Programmatic GraphQL Schema

Mikhail Novikov

@freiksenet

Mikhail Novikov

@freiksenet

Software Engineer at Gatsby Inc

Reindex, React Finland, GraphQL

Two cats

Contents

What is "programmatic"?

Programmatic <> Declarative

SDL is declarative

```
type Foo {
  id: ID!
}
```

GraphQL Frameworks are declarative

Prisma

AWS AppSync

GraphCMS

graphql-js declarative too

Oh the irony

```
const Foo = new GraphQLObjectType({
  name: 'Foo',
  fields: {
    id: {
      type: GraphQLID,
    },
  },
})
```

Programmatic is

using functions to create types

from other sources than a type config

```
function renameObjectType(type, newName) {
  const config = type.toConfig()
  return new GraphQLObjectType({
    ...config,
    name: newName,
  })
}
```

```
function createTypeFromSdl(sdl) {
  // ...
}
```

```
function createEdgeType(type) {
  // ...
}

function createConnectionType(type) {
  // ...
}
```

POV is important

User vs Library author

Why and why not?

NB: You might not need it

Use case

Different source of truth

Examples

AppSync - user config

PostGraphile - DB

Gatsby - nodes

SDL - SDL text

Use case

Multiple sources of truth

Examples

Gatsby

Schema Stitching

Use case

Repetitive/generic types

Examples

Relay Connections

Input objects

Use case

Creating types partially

Examples

Back-references

Gatsby

How?

JavaScript edition

Programmatic graphql-js

requires some tricks

Types must be defined

```
const Foo = new GraphQLObjectType({
  name: 'Foo',
  fields: {
    bar: {
      type: Bar // undefined, error
    }
  }
})
```

Workaround - Thunks

```
// ---
fields: () => ({
  bar: {
    type: Bar
  }
})
```

Circular modules

```
// Bar.js
const Foo = require('./Foo')

// Foo.js
const Bar = require('./Bar')
```

Workaround - Thunks 2

```
// ---
fields: () => {
  const { Bar } = require('./Bar')
  return {
    bar: {
      type: Bar
    }
  }
}
```

