**SQLAlchemy 简单笔记**

[](https://www.jianshu.com/u/5qrPPM)

[人世间](https://www.jianshu.com/u/5qrPPM) 关注

2014.07.07 15:07\* 字数 2071 阅读 44430评论 12喜欢 69赞赏 1

**ORM 江湖#####**

曾几何时，程序员因为惧怕SQL而在开发的时候小心翼翼的写着sql，心中总是少不了恐慌，万一不小心sql语句出错，搞坏了数据库怎么办？又或者为了获取一些数据，什么内外左右连接，函数存储过程等等。毫无疑问，不搞懂这些，怎么都觉得变扭，说不定某天就跳进了坑里，叫天天不应，喊地地不答。

ORM 的出现，让畏惧SQL的开发者，在坑里看见了爬出去的绳索，仿佛天空并不是那么黑暗，至少再暗，我们也有了眼睛。顾名思义，ORM 对象关系映射，简而言之，就是把数据库的一个个table(表)，映射为编程语言的class(类)。

python中比较著名的ORM框架有很多，大名顶顶的 SQLAlchemy 是python世界里当仁不让的ORM框架。江湖中peewee，strom， pyorm，SQLObject 各领风骚，可是最终还是SQLAlchemy 傲视群雄。

**SQLAlchemy 简介#####**

SQLAlchemy 分为两个部分，一共用于 ORM 的对象映射，另外一个是核心的 SQL expression 。第一个很好理解，纯粹的ORM，后面这个不是 ORM，而是DBAPI的封装，当然也提供了很多方法，避免了直接写sql，而是通过一些sql表达式。使用 SQLAlchemy 则可以分为三种方式。

* 使用 sql expression ，通过 SQLAlchemy 的方法写sql表达式，简介的写sql
* 使用 raw sql， 直接书写 sql
* 使用 ORM 避开直接书写 sql

本文先探讨 SQLAlchemy的 sql expresstion 部分的用法。主要还是跟着官方的 SQL Expression Language Tutorial.介绍

为什么要学习 sql expresstion ，而不直接上 ORM？因为后面这个两个是 orm 的基础。并且，即是不使用orm，后面这两个也能很好的完成工作，并且代码的可读性更好。纯粹把SQLAlchemy当成dbapi使用。首先SQLAlchemy 内建数据库连接池，解决了连接操作相关繁琐的处理。其次，提供方便的强大的log功能，最后，复杂的查询语句，依靠单纯的ORM比较难实现。

**实战#####**

**连接数据库######**

首先需要导入 sqlalchemy 库，然后建立数据库连接，这里使用 mysql。通过create\_engine方法进行

from sqlalchemy import create\_engine

engine = create\_engine("mysql://root:@localhost:3306/webpy?charset=utf8",encoding="utf-8", echo=True)

create\_engine 方法进行数据库连接，返回一个 db 对象。里面的参数表示

数据库类型://用户名:密码（没有密码则为空，不填）@数据库主机地址/数据库名?编码  
echo = True 是为了方便 控制台 logging 输出一些sql信息，默认是False

通过这个engine对象可以直接execute 进行查询，例如 engine.execute("SELECT \* FROM user") 也可以通过 engine 获取连接在查询，例如 conn = engine.connect() 通过 conn.execute()方法进行查询。两者有什么差别呢？

* 直接使用engine的execute执行sql的方式, 叫做connnectionless执行,
* 借助 engine.connect()获取conn, 然后通过conn执行sql, 叫做connection执行  
  主要差别在于是否使用transaction模式, 如果不涉及transaction, 两种方法效果是一样的. 官网推荐使用后者。

**定义表#####**

定义数据表，才能进行sql表达式的操作，毕竟sql表达式的表的确定，是sqlalchemy制定的，如果数据库已经存在了数据表还需要定义么？当然，这里其实是一个映射关系，如果不指定，查询表达式就不知道是附加在那个表的操作，当然定义的时候，注意表名和字段名，代码和数据的必须保持一致。定义好之后,就能创建数据表，一旦创建了，再次运行创建的代码，数据库是不会创建的。

