18丨SQLAlchemy：如何使用Python ORM框架来操作MySQL？

2019-07-22 陈旸

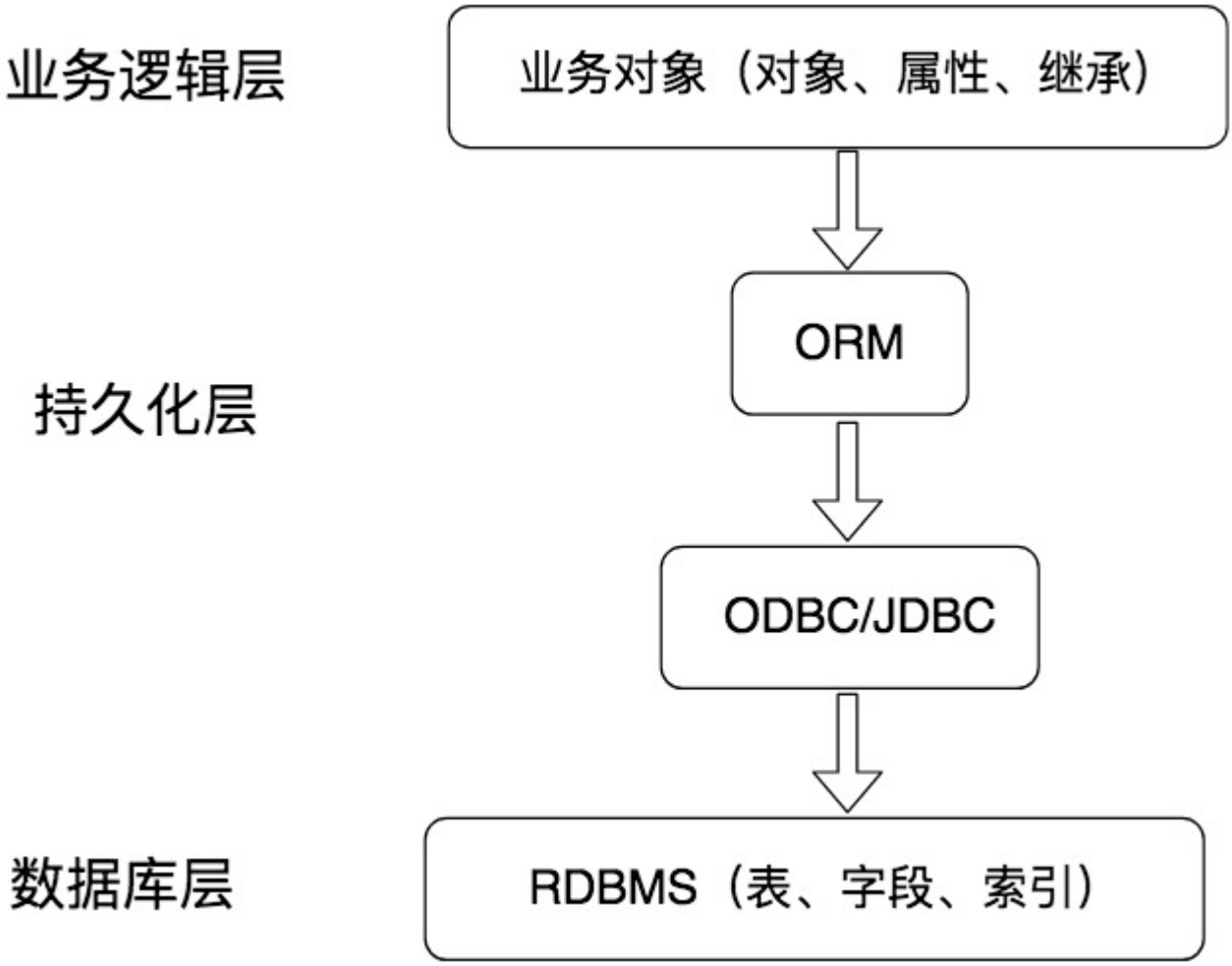
上节课，我介绍了Python DB API规范的作用，以及如何使用MySQL官方的mysql-connector驱动来完成数据库的连接和使用。在项目比较小的时候，我们可以直接使用SQL语句，通过mysql-connector完成与MySQL的交互，但是任何事物都有两面性，随着项目规模的增加，代码会越来越复杂，维护的成本也越来越高，这时mysql-connector就不够用了，我们需要更好的设计模式。

