GraphQL

使用GraphQL的前段能够向支持 GraphQL 的后端数据服务声明式传达数据需求。

高效

一个视图需的多个资源，rest api会需要多次请求，GraphQL则发起发起单个请求获取复杂的内容

|  |  |
| --- | --- |
| 请求 | 响应 |
| {  hero {  name  # 查询可以有备注！  friends {  name  }  }  } | {  "data": {  "hero": {  "name": "R2-D2",  "friends": [  {  "name": "Luke Skywalker"  },  {  "name": "Han Solo"  },  {  "name": "Leia Organa"  }  ]  }  }  } |
| {  human(id: "1000") {  name  height  }  } | {  "data": {  "human": {  "name": "Luke Skywalker",  "height": 1.72  }  }  } |

标准化

GraphQL 可以让多个客户端向多个服务请求数据的场景通信简化、标准化

灵活

问题和对策

* 资源耗尽型攻击

GraphQL 服务器可能会受到伴随着极其复杂的查询的攻击，造成服务器资源耗尽。很容易就能构造一个深度嵌套关系链（用户 -> 好友 -> 好友的好友。) 或者多次通过字段别名请求同一字段的查询。

缓解措施：

对单个用户请求的数据量做某种限制；

对需要很长时间处理的请求进行超时处理；

在 GraphQL 之下的更底层进行速率限制；

对于非公开的api，使用白名单策略并预先审核服务器能够处理的查询，客户端则仅能通过唯一查询标识码向服务器发起审核过的查询。（Facebook的策略）

* 客户端数据缓存

若使用查询字段作为响应数据的缓存键，效率低下，还容易造成数据一致性方面的问题，原因是多个 GraphQL 查询的结果很容易重叠，而这种缓存策略并没有考虑到这种重叠。

使用图缓存。将一个 GraphQL 查询的响应数据正则化为一个平铺的记录集合，为每个记录设置一个全局唯一 ID，我们就能够只缓存这些记录而不用缓存整个响应了。 [Relay.js](https://facebook.github.io/relay/) 就是一个采用这种缓存策略并在内部进行自动管理的框架。

* N+1 SQL 查询问题

例：有如下关系

public class Group {

private List<User> users;

}

public class User {

private Group group;

}

在一般ORM查询框架下查询Group下的User时会产生如下的逻辑

1、select \* from group;--->组装成List<Group>结合

2、for(Group group:groups){

      select \* from user where group\_id = ?(group.getId()) --->组装成List<User> users;

       group.setUsers(users)

    }

对策：使用GraphQL 的DataLoader。原理就是，在需要查询数据库的时候将查询进行延迟，等到拿到所有的查询需求之后再一次性查询出来。在graphene里面，批量查询可以这样写:

|  |
| --- |
| # managers/author.py  class AuthorsDataLoader(DataLoader):  def batch\_load\_fn(self, ids):  query = DBSession().query(AuthorModel).filter(AuthorModel.id.in\_(ids))  articles = dict([(article.id, article) for article in query.all()])  return Promise.resolve([articles.get(id, None) for id in ids]) |

最终，仅需要两次数据库查询就完成了两个批量查询，即:

|  |
| --- |
| SELECT \* FROM `articles` LIMIT 0, 20;  SELECT \* FROM `authors` WHERE `id` IN (1, 2, 3, ..., N); |

使用GraphQL 的DataLoader查询缓存。本身具有缓存结果的功能，它缓存的是SELECT \* FROM authors WHERE id =N的结果，而不是批量查询的结果，所以，下一次即使是不一样的IN列表，依然会有部分能够使用缓存。DataLoader默认本身是开启缓存的，如果想自己用Redis等来实现对象的缓存，可以在DataLoader初始化的时候将cache设置为False

常用封装框架

具体可以看[github上搜集的graphql资料 全](https://github.com/chentsulin/awesome-graphql)

**relay**

facebook官方前端缓存框架

**apollo-boost**

包含下图5个框架，与react-apollo一起搭配使用，是apollo社区推荐的前端使用方式，开发起来快，但是拓展修改不太方便，不适合自定义的公司项目

**apollo-client**

上图5个框架中最核心的一个，将各类接口微服务和数据库操作进行封装，集成redux，更易使用，但是略显臃肿

**apollo-server**

apollo-server是一个在nodejs上构建grqphql服务端的web中间件，支持express、koa等

**prisma**

一个orm库外加轻量的后端服务，可自动生成gql接口暴露出来，围绕prisma的生态已经初步建立，社区正在推动prisma2，现在正在待发布阶段，未来可期

参考

<https://graphql.org>

<https://graphql.cn>

<https://juejin.im/post/6844903491106930702>

<https://zhuanlan.zhihu.com/p/27323883>