采用什么方式进行分页

connection+edge+node(游标分页法)

Relay Connections Spec

设计思路

> connection提供了一个标准的切片和分页的机制来展示结果，在结果展示中，connection模式提供了一个游标，以及一种方式告诉客户端是否存在更多的数据。

example

```
{
  user {
    id
    name
    friends(first: 10, after: "opaqueCursor") {
      edges {
        cursor
        node {
          id
          name
        }
      }
      pageInfo {
        hasNextPage
      }
    }
  }
}
```

在这个例子中，friends是一个connection，这个查询展示了四个上述的特点

- 使用first参数进行切片，这告诉connection返回10个朋友
- 使用after参数来分页，通过cursor游标，我们可以要求服务端返回朋友(从这个游标处开始)
- 对于每一个edge，提供了一个cursor，这个cursor是不透明的，从上述例子可以看出来
- hasNextPage用来告诉是否存在更多的edges，或者告诉我们是否到达了connection的end

api

- Connection Types

> 所有以connection名字结尾的都是connection Type，connection必须是object类型

- Fields

> Connection types must have fields named edges and pageInfo. They may have additional fields related to the connection, as the schema designer sees fit.(ConnectionTYpe必须有两个Field(edge和pageInfo))，可能还会存在其他的参数，可以自己设计

- edge

> A “Connection Type” must contain a field called edges. This field must return a list type that wraps an edge type, where the requirements of an edge type are defined in the “Edge Types” section below.(这个field返回一个list的type)

- 自省

> 如果ExamppleConnection存在于这个类型系统中(初始化后把schema已加载到内存中)，因为它的名字以connection结尾，

```
{
  __type(name: "ExampleConnection") {
    fields {
      name
      type {
        name
        kind
        ofType {
          name
          kind
        }
      }
    }
  }
}
```

输出类型

```
{
  "data": {
    "__type": {
      "fields": [
        // May contain other items
        {
          "name": "pageInfo",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType": {
              "name": "PageInfo",
              "kind": "OBJECT"
            }
          }
        },
        {
          "name": "edges",
          "type": {
            "name": null,
            "kind": "LIST",
            "ofType": {
              "name": "ExampleEdge",
              "kind": "OBJECT"
            }
          }
        }
      ]
    }
  }
}
```

list type+total属性

list 为何不能进行分页

issue：how to add totalcount with list type(虽然list可以直接查出数据出来，但是并未进行分页处理)

e.g

```
  allusers {
      totalcount
      [
    id,
        name
      ]}
```

开源作者给出的答案：

> List types can only represent lists and can’t have any other attributes so what you’re trying to do in the example is not possible.

List类型仅仅是列表并没有像Field一样有任何其他的属性。可以采用ObjectType来存储totalCount字段

```
type Query {
        allUsers: AllUsersConnection
    }

type AllUsersConnection {
    totalCount: Int!
    edges: [User]
}

type User {
    id: ID!
    name: String!
}
```

所以就可以这样查询：

```
query {
allUsers: {
    totalCount
    edges: {
        id
        name
    }
}
}
```

采用python的graphql实现框架graphene的代码应该是如下：

```
class User(graphene.ObjectType):
    id = graphene.ID(required=True)
    name = graphene.String(required=True)

class AllUsersConnection(graphene.ObjectType):
    total_count = graphene.Int(required=True)
    edges = graphene.List(User)

class Query(graphene.ObjectType):
    all_users = graphene.Field(AllUsersConnection)
```

游标分页法 算法设计思想

> To determine what edges to return, the connection evaluates the before and after cursors to filter the edges, then evaluates first to slice the edges, then last to slice the edges.（connection通过解析before和after游标来过滤edges，然后通过fisrt和last来进行切片）

- 如果argment包含fisrt：
  a. 如果first小于0，throw error
  b. 如果edge的长度大于first：Slice edges to be of length first by removing edges from the end of edges.
- 如果argment包含last
  a.如果last小于0.throw error
  b.如果edge的长度大于last：Slice edges to be of length last by removing edges from the start of edges.

example

采用**list（limit+offset）**

定义的类型

```
type Playlist {
  id: Int!
  title: String
  userId: Int
  user: User
}

type RootQueryType {
  user(id: Int!): User
  playlist(id: Int!): Playlist
}

type User {
  id: Int!
  firstName: String
  lastName: String
  playlists(limit: Int, offset: Int): [Playlist]
}
```

输入请求

```
{
  user(id: 1) {
    lastName
    playlists(limit:2, offset:1) {
      title
    }
  }
}
```

输出

```
{
  "data": {
    "user": {
      "lastName": "Zidane",
      "playlists": [
        {
          "title": "Zidane's playlist #2"
        },
        {
          "title": "Zidane's playlist #3"
        }
      ]
    }
  }
}
```

采用GraphQL relations – relay mode

connecton方式类型定义

```
interface Node {
  id: ID!
}

type PageInfo {
  hasNextPage: Boolean!
  hasPreviousPage: Boolean!
  startCursor: String
  endCursor: String
}

type Playlist implements Node {
  id: ID!
  title: String
  userId: Int
  user: User
}

type PlaylistConnection {
  pageInfo: PageInfo!
  edges: [PlaylistEdge]
}

type PlaylistEdge {
  node: Playlist
  cursor: String!
}

type RootQueryType {
  node(id: ID!): Node
}

type User implements Node {
  id: ID!
  firstName: String
  lastName: String
  playlists(before: String, after: String, first: Int, last: Int): PlaylistConnection
}
```