Python还有另一种方式可以与MySQL进行交互，这种方式采用的是ORM框架。我们今天就来讲解如何使用ORM框架操作MySQL，那么今天的课程你需要掌握以下几个方面的内容：

1. 什么是ORM框架，以及为什么要使用ORM框架？
2. Python中的ORM框架都有哪些？
3. 如何使用SQLAlchemy来完成与MySQL的交互？

我们为什么要使用ORM框架？

在讲解ORM框架之前，我们需要先了解什么是持久化。如下图所示，持久化层在业务逻辑层和数据库层起到了衔接的作用，它可以将内存中的数据模型转化为存储模型，或者将存储模型转化为内存中的数据模型。



你可能会想到，我们在讲事务的4大特性ACID时，提到过持久性。你可以简单地理解为，持久性就是将对象数据永久存储在数据库中。通常我们将数据库的作用理解为永久存储，将内存理解为暂时存储。我们在程序的层面操作数据，其实都是把数据放到内存中进行处理，如果需要数据就会通过持久化层，从数据库中取数据；如果需要保存数据，就是将对象数据通过持久化层存储到数据库中。

那么ORM解决的是什么问题呢？它提供了一种持久化模式，可以高效地对数据库进行访问。ORM的英文是Object Relation Mapping，中文叫对象关系映射。它是RDBMS和业务实体对象之间的一个映射，从图中你也能看到，它可以把底层的RDBMS封装成业务实体对象，提供给业务逻辑层使用。程序员往往关注业务逻辑层面，而不是底层数据库该如何访问，以及如何编写SQL语句获取数据等等。采用ORM，就可以从数据库的设计层面转化成面向对象的思维。

我在开篇的时候提到过，随着项目规模的增大，在代码层编写SQL语句访问数据库会降低开发效率，也会提升维护成本，因此越来越多的开发人员会采用基于ORM的方式来操作数据库。这样做的好处就是一旦定义好了对象模型，就可以让它们简单可复用，从而不必关注底层的数据库访问细节，我们只要将注意力集中到业务逻辑层面就可以了。由此还可以带来另一点好处，那就是即便数据库本身进行了更换，在业务逻辑代码上也不会有大的调整。这是因为ORM抽象了数据的存取，同时也兼容多种DBMS，我们不用关心底层采用的到底是哪种DBMS，是MySQL，SQL Server，PostgreSQL还是SQLite。

但没有一种模式是完美的，采用ORM当然也会付出一些代价，比如性能上的一些损失。面对一些复杂的数据查询，ORM会显得力不从心。虽然可以实现功能，但相比于直接编写SQL查询语句来说，ORM需要编写的代码量和花费的时间会比较多，这种情况下，直接编写SQL反而会更简单有效。

