GraphQL Ruby 实现入门和实践

@bastengao 2019.08

- API 工程师
- 有数派
- 纺织行业 SasS
- 管理布匹 1, 199, 055, 527 米，绕地球 30 圈
- 终端覆盖 Web、App、TV、小程序、PDA、一体机

GraphQL

- At Facebook in 2012
- Open standard started in 2015
- GitHub, Shopify

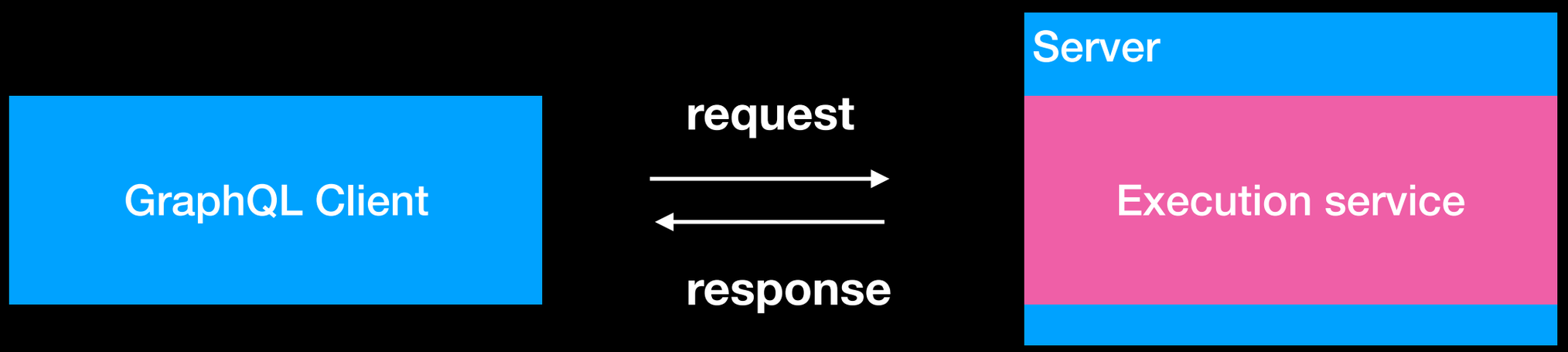
Prerelease	Working Draft	Thu, Jun 20, 2019	
Latest Release	June 2018	Sun, Jun 10, 2018	Release Notes
	October 2016	Mon, Oct 31, 2016	Release Notes
	April 2016	Thu, Apr 7, 2016	Release Notes
	October 2015	Thu, Oct 1, 2015	Release Notes
	July 2015	Thu, Jul 2, 2015	Release Notes

Ready for Production

- 纺织行业 SasS
- Web、App、小程序(5+ clients)
- 300+ queries
- 500+ mutations

GraphQL

- A query language
- Execution service



REST API

> curl https://api.github.com/search/repositories?q=ruby


```
{
    "total_count": 294596,
    "incomplete_results": false,
    "items": [
        {
            "id": 8514,
            "node_id": "MDEwOlJlcG9zaXRvcnk4NTE0",
            "name": "rails",
            "full_name": "rails/rails",
            "private": false,
            "owner": {
                "login": "rails",
                "id": 4223,
                "node_id": "MDEyOk9yZ2FuaXphdGlvbjQyMjM=",
                "avatar_url": "https://avatars1.githubusercontent.com/u/4223?v=4",
                "gravatar_id": "",
                "url": "https://api.github.com/users/rails",
                "html_url": "https://github.com/rails",
                "followers_url": "https://api.github.com/users/rails/followers",
                "following_url": "https://api.github.com/users/rails/following{/other_user}",
                "gists_url": "https://api.github.com/users/rails/gists{/gist_id}",
                "starred_url": "https://api.github.com/users/rails/starred{/owner}{/repo}",
                "subscriptions_url": "https://api.github.com/users/rails/subscriptions",
                "organizations_url": "https://api.github.com/users/rails/orgs",
                "repos_url": "https://api.github.com/users/rails/repos",
                "events_url": "https://api.github.com/users/rails/events{/privacy}",
                "received_events_url": "https://api.github.com/users/rails/received_events",
                "type": "Organization",
                "site_admin": false
            },
            "html_url": "https://github.com/rails/rails",
            "description": "Ruby on Rails",
            "fork": false,
            "url": "https://api.github.com/repos/rails/rails",
```


