GraphQL for D: Do the Boring Things

Dr. Robert Schadek

October 22, 2021

Introduction in GraphqlD

Graphql is a query language for the web
GraphqlD is a server implementation of the Graphql standard

Introduction in GraphqlD

Graphql is a query language for the web
GraphqlD is a server implementation of the Graphql standard

Think, a bad SQL implementation returning JSON

The Interesting Bits

The Interesting Bits (Parsing)

- graphql is a LL(1) language

```
mutation updateUser($userId: String!, $name: String!) {
  updateUser(id: $userId, name: $name) {
    name
  }
}
```

- creating a parser generator for LL(1) grammars is good fun

The Interesting Bits (Validation)

- Known type names
- No undefined variables
- Unique argument names
- No fragment cycles
- Scalar leafs
- ...

```
mutation updateUser($name: String!) {
  updateUser(id: $userId, name: $name) {
    name
  }
}
```

- AST traversal, etc.

The Interesting Bits (Transformation)

- Turning D `structs` into graphql types

```d
struct Query {
  Starship[] shipsSelection(long[] ids);
  Character captain(Series series);
  SearchResult search(string name);
  Nullable!Starship starship(long id);
  Starship starshipDoesNotExist();
  Starship[] starships(float overSize = 100.0);
  Nullable!Character character(long id);
  Character[] character(Series series);
  Humanoid[] humanoids();
  Android[] androids();
  Android[] resolverWillThrow();
  GQLDCustomLeaf!(DateTime, dtToString, stringToDT)
      currentTime();
  int currentTime();
  Starship numberBetween(Input searchInput);

  @GQLDUda(Ignore.yes)
  void ignoreMe() {
  }
}
```

```
1  type queryType {
2    shipsSelection(ids: [Int!]!): [Starship!]!
3    currentTime: DateTime!
4    starships(overSize: Float!): [Starship!]!
5    captain(series: Series!): Character!
6    humanoids: [Humanoid!]!
7    starship(id: Int!): Starship
8    androids: [Android!]!
9    resolverWillThrow: [Android!]!
10   numberBetween(searchInput: Input!): Starship!
11   search(name: String!): SearchResult!
12   character(id: Int!): Character
13 }
```

The Interesting Bits

- Most interesting bits are done
- We can actually get some work done

The Boring Bits

Resolving a request

```
{
  starships(overSize: 120.0) {
    id
    designation
  }
}
```

Resolving a request

```
1  struct CustomContext {
2    int userId;
3  }
4
5  GQLDOptions opts;
6
7  auto graphqld = new GraphQLD!(Schema,CustomContext)(opts);
8
9  string toParse = "...";
10 Json vars;
11
12 auto l = Lexer(toParse);
13 auto p = Parser(l);
14 Document d = p.parseDocument();
15
16 CustomContext con;
17
18 Json result = graphqld.execute(d, vars, con);
```

Writing a Resolver

```
graphqld.setResolver("queryType", "starships",
    delegate(string name, Json parent, Json args
        , ref CustomContext con) @safe
    {
      Json ret = Json.emptyObject;
      ret["data"] = Json.emptyArray;
      float overSize = args["overSize"].to!float();
      foreach(ship; database.ships) {
        if(ship.size > overSize) {
          Json tmp = starshipToJson(ship);
          ret["data"] ~= tmp;
        }
      }
      return ret;
    }
  );
```

starshipToJson

```
1  Json starshipToJson(Starship s) {
2    Json ret = Json.emptyObject();
3
4    ret["data"]["__typename"] = "Starship";
5    ret["data"]["id"] = s.id;
6    ret["data"]["designation"] = s.designation;
7    ret["data"]["name"] = s.name;
8    ret["data"]["size"] = s.size;
9
10   return ret;
11 }
```

Default Resolver

```
{
  starships(overSize: 120.0) {
    id
    designation
  }
}
```

Default Resolver

```
1
2  Json defaultResolver (string name, Json parent, Json args
3      , ref Con context)
4  {
5    Json ret = Json.emptyObject();
6    if(parent.type == Json.Type.object && name in parent) {
7      ret["data"] = Json.emptyObject();
8      ret["data"] = parent[name];
9    } else {
10     ret["errors"] = Json.emptyArray();
11     ret["errors"] ~= Json(["message" : format(
12       "no field name '%s' found on type '%s'"
13         , name, parent.getWithDefault!string("__typename")
14       )]);
15   }
16   return ret;
```

Registering a Resolver

```
1  alias ResolverType = delegate(string name, Json parent
2      , Json args, ref CustomContext con) @safe;
3
4  void registerResolver(
5      GraphQLD!(Schema,CustomContext) endpoint
6      , string type, string name, ResolverType resolver)
```