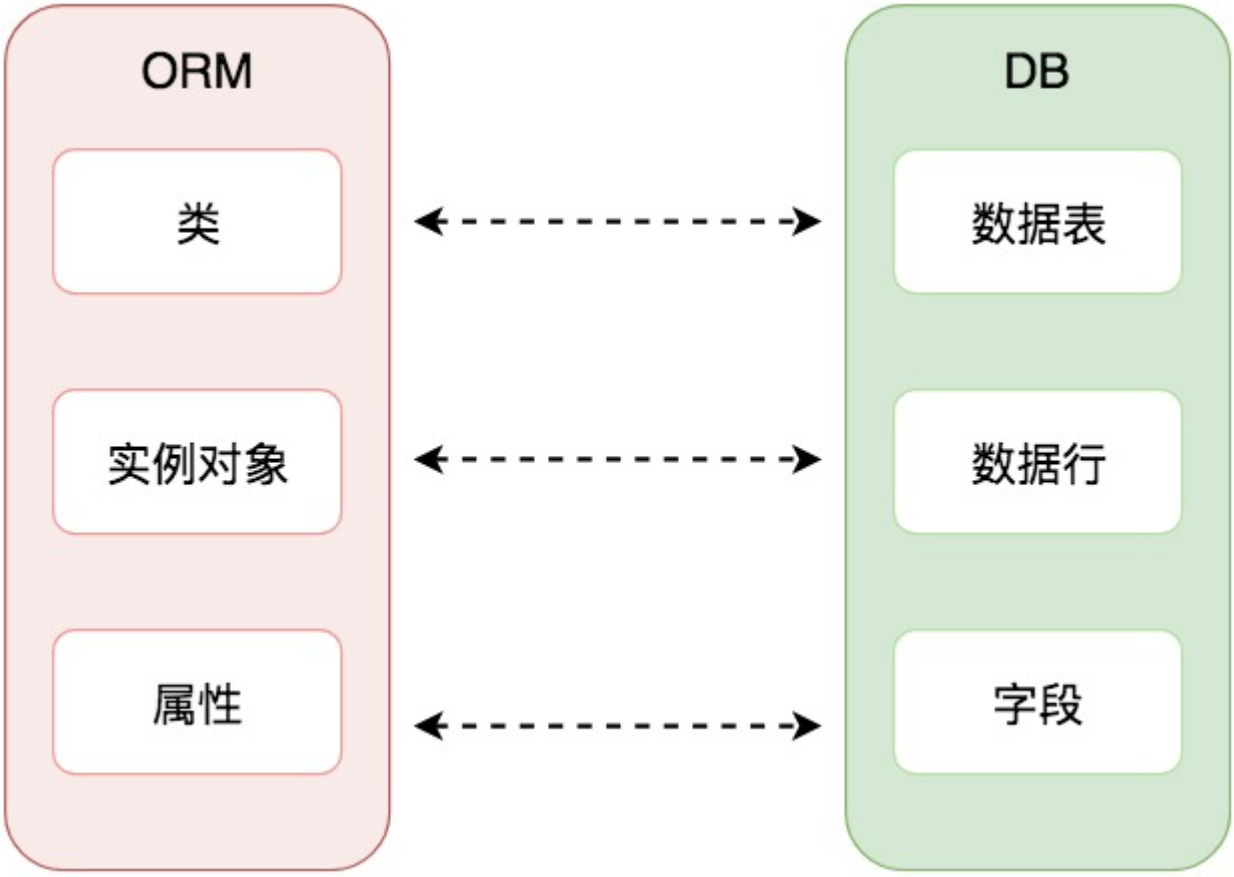
其实你也能看出来，没有一种方式是一劳永逸的，在实际工作中，我们需要根据需求选择适合的方式。

Python中的ORM框架都有哪些

ORM框架帮我们适配了各种DBMS，同时我们也可以选择不同的ORM框架。如果你用Python的话，有三种主流的ORM框架。

第一个是Django，它是Python的WEB应用开发框架，本身走大而全的方式。Django采用了MTV的框架模式，包括了Model（模型），View（视图）和Template（模版）。Model模型只是Django的一部分功能，我们可以通过它来实现数据库的增删改查操作。

一个Model映射到一个数据表，如下图所示：



从这张图上你能直观地看到，ORM的作用就是建立了对象关系映射。模型的每个属性代表数据表中的一个字段，我们通过操作类实例对象，对数据表中的数据行进行增删改查等操作。

第二个是SQLALchemy，它也是Python中常用的ORM框架之一。它提供了SQL工具包及ORM工

具，如果你想用支持ORM和支持原生SQL两种方式的工具，那么SQLALchemy是很好的选择。另外SQLALchemy的社区更加活跃，这对项目实施会很有帮助。

第三个是peewee，这是一个轻量级的ORM框架，简单易用。peewee采用了Model类、Field实例和Model实例来与数据库建立映射关系，从而完成面向对象的管理方式。使用起来方便，学习成本也低。

如何使用SQLAlchemy来操作MySQL

下面我们来看下如何使用SQLAlchemy工具对player数据表进行增删改查，在使用前，你需要先安装相应的工具包：

```
pip install sqlalchemy
初始化数据库连接
from sqlalchemy import create_engine
# 初始化数据库连接，修改为你的数据库用户名和密码
engine = create_engine('mysql+mysqlconnector://root:password@localhost:3306/wucai')
```

create_engine的使用方法类似我们在上篇文章中提到的mysql.connector，都需要提供数据库+数据库连接框架，即对应的是mysql+mysqlconnector，后面的是用户名:密码@IP地址:端口号/数据库名称。

创建模型

我们已经创建了player数据表，这里需要创建相应的player模型。

```
# 定义Player对象:
class Player(Base):
    # 表的名字:
    __tablename__ = 'player'

    # 表的结构:
    player_id = Column(Integer, primary_key=True, autoincrement=True)
    team_id = Column(Integer)
    player_name = Column(String(255))
    height = Column(Float(3,2))
```

这里需要说明的是，__tablename__ 指明了模型对应的数据表名称，即player数据表。同时我们在Player模型中对采用的变量名进行定义，变量名需要和数据表中的字段名称保持一致，否则会

