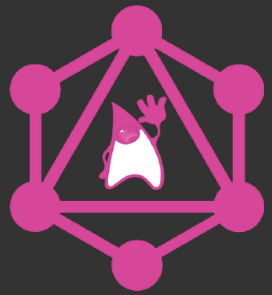


GraphQL & Java

Keep it dynamic

What?

- Data description + query language
- Type system ~~(think WSDL/SOAP)~~
- Open source spec (tnx Facebook!)
- Server-side runtime for executing queries
- Unrelated to graph DBs and their query languages (Cypher, Gremlin etc)
- Storage / language independent (think REST)

Why?

Describe data

```
type Query {
  hero (episode: Int): Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
}

type Planet {
  name: String
  type: String
}
```

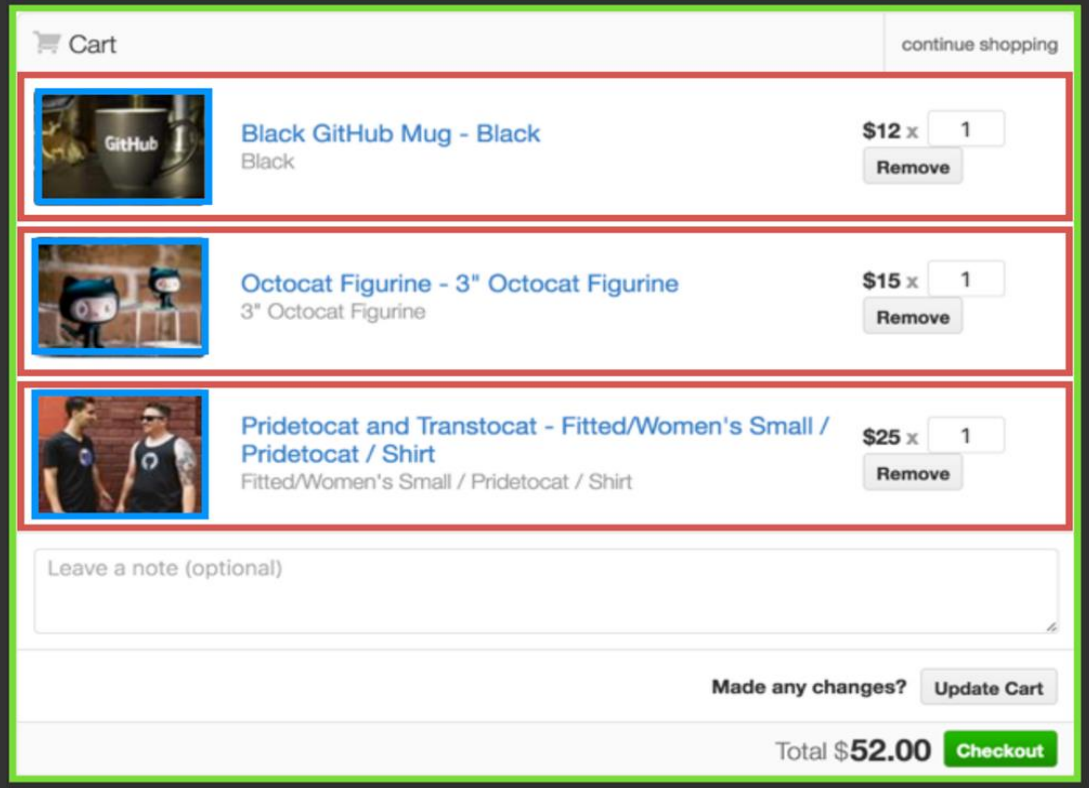
No under/over-fetch

```
{
    hero (episode: 4) {
        name
        homeWorld {
            name
            type
        }
    }
}
```

Predictable results

```
{
  "data" : {
      "hero" : {
        "name" : "Luke Skywalker"
        "homeWorld" : {
            "name" : "Tatooine"
            "type" : "desert"
        }
      }
}
```

GraphQL vs REST



Credit: [Marc-Andre Giroux]

/carts/1

/products/1

/products/2

/products/3

/product_images/1

/product_images/2

/product_images/3

GraphQL vs REST

/carts/1?expand=products

/carts/1?fields=products(name, description, price)

