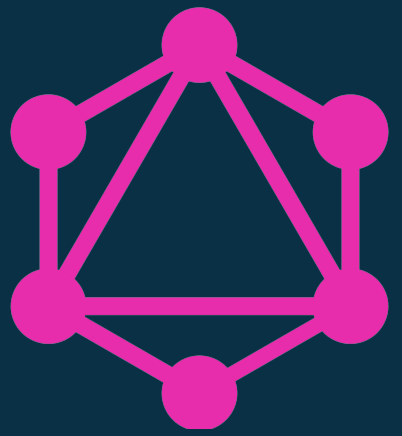
基于Slick的GraphQL服务



+

庄名洲<mingzhou.zhuang@gmail.com>

2018-06-30

目录

- GraphQL
- Sangria
- Slick

GraphQL是什么

- Facebook 2012年开发，2015年开源，用于替代REST API的，
- 更严格、可扩展、可维护的数据查询规范。

如何设计API?

```
type Author {
  id: Int
  firstName: String
  lastName: String
  posts: [Post]
}
type Post {
  id: Int
  title: String
  text: String
  views: Int
  author: Author
}
type Query {
  author(id: Int): Author
  allAuthors: [Author]
}
```

Public API

/authors/1234

/authors/1234/firstName

/authors/1234/posts

/posts/5678

REST API存在的问题

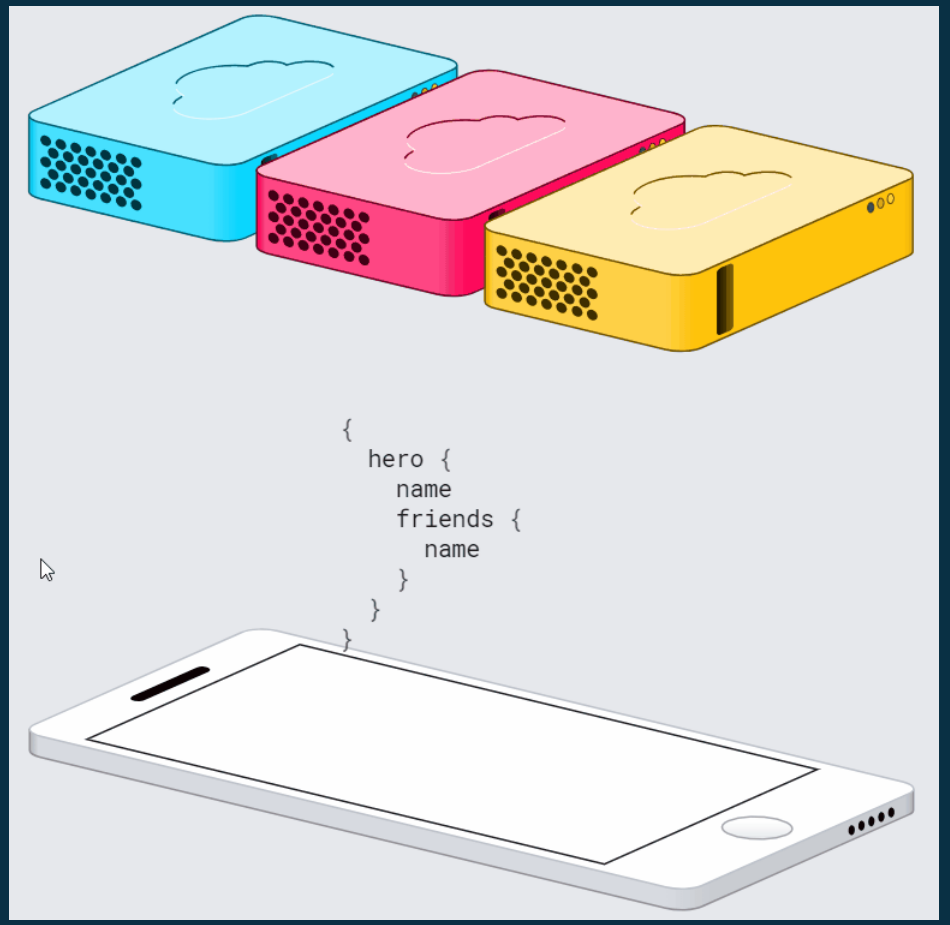
- 字段冗余
  - 明明只要一棵树，你却给我整个森林?
- 类型校验
  - 参数是必选还是可选，数字还是字符串?
- 多个接口
  - 接口之间没有显式的逻辑关系，一个简单的逻辑也需要多次调用才能满足。
- 接口升级
  - 增改字段，要么接口数量爆炸，要么旧程序原地爆炸。
- 维护文档
  - 接口修改一时爽，文档维护火葬场。
- 全栈调用
  - 跨语言，以统一的格式进行数据交换。