Registering a Resolver

```
1  void registerResolver(
2      GraphQLD!(Schema,CustomContext) endpoint
3      , string type, string name, ResolverType resolver)
4  {
5    auto histo = new Histogram(format("%s_%s_histo", type, name)
6        , "", [], restBuckets()
7      );
8    histo.register();
9
10   endpoint.setResolver(type, name,
11     delegate(string name, Json parent, Json args
12       , ref CustomContext context) @safe
13     {
14       auto sw = StopWatch(AutoStart.yes);
15       scope(exit) {
16         histo.observe(sw.peek().total!"msecs"());
17       }
18       return resolver(name, parent, args, context);
19     });
20 }
```

Registering a Resolver

```
registerResolver(graphqld, "queryType", "starships"
    , delegate(string name, Json parent, Json args
        , ref CustomContext con) @safe
    {
      Json ret = Json.emptyObject;
      ret["data"] = Json.emptyArray;
      float overSize = args["overSize"].to!float();
      foreach(ship; database.ships) {
        if(ship.size > overSize) {
          Json tmp = starshipToJson(ship);
          ret["data"] ~= tmp;
        }
      }
      return ret;
    }
  );
```

Registering a Resolver

```
1  registerResolver(graphqld, "queryType", "starships"
2      , delegate(string name, Json parent, Json args
3          , ref CustomContext con) @safe
4      {
5        Json ret = Json.emptyObject;
6        ret["data"] = Json.emptyArray;
7        float overSize = args["overSize"].to!float();
8        foreach(ship; database.ships) {
9          if(ship.size > overSize) {
10           Json tmp = starshipToJson(ship);
11           ret["data"] ~= tmp;
12         }
13       }
14       return ret;
15     }
16   );
```

- auth not handled

Authentification forwarder

```
ResolverType checkAuth(ResolverType resolver) {
  ResolverType ret = delegate(string name, Json parent
      , Json args, ref CustomContext context) @safe
    {
      if(context.userId == 0) {
        Json ret = Json.emptyObject();
        ret["errors"] = Json([
            Json([ "message": "You are not authenticated" ])
          ]);
        return ret;
      }

      return resolver(name, parent, args, context);
    };
  return ret;
}
```

Refactoring out the actual function

```
1  Json getStarships(string name, Json parent, Json args
2      , ref CustomContext con) @safe
3  {
4    Json ret = Json.emptyObject;
5    ret["data"] = Json.emptyArray;
6    float overSize = args["overSize"].to!float();
7    foreach(ship; database.ships) {
8      if(ship.size > overSize) {
9        Json tmp = starshipToJson(ship);
10       ret["data"] ~= tmp;
11     }
12   }
13   return ret;
14 }
```

Using auth forwarder

```
1  registerResolver(graphqld, "queryType", "starships"
2      , checkAuth(toDelegate(&getStartships));
```

The graph in graphql

```
{
  starships {
    name
    commander {
      name
      ship {
        name
      }
    }
  }
}
```

```
struct Starship {
  long id;
  string name;
  string designation;
  double size;
  Character commander;
  Character[] crew;
}

struct Character {
  long id;
  string name;
  Character[] commands;
  Starship ship;
}
```

The graph in graphql

```
{
  starships {
    name
    commander {
      name
      ship {
        name
      }
    }
  }
}
```

```
struct Starship {
  long id;
  string name;
  string designation;
  double size;
  Character commander();
  Character[] crew;
}

struct Character {
  long id;
  string name;
  Character[] commands;
  Starship ship();
}
```

The graph in graphql

```
{
  starships {
    name
    commander {
      name
      ship {
        name
      }
    }
  }
}
```

```
struct Starship {
  long id;
  string name;
  string designation;
  double size;
  Character commander();
  Character[] crew;
}

struct Character {
  long id;
  string name;
  Character[] commands;
  Starship ship();
}
```

- error: undefined reference to _D14implementation9Character4shipMFZSQBj8Starship

The graph in graphql

```
{
  starships {
    name
    commander {
      name
      ship {
        name
      }
    }
  }
}
```

```
struct Starship {
  long id;
  string name;
  string designation;
  double size;
  NullableStore!(Character) commander;
  Character[] crew;
}

struct Character {
  long id;
  string name;
  Character[] commands;
  NullableStore!(Starship) ship;
}
```

Going deep

```
{
  starships {
    name
    crew {
      name
      ship {
        name
        crew {
          name
          ship {
            name
            crew {
              name
            }
          }
        }
      }
    }
  }
}
```

```
alias ResolverType = delegate(string name, Json parent
    , Json args, ref CustomContext con) @safe;

struct CustomContext {
  int userId;
}
```

Commercial break

Symmetry Investments

Custom Primitive Types

- Int
- Float
- Boolean
- String
- ID (String)

Custom Primitive Types

- Int
- Float
- Boolean
- String
- ID (String)
- DateTime ?

Custom Primitive Types

```
QDateTime = GQLDCustomLeaf!(DateTime, dtToString, stringToDT);

string dtToString(DateTime dt) {
  return dt.toISOExtString();
}

DateTime stringToDT(string s) {
  return DateTime.fromISOExtString(s);
}
```

Conclusion

Conclusion

- One more layer of indirection
- D + Graphql = good fit
- Not just a toy anymore

https://github.com/burner/graphqld

The End