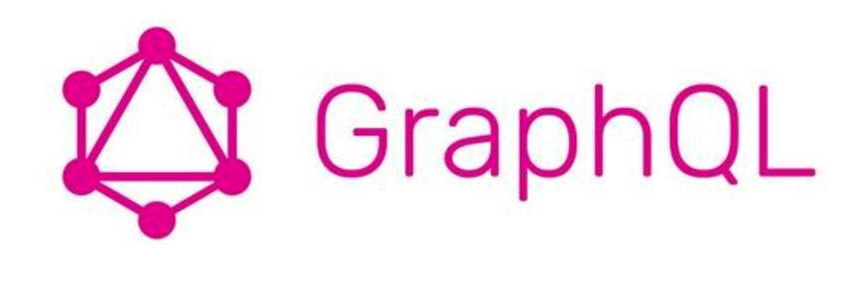
01\_Graphql的介绍

什么是Graphql

GraphQL 是Facebook于 2012 年在内部开发的数据查询语言，在 2015 年开源。

其数据由服务器上的一个Scheme提供，其查询返回的数据依赖请求的时候用户需要的精确数据。

官网：https://graphql.cn/



为什么使用Graphql

GraphQL和RESTful一样，都是一种网站架构，一种前后端通信规范，不涉及语言，不同语言有不同的实现方案。

GraphQL目前被认为是革命性的API工具，因为它可以让客户端在请求中指定希望得到的数据，而不像传统的RESTful那样只能呆板地在服务端进行预定义。

这样它就让前、后端团队的协作变得比以往更加的通畅，从而能够让组织更好地运作。

而实际上，GraphQL与RESTful都是基于HTTP进行数据的请求与接收，而且GraphQL也内置了很多RESTful模型的元素在里面。

那么在技术层面上，GraphQL和RESTful这两种API模型到底有什么异同呢？他们归根到底其实没多大区别，只不过GraphQL做了一些小改进，使得开发体验产生了较大的改变。

GraphQL VS RESTful

资源

RESTful的核心思想就是资源，每个资源都能用一个URL来表示，你能通过一个GET请求访问该URL从而获取该资源。

根据当今大多数API的定义，你得到一份JSON格式的数据响应，整个过程大概是这样：

|  |
| --- |
| GET /books/1 {  "title": "Python课程",  "author": {   "firstName": "尚学堂",  "lastName": "李老师"  }  // ... more fields here } |

注：上面的例子里的"author"也会作为一个单独的资源在其他RESTful API中被用到

需要注意的是，在RESTful中，一个资源的种类与你获取它的方式是耦合的。比如上面这个例子中的API就可以称之为“book端点”（book endpoint)。

在这一点上GraphQL就大为不同，因为在GraphQL里这两个概念是完全分离的。比如在你的schema定义中，你可能会有Book和Author两个类型（type）：

|  |
| --- |
| type Book {  id: ID  title: String  published: Date  price: String  author: Author } type Author {  id: ID  firstName: String  lastName: String  books: [Book] } |

注意这里我们虽然定义了数据类型，但却不知道该如何获取这些数据。

这是RESTful与GraphQL的一个核心差异：资源的描述信息与其获取方式相分离。

如果要去访问某个特定的book或者author资源，后端需要在schema中创建一个Query类型：

|  |
| --- |
| type Query {  book(id: ID!): Book  author(id: ID!): Author } |

然后，前端就可以像RESTful那样发送请求了：

|  |
| --- |
| GET /graphql?query={ book(id: "1") { title, author { firstName } } } {  "title": "Python课程",  "author": {  "firstName": "尚学堂",  } } |

虽然都是通过请求某个URL来得到相同的响应，但这里我们已经看到GraphQL与RESTful的差异之处了。

首先，我们看到GraphQL的URL请求里面指定了我们所需要的资源以及在该资源中我们所关心的字段。

另外，我们是主动请求得到与book相关的author数据的，而不是服务端替我们决定的。

最重要的是，在请求中我们不需要关心资源的主键和资源之间的关系定义，我们可以通过除id以外的其他字段来获取到相同的Book资源。

**相同点：**

都有资源这个概念，而且都能通过ID去获取资源

都可以通过HTTP GET方式来获取资源

都可以使用JSON作为响应格式

**差异点：**

在RESTful中，你所访问的路径就是该资源的唯一标识（ID）；在GraphQL中，该标识与访问方式并不相关

在RESTful中，资源的返回结构与返回数量是由服务端决定；在GraphQL，服务端只负责定义哪些资源是可用的，由客户端自己决定需要得到什么资源

路由

一个具有可预见性的API才是好的API。

因为你通常会把一个API当做程序的一部分来使用，所以你必须要知道它需要接收什么参数并预期能够获取到什么样的结果。

这时候，对API的访问描述信息就显得很重要。通常我们会通过阅读API文档来获取信息，但通过GraphQL的Introspection机制、以及Swagger这样的RESTful API工具，这些信息就能可以自动获取到。