143 KB

REST API

- 不喜欢别人不写文档
- 自己不喜欢写文档

REST vs GraphQL

- 需要额外的文档描述
- 弱类型、弱检验
- 返回的数据不一定用的上

- 对外暴露 schema, 文档清晰
- 强类型，同时可以做验证
- 按需获取数据
- 一次请求多个查询，避免多次网络请求
- API 变更相对友好

A query language for your API

```graphql
query {
  search(query:"ruby", type: REPOSITORY, first: 10) {
    nodes {
      ... on Repository {
        name
        url
        description
        forkCount
      }
    }
  }
}
```

QUERY VARIABLES


```json
{
  "data": {
    "search": {
      "nodes": [
        {
          "name": "ruby",
          "url": "https://github.com/ruby/ruby",
          "resourcePath": "/ruby/ruby",
          "description": "The Ruby Programming Language [mirror]",
          "forkCount": 4215
        },
        {
          "name": "ruby",
          "url": "https://github.com/airbnb/ruby",
          "resourcePath": "/airbnb/ruby",
          "description": "Ruby Style Guide",
          "forkCount": 602
        },
        {
          "name": "ruby",
          "url": "https://github.com/exercism/ruby",
          "resourcePath": "/exercism/ruby",
          "description": "Exercism exercises in Ruby.",
          "forkCount": 357
        },
        {
          "name": "rails",
          "url": "https://github.com/rails/rails",
```


GitHub API v4

Query

```
type User {
  name: String!
  email: String
  avatarUrl: String
  location: String
}

type Query {
  viewer: User!
}
```

```
query {
  viewer {
    name
    email
    location
  }
}
```

```
{
  "data": {
    "viewer": {
      "name": "Basten Gao",
      "email": "bastengao@gmail.com",
      "location": "China"
    }
  }
}
```

Query / OperationName

```
query currentUser{
  viewer {
    name
    email
    location
  }
}
```

Good

- 代码易于阅读
- 日志清晰
- 方便追踪和监控

```
query {
  viewer {
    name
    email
    location
  }
}
```

```
{
  viewer {
    name
    email
    location
  }
}
```

Bad

Query / Alias

```graphql
query {
  viewer {
    aLongName: name
  }
}
```

```json
{
  "data": {
    "viewer": {
      "aLongName": "foo"
    }
  }
}
```

Query / Argument

```
{
  user(name: "foo") {
    name
    email
    posts(first: 10) {
      nodes {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "user": {
      "name": "foo",
      "email": "foo@bar",
      "posts": {
        "nodes": [
          {
            "name": "GraphQL"
          }
        ]
      }
    }
  }
}
```

Query Variables

```graphql
query ($name: String!, $first: Int) {
  user(name: $name) {
    name
    email
    posts(first: $first) {
      nodes {
        name
      }
    }
  }
}
```

QUERY VARIABLES

```json
{
  "name": "foo",
  "first": 10
}
```

```json
{
  "data": {
    "user": {
      "name": "foo",
      "email": "foo@bar",
      "posts": {
        "nodes": [
          {
            "name": "GraphQL"
          }
        ]
      }
    }
  }
}
```

Query Fragment

```
query {
  viewer {
    ... userFields
  }

  user(name: "foo") {
    ... userFields
  }
}

fragment userFields on User {
  name
  email
  avatarUrl
}
```

```
{
  "data": {
    "viewer": {
      "name": "foo",
      "email": "foo@bar",
      "avatarUrl": "awesome.jpg"
    },
    "user": {
      "name": "foo",
      "email": "foo@bar",
      "avatarUrl": "awesome.jpg"
    }
  }
}
```

Mutation

```graphql
mutation ($input: CreatePostInput!) {
  createPost(input: $input) {
    post {
      name
      content
    }
  }
}
```

