dataloader

可用作应用程序数据获取层的一部分，通过**批处理和缓存**在各种远程数据源（如数据库或Web服务）上提供简化且一致的API,

batching

Create loaders by providing a batch loading function(下面).

```
 from promise import Promise
 from promise.dataloader import DataLoader
 class UserLoader(DataLoader):
    def batch_load_fn(self, keys):
        # Here we return a promise that will result on the
        # corresponding user for each key in keys
        return Promise.resolve([get_user(id=key) for key in keys])
```

返回类型为Promise，DataLoader will coalesce（合并） all individual loads which occur within a single frame of execution (executed once the wrapping promise is resolved) and then call your batch function with all requested keys.

```
class User(graphene.ObjectType):
    name = graphene.String()
    best_friend = graphene.Field(lambda: User)
    friends = graphene.List(lambda: User)

def resolve_best_friend(self, info):
    return user_loader.load(self.best_friend_id)

def resolve_friends(self, info):
    return user_loader.load_many(self.friend_ids)
```

example:

1.继承dataloader类，实现batch_load_fn()方法，在batch_loader_fn()方法中用promise进行处理
e.g:

```
from  promisepromise import Promise
from promise.dataloader import DataLoader

class UserLoader(DataLoader):
    def batch_load_fn(self, keys):
```

```
        # Here we return a promise that will result on the
        # corresponding user for each key in keys
        print(f'self.batch_load_fn({keys})')
        return Promise.resolve(get_users(ids=keys))
    # get_users is THE function that send SQL query. It accept a list of ids and
return a list if User who inherit ObjectType.

def get_users(ids):
    print(f"get_users({ids}) SELECT * FROM table_user WHERE id IN {ids}")
    return [User(name=f"User{id}") for id in ids]
```

我们可以发现：在resolve函数中，我们需要使用定义的dataloder来load_many所需要的keys，之后传向dataloader中的batch_load_fn函数中，这里用来进行数据层的处理。

In “databse”, there is one school, who has two classrooms. Each classroom has two students

```
class Student(graphene.ObjectType):
    name = graphene.String()

    def __repr__(self):
        return "<Student name={}>".format(self.name)

class Classroom(graphene.ObjectType):
    id = graphene.Int()
    students = graphene.Field(graphene.List(Student))

    def __repr__(self):
        return "<Classroom id={}>".format(self.id)

    def resolve_students(self, info):

        classroom_id_2_student_idss_promise = lambda classroom_id: student_class
room_loader.load_many( [classroom_id] )
        student_ids_2_student_promise = lambda students_ids: student_loader.load
_many( students_ids )


        return classroom_id_2_student_idss_promise(self.id).then(lambda student_
idss: student_ids_2_student_promise(student_idss[0]))


class School(graphene.ObjectType):
    id = graphene.Int()
    classrooms = graphene.Field(graphene.List(Classroom))

    def resolve_classrooms(self, info):
        classroom_ids = (10, 20)
        print(f'SELECT classroom_id FROM table_classroom_school_relationship WHE
RE school_id = {self.id}')
        print('    result:', classroom_ids)

        # classroom_loader defined below
        return classroom_loader.load_many(classroom_ids)
```

```
class SchoolQuery(graphene.ObjectType):

school = graphene.Field(School)

def resolve_school(self, info):
    the_only_school = School(id = 0)
    return the_only_school
```

定义的数据获取函数：

```
 def get_students_from_ids(ids):
    print(f'SELECT * FROM table_student WHERE id IN {ids}')
    result = [Student(name = f'NO.{id}') for id in ids]
    print('    result:', result)
    return result

def get_classrooms_from_ids(ids):
    print(f'SELECT * FROM table_classroom WHERE id IN {ids}')
    result = [Classroom(id = id) for id in ids]
    print('    result:', result)
    return result

def get_student_ids_from_classroom_ids(ids):
    # argument:
    #     classroom_ids with length N
    #     example: [classroom_id_1, classroom_id_2]
    # return:
    #     a list of tuple. Length of list id alse N.
    #     example: [ (student_ids_1, student_ids_2), (student_ids_2, student_ids
_3) ]
    #     student_ids_1, student_ids_2 are student_ids for classroom_id_1
    #     student_ids_2, student_ids_3 are student_ids for classroom_id_2


print(f'SELECT student_id FROM table_station_classroom_relationship WHERE classr
oom_id in {ids}')

result = []
for classroom_id in ids:
    students_ids = (classroom_id + 1, classroom_id + 2)  # each classroom has 2
students
    result.append(students_ids)

print('    result:', result)
return result

 class StudentLoader(DataLoader):
    def batch_load_fn(self, keys):
        # Here we return a promise that will result on the
        # corresponding user for each key in keys

        # print(f'StudentLoader.batch_load_fn({keys})')
```

```
        return Promise.resolve(get_students_from_ids(ids=keys))

class ClassroomLoader(DataLoader):
    def batch_load_fn(self, keys):
        # Here we return a promise that will result on the
        # corresponding user for each key in keys

        # print(f'ClassroomLoader.batch_load_fn({keys})')
        return Promise.resolve(get_classrooms_from_ids(ids=keys))


class StudentClassroomLoader(DataLoader):
    def batch_load_fn(self, keys):
        # Here we return a promise that will result on the
        # corresponding user for each key in keys

        # print(f'StudentClassroomLoader.batch_load_fn({keys})')
        return Promise.resolve(get_student_ids_from_classroom_ids(ids=keys))
```

运行query：

```
run(SchoolQuery, """
    {
      school {        # one school
        id
        classrooms {  # N classrooms
          id
          students {  # N*N students
            name
          }
        }
      }
    }
    """)
```

得到的结果：

```
SELECT classroom_id FROM table_classroom_school_relationship WHERE school_id = 0
    result: (10, 20)
SELECT * FROM table_classroom WHERE id IN [10, 20]
    result: [<Classroom id=10>, <Classroom id=20>]
SELECT student_id FROM table_station_classroom_relationship WHERE classroom_id i
n [10, 20]
    result: [(11, 12), (21, 22)]
SELECT * FROM table_student WHERE id IN [11, 12, 21, 22]
    result: [<Student name=NO.11>, <Student name=NO.12>, <Student name=NO.21>, <
Student name=NO.22>]
{
    "school": {
        "id": 0,
        "classrooms": [
            {
                "id": 10,
                "students": [
```

```
                    {
                        "name": "NO.11"
                    },
                    {
                        "name": "NO.12"
                    }
                ]
            },
            {
                "id": 20,
                "students": [
                    {
                        "name": "NO.21"
                    },
                    {
                        "name": "NO.22"
                    }
                ]
            }
        ]
    }
}
```

可以看出，不再是数据库获取一条一条的执行，而是每进入每个数据库，而把所有的key值放在一个list中，然后批量执行。

版本存在的严重问题

在 promise的版本中，2.1才能开始使用dataloader功能，如果是2.1之前的版本，所有从resolve获取到的key值都不会收集在一起，而是一个一个执行，失去了dataloader的公用，需要引起相当的重视

相关的链接：

Optimizing GraphQL Queries with DataLoader
GraphQL Dataloader with Node.js & MongoDB Tutorial
Batching GraphQL Queries with DataLoader