CoDia 课达编程

GraphQL 接口设计

GraphQL 接口设计

目录

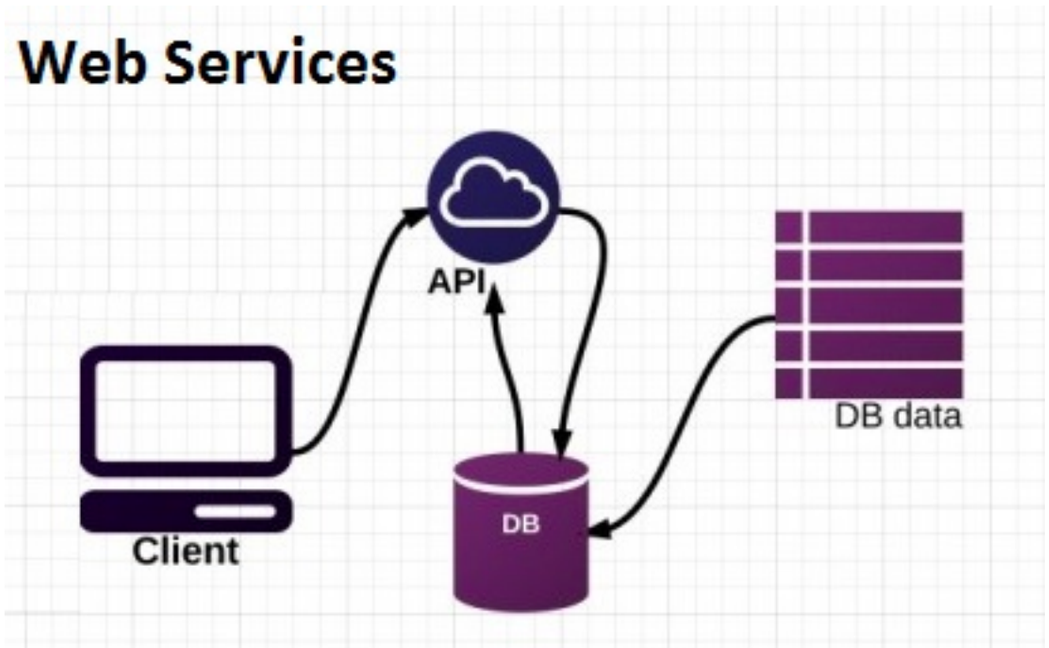
背景知识

- 后端接口
- GraphQL 介绍
- GraphQL API 设计

背景知识

后端接口

- 后端接口是什么?
  - 浏览器和服务器间的交互规范
  - 底层数据和实际产品间的抽象层次



背景知识

GraphQL 介绍

- GraphQL 是一种**查询**语言
  - 负责前后端交互
  - 提供统一接口

```
{
  hero {
    name
    height
    mass
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker",
    "height": 1.72,
    "mass": 77
  }
}
```

- GraphQL 是一种**数据描述**语言
  - 表达数据
  - 完成业务抽象

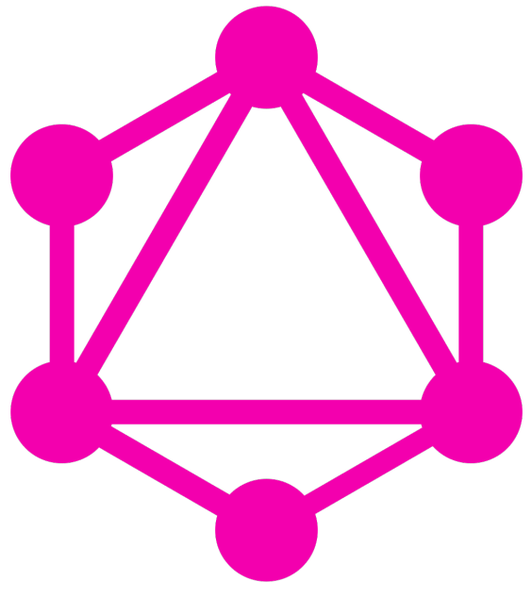
```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