# -\*- coding: utf-8 -\*-

\_\_author\_\_ = 'ghost'

from sqlalchemy import create\_engine, Table, Column, Integer, String, MetaData, ForeignKey

# 连接数据库

engine = create\_engine("mysql://root:@localhost:3306/webpy?charset=utf8",encoding="utf-8", echo=True)

# 获取元数据

metadata = MetaData()

# 定义表

user = Table('user', metadata,

Column('id', Integer, primary\_key=True),

Column('name', String(20)),

Column('fullname', String(40)),

)

address = Table('address', metadata,

Column('id', Integer, primary\_key=True),

Column('user\_id', None, ForeignKey('user.id')),

Column('email', String(60), nullable=False)

)

# 创建数据表，如果数据表存在，则忽视

metadata.create\_all(engine)

# 获取数据库连接

conn = engine.connect()

**插入 insert**

有了数据表和连接对象，对应数据库操作就简单了。

>>> i = user.insert() # 使用查询

>>> i

<sqlalchemy.sql.dml.Insert object at 0x0000000002637748>

>>> print i # 内部构件的sql语句

INSERT INTO "user" (id, name, fullname) VALUES (:id, :name, :fullname)

>>> u = dict(name='jack', fullname='jack Jone')

>>> r = conn.execute(i, \*\*u) # 执行查询，第一个为查询对象，第二个参数为一个插入数据字典，如果插入的是多个对象，就把对象字典放在列表里面

>>> r

<sqlalchemy.engine.result.ResultProxy object at 0x0000000002EF9390>

>>> r.inserted\_primary\_key # 返回插入行 主键 id

[4L]

>>> addresses

[{'user\_id': 1, 'email': 'jack@yahoo.com'}, {'user\_id': 1, 'email': 'jack@msn.com'}, {'user\_id': 2, 'email': 'www@www.org'}, {'user\_id': 2, 'email': 'wendy@aol.com'}]

>>> i = address.insert()

>>> r = conn.execute(i, addresses) # 插入多条记录

>>> r

<sqlalchemy.engine.result.ResultProxy object at 0x0000000002EB5080>

>>> r.rowcount #返回影响的行数

4L

>>> i = user.insert().values(name='tom', fullname='tom Jim')

>>> i.compile()

<sqlalchemy.sql.compiler.SQLCompiler object at 0x0000000002F6F390>

>>> print i.compile()

INSERT INTO "user" (name, fullname) VALUES (:name, :fullname)

>>> print i.compile().params

{'fullname': 'tom Jim', 'name': 'tom'}

>>> r = conn.execute(i)

>>> r.rowcount

1L

**查询 select**

查询方式很灵活，多数时候使用 sqlalchemy.sql 下面的 select方法

>>> s = select([user]) # 查询 user表

>>> s

<sqlalchemy.sql.selectable.Select at 0x25a7748; Select object>

>>> print s

SELECT "user".id, "user".name, "user".fullname

FROM "user"

如果需要查询自定义的字段，可是使用 user 的cloumn 对象，例如  
>>> user.c # 表 user 的字段column对象  
<sqlalchemy.sql.base.ImmutableColumnCollection object at 0x0000000002E804A8>  
>>> print user.c  
['user.id', 'user.name', 'user.fullname']  
>>> s = select([user.c.name,user.c.fullname])  
>>> r = conn.execute(s)  
>>> r  
<sqlalchemy.engine.result.ResultProxy object at 0x00000000025A7748>  
>>> r.rowcount # 影响的行数  
5L  
>>> ru = r.fetchall()  
>>> ru  
[(u'hello', u'hello world'), (u'Jack', u'Jack Jone'), (u'Jack', u'Jack Jone'), (u'jack', u'jack Jone'), (u'tom', u'tom Jim')]  
>>> r  
<sqlalchemy.engine.result.ResultProxy object at 0x00000000025A7748>  
>>> r.closed # 只要 r.fetchall() 之后，就会自动关闭 ResultProxy 对象  
True

