ typeDefs, resolvers });

server.start(() =>
  console.log('Server is running on localhost:4000'));
```

Why

- Performance
  - Less roundtrips
- Development experance
  - Self documented
  - Less endpoints
  - Ask for what you want
  - Real-time data push (subscription)

Issues and Solutions

- N+1 problem
- Writing test
- Similar code for normal usage

Issue: N+1 problem

```
# Will do N + 1 database query if there is N blogs
query {
  blogs {
    id
    title
    createdBy {
      id
      name
    }
  }
}
```

Situation can be worse when the query becomes more complex.

Can we do the N user query together?

Solution: `Dataloader` (1)

```
const DataLoader = require('dataloader');

// Provid a batch loading function
const myBatchGetUsers = ids =>
  daos.User.whereIn('id', ids);

// Create your data loader
const userLoader =
  new DataLoader(myBatchGetUsers);
```

Solution: `Dataloader` (2)

Update resolver

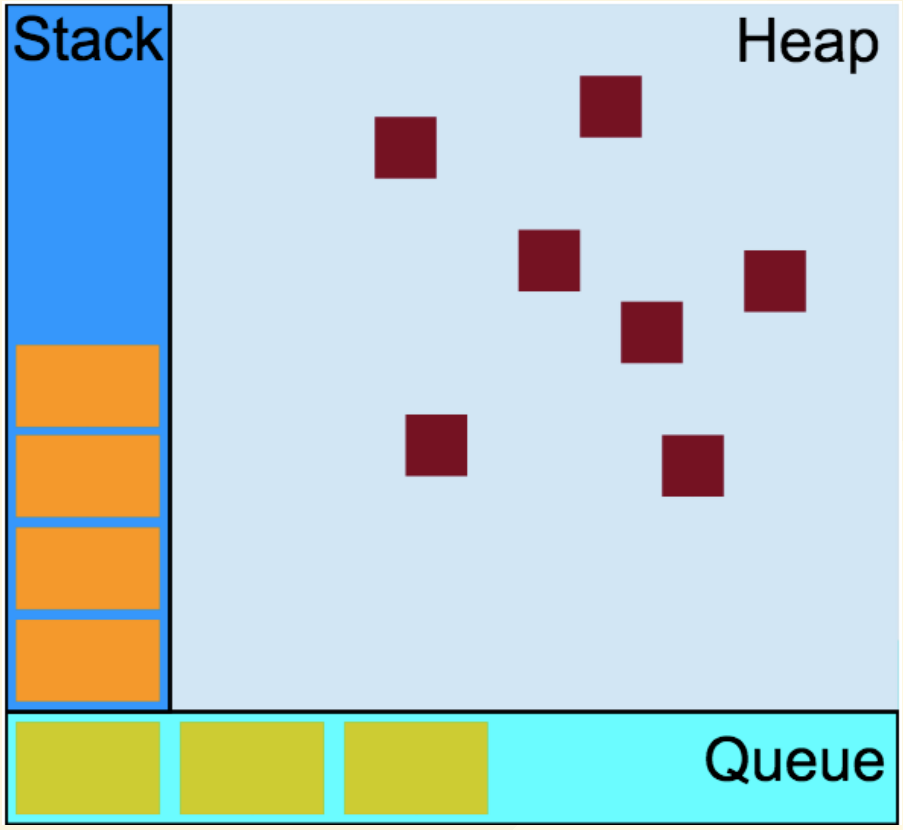
```
const resolver = {
  Query: {
    user: (obj, args) => daos.User.get(args.id),
  },
  User: {
    blogs: (obj, args) => daos.Blog.getByUser(obj.id),
  },
  Blog: {
-   createdBy: (obj, args) => daos.User.get(obj.createdBy),
+   createdBy: (obj, args) =>
+     userLoader.load(obj.createdBy),
  },
}
```

`Dataloader` Caching

```
load(key)

clear(key)

loadMany(keys)

clearAll()
```

How does Dataloader work

Let's revise how event loop work first




```
while(queue.waitForMessage()){
  queue.processNextMessage();
}
```

How does Dataloader work

DataLoader will coalesce all individual loads which occur within a single frame of execution (a single tick of the event loop) and then call your batch function with all requested keys.

push key to a queue(array):
https://github.com/facebook/dataloader/blob/master/src/index.js#L96
batch query in next tick:
https://github.com/facebook/dataloader/blob/master/src/index.js#L104

Application level dataloader?

The official Readme encourage user to create a new DataLoader per request. Because:

1. Many different users with different access permissions. It may be dangerous to use one cache across many users
2. In memary cache can not scale among servers

But they are actually solvable

Application level dataloader?

1. Use dataloader in the dao layer of your app and do ACL on resolver
2. Use redis/memcached as the cache of dataloader

Refs

- Disscussions on an issue of dataloader repo
- Use redis instead of memary as the cache

Writing tests

```
# Sample schema
type Query {
    user(id: Int!): User!
}

type User {
    id: Int!
    username: String!
    email: String!
    createdAt: String!
}
```

```
# Sample query
query user($id: Int!) {
    user(id: $id) {
        id
        username
        email
        createdAt
    }
}
```

```
# Sample query
query user($id: Int!) {
    user(id: $id) {
        id
        username
        email
        createdAt
    }
}
```

gql-generator: generate sample queries for you based on the schema

Automatically mapping your API to database: Prisma

Define your types and it will do the resolves for you.