/cart_with_all_the_stuff_i_need

/cart_version_2_with_all_the_things

/cart_with_all_the_stuff_i_need_but_not_description

Server

Update endpoints

Update models

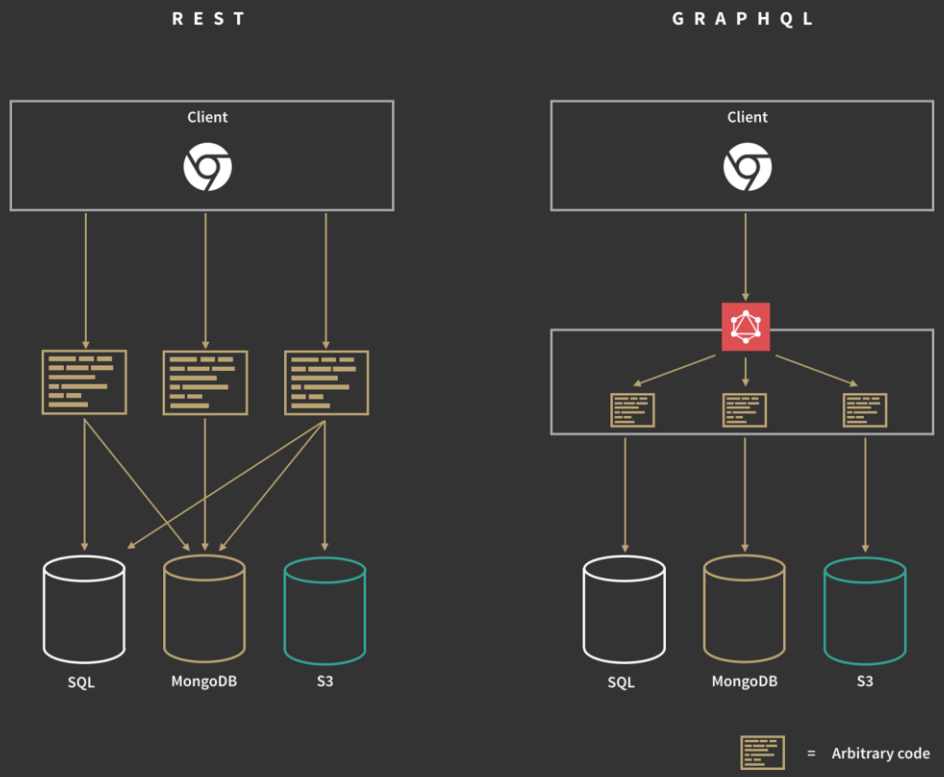
HATEOAS

GraphQL vs REST

```
{
    cart {
        total
        products {
            name
            price
            quantity
        }
    }
}
```

```
{
  "data": {
      "cart" : {
        "total" : 27.00
        "products" : [
            {
                "name" : "Mug Black"
                "price" : 12.00
                "quantity" : 1
            },
            {
                "name" : "Octocat figurine"
                "price" : 15.00
                "quantity" : 1
            }
        ]
    }
}
```

GraphQL vs REST



Credit: [Jonas Helfer]

Defining the schema : Code

```
GraphQLObjectType address = newObject()
        .name("Address")
        .field(newFieldDefinition()
                .name("streetName")
                .type(Scalars.GraphQLString)
                .build())
        .field(newFieldDefinition()
                .name("houseNumber")
                .type(Scalars.GraphQLInt)
                .build())
        .build();
```

```
GraphQLObjectType person= newObject()
        .name("Person")
        .field(newFieldDefinition()
                .name("addresses")
                .type(new GraphQLList(address))
                .dataFetcher(env ->
                        dataBase.query(...)))
        .field(newFieldDefinition()
                .name("name")
                .type(Scalars.GraphQLString))
        .build();
```

Defining the schema : Code

```
GraphQLObjectType queryType = newObject()
        .name("ROOT_QUERY")
        .field(newFieldDefinition()
            .name("peopleByName")
            .argument(newArgument()
                .name("name")
                .type(Scalars.GraphQLString))
            .type(new GraphQLList(person))
            .dataFetcher(env ->
                personService.findByName(...)))
        .build();
```

```
class Person {

    private String name;
    private List<Address> addresses;

    //getters & setters
}

class Address {
    ...
}

class PersonService {

    public List<Person> findByName(String name) {
        ...
    }
}
```

Defining the schema : SDL

```
schema {
    query: Query
}

type Query {
    peopleByName(name: String): [Person]
}

type Person {
    name: String
    address: Address
}

type Address {
    streetName: String
    houseNumber: Int
}
```

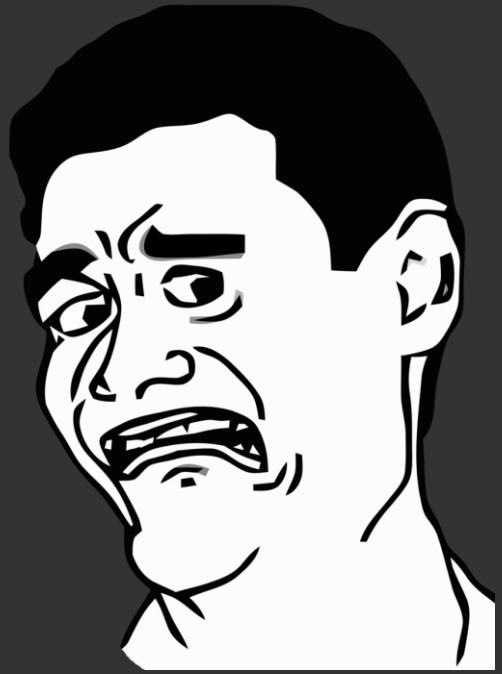
```
TypeDefinitionRegistry typeRegistry =
schemaParser.parse(schemaFile);

RuntimeWiring wiring = RuntimeWiring.newRuntimeWiring()
    .type("Query", typeWiring -> typeWiring
        .dataFetcher("peopleByName", env -> {
            String name = (String) env.getArguments().get("name");
            return personService.findByName(name))
    })
    .build();

GraphQLSchema schema = new SchemaGenerator()
        .makeExecutableSchema(typeRegistry, wiring);
```

