Intro to GraphQL

Josh Price

github.com/joshprice
@joshprice

Background and motivation

REST is great

- HTTP transport (request/response)
- JSON data representation (vs XML)
- Resource oriented (easy to model)

REST is webby

- URL based (easy to develop against/debug)
- APIs work like the web does
- Discoverability
- Caching for free
- Obvious how it works (for web devs)

REST is ***hard*** in practice

- Big upfront design required
  - need to anticipate all future clients and their needs
  - impossible to get right
- Hard to change (versioning problems)
- Maintainability issues

REST is ***hard*** in practice

- No one true way to do REST
- Badly designed APIs can significantly hamper the design of great frontends
- No type information
    - `user` and `friend` are different resources but share some interface
- Single way to describe relationship hierarchy (ie nesting)

REST is *hard* in practice

- Poor performance (1 + N resource traversals)
    - fetch user, then fetch friends, then fetch pets
- or complex handrolled API to batch requests

```
GET /users/1/friends/1/pets/1?
include=user.name,friend.name,pet.age
```

REST limitations part 1

- REST assumes that like the web the client hasn't ever seen API before
  - Internal mobile/web clients saw it <10ms ago
- Can't easily model realtime updates via server push
  - HTTP 2.0 / Websockets
  - realtime web is coming

REST limitations part 2

- Too much overhead for tightly coupled clients & servers
  - clients traversing resource tree need to make > 1 request in serial
  - dynamic traversals
- No one true way means lot's of handrolled code and documentation

Complex query in REST

```
GET /users/1/friends/1/pets/1?
include=user.name,friend.name,dog.age
```

Complex query in GraphQL

```
{
  user(id: 1) {
    name
    friends {
      name
      pets {
        age
      }
    }
  }
}
```

What is GraphQL?

What is GraphQL?

- It's a *specification* for client/server interaction
- Language independent
- A DSL for defining queries for data
- A DSL for defining data types in a schema
- A specification for the query execution engine

What is GraphQL?

- Strongly typed
- Super flexible
- Queries are declarative
- They have the same shape as the response
- Queries can provide all data required by view in single query

What GraphQL is not

- Nothing to do with graphs
- Not language specific (many implementations)

Properties of GraphQL core

- Accepts queries and executes them against a schema
- Not tied to HTTP, JSON or data store
- Single `GraphQLexecute(schema, query)` function

Plug GraphQL

- Single endpoint (entire API at a single URL)
- GET or POST queries
- POST mutations
- Websockets for subscriptions

Lifecycle of a query

- Parse query to AST
  - we use leex and yecc
- Validation of query
  - ie fields which don't match schema data types
- Execution of query
  - executes resolve functions for each required data type or

Simple Schema

```
%Schema{
  query: %ObjectType{
    name: "SimpleQuery",
    fields: %{
      greeting: %{
        type: %String{},
        resolve: fn (_, _, _) -> "Hello, world!" end
      }
    }
  }
}
```

Simple Query

Query

```
{ greeting }
```

JSON response

```
{
  "data": {
    "greeting": "Hello, World!"
  }
}
```

Data Access Schema

```
@items %{"a" => %{id: "a", name: "Foo"}, "b" => %{id: "b", name: "Bar"}}

%Schema{
  query: %ObjectType{
    fields: %{
      item: %{
        type: %Item{},
        args: %{id: %{type: %String{}}},
        resolve: fn(_, %{id: id}, _) -> Map.get(@items, id) end
      }
    }
  }
}
```

Data Access Query

Query

```
{
  item(id: "a") {
    id
    name
  }
}
```

JSON response

```
{
  "data": {
    "id": "a",
    "name": "Foo"
  }
}
```

Demo

Resources

- http://graphql-elixir.org
- https://github.com/joshprice/graphql-elixir
  - Hex: graphql
- https://github.com/joshprice/plug_graphql
  - Hex: plug_graphql
- http://playground.graphql-elixir.org

Fin