Inspecting fields

```
foo.getFields() // ERROR!
```

Workaround - create all types

Types need to be *complete*

Unknown type names

```
type Foo {
  bar: Bar
}
```

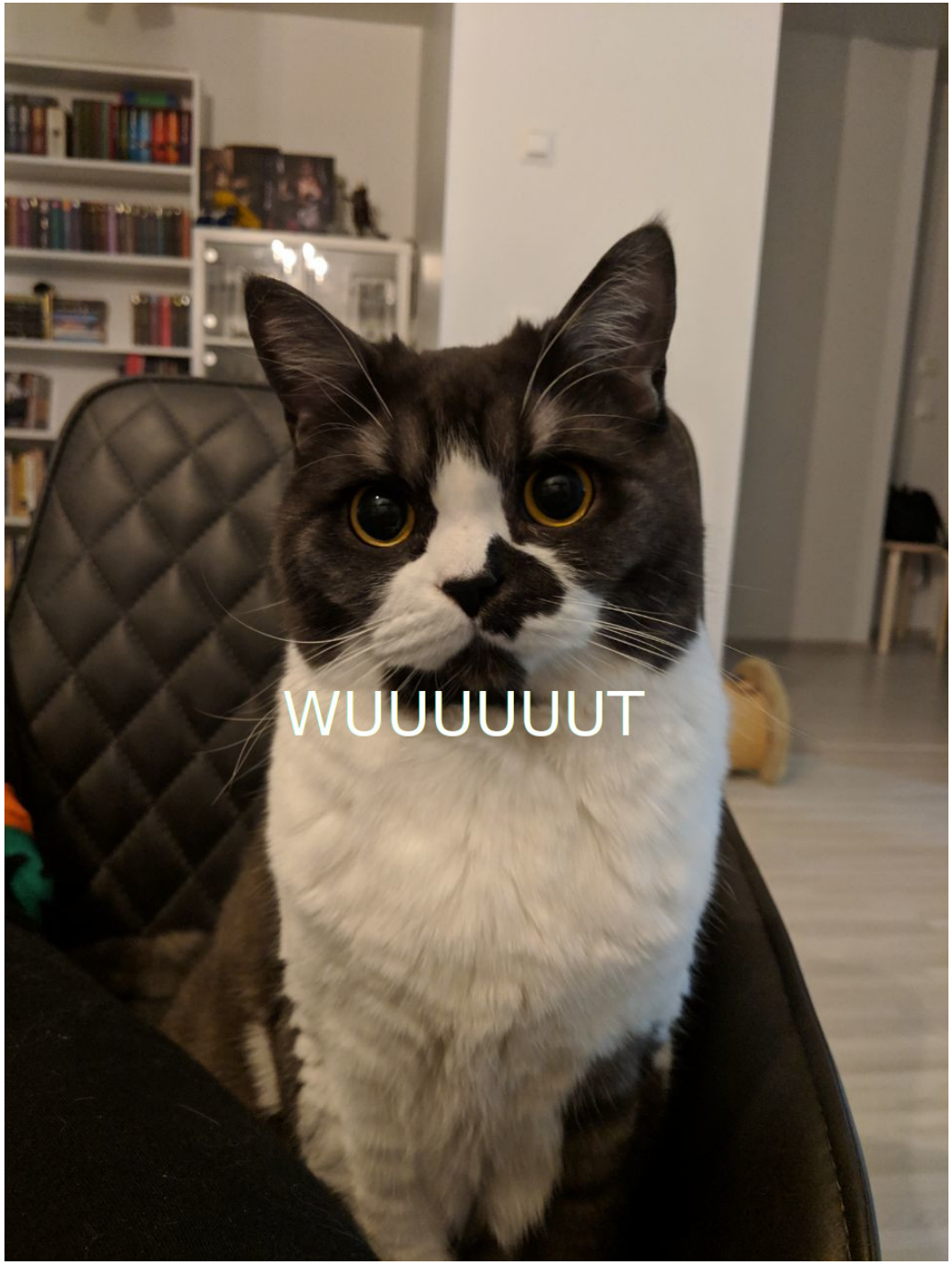
Unknown type names

```
sdlAst.fields.forEach((field) => {
  field[field.name.value] = {
    type: field.type.name.value, // STRING!
  }
})
```

Workaround - ???

Enter Type Registry

```
export const typeRegistry = {}
```


WUUUUUUT

Type registry - basic usage

```
  // ---
  fields: () => ({
    bar: {
      type: typeRegistry['Bar']
    })
  }
  // ---
typeRegistry['Foo'] = Foo
```

Type registry - unknown type names

```
sdlAst.fields.forEach((field) => {
  field[field.name.value] = {
    type:
      typeRegistry[field.type.name.value],
  }
})
```

Type registry

State of schema creation

Naively just holds types by name

Could be much more

Better TR - not just an object

```
const typeRegistry = new TypeRegistry()
typeRegistry.addType(Bar)
typeRegistry.getType('Bar')
```

Better TR - autoadd types

```
typeRegistry.createObjectType({
  name: "Foo",
  fields: () => ({
    bar: {
      type: typeRegistry.getType('Bar')
    }
  }),
})
```

Better TR - resolve string types

```
typeRegistry.createObjectType({
  name: "Foo",
  fields: {
    bar: {
      type: 'Bar'
    }
  },
})
```

Better TR - extend types

```
typeRegistry
  .getType('Bar')
  .extendFields({
    bars: {
      type: 'Foo'
    },
  })
```

Better TR - getOrCreate

```
typeRegistry
  .getOrCreateObjectType(
    'Foo',
    () => {
      // config if doesn't exist
    }
  )
```

Better TR - Build types to a schema

```
typeRegistry.buildSchema()
```

Better TR - Creating derived types

```
typeRegistry
  .getType('Bar')
  .toInputObjectType()
```

Better Type Registry

Holds all types

Concise programmatic syntax

Type modification after creation

Helpes to create types

Should I create my own?

Maybe not!

graphql-compose

graphql-compose

Library implementing type registry

Features described above

Used extensively at Gatsby

Usage at Gatsby?