找不到数据表中的字段。在SQLAlchemy中，我们采用Column对字段进行定义，常用的数据类型如下：

Integer	整数型
Float	浮点类型
Decimal	定点类型
Boolean	布尔类型
Date	datetime.date 日期类型
Time	datetime.time 时间类型
String	字符类型，使用时需要指定长度，区别于Text类型
Text	文本类型

除了指定Column的数据类型以外，我们也可以指定Column的参数，这些参数可以帮我们对对象创建列约束：

default	默认值
primary_key	是否为主键
unique	是否唯一
autoincrement	是否自动增长

这里需要说明的是，如果你使用相应的数据类型，那么需要提前在SQLAlchemy中进行引用，比如：

```
from sqlalchemy import Column, String, Integer, Float
```

对数据表进行增删改查

假设我们想给player表增加一名新球员，姓名为“约翰·科林斯”，球队ID为1003（即亚特兰大老鹰），身高为2.08。代码如下：

```
# 创建DBSession类型:
DBSession = sessionmaker(bind=engine)
# 创建session对象:
session = DBSession()



# 创建Player对象:
new_player = Player(team_id = 1003, player_name = "约翰-科林斯", height = 2.08)
# 添加到session:
session.add(new_player)
# 提交即保存到数据库:
session.commit()
# 关闭session:
session.close()
```

这里，我们首先需要初始化DBSession，相当于创建一个数据库的会话实例session。通过session来完成新球员的添加。对于新球员的数据，我们可以通过Player类来完成创建，在参数中指定相应的team_id, player_name, height即可。

然后把创建好的对象new_player添加到session中，提交到数据库即可完成添加数据的操作。

接着，我们来看一下如何查询数据。

添加完插入的新球员之后，我们可以查询下身高≥2.08m的球员都有哪些，代码如下：

```
#增加to_dict()方法到Base类中
def to_dict(self):
    return {c.name: getattr(self, c.name, None)
            for c in self.__table__.columns}
#将对象可以转化为dict类型
Base.to_dict = to_dict
# 查询身高>=2.08的球员有哪些
rows = session.query(Player).filter(Player.height >= 2.08).all()
print([row.to_dict() for row in rows])
```

运行结果：

```
[{'player_id': 10003, 'team_id': 1001, 'player_name': '安德烈-德拉蒙德', 'height': Decimal('2.1100000000')}, {'player_id
```

如果我们对整个数据行进行查询，采用的是session.query(Player)，相当于使用的是SELECT *。这时如果我们想要在Python中对query结果进行打印，可以对Base类增加to_dict()方法，相当于将对象转化成了Python的字典类型。

在进行查询的时候，我们使用的是filter方法，对应的是SQL中的WHERE条件查询。除此之外，filter也支持多条件查询。

如果是AND的关系，比如我们想要查询身高 ≥ 2.08，同时身高 ≤ 2.10的球员，可以写成下面这样：

```
rows = session.query(Player).filter(Player.height >=2.08, Player.height <=2.10).all()
```

如果是OR的关系，比如我们想要查询身高 ≥ 2.08，或者身高 ≤ 2.10的球员，可以写成这样：

```
rows = session.query(Player).filter(or_(Player.height >=2.08, Player.height <=2.10)).all()
```

这里我们使用了SQLAlchemy的or_操作符，在使用它之前你需要进行引入，即：from sqlalchemy import or_。

除了多条件查询，SQLAlchemy也同样支持分组操作、排序和返回指定数量的结果。

比如我想要按照team_id进行分组，同时筛选分组后数据行数大于5的分组，并且按照分组后数据行数递增的顺序进行排序，显示team_id字段，以及每个分组的数据行数。那么代码如下：

```
from sqlalchemy import func
rows = session.query(Player.team_id, func.count(Player.player_id)).group_by(Player.team_id).having(func.count(Pl
print(rows)
```

运行结果：

```
[(1001, 20), (1002, 17)]
```

这里有几点需要注意：

1. 我们把需要显示的字段Player.team_id, func.count(Player.player_id)作为query的参数，其中我们需要用到sqlalchemy的func类，它提供了各种聚集函数，比如func.count函数。
2. 在query()后面使用了group_by()进行分组，参数设置为Player.team_id字段，再使用having对分组条件进行筛选，参数为func.count(Player.player_id)>5。
3. 使用order_by进行排序，参数为func.count(Player.player_id).asc()，也就是按照分组后的数据行数递增的顺序进行排序，最后使用.all()方法需要返回全部的数据。