QUERY VARIABLES

```json
{
  "input": {
    "name": "GrapphQL",
    "content": "..."
  }
}
```

```json
{
  "data": {
    "createPost": {
      "post": {
        "name": "GrapphQL",
        "content": "..."
      }
    }
  }
}
```

GraphQL Schema

Describe your API

- **Query**
- **Mutation**
- **Subscription**

```
schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}
```

GraphQL Schema

```
type User {
  name: String!
  email: String
  avatarUrl: String
  location: String
}

type Query {
  viewer: User!
}

type Mutation {
  createUser(input: CreateUserInput!) CreateUserPayload
}

schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}
```

Types

- Scalar
- Enumeration
- List
- Non-Null
- Object
- Input
- Union
- Interface

Scalar Types

- Int
- Float
- String
- Boolean
- ID (represents a identifier, serialized as string)

Object type & fields

```
type User {
  id: ID!
  name: String!
  age: Int
  email: String
}
```

List type

```
type User {
  id: ID!
  name: String!
  age: Int
  email: String
  posts: [Post]
}
```

Non-null

```
type User {
  id: ID!
  name: String!
  age: Int
  email: String
  posts: [Post!]!
}
```

```
[Post]   => [post] or [] or [null] or null
[Post]!  => [post] or [] or [null]
[Post!]! => [post] or [] or
```

Enumeration type

```
type User {
  id: ID!
  name: String!
  age: Int
  email: String
  gener: Gender
}

enum Gender {
  Female
  Male
}
```

Union type

```
union SearchResult = User || Post

type Query {
  search(keyword: String) [SearchResult!]!
}
```

```
query {
  search(keyword: “foo”) {
    __typename
    ... on User {
      name
      gender
    }
    ... on Post {
      name
      content
    }
  }
}
```

Interface type

```
interface Namable {
  name: String!
}

type User implements Namable {
  name: String!
  email: String
}

type Post implements Namable {
  name: String!
  content: String!
}

type Query {
  searchNames(keyword: String) [Namable!]!
}
```

```
query {
  searchNames(keyword: “foo”) {
    __typename
    name
    ... on User {
      email
    }
    ... on Post {
      content
    }
  }
}
```

Input type

```
input CreatePostInput {
  name: String!
  content: String!
}

type Mutation {
  createPost(input: CreatePostInput!) CreatePostPayload
}
```

```
mutation ($input: CreatePostInput!) {
  createPost(input: $input) {
    post {
      name
      content
    }
  }
}
```

QUERY VARIABLES

```
{
  "input": {
    "name": "GrapphQL",
    "content": "..."
  }
}
```

```
{
  "data": {
    "createPost": {
      "post": {
        "name": "GrapphQL",
        "content": "..."
      }
    }
  }
}
```

Introspection

```
{
  __schema {
    queryType {
      name
      fields {
        name
      }
    }
    mutationType {
      name
      fields {
        name
      }
    }
  }
}
```

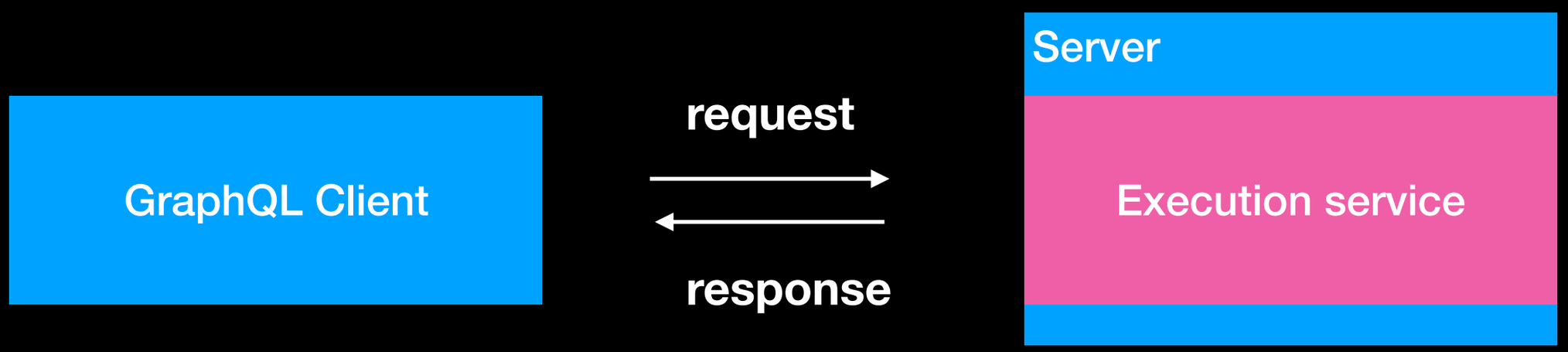
```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query",
        "fields": [
          {
            "name": "user"
          },
          {
            "name": "users"
          },
          {
            "name": "viewer"
          }
        ]
      },
      "mutationType": {
        "name": "Mutation",
        "fields": [
          {
            "name": "createComment"
          },
          {
            "name": "createPost"
          }
        ]
      }
    }
  }
}
```

