**sqlalchemy 学习（一）创建Session和Engine**

2017年02月04日 18:04:23

阅读数：4077

**基本概念**

说到数据库，就离不开Session。Session的主要目的是建立与数据库的会话，它维护你加载和关联的所有数据库对象。它是数据库查询（Query）的一个入口。

在Sqlalchemy中，数据库的查询操作是通过Query对象来实现的。而Session提供了创建Query对象的接口。

Query对象返回的结果是一组同一映射（Identity Map）对象组成的集合。事实上，集合中的一个对象，对应于数据库表中的一行（即一条记录）。所谓同一映射，是指每个对象有一个唯一的ID。如果两个对象（的引用）ID相同，则认为它们对应的是相同的对象。

要完成数据库查询，就需要建立与数据库的连接。这就需要用到Engine对象。一个Engine可能是关联一个Session对象，也可能关联一个数据库表。

当然Session最重要的功能还是实现原子操作。

**创建Session**

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54863830) [copy](https://blog.csdn.net/hedan2013/article/details/54863830)

1. **from** sqlalchemy **import** create\_engine
2. **from** sqlalchemy.orm **import** sessionmaker
4. some\_engine = create\_engine('nysql://username:password@localhost/mydb?charset=utf8')
6. ession = sessionmaker(bind=some\_engine)
8. session = Session()

代码第4行创建一个Engine对象，我们提供的参数是数据库连接的url。

代码第6行通过sessionmaker方法创建了一个Session工厂。

代码第8行通过调用工厂方法来创建一个Session对象。

**Session的生命期**

Session通常在我们需要对数据库进行操作时创建。

一旦一个Session创建成功，我们在这个Session下完成多个事务（transaction）。

究竟何时创建和关闭session，不能一概而论。但是一个原则是session不应该在操作事务的方法中创建。

例如下面的方法定义就是不规范的。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54863830) [copy](https://blog.csdn.net/hedan2013/article/details/54863830)

1. **class** SomeThing(object):
2. **def** go(self):
3. session = Session()
4. **try**:
5. # Do something
6. session.commit()
7. **except**:
8. session.rollback()
9. **raise**

我们应该做如下改正。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54863830) [copy](https://blog.csdn.net/hedan2013/article/details/54863830)

1. **class** SomeThing(object):
2. **def** go(self, session):
3. # Do something
5. **def** run\_my\_program():
6. session = Session()
7. **try**:
8. SomeThing().go(session)
9. session.commit()
10. **except**:
11. session.rollback()
12. **raise**
13. **finally**:
14. session.close()

**sqlalchemy 学习（二）scoped session**

2017年02月04日 19:16:45

阅读数：3037

数据库设计的难点之一，是session生命周期的管理问题。sqlalchemy提供了一个简单的session管理机制，即scoped session。它采用的注册模式。所谓的注册模式，简单来说，是指在整个程序运行的过程当中，只存在唯一的一个session对象。

创建scoped session的方法如下

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54865144) [copy](https://blog.csdn.net/hedan2013/article/details/54865144)

1. **from** sqlalchemy.orm **import** scoped\_session
2. **from** sqlalchemy.orm **import** sessionmaker
4. session\_factory = sessionmaker(bind=some\_engine)
5. Session = scoped\_session(session\_factory)

scoped\_session本质上也是一个方法，它首先通过session工厂创建session对象，然后对session对象进行相应的管理。

如果需要创建session对象，可执行下面的代码。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54865144) [copy](https://blog.csdn.net/hedan2013/article/details/54865144)

1. some\_session = Session()

注意，由于scoped session采用的是注册模式，因此下面代码的session1和session2是一个相同的对象引用。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54865144) [copy](https://blog.csdn.net/hedan2013/article/details/54865144)

1. session1 = Session()
2. session2 = Session()

但是下面代码中的session1和session2是不同的对象引用。因为session2创建之前，session1已经被销毁。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54865144) [copy](https://blog.csdn.net/hedan2013/article/details/54865144)

1. session1 = Session()
2. session1.remove()
3. session2 = Session()

介绍到这里，我们知道scoped session本质上是一个全局变量。可是，如果直接把session定义成全局变量，在多线程的环境下，会造成线程同步的问题。为此，scoped session在默认情况下，采用的线程本地化存储方式。也就是说，每个线程的session对象是不同的。这样，不同线程对数据库的操作不会相互影响。

**sqlalchemy学习（三）：创建数据库表格**

2017年02月08日 11:58:08

阅读数：1610

在创建数据库表格之前，首先请保证数据库已经通过mysql命令创建完毕。

**[plain]** [view plain](https://blog.csdn.net/hedan2013/article/details/54914822) [copy](https://blog.csdn.net/hedan2013/article/details/54914822)

1. mysql> create database mydb character set utf8;
2. Query OK, 1 row affected (0.08 sec)

数据库创建好以后，我们需要定义数据库表类型。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54914822) [copy](https://blog.csdn.net/hedan2013/article/details/54914822)

1. **from** sqlalchemy.dialects.mysql **import** (INTEGER, CHAR)
2. **from** sqlalchemy **import** Column, Table, MetaData
4. meta = MetaData()
6. users\_table = Table('users', meta,
7. Column('id', INTEGER, primary\_key=True),
8. Column('name', CHAR(127)),
9. Column('password', CHAR(127))
10. )