```
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
}
```

背景知识

GraphQL API 设计

- 如何设计 GraphQL 接口？三个步骤：
    1. 利用图结构来建模业务
    2. 规定所支持的查询和操作
    3. 保持稳定的情况下扩展业务



GraphQL 业务建模

- 概述
- 对象和属性
- 关联关系
- 联合和接口类型

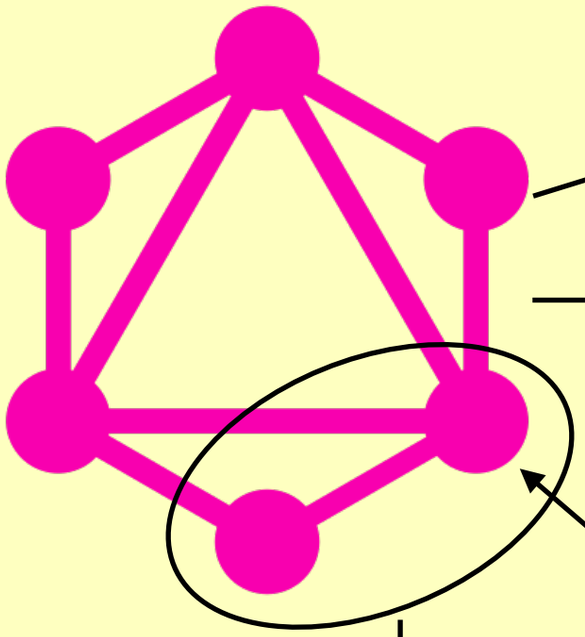
GraphQL 业务建模

概述

- 业务的表示和建模
  - 业务本身是模糊抽象的，需要一种具体的表示
  - 传统上通过特定函数表达业务中的特定用例、场景等
  - REST 风格接口围绕资源（对象）及对资源的操作表达业务
- GraphQL 建模业务
  - 区分建模本身和具体操作
  - 将业务建模为图结构

GraphQL 业务建模

GraphQL 建模



点：建模业务中的具体对象

边：建模业务中的关联关系

业务建模

权限控制

查询：提供图的一个入口、通过关联实现全图的查询

操作：专门提供修改和操作业务的接口

查询和操作

GraphQL 业务建模

对象和属性

- 对象
  - **自身信息**：基本类型字段
  - **关联的对象**：其他对象类型字段

```
type User {
  id: ID!
  login: String
  profile: Profile
}
```

```
type Profile {
  name: String
  studentId: String
  # ...
}
```

GraphQL 业务建模

关联关系

- 关联对象 vs 关联关系

关联对象	关联关系
用户创建了哪些**题目**?	用户**何时**创建的题目?
题目有哪些**同学**参与?	题目有**几个**同学通过?
群组中有哪些**成员**?	群组成员在群组资源上的**表现**?

GraphQL 业务建模

关联关系

- 关系的表达
  - 作为对象信息
  - 关系对象类型：connection 和 edge
- 关系建模最佳实践：
  - 1-1 关系
  - 1-n 关系
  - m-n 关系

关联关系
用户**何时**创建的题目?
题目有**几个**同学通过?
群组成员在群组资源上的**表现**?

GraphQL 业务建模

1-1 关系

```graphql
type CodingExerciseStatus {
  # ...
  submission: Submission
}
```

```graphql
type Submission {
  # ...
  status: CodingExerciseStatus
}
```

GraphQL 业务建模

1-n 关系：错误示范

```
type Group {
  # ...
  owner: User
  createdAt: DateTime
}
```

对象字段表达关联信息

```
type User {
  # ...
  createdGroups: [Group]
}
```

```
type Group {
  # ...
  owner: User
  createdAt: DateTime
}
```

统计信息难以表示

.....

GraphQL 业务建模

1-n 关系：connection 和 edge

edge 对象仅用于基本的分页功能

关联信息仍表示为对象字段

```
type GroupEdge {
  cursor: String
  node: Group
}
```

```
type Group {
  # ...
  owner: User
  createdAt: DateTime
}
```

```
type User {
  # ...
  createdGroups: GroupConnection
}
```

```
type GroupConnection {
  nodes: [Group]
  edges: [GroupEdge]
  pageInfo: PageInfo
  totalCount: Int
  viewerJoinedCount: Int
  # ...
}
```

```
type Group {
  # ...
  owner: User
  createdAt: DateTime
}
```

.....

connection 对象表达关联关系的整体统计

GraphQL 业务建模

m-n 关系

通过 edge 对象表达
关联关系的详细信息

```
type GroupMemberEdge {
  cursor: String
  node: User
  joinTime: DateTime
  joinedCodingExercises:
    GroupMemberJoinedCodingExerciseConnection
}
```

```
type User {
  # ...
  joinedGroups:
    UserJoinedGroupConnection
}
```

```
type Group {
  # ...
  members: GroupMemberConnection
}
```

```
type GroupMemberConnection {
  nodes: [User]
  edges: [GroupMemberEdge]
  pageInfo: PageInfo
  totalCount: Int
  # ...
}
```

```
type User {
  # ...
  joinedGroups:
    UserJoinedGroupConnection
}
```

connection 和
edge 根据字段命名

两种 connection 对象
维护双向联系

.....