如今的RESTful API通常会由一系列的URL端点组成：

|  |
| --- |
| GET /books/:id GET /authors/:id GET /books/:id/comments POST /books/:id/comments |

你可以把这种API的形态称之为线性结构——因为这就是一个列表嘛。

当你要获取数据时，第一个事情就是搞清楚你要访问的是哪个端点。

而在GraphQL中——其实在上一节里你也看到了——可以通过查看GraphQL schema获得相关信息：

|  |
| --- |
| type Query {  book(id: ID!): Book  author(id: ID!): Author } type Mutation {  addComment(input: AddCommentInput): Comment } type Book { ... } type Author { ... } type Comment { ... } input AddCommentInput { ... } |

RESTful会使用类似GET、POST这样的动词去请求相同的URL来表示这到底是一个读操作还是写操作，而GraphQL会使用不同的预定义类型：Mutation和Query。

在GraphQL请求中，你可以通过不同的关键字进行不同的操作：

|  |
| --- |
| query { ... } mutation { ... } |

这里的Query类型定义与上面的RESTful路由是完全契合的，同样表示了数据的访问入口，因此这是GraphQL中最能与RESTful的URL端点所对应的概念。

如果是对资源的简单查询，GraphQL与RESTful是类似的，都是通过指定资源的名称以及相关参数来取得，但不同的是，你可以根据资源之间的关联关系来发起一个复杂请求，而在RESTful中你只能定义一些特殊的URL参数来获取到特殊的响应，或者是通过发起多个请求、再自行把响应得到的数据进行组装才行。

RESTful对数据的描述形式是一连串的URL端点，而GraphQL则是由相互之间有所关联的schema组成。 相同点：

RESTful API的URL端点列表与GraphQL的Query/Mutation中的字段类似，都表示数据的访问入口

都能用不同的方式描述一个API请求到底是读操作还是写操作

**差异点：**

GraphQL让你可以通过一个资源入口访问到关联的其他资源，只要事先在schema中定义好资源之间的关系即可；而RESTful则提供了多个URL端点来获取相关的资源

在GraphQL中，Query类型可以在一个请求的根节点中被访问，除此以外它跟其他类型没有区别，比如你也可以对一个query中的字段添加参数。而在RESTful中，即使响应结果是嵌套关系，但在请求中并没有嵌套的概念

RESTful使用POST这样的HTTP方法名称来定义写操作，GraphQL则是查询结构中的关键字

正因为上述的第一个点，人们通常会把Query类型中的字段称为GraphQL中的“端点”或“查询条件”。

虽然这是一个合理的解释，但同时也会对其他人造成误导，让人以为Query类型是一个非常特殊的类型。

路由处理器（Route Handlers）vs 解析器（Resolvers）

想象一下，当你调用一个API的时候，实际上会发生什么事情？

应该是在服务器上面执行了一些代码来处理这个请求，可能是进行了一些计算，可能从数据库中加载了一些数据，也可能是再次调用了一个别的API。

虽然总的来说，作为调用方你并不需要知道内部发生了什么事情，不过由于RESTful和GraphQL都提供了标准的API实现方法，我们可以通过对比来感受一下两者之间的差异。

为了突出重点，我会忽略掉一些构建服务用的过程代码。

首先实现一个 hello world：

|  |
| --- |
| class HelloWorld(Resource):  def get(self):  return {'say': 'hello world'}  api.add\_resource(HelloWorld, '/hello') |

这里我们得到了一个可以返回“world”这个字符串的/hello端点。从这个例子我们可以看到一个RESTful API请求的的生命周期：

服务器收到请求并提取出HTTP方法名（比如这里就是GET方法）与URL路径

API框架找到提前注册好的、请求路径与请求方法都匹配的代码

该段代码被执行，并得到相应结果

API框架对结果进行序列化，添加上适当的状态码与响应头后，返回给客户端

GraphQL差不多也是这样工作的，我们来看下这个对应的 hello world例子:

|  |
| --- |
| class Query(ObjectType):  """定义一个字符串属性域hello"""  say = String()  def resolve\_say(root,info):  return f'Hello World!' |