你能看到SQLAlchemy使用的规则和使用SELECT语句的规则差不多，只是封装到了类中作为方法进行调用。

接着，我们再来看下如何删除数据。如果我们想要删除某些数据，需要先进行查询，然后再从session中把这些数据删除掉。

比如我们想要删除姓名为约翰·科林斯的球员，首先我们需要进行查询，然后从session对象中进行删除，最后进行commit提交，代码如下：

```
row = session.query(Player).filter(Player.player_name=='约翰-科林斯').first()
session.delete(row)
session.commit()
session.close()
```

需要说明的是，判断球员姓名是否为约翰·科林斯，这里需要使用（==）。

同样，如果我们想要修改某条数据，也需要进行查询，然后再进行修改。比如我想把球员索恩·马克的身高改成2.17，那么执行完之后直接对session对象进行commit操作，代码如下：

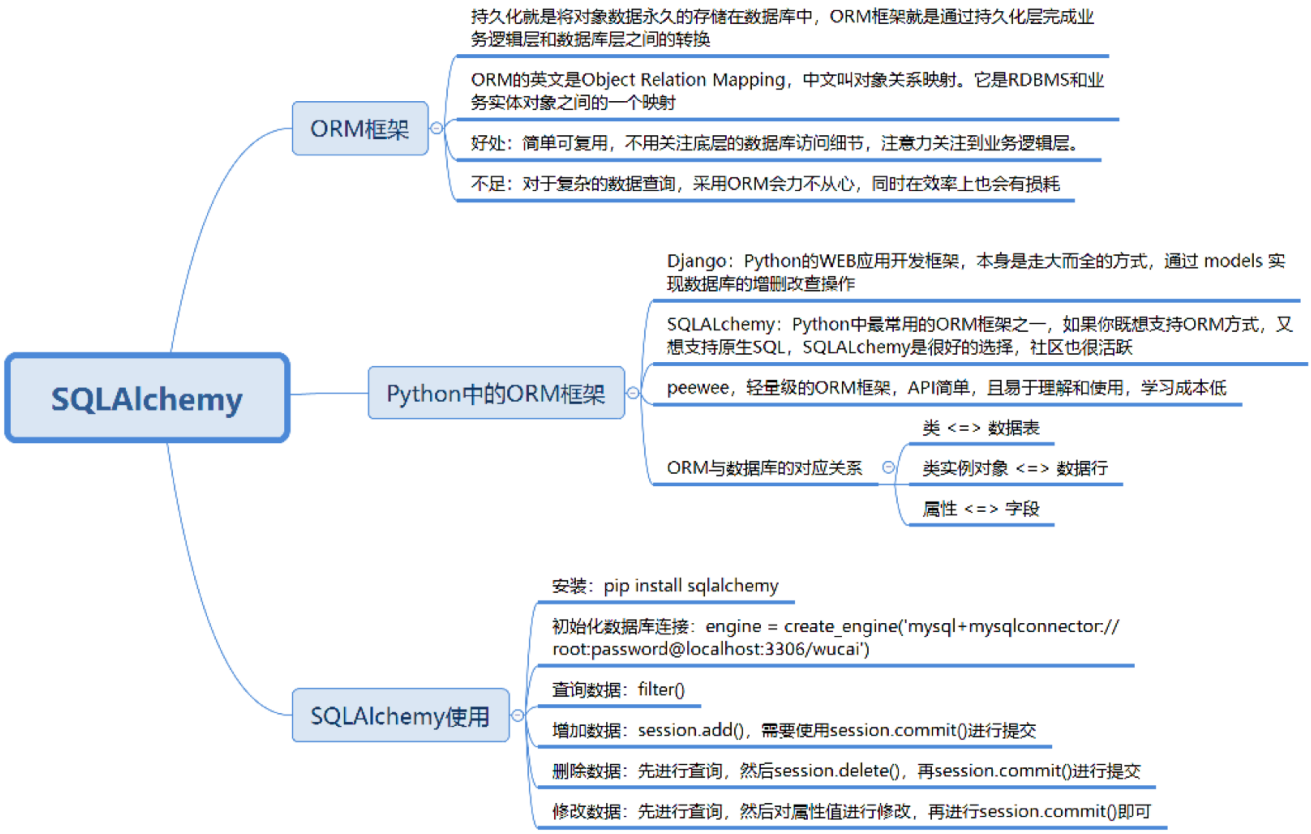
```
row = session.query(Player).filter(Player.player_name=='索恩-马克').first()
row.height = 2.17
session.commit()
session.close()
```