```
type UserJoinedGroupConnection {
  nodes: [Group]
  edges: [UserJoinedGroupEdge]
  pageInfo: PageInfo
  totalCount: Int
  # ...
}
```

GraphQL 业务建模

建模练习

GraphQL 业务建模

建模练习

- 某题目有几个用户通过

```
type CodingExercise {
  joinedUsers:
    CodingExerciseJoinedUserConnection
}
```

```
type CodingExerciseJoinedUserConnection {
  nodes: [User]
  edges: [CodingExerciseJoinedUserEdge]
  pageInfo: PageInfo
  totalCount: Int
  passedCount: Int
}
```

m-n 关系，注意命名方式

GraphQL 业务建模

建模练习

- 某题目上某用户的提交记录

```
type CodingExercise {
  joinedUsers:
    CodingExerciseJoinedUserConnection
  joinedUser(userId): CodingExerciseJoinedUserEdge
}
```

```
type CodingExerciseJoinedUserEdge {
  cursor: String
  node: User
  codingExerciseStatuses:
    CodingExerciseStatusConnection
}
```

对于 m-n 关系，除提供列表查询外，还应提供单点查询

GraphQL 业务建模

建模练习

- 参与某题包的每个用户的最后一次作答记录

```
type ExercisePack {
  joinedUsers:
    ExercisePackJoinedUserConnection
}
```

```
type ExercisePackJoinedUserConnection {
  nodes: [User]
  edges: [ExercisePackJoinedUserEdge]
  pageInfo: PageInfo
  totalCount: Int
}
```

```
type ExercisePackJoinedUserEdge {
  cursor: String
  node: User
  lastSession: ExercisePackSession
}
```

注意 ExercisePackJoinedUserEdge 和 ExercisePackSession 之间是 1-n 关系

GraphQL 业务建模

建模练习

- 某次作答记录中每道题目上的提交次数

作答记录中的提交是题目上提交的子集，必须在 edge 上表达

```
type ExercisePackSession {
  codingExercises:
    ExercisePackSessionCodingExercisesConnection
}
```

```
type ExercisePackSessionCodingExercisesConnection {
  nodes: [CodingExercise]
  edges: [ExercisePackSessionCodingExercisesEdge]
  pageInfo: PageInfo
  totalCount: Int
}
```

```
type ExercisePackSessionCodingExercisesEdge {
  cursor: String
  node: CodingExercise
  codingExerciseStatuses:
    CodingExerciseStatusConnection
}
```

```
type CodingExerciseStatusConnection {
  # ...
  totalCount
}
```

GraphQL 业务建模

建模练习

- 提交错误类型分布

```
type CodingExerciseStatusConnection {
  # ...
  totalCount: Int
  passedCount: Int
  submissionDist: [SubmissionDistItem]
}
```

```
type SubmissionDistItem {
  type: Int
  count: Int
}
```

此处列表并非表达关系，
只是用于表达分布

GraphQL 业务建模

建模练习

- 用户在不同语言上的提交通过率

```
type User {
  codingExerciseStatusesByLang:
    [CodingExerciseStatusesByLang]
}
```

Language 并非对象，且数量有限，

```
type CodingExerciseStatusesByLang {
  lang: Language
  codingExerciseStatuses:
    CodingExerciseStatusConnection
}
```

GraphQL 业务建模

建模练习

- 群组中每个用户完成的群组内题包数量

```
type Group {
  members: GroupMemberConnection
}
```

```
type GroupMemberConnection {
  # ...
  edges: [GroupMemberEdge]
}
```

```
type GroupMemberEdge {
  # ...
  joinedExercisePacks:
    GroupMemberJoinedExercisePackConnection
}
```

```
type GroupMemberJoinedExercisePackConnection {
  totalCount: Int
  passedCount: Int
}
```

GroupMemberEdge 和 ExercisePack 是 m-n 关系

GraphQL 业务建模

建模练习

- 群组中每个用户在群组内题包上最后一次作答记录的总得分

```
type GroupMemberJoinedExercisePackConnection {
  # ...
  edges: [GroupMemberJoinedExercisePackEdge]
}
```

```
type GroupMemberJoinedExercisePackEdge {
  # ...
  lastSession: ExercisePackSession
}
```

GraphQL 业务建模

一些原则

- 首先建模为点和关联，然后再考虑接口设计
- 接口名称中体现关联的含义（加入如 joined, created 等）
- 不同接口允许冗余，只要建模的思维路径合理即可

查询和修改

- 节点查询
- 列表查询
- 修改操作
- 错误反馈
- 权限控制

查询和修改

节点查询

接口	含义
`node(id)`	查询任意 id 所对应的节点
`me`	查询当前登录用户对应节点
`user(login)`	查询用户名对应的用户节点

查询和修改

节点查询

- node 接口：指定类型查询