我们看到，这里并没有针对某个URL路径提供函数，而是把Query类型中的hello字段映射到一个函数上了。

在GraphQL中这样的函数我们称之为解析器（Resolver）。

然后我们就可以这样发起一个查询：

|  |
| --- |
| query {  say } |

至此，总结一下服务器对一个GraphQL请求的执行过程：

服务器收到HTTP请求，取出其中的GraphQL查询

遍历查询语句，调用里面每个字段所对应的Resolver。在这个例子里，只有Query这个类型中的一个字段hello

Resolver函数被执行并返回相应结果

GraphQL框架把结果根据查询语句的要求进行组装

因此我们将会得到如下响应：

|  |
| --- |
| { "say": "Hello world!" } |

这里有个小技巧：我们其实可以多次调用同一个Resolver：

|  |
| --- |
| query {  say  secondSay: say } |

在这个例子中的生命周期跟上面的是类似的，但因为我们通过别名来两次请求了同一个字段，所以对应Resolver函数hello也会被执行两次。

虽然这个例子举得不是很好，不过这里主要想表达的是在一个请求中可以解析多个字段，即使是相同的字段也可以在查询的不同地方被多次访问。

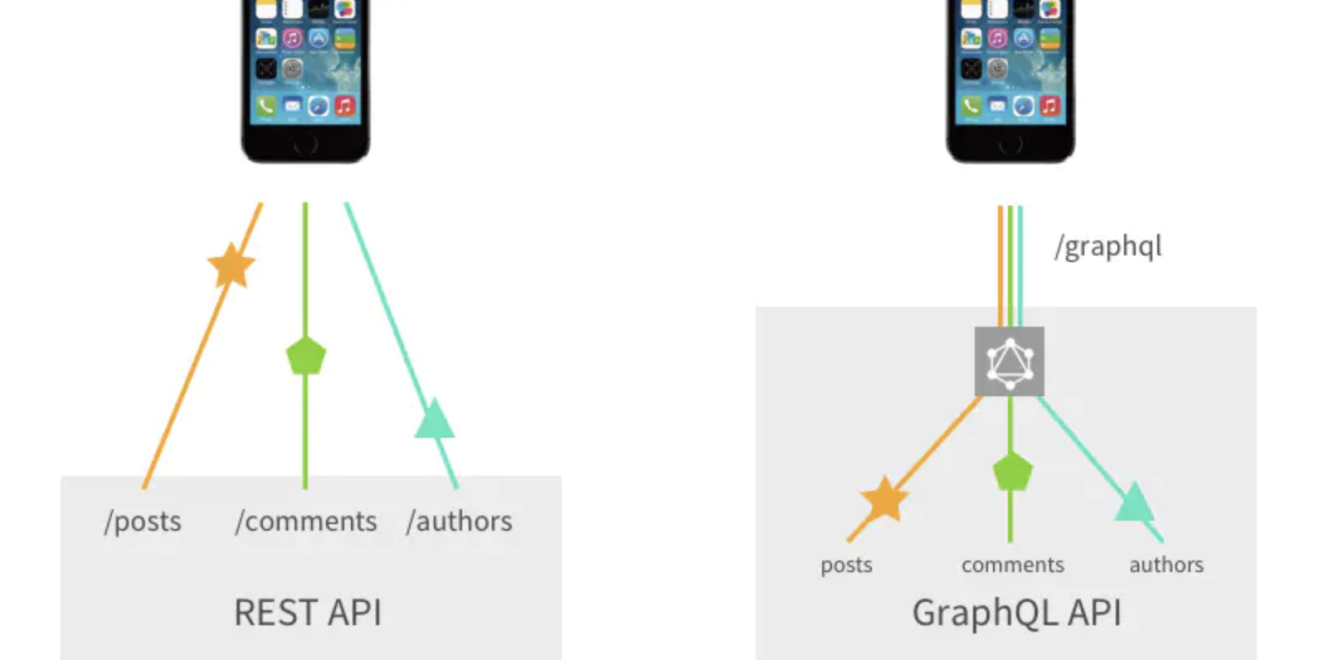
再来看下“嵌套”解析器是怎样的：

|  |
| --- |
| {  Query: {  author: (root, { id }) => find(authors, { id: id }),  },  Author: {  posts: (author) => filter(posts, { authorId: author.id }),  }, } |

这样的解析器可以处理如下查询请求：

|  |
| --- |
| query {  author(id: 1) {  firstName  posts {  title  }  } } |

即使解析器的结构是扁平的，但由于它们被不同的类型所引用，所以你还是可以利用它们来实现嵌套查询



上图形象地说明了使用RESTful和GraphQL进行多种资源获取的方式的差异

小结

总的来说，RESTful和GraphQL都提供了很好的API调用方式。如果你对如何构建一个RESTful API足够熟悉，使用GraphQL来实现同样的API功能对你来说并不是一件难事。但GraphQL的一大优势是让你可以在不需要发起多次请求的情况下调用多个函数来获取资源数据。