总结

今天我们使用SQLAlalchemy对MySQL进行了操作，你能看到这些实现并不复杂，只是需要事先掌握一些使用方法，尤其是如何创建seesion对象，以及如何通过session对象来完成对数据的增删改查等操作。建议你把文章里的代码都跑一遍，在运行的过程中一定会有更深入的体会。

当然除了学习掌握SQLAlalchemy这个Python ORM工具以外，我还希望你能了解到ORM的价值和不足。如果项目本身不大，那么自己动手写SQL语句会比较简单，你可以不使用ORM工具，

而是直接使用上节课讲到的mysql-connector。但是随着项目代码量的增加，为了在业务逻辑层与数据库底层进行松耦合，采用ORM框架是更加适合的。



我今天讲解了SQLAlalchemy工具的使用，为了更好地让你理解，我出一道练习题吧。还是针对player数据表，请你使用SQLAlalchemy工具查询身高为2.08米的球员，并且将这些球员的身高修改为2.09。

欢迎你在评论区写下你的答案，也欢迎把这篇文章分享给你的朋友或者同事，一起交流。

精选留言

ttttt

6

缺少一些代码，可以参考廖雪峰的这个。
https://www.liaoxuefeng.com/wiki/1016959663602400/1017803857459008

2019-07-22

墨禾

3

以下从ORM的作用，是什么，优缺点以及一些比较流行的ORM的对比的个人总结：

1.ORM的作用
对象关系映射，能够直接将数据库对象进行持久化。

在没有ORM前，我们要自己写数据库连接方法，自己在方法里面嵌入原生的sql语句去访问数据表.....

这时问题就来了：
数据库名，数据表名完全暴露在代码中，有脱库的风险；
需要我们自己处理数据表对象，比如说把数据表中取出的数据转化为标准json等，sql语句安全过滤，数据表、字段别名、兼容多种数据库等一系列的数据处理工作；

下面介绍一下ORM到底是啥？

2、ORM是什么？
ORM作为数据库层与业务逻辑层之间的一个抽象，能够将业务逻辑的处理持久化为内存对象，交由数据库去处理。其封装了数据库的连接，数据表的操作细节.....在文中我们可以看到ORM将sql语句做了封装，我们可以通过filter实现过滤，而不是写where子句。

ORM真的那么好？

3、优缺点
优点：
安全：因为屏蔽了数据库的具体操作细节以及对sql做了严格的过滤，因此能够保证数据库信息的隐蔽性，同时防止sql注入。
简单：屏蔽了数据层的访问细节，我们只需要集中注意力处理业务逻辑就可以了。

缺点：
性能低：自动化意味着加载很多即使没有必要的关联和映射，牺牲性能。但ORM也采取了一些补救措施：对象懒加载，缓存技术等。

学习成本高：面向对象的封装设计，是的我们必须要去了解对象的处理细节。

难以实现复杂查询：ORM实现的是一些通用的数据处理方法，一些负责的业务处理还是需要自己组装sql。

那么还有哪些比较流行的ORM呢？
hibernate:强调对单条数据的处理
mybits:基于自定义配置的sql操作

2019-07-23

ttttt 3

错误解决：
如果报如下错误：Authentication plugin 'caching_sha2_password' is not supported
sqlalchemy.exc.NotSupportedError: (mysql.connector.errors.NotSupportedError) Authentication plugin 'caching_sha2_password' is not supported (Background on this error at: http://sqlalche.me/e/tw8g)
可以参考下面的链接处理：
https://stackoverflow.com/questions/51783313/how-do-i-get-sqlalchemy-create-engine-with-mysqlconnector-to-connect-using-mysql