Client tools

- Graph*i*QL
- GraphQL Playground
- Insomnia

GraphQL

- A query language
- Execution service



rmosolgo / graphql-ruby

Used by 3,249 | Watch 83 | Star 3,805 | Fork 692

Code | Issues 147 | Pull requests 26 | Wiki | Security | Insights

Ruby implementation of GraphQL http://graphql-ruby.org

ruby graphql relay rails

4,902 commits | 33 branches | 165 releases | 254 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

rmosolgo Merge pull request #2377 from eapache/another-unicode-lexer-bug ... Latest commit 80e9c87 2 days ago

.github	doc(CONTRIBUTING) Add contributing.md	2 years ago
benchmark	Add memory benchmark	10 months ago
cop	Add a cop to catch misused .none?	5 months ago
gemfiles	Add sqlite version constraint	5 months ago
guides	Fix file paths and add types prefix	5 days ago
javascript_client	remove yarn.lock	5 days ago
lib	Merge pull request #2377 from eapache/another-unicode-lexer-bug	2 days ago
spec	Merge pull request #2377 from eapache/another-unicode-lexer-bug	2 days ago
.codeclimate.yml	Don't run code climate on tests	last year

Hello World

```ruby
require 'graphql'

class DemoSchema < GraphQL::Schema
  query QueryType
end

class QueryType < GraphQL::Schema::Object
  field :hello, String, null: false
    resolve: ->(obj, args, ctx) { "world" }
end

puts Schema.execute("{ hello }").to_json
```

```
{
  "data": {
    "hello": "world"
  }
}
```

Rails Controller

```ruby
# config/routes.rb
post "/graphql", to: "graphql#execute"
```

```ruby
# graphql_controller.rb
class GraphqlController < ActionController::API
  def execute
    result = DemoSchema.execute(
      query,
      variables: variables,
      context: context,
      operation_name: operation_name
    )
    render json: result
  end

  ...

end
```

Dir structure

```
app/
  graphql/
    mutations/
    queries/
    types/
    demo_schema.rb
```

Define an Object Type

```ruby
# app/models/user.rb
class User < ApplicationRecord
end

# table users
#   id          :integer
#   name        :string
#   avatar_url  :string
```

```ruby
class Types::UserType < Types::BaseObject
  graphql_name "User"
  description "用户"

  field :id, ID, null: false
  field :name, String, null: false
  field :avatar_url, String, '头像', null: true
end
```

Define a Query

```ruby
class QueryType < Types::BaseObject
  field :user, Types::UserType, null: false do
    argument :id, ID, required: true
  end

  def user(id:)
    User.find(id)
  end
end
```

Field Documentation

```
field :avatar_url, String, "头像", null: true
field :avatar_url, String, description: "头像"null: true

field :old_avatar, String, "头像" null: true,
  deprecation_reason: "由 avatar_url 代替"
```

Field Resolution

- object.public_send(name)
- Hash
  - object[name.to_sym] or object[name.to_s]

```ruby
class Types::UserType < Types::BaseObject
  # @object => User instance
  # @context

  field :id, ID, null: false
  # object.id or object[:id] or object["id"]
end
```

Field Resolution

- **method:**
- **hash_key:**
- **override method**

```ruby
class Types::UserType < Types::BaseObject
  field :nickname, String, null: false, method: :name    # => object.name
  field :fullname, String, null: true,  hash_key: "name" # => object["name"]

  field :first_letter, String, null: false

  def first_letter
    object.name[0]
  end
end
```