按需请求、按需响应

```
{
  hero {
    name
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

多个接口，一次查询



类型校验，必选可选

强类型系统，明确定义业务Schema

```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

```
type Query {
  hero: Character
}

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

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

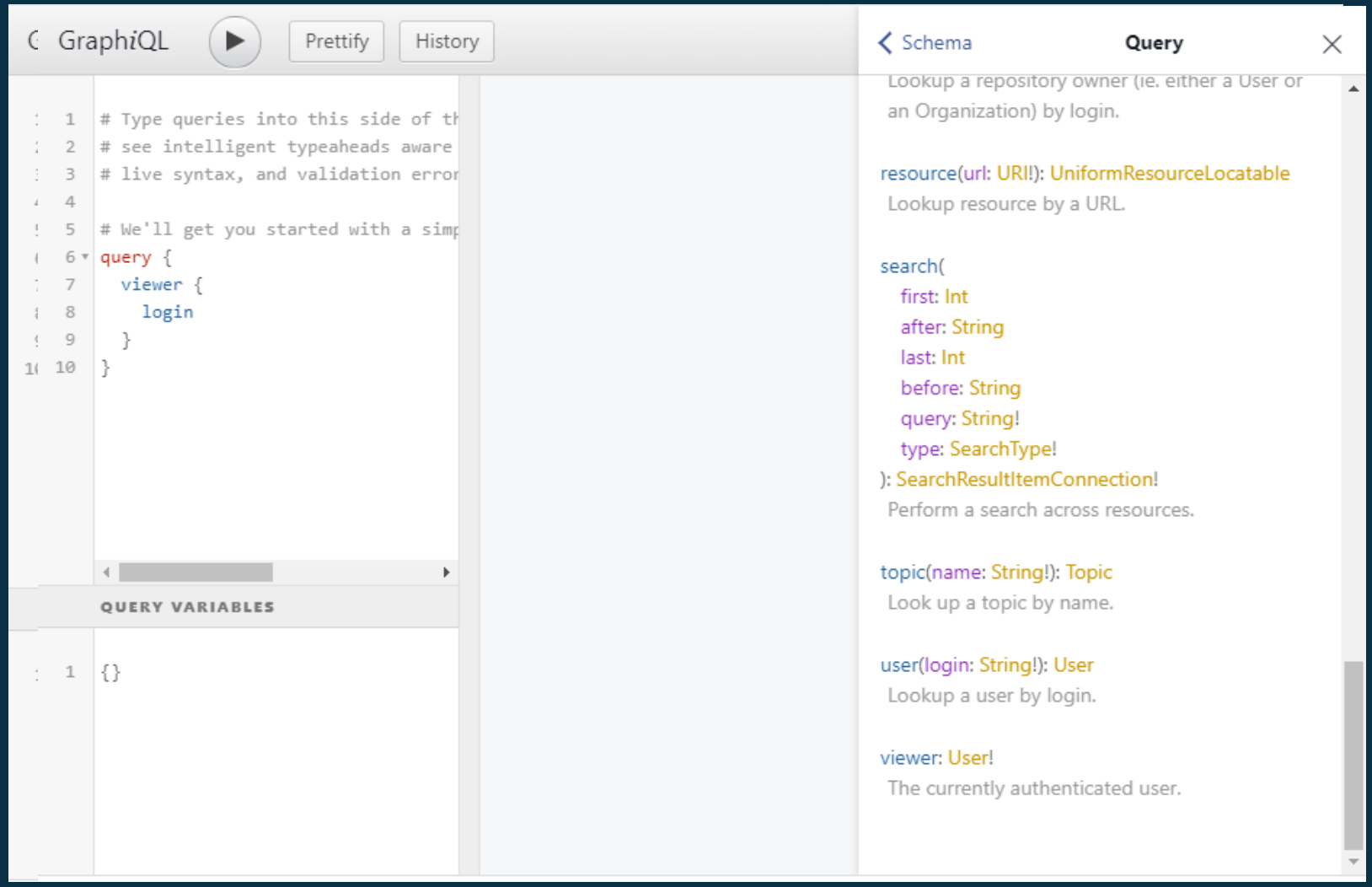
接口升级，随心所欲

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



}
```