相同点：

RESTful的端点与GraphQL查询字段都在服务端调起函数执行

RESTful和GraphQL都使用框架和类库来进行一些通用的网络协议处理

差异点：

一个RESTful请求对应一个路由处理器（Route Handler）；而一个GraphQL的请求可以唤起多个解析器（Resolver）在一次响应中访问多种资源

RESTful需要你自己构建整个请求的响应；而GraphQL的请求响应是由查询方指定结构、并由GraphQL进行构建组装的

你可以把GraphQL理解为一个可以在一次请求中进行多个端点调用的系统，差不多算是RESTful的多路复用版。

总的主要的不同点就是：

RESTful一个接口只能返回一个资源，GraphQL一个可以获取多个资源。

RESTful用不同的URL来区分资源，GraphQL用类型区分资源。



02\_Python使用GraphQL

Graphene

Graphene是一个提供在Python中实现GraphQL API的工具的库

Graphene具有与最流行的Web框架和ORM集成的全部功能。Graphene生成的方案完全符合GraphQL规范，并提供了用于构建符合中继标准的API的工具和模式。

安装

Python (2.7, 3.4, 3.5, 3.6, pypy)

Graphene (2.0)

|  |
| --- |
| pip install "graphene>=2.0" |

第一个程序-HelloWorld

|  |
| --- |
| from graphene import ObjectType, String, Schema  class Query(ObjectType):  """定义一个字符串属性域hello"""   hello = String()   def resolve\_hello(root,info):  return f'Hello World!'  schema = Schema(query=Query)  if \_\_name\_\_ == '\_\_main\_\_':  query\_string = '''  {  hello  }  '''  result = schema.execute(query\_string)  print(result.data) |

属性

* name
* required
* default\_value

|  |
| --- |
| from graphene import ObjectType, String, Schema  class Query(ObjectType):  """定义一个字符串属性域hello"""   hello = String(name = 'text',sex=String(default\_value="man"),desc = String(required = True))   def resolve\_hello(root,resolve,sex,desc = 'python'):  return f'Hello World! sex：{sex} desc: {desc}'  schema = Schema(query=Query)  if \_\_name\_\_ == '\_\_main\_\_':  query\_string = '''  {  text(desc:"graphql")  }  '''  result = schema.execute(query\_string)  print(result.data) |

数据类型

* graphene.String
* graphene.Int
* graphene.Float
* graphene.Boolean
* graphene.ID

graphene.Date

|  |
| --- |
| import datetime from graphene import Schema, ObjectType, Date  class Query(ObjectType):  one\_week\_from = Date(required=True, date\_input=Date(required=True))  def resolve\_one\_week\_from(root, info, date\_input):  assert date\_input == datetime.date(2050, 1, 2)  return date\_input + datetime.timedelta(weeks=1) schema = Schema(query=Query) results = schema.execute("""  query {  oneWeekFrom(dateInput: "2050-01-02")  } """) assert results.data == {"oneWeekFrom": "2050-01-09"} |

graphene.DateTime

|  |
| --- |
| import datetime from graphene import Schema, ObjectType, DateTime  class Query(ObjectType):  one\_hour\_from = DateTime(required=True, datetime\_input=DateTime(required=True))  def resolve\_one\_hour\_from(root, info, datetime\_input):  assert datetime\_input == datetime.datetime(2050, 1, 2, 3, 4, 5)  return datetime\_input + datetime.timedelta(hours=1) schema = Schema(query=Query) results = schema.execute("""  query {  oneHourFrom(datetimeInput: "2050-01-02T15:04:05")  } """) assert results.data == {"oneHourFrom": "2050-01-02T16:04:05"} |

graphene.Time

|  |
| --- |
| import datetime from graphene import Schema, ObjectType, Time  class Query(ObjectType):  one\_hour\_from = Time(required=True, time\_input=Time(required=True))  def resolve\_one\_hour\_from(root, info, time\_input):  assert time\_input == datetime.time(15, 4, 5)  tmp\_time\_input = datetime.datetime.combine(datetime.date(1, 1, 1), time\_input)  return (tmp\_time\_input + datetime.timedelta(hours=1)).time() schema = Schema(query=Query) results = schema.execute("""  query {  oneHourFrom(timeInput: "15:04:05")  } """) assert results.data == {"oneHourFrom": "16:04:05"} |