Arguments

- description:
- required:
- default_value:
- as:
- prepare:

```ruby
class Types::UserType < Types::BaseObject
  field :posts, [Types::PostType], null: false do
    argument :category, String, "分类", required: false,
      default_value: "GraphQL"
  end

  def posts(category: nil)
    if category
      object.posts.where(category: category)
    else
      object.posts
    end
  end
end
```

Auto camelization

- 蛇形写，对外将转换为驼峰
- camelize: false 禁用

```ruby
class Types::UserType < Types::BaseObject
  field :id, ID, null: false
  field :name, String, null: false
  field :avatar_url, String, '头像', null: true
end
```

```
type User {
  id: ID!
  name: String!
  avatarUrl: String
}
```

Resolver

```ruby
class QueryType < Types::BaseObject
  field :posts, [Types::PostType], null: false do
    argument :category, String, "分类", required: false,
      default_value: "GraphQL"
  end

  def posts(category: nil)
    if category
      Post.where(category: category)
    else
      Post.all
    end
  end
end
```

```ruby
class QueryType < Types::BaseObject
  field :posts, resolver: Resolvers::Posts
end

class Resolvers::Posts < GraphQL::Schema::Resolver
  type [Types::PostType], null: false

  argument :category, String, "分类", required: false,
    default_value: "GraphQL"

  def resolve(category: nil)
    if category
      Post.where(category: category)
    else
      Post.all
    end
  end
end
```

Define a Mutation

```ruby
# apps/graphql/muttaions/create_post.rb
class Mutations::CreatePost < Mutations::BaseMutation
  graphql_name "CreatePost"
  description "Create a post"

  argument :name, String, required: true
  argument :content, String, required: true

  field :post, Types::PostType, null: true

  def resolve(**args)
    params = args.slice(:name, :content)
    params[:user] = context[:current_user]
    { post: Post.create!(params) }
  end
end
```

```ruby
class MutationType < Types::BaseObject
  field :create_post, mutation: Mutations::CreatePost
end
```

Define a Subscription

```ruby
# app/graphql/subscription_type.rb
class SubscriptionType < Types::BaseObject
  field :post_created, Types::PostType, null: false,
    description: "A post was created to the blog"
end

# app/graphql/schema.rb
class Schema < GraphQL::Schema
  subscription SubscriptionType
end
```

push

```ruby
new_post = Post.create(params)
Schema.subscriptions.trigger("postCreated", {}, new_post)
```

Errors

- validation error
- type error
- runtime error

```
{
  "data": ...
  "errors": [{
    "message": "Can't continue with this query",
    "locations": [{ "line": 6, "column": 7 }],
    "path": ["viewer"],
  }]
}
```

Raise top-level errors

```
def resolve
  raise GraphQL::ExecutionError "Can't continue with this query"
  ...
end
```

Errors as data

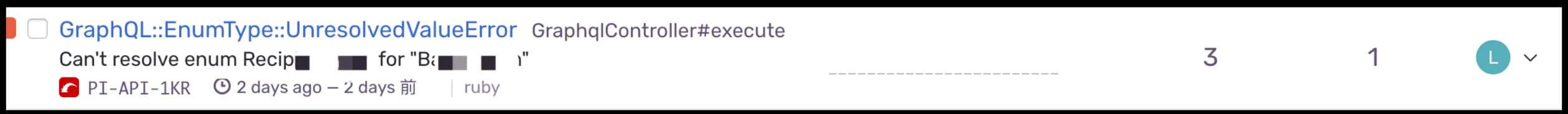
```ruby
class Mutations::CreatePost < Mutations::BaseMutation
  # ...
  field :post,   Types::PostType,    null: true
  field :errors, [Types::UserError], null: false

  def resolve
     {
       post: post,
       errors: user_errors
     }
  end
end
```

Track error

```ruby
# integrate with sentry
class DemoSchema < GraphQL::Schema
  ...

  def self.type_error(err, query_ctx)
    Raven.capture_message(err)
    super
  end
end
```