2019-07-22

一叶知秋 3

日常交作业~~~

```
# -*- coding:utf-8 -*-
from sqlalchemy import and_
```

```
from sqlalchemy import Column, INT, FLOAT, VARCHAR
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()


class Test_db:
def __init__(self):
"""此处填上自己的连接配置"""
self.engine = create_engine(
'mysql+pymysql://UserName:Password@host:port/Db_Name?charset=utf8')
db_session = sessionmaker(bind=self.engine)
self.session = db_session()

def update(self, target_class, query_filter, target_obj):
"""
更新操作通用方法
:param target_class: 表对象
:param query_filter: 查询条件
:param target_obj: 更新目标对象
:return:
"""
try:
self.session.query(target_class).filter(query_filter).update(target_obj)
self.session.commit()
self.session.close()
return True
except Exception as e:
print(e)


class Player(Base):
"""定义表结构"""
__tablename__ = 'player'
player_id = Column(INT(), primary_key=True)
team_id = Column(INT())
player_name = Column(VARCHAR(255))
height = Column(FLOAT())

def __init__(self, player_id, team_id, player_name, height):
self.player_id = player_id
```

```
self.team_id = team_id
self.player_name = player_name
self.height = height


if __name__ == '__main__':
db_obj = Test_db()
query_filter = and_(Player.height == 2.08)
target_obj = {'height': 2.09}
update_result = db_obj.update(Player, query_filter, target_obj)
```

后续更新数量、更新结果等等判断就略过了...
（小声bb：什么时候极客时间评论也能支持markdown啊。。）

2019-07-22

ABC

2

翻了一下SQLAlchemy的官方文档,看到一个简单的办法,作业如下:

'''

作业:

使用SQLAlchemy工具查询身高为2.08米的球员,并且将这些球员的身高修改为2.09;

参考:

https://docs.sqlalchemy.org/en/13/core/dml.html

'''

```
from sqlalchemy import Column, String, Integer, Float,create_engine,update
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
Base = declarative_base()
engine = create_engine('mysql+mysqlconnector://root:123456@localhost:3306/geektime-sql')


class Player(Base):
__tablename__ = 'player'

player_id = Column(Integer, primary_key=True, autoincrement=True)
```

```
team_id = Column(Integer)
player_name = Column(String(255))
height = Column(Float(3,2))

def to_dict(self):
return {c.name: getattr(self, c.name, None)
for c in self.__table__.columns}

if __name__ == '__main__':
DBSession = sessionmaker(bind=engine)
session = DBSession()
Base.to_dict = to_dict
print("更新前:")
rows = session.query(Player).filter(Player.height == 2.08).all()
print([row.to_dict() for row in rows])
# 参考: https://docs.sqlalchemy.org/en/13/core/dml.html#sqlalchemy.sql.expression.update
stmt = update(Player).where(Player.height == 2.08).values(height=2.09)
engine.execute(stmt)
session.commit()
rows = session.query(Player).filter(Player.height == 2.09).all()
print("更新后:")
print([row.to_dict() for row in rows])
session.close()
```

太长,省略了部分执行结果.自己执行一下,就可以看到完整结果了..

更新前:

[{'player_id': 10010, 'team_id': 1001, 'player_name': '乔恩-洛伊尔', 'height': Decimal('2.0800000000')}.....

更新后:

[{'player_id': 10010, 'team_id': 1001, 'player_name': '乔恩-洛伊尔', 'height': Decimal('2.0900000000')}.....

[Finished in 0.9s]

2019-07-22

ABC

2

文章中的示例代码.完整可运行.

```
from sqlalchemy import Column, String, Integer, Float,create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
Base = declarative_base()
# 初始化数据库连接，修改为你的数据库用户名和密码
engine = create_engine('mysql+mysqlconnector://root:123456@localhost:3306/geektime-sql')
```

```
# 定义 Player 对象:
class Player(Base):
# 表的名字:
__tablename__ = 'player'

# 表的结构:
player_id = Column(Integer, primary_key=True, autoincrement=True)
team_id = Column(Integer)
player_name = Column(String(255))
height = Column(Float(3,2))


# 增加 to_dict() 方法到 Base 类中
def to_dict(self):
return {c.name: getattr(self, c.name, None)
for c in self.__table__.columns}



if __name__ == '__main__':
# 创建 DBSession 类型:
DBSession = sessionmaker(bind=engine)
# 创建 session 对象:
session = DBSession()

# 创建 Player 对象:
new_player = Player(team_id = 1003, player_name = "约翰 - 科林斯", height = 2.08)
# 添加到 session:
session.add(new_player)
# 提交即保存到数据库:
session.commit()
# 关闭 session:
session.close()
# 将对象可以转化为 dict 类型
Base.to_dict = to_dict
# 查询身高 >=2.08 的球员有哪些
# rows = session.query(Player).filter(Player.height >=2.08, Player.height <=2.10).all()
# from sqlalchemy import or_
# rows = session.query(Player).filter(or_(Player.height >=2.08, Player.height <=2.10)).all()
rows = session.query(Player).filter(Player.height >= 2.08).all()
print([row.to_dict() for row in rows])
from sqlalchemy import func
```

```
rows = session.query(Player.team_id, func.count(Player.player_id)).group_by(Player.team_id).having(func.count(Player.player_id)>5).order_by(func.count(Player.player_id).asc()).all()
print(rows)
row = session.query(Player).filter(Player.player_name=='约翰 - 科林斯').first()
session.delete(row)
session.commit()
session.close()
row = session.query(Player).filter(Player.player_name=='索恩-马克').first()
row.height = 2.17
session.commit()
session.close()
```