graphene.Decimal

|  |
| --- |
| import decimal from graphene import Schema, ObjectType, Decimal  class Query(ObjectType):  add\_one\_to = Decimal(required=True, decimal\_input=Decimal(required=True))  def resolve\_add\_one\_to(root, info, decimal\_input):  assert decimal\_input == decimal.Decimal("10.50")  return decimal\_input + decimal.Decimal("1") schema = Schema(query=Query) results = schema.execute("""  query {  addOneTo(decimalInput: "10.50")  } """) assert results.data == {"addOneTo": "11.50"} |

graphene.JSONString

|  |
| --- |
| from graphene import Schema, ObjectType, JSONString, String  class Query(ObjectType):  update\_json\_key = JSONString(  required=True,  json\_input=JSONString(required=True),  key=String(required=True),  value=String(required=True)  )  def resolve\_update\_json\_key(root, info, json\_input, key, value):  assert json\_input == {"name": "Jane"}  json\_input[key] = value  return json\_input schema = Schema(query=Query) results = schema.execute("""  query {  updateJsonKey(jsonInput: "{\\"name\\": \\"Jane\\"}", key: "name", value: "Beth")  } """) assert results.data == {"updateJsonKey": "{\"name\": \"Beth\"}"} |

数据返回不可能为空

|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask  class Query(graphene.ObjectType):  uname = graphene.NonNull(graphene.String) #   user\_name = graphene.String(required = True)  score = graphene.List(graphene.Int) #   score\_list = graphene.List(graphene.NonNull(graphene.String))  score\_list2 = graphene.NonNull(graphene.List(graphene.Int))   person = graphene.Field(Person)  def resolve\_uname(root,resolve):  return 'test1'  def resolve\_user\_name(root,resolve):  return 'test'  def resolve\_score(root,resolve):  return [90,95]  def resolve\_score\_list(root,resolve):  return [1,2]  def resolve\_score\_list2(root,resolve):  return [1]   if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql', view\_func=GraphQLView.as\_view('graphql',  schema=schema, graphiql=True))   app.run(debug=True) |

自定义数据类型

|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask  class Person(graphene.ObjectType):  first\_name = graphene.String()  last\_name = graphene.String()  class Query(graphene.ObjectType):  person = graphene.Field(Person)  person2 = graphene.Field(Person)  persons = graphene.List(Person)   def resolve\_person(self,info):  return {"first\_name": "R2", "last\_name": "D2"}  def resolve\_person2(self,info):  return Person(first\_name="Luke", last\_name="Skywalker")  def resolve\_persons(self,info):  return [  Person(first\_name='Sxt',last\_name = 'Mr.li'),  Person(first\_name='Sxt',last\_name = 'Mr.Gao'),  ]  if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql', view\_func=GraphQLView.as\_view('graphql',  schema=schema, graphiql=True))   app.run(debug=True) |

03\_接口的使用

|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask  class Animal(graphene.Interface):  id = graphene.ID(required=True)  name = graphene.String(required=True)   class Mouse(graphene.ObjectType):  class Meta:  interfaces = (Animal, )   run = graphene.String()  class Bird(graphene.ObjectType):  class Meta:  interfaces = (Animal, )   fly = graphene.String()   class Query(graphene.ObjectType):   mouse = graphene.Field(Mouse)  bird = graphene.Field(Bird)  animal = graphene.Field(Animal,type\_ = graphene.Int(required= True))   def resolve\_mouse(root, info):  return {'id':1,'name':'杰瑞','run':'跑呀~汤姆来啦！'}  def resolve\_bird(root,info):  return {'id':1,'name':'鹦鹉','run':'飞呀~杰瑞来啦！'}   def resolve\_animal(root,info,type\_):  if type\_ == 1:  return Mouse(id=1,name='杰瑞',run='跑呀~汤姆来啦！')  elif type\_ == 2:  return Bird(id=1,name='鹦鹉',fly='飞呀~汤姆来啦！')  else:  return Mouse(id=1,name='杰瑞',run='跑呀~斯派克来啦！')   if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql', view\_func=GraphQLView.as\_view('graphql',  schema=schema, graphiql=True))   app.run(debug=True) |

04\_枚举

|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask  class Role(graphene.Enum):  JR = 1  TM = 2  SPK = 3  class Animal(graphene.Interface):  id = graphene.ID(required=True)  name = graphene.String(required=True)   class Mouse(graphene.ObjectType):  class Meta:  interfaces = (Animal, )   run = graphene.String()  class Bird(graphene.ObjectType):  class Meta:  interfaces = (Animal, )   fly = graphene.String()   class Query(graphene.ObjectType):   mouse = graphene.Field(Mouse)  bird = graphene.Field(Bird)  animal = graphene.Field(Animal,role = Role())   def resolve\_mouse(root, info):  return {'id':1,'name':'杰瑞','run':'跑呀~汤姆来啦！'}  def resolve\_bird(root,info):  return {'id':1,'name':'鹦鹉','run':'飞呀~杰瑞来啦！'}   def resolve\_animal(root,info,role):  if role == Role.JR:  return Mouse(id=1,name='杰瑞',run='跑呀~汤姆来啦！')  elif role == Role.TM:  return Bird(id=2,name='鹦鹉',fly='飞呀~汤姆来啦！')  else:  return Mouse(id=3,name='杰瑞',run='跑呀~斯派克来啦！')   if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql', view\_func=GraphQLView.as\_view('graphql',  schema=schema, graphiql=True))   app.run(debug=True) |