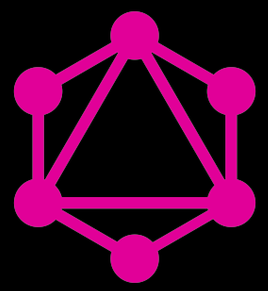


Relay

A JavaScript framework based on React + GraphQL

React

GraphQL



Relay

GraphQL Server Specification

- **Object Identification**
- **Connections**
- **Mutations**

Relay

Object Identification

```
interface Node {
  id: ID!
}

type User implements Node {
  id: ID!
  name: String!
}

type Post implements Node {
  id: ID!
  name: String!
}

type Query {
  node(id: ID!): Node
}
```

```
{
  node(id: "VXNlci01") {
    id
    ... on User {
      name
      email
    }
  }
}
```

```
{
  "data": {
    "node": {
      "id": "VXNlci01",
      "name": "foo",
      "email": "foo@bar"
    }
  }
}
```

Relay

Object Identification

```
class DemoSchema < GraphQL::Schema
  def self.id_from_object(object, type_definition, query_ctx)
    ...
  end

  def self.object_from_id(id, query_ctx)
    ...
  end

  def self.resolve_type(type, obj, ctx)
    case obj
    when User
      Types::UserType
    else
      raise("Unexpected object: #{obj}")
    end
  end
end

class Types::UserType < Types::BaseObject
  implements GraphQL::Relay::Node.interface

  global_id_field :id
end

class QueryType < Types::BaseObject
  field :node, field: GraphQL::Relay::Node.field
end
```

Relay

Connections

cursor-based pagination

```
query {
  users(first: 2) {
    pageInfo {
      startCursor
      endCursor
      hasPreviousPage
      hasNextPage
    }
    edges {
      cursor
      node {
        id
        name
      }
    }
  }
}
```

```
{
  "data": {
    "users": {
      "pageInfo": {
        "startCursor": "MQ==",
        "endCursor": "Mg==",
        "hasPreviousPage": false,
        "hasNextPage": true
      },
      "edges": [
        {
          "cursor": "MQ==",
          "node": {
            "id": "VXNlci0x",
            "name": "A"
          }
        },
        {
          "cursor": "Mg==",
          "node": {
            "id": "VXNlci0y",
            "name": "B"
          }
        }
      ]
    }
  }
}
```

```
query {
  users(first: 2, after: "Mg==")
    pageInfo {
      startCursor
      endCursor
      hasPreviousPage
      hasNextPage
    }
    edges {
      cursor
      node {
        id
        name
      }
    }
  }
}
```

```
{
  "data": {
    "users": {
      "pageInfo": {
        "startCursor": "Mw==",
        "endCursor": "NA==",
        "hasPreviousPage": false,
        "hasNextPage": true
      },
      "edges": [
        {
          "cursor": "Mw==",
          "node": {
            "id": "VXNlci0z",
            "name": "C"
          }
        },
        {
          "cursor": "NA==",
          "node": {
            "id": "VXNlci00",
            "name": "D"
          }
        }
      ]
    }
  }
}
```

Relay

Connections

```
field :users, UserType.connection_type, null: false do
  argument :search, String, required: false
end
```

Relay

Mutations

wrap input & payload

```
input CreatePostInput {
  name: String!
  content: String!
  clientMutationId: String
}

type CreatePostPayload {
  post: Post
  clientMutationId: String
}

type Mutation {
  createPost(input: CreatePostInput!): CreatePostPayload
}
```

Relay

Mutations

```
mutation($input: CreatePostInput!) {
  createPost(input: $input) {
    post {
      id
      name
    }
    clientMutationId
  }
}
```

QUERY VARIABLES

```
{
  "input": {
    "name": "GraphQL",
    "content": "...",
    "clientMutationId": "asdf"
  }
}
```

```
{
  "data": {
    "createPost": {
      "post": {
        "id": "UG9zdC00",
        "name": "GraphQL"
      },
      "clientMutationId": "asdf"
    }
  }
}
```

Relay

Mutations

```ruby
class Mutations::CreatePost < GraphQL::Schema::RelayClassicMutation
  ...
end
```