Gatsby GraphQL is special

Build time

Multiple sources of truth

Inferred from data

Generated roots

Inferrence

Inferrence - source

```
{
  "id": "5",
  "title": "Gatsby is the best!",
  "date": "2019-09-06"
}
```

Inferrence - result

```
type FooJson implements Node {
  id: ID!
  title: String
  date: Date
}
```

Inferrence - markdown

```
---
"title": "Gatsby is the best!",
"date": "2019-09-06"
---
```

Inferrence - result

```
type MarkdownRemark implements Node {
  id: ID!
  frontmatter: Frontmatter
}

type Frontmatter {
  title: String
  date: Date
}
```

Defined types

```
type FooJson implements Node {
  id: ID!
  title: String
  date: Date
}
```

Schema control

```
type FooJson implements Node @dontInfer {
  id: ID!
  title: String
  date: Date @dateformat
  comments: [Comment] @link(by: "fooId")
}
```

Declarative, composable resolvers

```
type FooJson {
  date: Date @dateformat
  localizedDate: Date
    @proxy(from: "date")
    @dateformat(locale: "de")
}
```

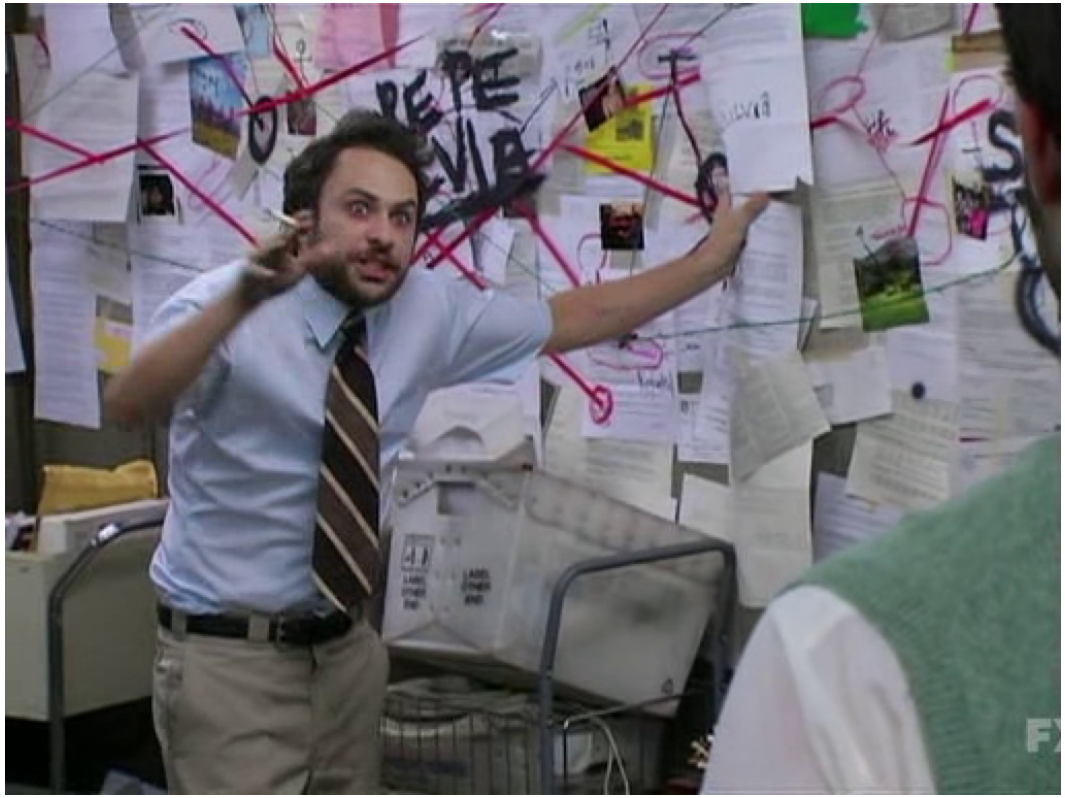
Generated query roots - source

```
type FooJson implements Node {
  id: ID!
  title: String
  date: Date
}
```

Generated query roots

```
allFoo(filter: {
  title: { eq: "Foo" }
  sort: { fields: date }
})
```

How it's done


PETE
EVIL

How it's done

1. User types
2. Inferred types
3. Merging
4. Service type generation
5. Query generation
6. Third-party schemas

Takeaways

Takeaway 1

You might not need it

Takeaway 2

Use if non-graphql source of truth

Takeaway 3

Use to simplify repeated types

Takeaway 4

Type Registry is a key pattern

Takeaway 5

Use graphql-compose ;)

Thank you!

@freiksenet

Mikhail Novikov