05\_GraphQL查询

服务端

|  |
| --- |
| human\_data = {} droid\_data = {}  ''' ===================== data ========================= ''' def setup():  global human\_data, droid\_data  luke = Human(  id="1000",  name="Luke Skywalker",  friends=["1002", "1003", "2000", "2001"],  appears\_in=[4, 5, 6],  home\_planet="Tatooine",  )   vader = Human(  id="1001",  name="Darth Vader",  friends=["1004"],  appears\_in=[4, 5, 6],  home\_planet="Tatooine",  )   han = Human(  id="1002",  name="Han Solo",  friends=["1000", "1003", "2001"],  appears\_in=[4, 5, 6],  home\_planet=None,  )   leia = Human(  id="1003",  name="Leia Organa",  friends=["1000", "1002", "2000", "2001"],  appears\_in=[4, 5, 6],  home\_planet="Alderaan",  )   tarkin = Human(  id="1004",  name="Wilhuff Tarkin",  friends=["1001"],  appears\_in=[4],  home\_planet=None,  )   human\_data = {  "1000": luke,  "1001": vader,  "1002": han,  "1003": leia,  "1004": tarkin,  }   c3po = Droid(  id="2000",  name="C-3PO",  friends=["1000", "1002", "1003", "2001"],  appears\_in=[4, 5, 6],  primary\_function="Protocol",  )   r2d2 = Droid(  id="2001",  name="R2-D2",  friends=["1000", "1002", "1003"],  appears\_in=[4, 5, 6],  primary\_function="Astromech",  )   droid\_data = {"2000": c3po, "2001": r2d2}  def get\_character(id):  return human\_data.get(id) or droid\_data.get(id)  def get\_friends(character):  return map(get\_character, character.friends)  def get\_hero(episode):  if episode == 5:  return human\_data["1000"]  return droid\_data["2001"]   def get\_human(id):  return human\_data.get(id)  def get\_droid(id):  return droid\_data.get(id)  ''' ===================== schema ========================= '''  import graphene from flask\_graphql import GraphQLView from flask import Flask  class Episode(graphene.Enum): # 剧集  NEWHOPE = 4 # 星球大战4 新希望  EMPIRE = 5 # 黑金帝国  JEDI = 6 # 星球大战 绝地  class Character(graphene.Interface): # 角色  id = graphene.ID()  name = graphene.String()  friends = graphene.List(lambda: Character)  appears\_in = graphene.List(Episode) # 出演   def resolve\_friends(self, info):  # The character friends is a list of strings  return [get\_character(f) for f in self.friends]  class Human(graphene.ObjectType): # 人类  class Meta:  interfaces = (Character,)   home\_planet = graphene.String() # 地球家园   class Droid(graphene.ObjectType): # 机器人  class Meta:  interfaces = (Character,)   primary\_function = graphene.String() # 主要功能  class Query(graphene.ObjectType):  hero = graphene.Field(Character, episode=Episode())  human = graphene.Field(Human, id=graphene.String())  droid = graphene.Field(Droid, id=graphene.String())   def resolve\_hero(root, info, episode=None):  return get\_hero(episode)   def resolve\_human(root, info, id):  return get\_human(id)   def resolve\_droid(root, info, id):  return get\_droid(id)  if \_\_name\_\_ == '\_\_main\_\_':  setup()   schema = graphene.Schema(query=Query)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql',   view\_func=GraphQLView.as\_view('graphql',schema=schema, graphiql=True))  app.run(debug=True) |

查询

|  |
| --- |
| # select name from table1 t1 left join table2 t2 on t1.id == t2.id ; {  hero {  name  friends {  name  }  } }   # select name from table1 where id = '??'; {  human(id:"1002"){  name  appearsIn  } }   # select name as n1 name as n2 from table1 {  h1: human(id:"1002"){  name  }  h2: human(id:"1003"){  name  } }  # 字段复用 {  h1: human(id:"1002"){  ...heroFields  }  h2: human(id:"1003"){  ...heroFields  } }  fragment heroFields on Character{  id  name  friends {  id  name  } }  # 设置查询名称 query queryHero{  h1: human(id:"1002"){  ...heroFields  }  h2: human(id:"1003"){  ...heroFields  } }  # 接口子类专属属性查询 {animal(type\_:1) {  id,  name,  ... on Mouse{  run  }  ... on Bird{  fly  } }}  # 参数输入  query queryHero($id:String!){  h1: human(id:$id){  name  id  } }  {  "id": "1002" } |