同时查询两个表  
>>> s = select([user.c.name, address.c.user\_id]).where(user.c.id==address.c.user\_id) # 使用了字段和字段比较的条件  
>>> s  
<sqlalchemy.sql.selectable.Select at 0x2f03390; Select object>  
>>> print s  
SELECT "user".name, address.user\_id  
FROM "user", address  
WHERE "user".id = address.user\_id

**操作符######**

>>> print user.c.id == address.c.user\_id # 返回一个编译的字符串

"user".id = address.user\_id

>>> print user.c.id == 7

"user".id = :id\_1 # 编译成为带参数的sql 语句片段字符串

>>> print user.c.id != 7

"user".id != :id\_1

>>> print user.c.id > 7

"user".id > :id\_1

>>> print user.c.id == None

"user".id IS NULL

>>> print user.c.id + address.c.id # 使用两个整形的变成 +

"user".id + address.id

>>> print user.c.name + address.c.email # 使用两个字符串 变成 ||

"user".name || address.email

**操作连接######**

这里的连接指条件查询的时候，逻辑运算符的连接，即 and or 和 not  
>>> print and\_(  
user.c.name.like('j%'),  
user.c.id == address.c.user\_id,  
or\_(  
[address.c.email](https://link.jianshu.com/?t=http://address.c.email) == '[wendy@aol.com](https://link.jianshu.com/?t=mailto:wendy@aol.com)',  
[address.c.email](https://link.jianshu.com/?t=http://address.c.email) == '[jack@yahoo.com](https://link.jianshu.com/?t=mailto:jack@yahoo.com)'  
),  
not\_(user.c.id>5))  
"user".name LIKE :name\_1 AND "user".id = address.user\_id AND (address.email = :email\_1 OR address.email = :email\_2) AND "user".id <= :id\_1  
>>>

得到的结果为 编译的sql语句片段，下面看一个完整的例子

>>> se\_sql = [(user.c.fullname +", " + address.c.email).label('title')]

>>> wh\_sql = and\_(

user.c.id == address.c.user\_id,

user.c.name.between('m', 'z'),

or\_(

address.c.email.like('%@aol.com'),

address.c.email.like('%@msn.com')

)

)

>>> print wh\_sql

"user".id = address.user\_id AND "user".name BETWEEN :name\_1 AND :name\_2 AND (address.email LIKE :email\_1 OR address.email LIKE :email\_2)

>>> s = select(se\_sql).where(wh\_sql)

>>> print s

SELECT "user".fullname || :fullname\_1 || address.email AS title

FROM "user", address

WHERE "user".id = address.user\_id AND "user".name BETWEEN :name\_1 AND :name\_2 AND (address.email LIKE :email\_1 OR address.email LIKE :email\_2)

>>> r = conn.execute(s)

>>> r.fetchall()

使用 raw sql 方式

遇到负责的sql语句的时候，可以使用 sqlalchemy.sql 下面的 text 函数。将字符串的sql语句包装编译成为 execute执行需要的sql对象。例如：、  
>>> text\_sql = "SELECT id, name, fullname FROM user WHERE id=:id" # 原始sql语句，参数用（ ：value）表示  
>>> s = text(text\_sql)  
>>> print s  
SELECT id, name, fullname FROM user WHERE id=:id  
>>> s  
<sqlalchemy.sql.elements.TextClause object at 0x0000000002587668>  
>>> conn.execute(s, id=3).fetchall() # id=3 传递：id参数  
[(3L, u'Jack', u'Jack Jone')]

**连接 join**

连接有join 和 outejoin 两个方法，join 有两个参数，第一个是join 的表，第二个是on 的条件，joing之后必须要配合select\_from 方法  
>>> print user.join(address)  
"user" JOIN address ON "user".id = address.user\_id # 因为开启了外键 ，所以join 能只能识别 on 条件  
>>> print user.join(address, address.c.user\_id==user.c.id) # 手动指定 on 条件  
"user" JOIN address ON address.user\_id = "user".id