Authentication

```
# HTTP headers
Authorization: Bearer <token>
```

```
class GraphqlController < ActionController::API
  ...

  def context
    {
      current_user: find_user_from_token(request)
    }
  end
end
```

just pass context to authorization part

Authorization

Type, Field, Argument, Mutation

- **visible?(context)**
- **accessible?(context)**
- **authorized?(object, context)**

Authorization

```ruby
class BaseField < GraphQL::Schema::Field
  def initialize(auth_required: true, **kwargs, &block)
    @auth_required = auth_required
    super(**kwargs, &block)
  end

  def accessible?(context)
    return true unless @auth_required

    !context[:current_user].nil?
  end
end

class BaseObject < GraphQL::Schema::Object
  field_class BaseField
end

class QueryType < Types::BaseObject
  field :viewer, Types::UserType, required: false, auth_required: false
end

class MutationType < Types::BaseObject
  field :login, mutation: Mutations::Login, auth_required: false
end
```

Authorization

```
gem 'graphql-guard'
```

```
class DemoSchema < GraphQL::Schema
  use GraphQL::Guard.new
end

class QueryType < Types::BaseObject
  field :viewer, Types::UserType, required: false,
    guard: ->(obj, args, ctx) { !context[:current_user].nil? }
end
```

Upload

```
gem 'apollo_upload_server'
```

```
argument :file, ApolloUploadServer::Upload
```

Not compatible with ActiveStorage now
https://github.com/jetruby/apollo_upload_server-ruby/issues/10

Download

Trough query to get normal download url

```ruby
class QueryType < Types::BaseObject
  field :download_link, Types::UserType, null: true

  def download_link
    Rails.application.routes.url_helpers.download_posts_url
  end
end
```

N + 1 query

```
# Gemfile
gem 'graphql-batch'
gem 'graphql-preload'
```

```
class DemoSchema < GraphQL::Schema
  use GraphQL::Batch
  enable_preloading

  ...
end

class Types::UserType < Types::BaseObject
  ...

  # User.includes(:posts)
  field :posts, [Types::PostType], null: false, preload: :posts
end
```

max_depth

```
{
  viewer {
    name
    repositories(last: 10) {
      nodes {
        name
        description
        issues(first: 10) {
          nodes {
            title
            comments {
              nodes {
                author {
                  login
                }
                body
                userContentEdits(first: 10) {
                  nodes {
                    diff
                    editor {
                      login
                    }
                  }
                }
              }
```

```
class DemoSchema < GraphQL::Schema
  max_depth 10
end
```

```
{
  "errors": [
    {
      "message": "Query has depth of 11,
                  which exceeds max depth of 10"
    }
  ]
}
```