06\_数据的增删改操作

增加

|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask  users = []  class Person(graphene.ObjectType):  name = graphene.String()  age = graphene.Int() # 1 class CreatePerson(graphene.Mutation):  person = graphene.Field(Person)  # 2  class Arguments:  name = graphene.String()  age = graphene.Int()  # 3  def mutate(self, info, name, age):  person = Person(name=name, age=age)  users.append(person)  return CreatePerson(person = person) # 4  class Mutation(graphene.ObjectType):  create\_person = CreatePerson.Field()   class Query(graphene.ObjectType):  persons = graphene.List(Person)   def resolve\_persons(self,info):  return users  if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query, mutation=Mutation)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql',  view\_func=GraphQLView.as\_view('graphql', schema=schema, graphiql=True))  app.run(debug=True) |

修改与删除

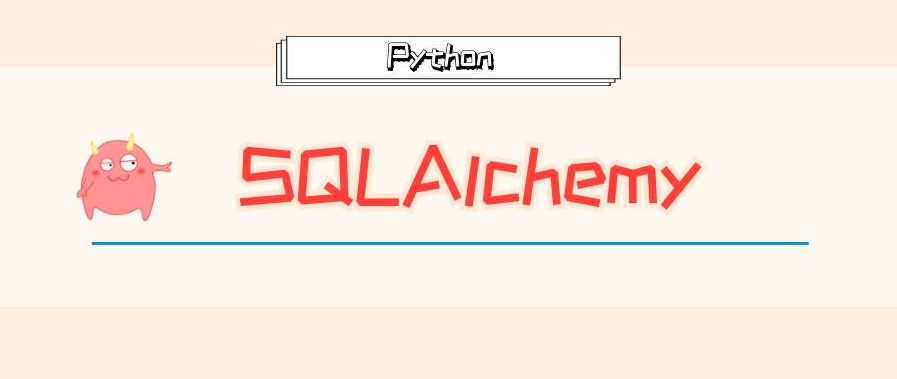
|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask   class Person(graphene.ObjectType):  name = graphene.String(required = True)  age = graphene.Int()  users = [  Person(name='zs',age=18),  Person(name='ls',age=20), ]  # 1 class UpdatePerson(graphene.Mutation):  person = graphene.Field(Person)  msg = graphene.String()  # 2  class Arguments:  name = graphene.String()  age = graphene.Int()  # 3  def mutate(self, info, name, age):  for p in users:  if p.name == name:  p.age = age  return UpdatePerson(person = p,msg ="success")  return UpdatePerson(msg = 'fail') # 4  class DeletePerson(graphene.Mutation):  person = graphene.Field(Person)  msg = graphene.String()  # 2  class Arguments:  name = graphene.String()  age = graphene.Int()  # 3  def mutate(self, info, name):  for p in users:  if p.name == name:  users.remove(p)  return UpdatePerson(person = p,msg ="success")  return UpdatePerson(msg = 'fail')  class Mutation(graphene.ObjectType):  update\_person = UpdatePerson.Field()  delete\_person = DeletePerson.Field()   class Query(graphene.ObjectType):  persons = graphene.List(Person)   def resolve\_persons(self,info):  return users  ''' mutation update{  updatePerson(name:"zs",age:16){  person{  name  age  }  msg  } }  mutation delete{  deletePerson(name:"ls"){  person {  name  age  }  msg  } }  query select{  persons{  name  age  } } '''  if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query, mutation=Mutation)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql',  view\_func=GraphQLView.as\_view('graphql', schema=schema, graphiql=True))  app.run(debug=True) |