```
query($id: String!) {
  Execute Query
  node(id: $id) {
    id
    ... on CodingExercise {
      title
    }
  }
}
```

查询和修改

返回多种类型

- Union 和 Interface
  - 通过 Union 可指定返回多种类型
  - 当多种类型有需要注意的共同特征时，通过 Interface 指明

查询和修改

列表查询

- 可通过节点关联查询列表（如查询用户创建的题包时）
- 平台提供一些全局的列表接口：
  - `publicCodingExercises` 等：公开资源列表
  - `search`：搜索接口（后续统一为一个接口）
  - `submissions`：平台所有提交信息

查询和修改

分页机制

- 分页参数
  - `first`
  - `last`
  - `before`
  - `after`
  - `skip`
- 限制每页最多返回 100 个元素

- 分页信息：`pageInfo`
  - `hasNextPage`
  - `endCursor`
  - `hasPreviousPage`
  - `beginCursor`

查询和修改

修改操作

- 命名参考:
  - `createCodingExercise(data)`
  - `updateExercisePack(exercisePackId, data)`
  - `updateCodingExerciseJudging(codingExerciseId, data)`
  - `submitData(codingExerciseId, data)`
- 返回:
  - 需要返回修改的对象本身
  - 例：`generatePackCode` 需要返回 `ExercisePack`

查询和修改

错误反馈

- 字段默认可以为空
- 有错误字段返回 null，其他字段仍可正常返回
- 错误类型
  - `AuthenticationError`：用户验证失败
  - `ForbiddenError`：用户无访问权限（可能未登陆）
  - `UserInputError`：用户参数输入有误（如长度不符合要求等）
  - `NotFoundError`：用户想要查询的对象未找到（以前同看作输入错误，后面要区分）
  - `PaginationError`：翻页错误（如请求超过 100 个节点）
  - `InternalError`：内部错误（如连接故障、评测脚本错误等）
  - `UnknownError`：未知原因错误，尽量少用

查询和修改

权限控制

- 节点权限和字段权限
  - 通过 node 接口和关联查询，可控制节点本身是否可见
  - 节点可见的前提下，节点的每个字段均可设置独立逻辑控制权限
- 资源权限管理
  - 通过节点权限控制
  - 机制：public、visible 和白名单
  - 列表接口需要**手动设置逻辑**以满足资源权限要求
    - 如 `publicCodingExercises` 需要筛选 `{public:true, visible:true}`

业务扩展

- 基本思想
- 扩展当前业务
- 新增子业务

业务扩展

基本思想

- GraphQL 可以不影响现有接口的基础上增量扩展
- 扩展业务时可以只关注新增的部分
- 业务尽量拆分思考，而不是聚合思考

业务扩展

扩展当前业务

- 查询更多信息
  - 建模为点和关系
  - 在对应的 connection 或 edge 上新增字段

业务扩展

扩展当前业务

- 增加对象
  - 增加对应对象类型和基础字段
  - 补充和其他对象的关联关系
  - 补充节点和列表查询

业务扩展

新增子业务

- 需求：为平台一些对象增加新业务功能
  - 可单独抽取出一套业务逻辑
  - 适用于一个或多个现有对象
    - 如：评分、评论、收藏、打标签等

- 我们使用 GraphQL 中的 Interface 来抽取独立的业务逻辑

业务扩展

新增子业务

- 例：评分业务
  - 评分业务可针对一类对象
  - 对象应能够查询当前评分，几人 upvote，几人 downvote 等
  - 用户可以给这些对象提交评分

业务扩展

新增子业务

```
interface Votable {
  votes: Int
  upvotes: Int
  downvotes: Int
}
```

通过 interface 表达
一类业务对象

```
mutation {
  vote(votableId: ID!, vote: Int): Votable
}
```

针对这类对象的操作

```
type CodingExercise implements Votable {
  # ...
}
```

指定哪些对象
支持该业务

业务扩展

新增子业务

- 使用 interface 思考和定义业务：
  - 局限在特定业务中思考，减少思维成本和沟通成本
  - 方便将业务延伸到更多对象
  - 方便不同业务并行规划和开发

总结和规划

- 历史遗留问题：命名混乱，权限控制混乱，依赖更新等
- platform-next
- 下一步：
  - 业务拆分开发
  - 针对拆分业务的集成测试
  - 前端调用迁移