代码第4行定义了一个元数据对象。它包含了与数据库有关的所有的属性。当然，在执行这一句时，数据库元数据对象还是没有任何内容的。

代码第6-10行定义了一个数据库表。在这个方法中：第一个参数‘users’是数据库表的名字；第二个参数，是我们刚刚创建的metadata对象。后面的几个参数分别定义了数据库表的各列。大家一看便知，不再一一介绍。

在执行这个方法时，会把users表的相关属性都添加进metadata对象。

数据库表定义好以后，便可以实行创建了。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54914822) [copy](https://blog.csdn.net/hedan2013/article/details/54914822)

1. **from** sqlalchemy **import** create\_engine
3. ENGINE = create\_engine("mysql://root:password@localhost:3306/mydb?charset=utf8",
4. convert\_unicode=True)
6. users\_table.create(bind=ENGINE)

上面的代码一目了然，无须介绍。

代码执行完后，可以看到数据库中已经新建了一个名为users表。

**[plain]** [view plain](https://blog.csdn.net/hedan2013/article/details/54914822) [copy](https://blog.csdn.net/hedan2013/article/details/54914822)

1. mysql> use mydb
2. Database changed
3. mysql> show tables;
4. +----------------+
5. | Tables\_in\_mydb |
6. +----------------+
7. | users          |
8. +----------------+
9. 1 row in set (0.00 sec)
11. mysql> desc users;
12. +----------+-----------+------+-----+---------+----------------+
13. | Field    | Type      | Null | Key | Default | Extra          |
14. +----------+-----------+------+-----+---------+----------------+
15. | id       | int(11)   | NO   | PRI | NULL    | auto\_increment |
16. | name     | char(127) | YES  |     | NULL    |                |
17. | password | char(127) | YES  |     | NULL    |                |
18. +----------+-----------+------+-----+---------+----------------+
19. 3 rows in set (0.01 sec)

**sqlalchemy学习（四）ORM model及数据库表的增删改查**

2017年02月08日 17:18:35

阅读数：3969

目前，许多主流的语言，都实现了对象关系映射（ORM Object Relational Mapper）的库包。ORM的主要功能是将数据库表中的每条记录映射成一个对象。所有的数据库操作，都转化为对象的操作。这样可以增加代码的可读性和安全性。当然性能上会低于直接执行sql命令。本文介绍Sqlalchemy的ORM操作。Sqlalchemy的版本需要在1.1以上。

在执行后面的代码前，请保证已经根据上一篇[博文](http://blog.csdn.net/hedan2013/article/details/54914822)的介绍创建好了名为users的表格。

在ORM中，数据库的每张表，都对应一个model类。而这些model类都需要继承一个名为declarative base class的基类。我们需要通过下面的代码来获得这个基类。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. **from** sqlalchemy.ext.declarative **import**  declarative\_base
3. BASE = declarative\_base()

有了这个基类，就可以为users表定义model类了。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. **class** User(BASE):
2. \_\_tablename\_\_ = "users"
4. id = Column(INTEGER, primary\_key=True)
5. name = Column(CHAR(127))
6. password = Column(CHAR(127))

上面的代码中，\_\_tablename\_\_类成员变量指明了User类对应的是数据库的users表。后面的3个类成员变量都是Column对象，分别对应表的三个域（field）。

注意：默认情况下，User中Column对象的名字与数据库表field的名字相一致。如果两者不一致时，需要在Column对象的构造方法中传入name参数。

**添加记录**

下面的代码介绍了往数据库表添加记录的方法。

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. **from** sqlalchemy.orm **import** sessionmaker
2. **from** sqlalchemy **import** create\_engine
4. ENGINE = create\_engine("mysql://root:zaq12wsx@localhost:3306/mydb?charset=utf8",
5. convert\_unicode=True)
6. Session = sessionmaker(bind=ENGINE, autocommit=False, autoflush=False)
8. session = Session()
10. user = User()
11. user.name = "user1"
12. user.password = "password"
13. session.add(user)
14. session.commit()

代码第4-8行是用来创建session对象。注意我们把session对象的自动提交功能禁用了。

代码第10-12行创建了一个User对象。这个对象有3个成员变量，即id, name和password。与三个User类中三个Column对象的类成员变量名字一致。

代码第14行，是将新建的user对象添加进来。但是注意，由于session的自动提交功能禁用了，所在执行完这行代码以后，数据库并没有真正更新。直到第15行代码执行完毕后，才会真正更新数据库。

下面是数据库的更新结果。

**[plain]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. mysql> select \* from users;
2. +----+-------+----------+
3. | id | name  | password |
4. +----+-------+----------+
5. |  1 | user1 | password |
6. +----+-------+----------+
7. 1 row in set (0.00 sec)

**查询记录**

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. user = session.query(User).filter\_by(name="user1").first()
2. **print** "%s %s" % (user.name, user.password)

代码中session.query方法会返回一个Query对象。此后，调用Query对象的filter\_by方法进行查询。它返回的依然是Query对象。最后调用Query对象的first方法获得查询记录的第1条记录。如果查询记录为空，则返回None。

**更新记录**

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. user = session.query(User).filter\_by(name="user1").first()
2. user.password = "newpassword"
3. session.commit()

**删除记录**

**[python]** [view plain](https://blog.csdn.net/hedan2013/article/details/54927160) [copy](https://blog.csdn.net/hedan2013/article/details/54927160)

1. user = session.query(User).filter\_by(name="user1").first()
2. session.delete(user)
3. session.commit()