输入请求：

```
{
  node(id: "VXNlcjox") {
    id
    ... on User {
      firstName
        lastName
      playlists {
        pageInfo {
          hasNextPage
          hasPreviousPage
          startCursor
          endCursor
        }
        edges {
          cursor
          node {
            title
          }
        }
      }
    }
  }
}
```

输出：

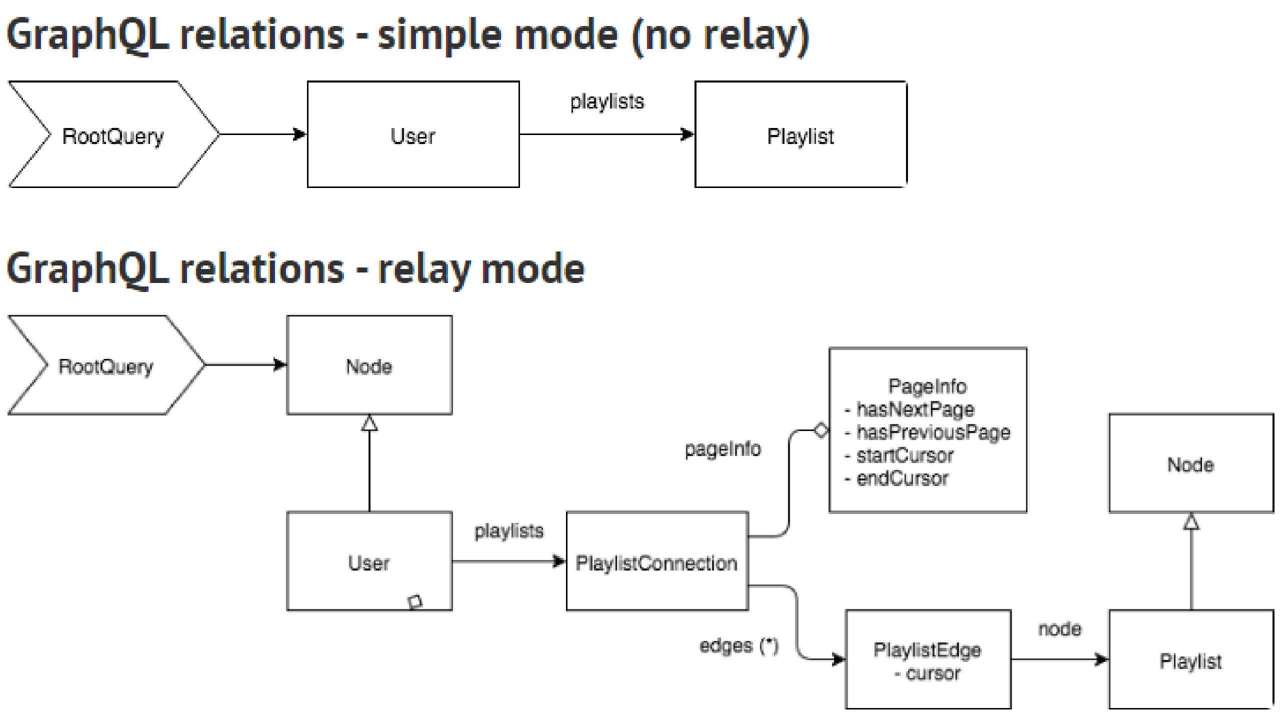
```
{
  "data": {
    "node": {
      "id": "VXNlcjox",
      "firstName": "Zinedine",
      "lastName": "Zidane",
      "playlists": {
        "pageInfo": {
          "hasNextPage": false,
          "hasPreviousPage": false,
          "startCursor": "Q29ubmVjdGlvbjow",
          "endCursor": "Q29ubmVjdGlvbjoy"
        },
        "edges": [
          {
            "cursor": "Q29ubmVjdGlvbjow",
            "node": {
              "title": "Zidane's playlist #1"
            }
          },
          {
            "cursor": "Q29ubmVjdGlvbjox",
            "node": {
              "title": "Zidane's playlist #2"
            }
          },
          {
            "cursor": "Q29ubmVjdGlvbjoy",
            "node": {
              "title": "Zidane's playlist #3"
            }
          }
        ]
      }
    }
  }
}
```

编解码 cursor

base64 encoded string 通过解码我们可以得到：

encode	decode
Q29ubmVjdGlvbjow	Connection:0
Q29ubmVjdGlvbjox	Connection:1
Q29ubmVjdGlvbjoy	Connection:2

difference 展示图


GraphQL relations - simple mode (no relay)
RootQuery
User
playlists
Playlist
GraphQL relations - relay mode
RootQuery
Node
User
playlists
PlaylistConnection
pageInfo
PageInfo
- hasNextPage
- hasPreviousPage
- startCursor
- endCursor
edges (*)
PlaylistEdge
- cursor
node
Playlist
Node