2019-07-22

夜路破晓 2

框架对实体的映射不难理解,数据库本身就是对现实世界的映射,借由映射将事实转换为数据.
代码部分有些基础的也不难理解;基础较弱硬钢的亲们,耐心一条条来缕也可以捋顺,都是基础的东西,无非花费时间长短问题.有几个坑这里记录下,供后来人借鉴:
1.关于初始化连接数据库问题.creat_engine的参数这块容易卡壳,可以参考以下文字说明:
create_engine("数据库类型+数据库驱动://数据库用户名:数据库密码@IP地址:端口/数据库"，其他参数)
2.数据库驱动这块,老师的参考代码是用mysqlconnector,沿承得是上篇中导入mysql-connector包;网上一些资料以及参考其他同学的答案有使用pymysql,要用这个需安装pip install pymysql.这两货对于本篇的学习内容在本质上是一样的,任选一个即可.
3.在代码复写过程中,删除操作一直报错.网上查了资料说是跟返回值有关.经过测试,发现问题所在,filter返回结果为None.也就是说没有查询到"约翰-科林斯".往回倒腾,发现开始新增数据那里,增加的" 约翰-科林斯 ",前后对比后者两侧多了个空格.统一前后,删除操作顺利完成.

2019-07-22

Destroy、 2

```
rows = session.query(Player).filter(Player.height==2.08).all()
for row in rows:
row.height = 2.09
session.commit()
session.close()
```

2019-07-22

Geek_5d805b 1

问一下，我们已经创建过了player表，能说一下对于已有的数据库表，怎么直接将存储模型转换为数据模型吗，而不是再按字段新建

2019-07-25

ABC 1

另外,SQLAlchemy和MyBatis有点像，唯一不同的是MyBatis可以把SQL语句写在XML文件里面(当然也可以写在Java方法上)，SQLAlchemy好像只能用字符串方式(在官网暂时没找到其它方式的示例)来写SQL语句.

2019-07-22

许童童 1

老师你好，能否多讲一些ORM框架的缺点，以及为什么互联网项目大多不用ORM的原因？

2019-07-22

阿锋 1

上面那个分组查询，按照分组后数据行数递增的顺序进行排序，怎么结果是[(1001, 20), (1002, 17)]，那不是递减？是不是写错了？

2019-07-22

作者回复

rows = session.query(Player.team_id, func.count(Player.player_id)).group_by(Player.team_id).having(func.count(Player.player_id)>5).order_by(func.count(Player.player_id).asc()).all()

这里使用的是asc()，所以结果应该是：[(1002, 17), (1001, 20)]，你可以再check下order_by的部分

2019-07-22

大牛凯 1

老师好，对于修改数据的事例有一点困惑还请您解答。对于下面这段代码中

```
row = session.query(Player).filter(Player.player_name=='索恩 - 马克').first()
row.height = 2.17
session.commit()
session.close()
```

我理解row是存在于内存中的对象，但是我们在修改后并没有传递到数据库中，如果直接commit可以进行修改的话，这个row是在哪里存放的对象呢？

2019-07-22

作者回复

我们对数据表进行的增删改查实际上都是通过 session对象来完成的。这里row是定位session对象中想要查询的位置，然后对row.height进行了修改，也就是对session对象中那个查询位置的height进行了修改。这时数据表中的内容还没有更新，需要采用session.commit()来完成持久化。所以重点是我们修改了session对象的数据，然后进行了session.commit()

2019-07-22

jxs1211 0

老师，我的表中有个时间字段，我想在插入数据时，自动生成时间应怎么设置该字段，是这样吗：

`create_time` timestamp(0) NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',

2019-07-25

Geek_5d805b

0

to_dict方法这块看不太懂，base类指的是player类吗，谁给讲讲

2019-07-25

Nixus

0

ORM的使用，更多的不都是通过查文档的吗？

2019-07-24