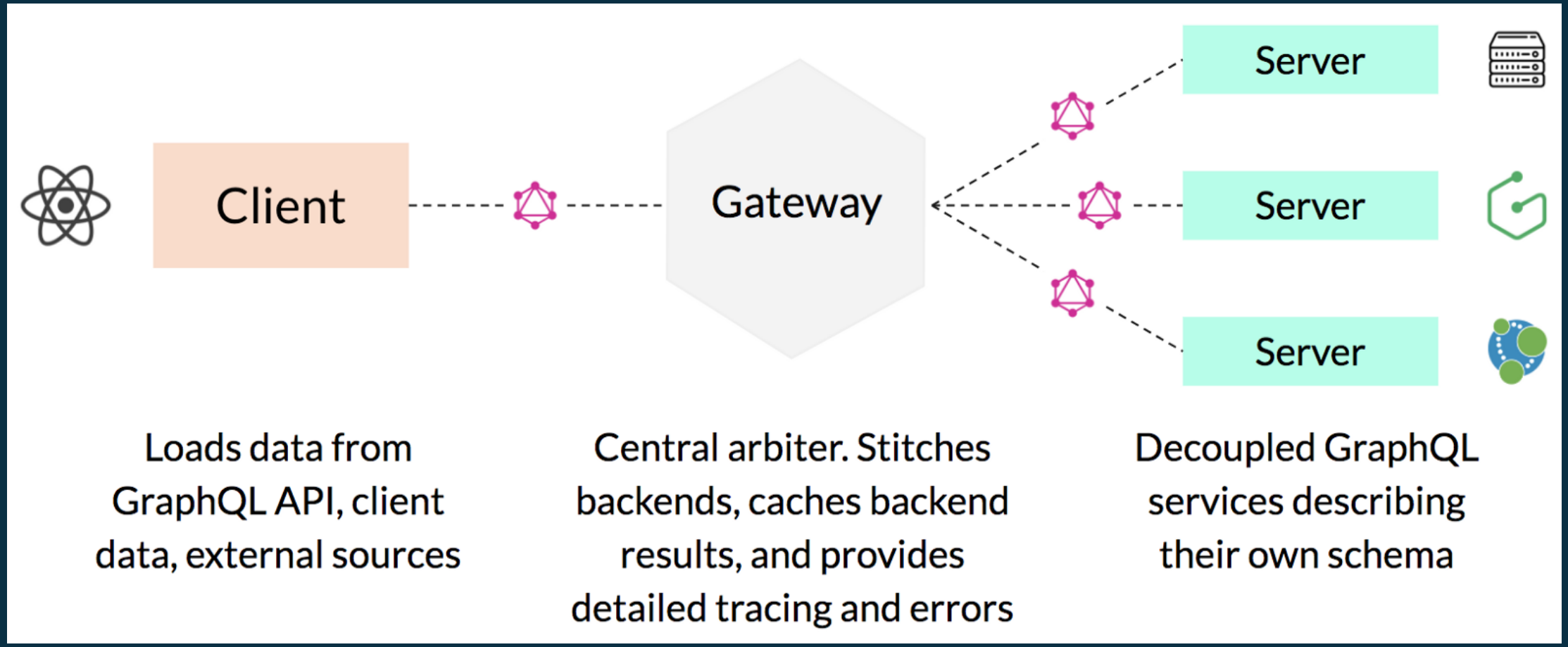
文档维护，新人培训，省心省力



数据交换，跨越语言

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

微服务接口联结


Client
Gateway
Server
Server
Server
Loads data from GraphQL API, client data, external sources
Central arbiter. Stitches backends, caches backend results, and provides detailed tracing and errors
Decoupled GraphQL services describing their own schema

GraphQL小结

- GraphQL != Framework/Library
- GraphQL = Specification
- 数据查询规范
  - 更严格
  - 可扩展
  - 可维护

Sangria 

- Sangria是GraphQL规范在Scala语言上的最佳实现

使用Sangria

- 定义Scalar Type
- 定义Schema
- 定义Resolver
- 定义Executor
- 定义Result Marshalling & Input Unmarshalling

Sangria
Schema

```
@GraphQLName("AuthUser")
@GraphQLDescription("A user of the system.")
case class User(
  @GraphQLDescription("User ID.")
  id: String,

  @GraphQLName("userPermissions")
  @GraphQLDeprecated("Will not be exposed in future")
  permissions: List[String],

  @GraphQLExclude
  password: String)

val UserType = deriveObjectType[MyCtx, User]()
val UserInputType = deriveInputObjectType[User](
  InputObjectTypeName("UserInput"))
```

Sangria
Resolver

```
val HeroOnlyQuery = ObjectType[CharacterRepo, Unit](
  "HeroOnlyQuery", fields[CharacterRepo, Unit](
    Field("hero", TestSchema.Character,
      arguments = TestSchema.EpisodeArg :: Nil,
      resolve = ctx ⇒ ctx.ctx.getHero(ctx.argOpt(TestSchema.EpisodeArg)))
  ))

val heroOnlySchema = Schema(HeroOnlyQuery,
  additionalTypes = TestSchema.Human :: TestSchema.Droid :: Nil)
```

Sangria
Executor

```
import sangria.execution.Executor

Executor.execute(TestSchema.StarWarsSchema, queryAst,
  userContext = new CharacterRepo,
  deferredResolver = new FriendsResolver,
  variables = vars)
```

Sangria