07\_原生SQL执行



|  |
| --- |
| import pymysql import graphene from flask\_graphql import GraphQLView from flask import Flask  def get\_conn():  client = pymysql.connect(host='192.168.30.151',port=3306,user='root',passwd='123',db='graphql',charset='utf8')  cursor = client.cursor()  return client,cursor  class Person(graphene.ObjectType):  id = graphene.ID()  name = graphene.String()  age = graphene.Int()  class DataInput(graphene.InputObjectType):  name = graphene.String()  age = graphene.Int()  class CreatePerson(graphene.Mutation):  person = graphene.Field(Person)   class Arguments:  name = graphene.String()  age = graphene.Int()    def mutate(self, info, name, age):  sql = 'insert into t\_user values (%s,%s)'  c1,c2 = get\_conn()  c2.execute(sql,[name,age])  c1.commit()  c2.close()  c1.close()   return CreatePerson(person = Person(name=name,age=age))  class UpdatePerson(graphene.Mutation):  person = graphene.Field(Person)  msg = graphene.String()   class Arguments:  p1 = DataInput(required=True)   def mutate(self, info, name, age):  try:  sql = 'update t\_user set age = %s where name = %s'  c1,c2 = get\_conn()  c2.execute(sql,[name,age])  c1.commit()  c2.close()  c1.close()  return UpdatePerson(person = p,msg ="success")  except Exception as e:  return UpdatePerson(msg = 'fail')  class DeletePerson(graphene.Mutation):  person = graphene.Field(Person)  msg = graphene.String()   class Arguments:  p1 = DataInput(required=True)   def mutate(self, info, name):  try:  sql = 'delete from t\_user where name = %s'  c1,c2 = get\_conn()  c2.execute(sql,[name,age])  c1.commit()  c2.close()  c1.close()  return UpdatePerson(person = p,msg ="success")  except Exception as e:  return UpdatePerson(msg = 'fail')  class Mutation(graphene.ObjectType):  create\_person = CreatePerson.Field()  update\_person = UpdatePerson.Field()  delete\_person = DeletePerson.Field()   class Query(graphene.ObjectType):  persons = graphene.List(Person)   def resolve\_persons(self,info):  return users  if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query, mutation=Mutation)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql',  view\_func=GraphQLView.as\_view('graphql', schema=schema, graphiql=True))  app.run(debug=True) |

08\_SqlAlchemy的使用



安装

|  |
| --- |
| pip install "graphene-sqlalchemy>=2.0" |

官网：

|  |
| --- |
| https://github.com/graphql-python/graphene-sqlalchemy |

**Tip 传递的数据需要 base64 加密**

berfore after id:<model:id>

|  |
| --- |
| import graphene from flask\_graphql import GraphQLView from flask import Flask  from sqlalchemy import create\_engine,Column,Integer,String from sqlalchemy.ext.declarative import declarative\_base from sqlalchemy.orm import scoped\_session, sessionmaker from graphene\_sqlalchemy import SQLAlchemyConnectionField, SQLAlchemyObjectType from uuid import uuid4  engine = create\_engine('sqlite:///graphql.sqlite3') db\_session = scoped\_session(sessionmaker(autocommit=False,  autoflush=False,  bind=engine)) Base = declarative\_base() Base.query = db\_session.query\_property()   class EmployeeModel(Base):  \_\_tablename\_\_ = 't\_employee'  # id = Column(Integer, primary\_key=True)  id = Column(String(128),primary\_key=True)  name = Column(String)   class Employee(SQLAlchemyObjectType):  class Meta:  model = EmployeeModel  interfaces = (graphene.relay.Node, )  class Query(graphene.ObjectType):  emps = graphene.List(Employee)  emp = graphene.Field(Employee,id = graphene.String())    node = graphene.relay.Node.Field()  employees = SQLAlchemyConnectionField(Employee.connection)    def resolve\_emps(self, info):  query = Employee.get\_query(info) # SQLAlchemy query  return query.all()  def resolve\_emp(self,info,id):  return Employee.get\_node(info =info ,id = id)   class CreateEmp(graphene.Mutation):  emp = graphene.Field(Employee)  msg = graphene.Field(graphene.String)   class Arguments:  ida = graphene.String()  names = graphene.String()   def mutate(self, info, ida, names):  e = EmployeeModel(id = ida,name = names)  db\_session.add(e)  db\_session.commit()  return CreateEmp(emp = e,msg ="success")   class Mutation(graphene.ObjectType):  create\_emp = CreateEmp.Field()  # 2 if \_\_name\_\_ == '\_\_main\_\_':  schema = graphene.Schema(query=Query,mutation= Mutation)  app = Flask(\_\_name\_\_)  app.add\_url\_rule('/graphql',  view\_func=GraphQLView.as\_view('graphql', schema=schema, graphiql=True))  app.run(debug=True)  ''' query q1{  emps {  id  name  }  emp(id:"425e0096c2814826a562d13c65132c58") {  id  name  } }  mutation ca {  createEmp(ida:"aaa",names:"c1") {  msg  emp{  id,  name  }  } } '''  ''' query{  node(id:"RW1wbG95ZWU6YWFh") { mode:id  id  ...on Employee{  id  name  }  } } ''' |