The problem : Duplication

- Duplicates type definitions
- Another layer to maintain



The solution : Dynamic schema generation

- Eliminate type duplication (DRY)
- Allow following best practices
- Be adaptable / customizable

Introducing GraphQL SPQR

SPQR

GraphQL SPQR

```
public class PersonService {

    @GraphQLQuery(name = "peopleByName")
    public List<Person> findByName(String name) {

        return dataBase.queryByName(name);
    }
    ...
}
```

GraphQL SPQR : Setup

```
public class Person {

    private String firstName;
    private String lastName;

    @GraphQLQuery(name = "firstName")
    public String getFirstName() {
        return firstName;
    }

    @GraphQLQuery(name = "lastName")
    public String getLastName() {
        return lastName;
    }
...
}
```

```
PersonService personService = new PersonService();


GraphQLSchema schema = new GraphQLSchemaGenerator()
            .withOperationsFromSingleton(personService)
            .generate();

GraphQL graphQL = new GraphQL(schema);


String query = "
      {
         peopleByName(firstName: "John") {
               firstName
               lastName
       }}";

  ExecutionResult result = graphQL.execute(query);
```

GraphQL SPQR : Queries

```
public class PersonService {

    @GraphQLQuery(name = "twitterProfile")
    public TwitterProfile getTwitterProfile (
            @GraphQLContext  Person person) {

        return twitterApi.getProfile(person);
    }

    @GraphQLQuery(name = "peopleByName")
    public List<Person> findByName( ... ) {
        ...
    }
}
```

```
{
    twitterProfile (
            person: {
                firstName : "John"
                lastName : "Doe"
            })
        handle
        numberOfTweets
    }
}
```

```
{
    peopleByName (firstName : "John") {
        firstName
        lastName
        twitterProfile {
            handle
            numberOfTweets
        }
    }
}
```

Demo

DEMO TIME

The problem: Malicious queries

- Arbitrarily complex query
- Arbitrary size of the result

- Analyze AST complexity
- Limit depth
- Limit execution time
- Whitelist queries

The problem: N + 1

```
{
    peopleByName (firstName : "John") {

        firstName
        lastName

        twitterProfile {
            handle
            numberOfTweets
        }

    }
}
```

```
public class PersonService {

    @Batched
    @GraphQLQuery(name = "twitterProfile")
    public List<TwitterProfile> getTwitterProfile (
                        List<Person> person) {

        return  twitterApi.getProfile(person);
    }
```

The problem: N + 1

```
BatchLoader<Long, Person> userBatchLoader = new BatchLoader<Long, Person>() {
    @Override
    public CompletionStage<List< Person>> load(List<Long> userIds) {
        return CompletableFuture.supplyAsync(() -> {
            return personService.loadByIds(userIds);
        });
    }
};

DataLoader<Long, Person> userLoader = new DataLoader<>(userBatchLoader); //request scoped?

CompletableFuture<Person> user = userLoader.load(1L); //inside DataFetcher

CompletableFuture<Person> user = userLoader.load(2L); //another DataFetcher

userLoader.dispatch(); //somewhere… IMPORTANT!
```

Batched

The problem: Optimize fetching

```
{
    peopleByName (name: "John") {
        firstName
        lastName
    }
}
```

```
@GraphQLQuery(name = "peopleByName")
public List<Person> getPeopleByName(String name) {
    SELECT * FROM Users WHERE name = 'John';
    return …;
}
```

☹

The problem: Optimize fetching

```
{
    peopleByName (name: "John") {
        firstName
        lastName
    }
}
```

```
@GraphQLQuery(name = "peopleByName")
public List<Person> getPeopleByName(String name,
        @GraphQLEnvironment List<String> subFields) {

    SELECT firstName, lastName FROM Users
                    WHERE name = 'John';

    return  …;
}
```

The problem: Authorization

- Inject security context into the resolver

```
@RequestMapping(value="/graphql")
public Object graphql(@RequestBody Map<String, Object> request) {

    String query = request.get("query").toString();
    User user = SecurityContextHolder.getContext().getAuthentication();

    ExecutionResult executionResult = graphQL.execute(query, user);
}


public String getProfile(..., @GraphQLRootContext User user) {
        if (current.getId() == ...) {
            return ...
        }
}
```

The problem: Authorization

- AOP style

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
@GraphQLMutation(name = "updateUser")
public String updateUser(...) {
    ...
}
```

Further topics

- Schema visibility
- Schema extension
- Caching
- …

Closing

Bojan Tomić

leangen.io

github.com/leangen

@kaqqao

Questions