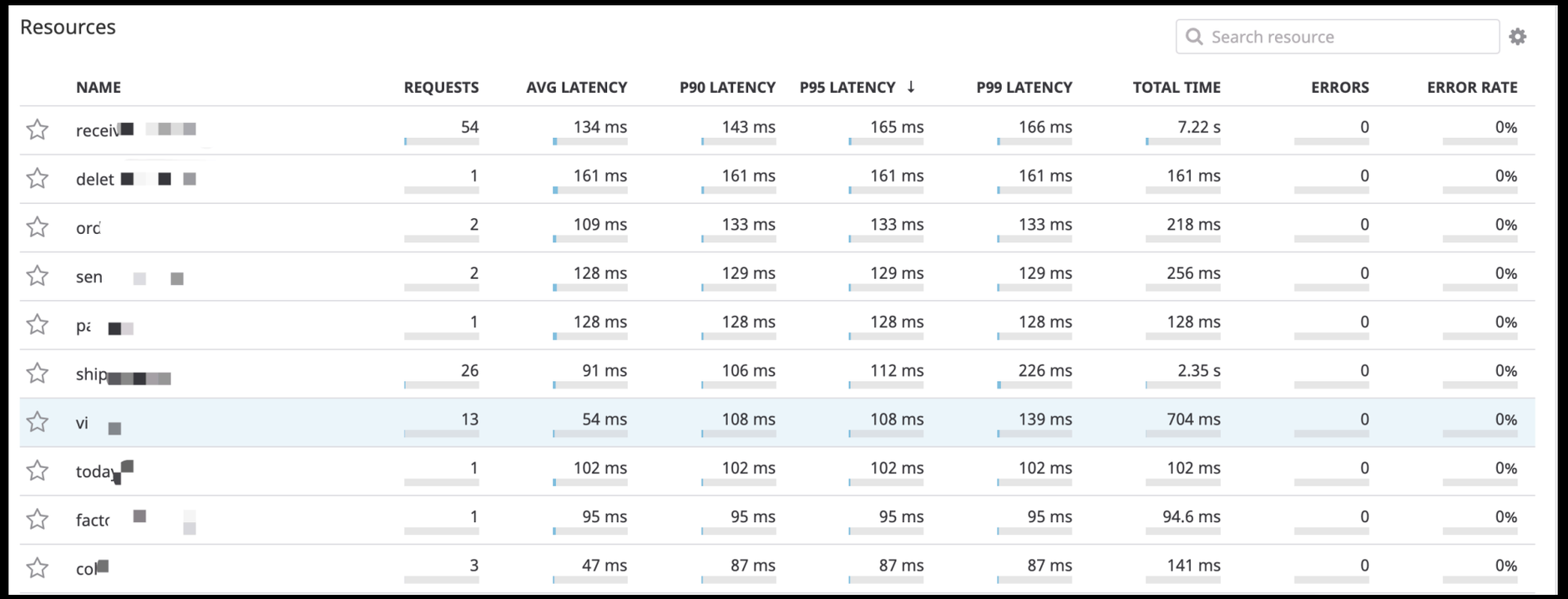
Tracing

built-in traces

- **AppSignal**
- **New Relic**
- **Scout**
- **Skylight**
- **Datadog**
- **Prometheus**

Tracing

```ruby
# app/graphql/demo_schema.rb
class DemoSchema < GraphQL::Schema
  use GraphQL::Tracing::DataDogTracing
end
```



Query log

```ruby
# app/graphql/demo_schema.rb
class DemoSchema < GraphQL::Schema
  query_analyzer GraphQL::Analysis::QueryDepth.new do |query, depth|
    logger.info("
      RequestId #{query.context[:request_id]}
      Depth #{depth}
      Query: #{query.query_string.squish}
    ".squish)
  end
end
```

```
2019-07-19T23:07:00.300835]: RequestId 92dba5fd-b233-4d28-a0ec-a6cd41721152, Depth 0, Query: query IntrospectionQuery { __schema { queryType { na
2019-07-19T23:07:34.603395]: RequestId fb9ac8fc-9237-49b2-94a9-da4b96801f48, Depth 3, Query: { orderItems { pageInfo { perPage } } }
2019-07-19T23:09:00.294628]: RequestId ee7c639e-49f7-45e4-8ce0-7ca2384bd8a5, Depth 3, Query: { orderItems(perPage: 8) { pageInfo { perPage } } }
2019-07-19T23:09:30.904968]: RequestId 28fe5ce3-6606-4e22-bf14-b21496227871, Depth 3, Query: { orderItems(perPage: 8) { pageInfo { perPage totalC
2019-07-19T23:09:43.855390]: RequestId 5d730152-ab06-4f1b-9fe0-275f601fdd1a, Depth 3, Query: { orderItems(perPage: 8) { pageInfo { perPage curren
```

Schema tracking

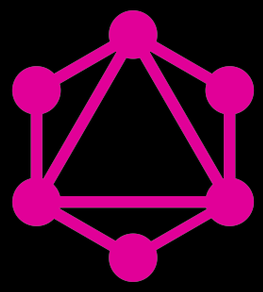
```
File.write(idl_file_path, DemoSchema.to_definition)
=> demo_schema.graphql
```

```
type Film {
  title: String
  episode: Int
  releaseDate: String
  openingCrawl: String
  director: String

}
```

```
 type Film {
   title: String
   episode: Int
   releaseDate: String
   openingCrawl: String
   director: String
+  directedBy: Person
 }

+type Person {
+  name: String
+  directed: [Film]
+  actedIn: [Film]
+}
```



GraphQL