Marshalling

```
import sangria.marshalling.sprayJson._

val result: Future[JsValue] = Executor.execute(TestSchema.StarWarsSchema, queryAst,
  variables = vars
  userContext = new CharacterRepo,
  deferredResolver = new FriendsResolver)
```

Sangria

```scala
case class ProjectionName(name: String) extends FieldTag
case object ProjectionExclude extends FieldTag

trait Projector[Ctx, Val, Res] extends (Context[Ctx, Val] ⇒ Action[Ctx, Res]) {
  val maxLevel: Int = Integer.MAX_VALUE
  def apply(ctx: Context[Ctx, Val], projected: Vector[ProjectedName]): Action[Ctx, Res]
}

object Projector {
  def apply[Ctx, Val, Res](fn: (Context[Ctx, Val], Vector[ProjectedName]) ⇒ Action[Ctx, Res]) : Projector[Ctx, Val, Res] =
    new Projector[Ctx, Val, Res] {
      def apply(ctx: Context[Ctx, Val], projected: Vector[ProjectedName]) = fn(ctx, projected)
      override def apply(ctx: Context[Ctx, Val]) = throw new IllegalStateException("Default apply should not be called on projector!")
    }

  def apply[Ctx, Val, Res](levels: Int, fn: (Context[Ctx, Val], Vector[ProjectedName]) ⇒ Action[Ctx, Res]) : Projector[Ctx, Val, Res] =
    new Projector[Ctx, Val, Res] {
      override val maxLevel = levels
      def apply(ctx: Context[Ctx, Val], projected: Vector[ProjectedName]) = fn(ctx, projected)
      override def apply(ctx: Context[Ctx, Val]) = throw new IllegalStateException("Default apply should not be called on projector!")
    }
}

case class ProjectedName(name: String, children: Vector[ProjectedName] = Vector.empty) {
  lazy val asVector = {
    def loop(name: ProjectedName): Vector[Vector[String]] =
      Vector(name.name) +: (name.children flatMap loop map (name.name +: _))

    loop( name = this)
  }
}
```