>>> s = select([user.c.name, address.c.email]).select\_from(user.join(address, user.c.id==address.c.user\_id)) # 被jion的sql语句需要用 select\_from方法配合

>>> s

<sqlalchemy.sql.selectable.Select at 0x2eb63c8; Select object>

>>> print s

SELECT "user".name, address.email

FROM "user" JOIN address ON "user".id = address.user\_id

>>> conn.execute(s).fetchall()

[(u'hello', u'jack@yahoo.com'), (u'hello', u'jack@msn.com'), (u'hello', u'jack@yahoo.com'), (u'hello', u'jack@msn.com'), (u'Jack', u'www@www.org'), (u'Jack', u'wendy@aol.com'), (u'Jack', u'www@www.org'), (u'Jack', u'wendy@aol.com')]

更复杂的连接参考 官方的文档了。

**排序 分组 分页**

排序使用 order\_by 方法，分组是 group\_by ，分页自然就是limit 和 offset两个方法配合

>>> s = select([user.c.name]).order\_by(user.c.name) # order\_by

>>> print s

SELECT "user".name

FROM "user" ORDER BY "user".name

>>> s = select([user]).order\_by(user.c.name.desc())

>>> print s

SELECT "user".id, "user".name, "user".fullname

FROM "user" ORDER BY "user".name DESC

>>> s = select([user]).group\_by(user.c.name) # group\_by

>>> print s

SELECT "user".id, "user".name, "user".fullname

FROM "user" GROUP BY "user".name

>>> s = select([user]).order\_by(user.c.name.desc()).limit(1).offset(3) # limit(1).offset(3)

>>> print s

SELECT "user".id, "user".name, "user".fullname

FROM "user" ORDER BY "user".name DESC

LIMIT :param\_1 OFFSET :param\_2

[(4L, u'jack', u'jack Jone')]

**更新 update**

前面都是一些查询，更新和插入的方法很像，都是 表下面的方法，不同的是，update 多了一个 where 方法 用来选择过滤  
>>> s = user.update()  
>>> print s  
UPDATE "user" SET id=:id, name=:name, fullname=:fullname  
>>> s = user.update().values(fullname=user.c.name) # values 指定了更新的字段  
>>> print s  
UPDATE "user" SET fullname="user".name  
>>> s = user.update().where(user.c.name == 'jack').values(name='ed') # where 进行选择过滤  
>>> print s  
UPDATE "user" SET name=:name WHERE "user".name = :name\_1  
>>> r = conn.execute(s)  
>>> print r.rowcount # 影响行数  
3

还有一个高级用法，就是一次命令执行多个记录的更新，需要用到 bindparam 方法

>>> s = user.update().where(user.c.name==bindparam('oldname')).values(name=bindparam('newname')) # oldname 与下面的传入的从拿书进行绑定，newname也一样

>>> print s

UPDATE "user" SET name=:newname WHERE "user".name = :oldname

>>> u = [{'oldname':'hello', 'newname':'edd'},

{'oldname':'ed', 'newname':'mary'},

{'oldname':'tom', 'newname':'jake'}]

>>> r = conn.execute(s, u)

>>> r.rowcount

5L

**删除 delete**

删除比较容易，调用 delete方法即可，不加 where 过滤，则删除所有数据，但是不会drop掉表，等于清空了数据表  
>>> r = conn.execute(address.delete()) # 清空表  
>>> print r  
<sqlalchemy.engine.result.ResultProxy object at 0x0000000002EAF550>  
>>> r.rowcount  
8L  
>>> r = conn.execute(users.delete().where(users.c.name > 'm')) # 删除记录  
>>> r.rowcount  
3L

至此，sqlalchemy sql表达式的基本用法介绍完毕，更深入的阅读可以查看官方的api SQL Statements and Expressions API