- Slick(Scala Language-Integrated Connection Kit)
- Functional Relational Mapping
- Typesafe
- Composable
- Reactive

Typesafe

```scala
/**
  * get a stock list use the giving pattern
  *
  * @param pattern stock code match pattern
  * @return a stock list, configured return max 10
  */
def getAShareStockCodeList(pattern: String, limit: Int = 10): Future[Seq[GL_AShareStockList]] = {
  val q = for {
    x <- Asharedescription.filter(_.sInfoWindcode.startsWith(pattern)).take(limit)
  } yield (x.sInfoWindcode.get, x.sInfoName)
    .mapTo[GL_AShareStockList]
  db.run(q.result)
}
```

Ugly

```
def get_AShareDescription(projections: Vector[ProjectedName], symbol: Option[String], name: Option[String]): Future[Seq[GL_AShareDescription]] = {
  val schemaFields = Seq("symbol", "name", "company_name", "company_name_en", "isin_code", "exchange", "list_board", "list_date", "delist_date", "cur
  val fieldMap = Map(
    "symbol" ->"""S_INFO_WINDCODE""",
    "name" ->"""S_INFO_NAME""",
    "company_name" ->"""S_INFO_COMPNAME""",
    "company_name_en" ->"""S_INFO_COMPNAMEENG""",
    "isin_code" ->"""S_INFO_ISINCODE""",
    "exchange" ->"""S_INFO_EXCHMARKET""",
    "list_board" ->"""S_INFO_LISTBOARD""",
    "list_date" ->"""S_INFO_LISTDATE""",
    "delist_date" ->"""S_INFO_DELISTDATE""",
    "currency" ->"""CRNCY_CODE""",
    "pin_yin" ->"""S_INFO_PINYIN""",
    "list_board_name" ->"""S_INFO_LISTBOARDNAME"""
  )
  val selectFields = genSelect(projections, schemaFields, fieldMap)
  implicit val getResult: AnyRef with GetResult[GL_AShareDescription] = GetResult(r =>
    GL_AShareDescription(r.<<, r.<<, r.<<, r.<<, r.<<, r.<<, r.<<, r.<<, r.<<, r.<<, r.<<, r.<<))
  val condition = ArrayBuffer[String]()
  if (symbol.isDefined) condition += buildWhereCondition( db_field = "S_INFO_WINDCODE",  op = "=", symbol.get)
  if (name.isDefined) condition += buildWhereCondition( db_field = "S_INFO_NAME",  op = "=", name.get)
  val where = condition.mkString("AND")
  val query = if (where.isEmpty) {
    sql"""
      SELECT #$selectFields FROM "dbo"."ASHAREDESCRIPTION" """.stripMargin.as[GL_AShareDescription]
  } else {
    sql"""
      SELECT #$selectFields FROM "dbo"."ASHAREDESCRIPTION"
      WHERE #$where""".stripMargin.as[GL_AShareDescription]
  }
  db.run(query)
}
```

相关链接

- https://graphql.org/
- https://www.apollographql.com/
- https://sangria-graphql.org/
- http://slick.lightbend.com/

谢谢大家！