**懒加载：**

在一对多，或者多对多关系的时候，如果想要获取多的一方这一部分的数据的时候，往往能通过一个属性就可以全部获取了。

如有一个作者，想要这个作者的所有文章，通过user.articles就可以获取所有的。

但有时候我们不想获取所有的数据，如只想获取这个作者今天发表的文章，那么这时候我们可以给relationship方法添加属性lazy='dynamic'，以后通过user.articles获取到的就不是一个列表，而是一个AppenderQuery对象了。这样就可以对这个对象再进行一层过滤和排序等操作。

通过`lazy='dynamic'`，获取出来的多的那一部分的数据，就是一个`AppenderQuery`对象了。这种对象既可以添加新数据，也可以跟`Query`一样，可以再进行一层过滤。

**lazy可用的选项：**

**1.** `select`：这个是默认选项。还是拿`user.articles`的例子来讲。如果你没有访问`user.articles`这个属性，那么sqlalchemy就不会从数据库中查找文章。一旦你访问了这个属性，那么sqlalchemy就会立马从数据库中查找所有的文章，并把查找出来的数据组装成一个列表返回。这也是懒加载。

**2.** `dynamic`：这个也是懒加载。就是在访问`user.articles`的时候返回来的不是一个列表，而是`AppenderQuery`对象。

总结：如果你在获取数据的时候，想要对多的那一边的数据再进行一层过滤，那么这时候就可以使用`lazy='dynamic'`。

**代码演示：**

|  |
| --- |
| **from** sqlalchemy **import** create\_engine,Column,Integer,Float,Boolean,DECIMAL,Enum,\  Date,DateTime,Time,String,Text,func,or\_,and\_,ForeignKey,Table  **from** sqlalchemy.dialects.mysql **import** LONGTEXT  **from** sqlalchemy.ext.declarative **import** declarative\_base  **from** sqlalchemy.orm **import** sessionmaker,relationship,backref  **import** random,time  **from** datetime **import** datetime  HOSTNAME = **'127.0.0.1'**  PORT = **'3306'**  DATABASE = **'first\_sqlalchemy'**  USERNAME = **'root'**  PASSWORD = **'root'**  DB\_URI =**"mysql+pymysql://{username}:{password}@{host}:{port}/{db}?charset=utf8"**.format(username=USERNAME,password=PASSWORD,host=HOSTNAME,port=PORT,db=DATABASE)  engine = create\_engine(DB\_URI)  Base = declarative\_base(engine)  session = sessionmaker(engine)()  **class** User(Base):  \_\_tablename\_\_ = **'user'**  id = Column(Integer, primary\_key=**True**, autoincrement=**True**)  uname = Column(String(50),nullable=**False**)  **class** Article(Base):  \_\_tablename\_\_ = **'article'**  id = Column(Integer, primary\_key=**True**, autoincrement=**True**)  title = Column(String(50), nullable=**False**)  create\_time = Column(DateTime, nullable=**False**, default=datetime.now)  uid = Column(Integer,ForeignKey(**"user.id"**))  # author = relationship("User", backref=backref("articles"))  #懒加载  author = relationship(**"User"**, backref=backref(**"articles"**,lazy=**"dynamic"**))  **def** \_\_repr\_\_(self):  **return "<Article(title:%s,create\_time:%s)>"** % (self.title,self.create\_time)  **def** add\_data():  Base.metadata.drop\_all()  Base.metadata.create\_all()  user = User(uname=**'莫莫'**)  **for** x **in** range(100):  article = Article(title=**"title %s"** % x)  article.author = user  session.add(article)  session.commit()  **from** sqlalchemy.orm.collections **import** InstrumentedList  **def** oper1():  user = session.query(User).first()  print(type(user.articles)) #<class 'sqlalchemy.orm.collections.InstrumentedList'>  print(user.articles)  #懒加载  **from** sqlalchemy.orm.dynamic **import** AppenderQuery  **def** oper2():  user = session.query(User).first()  print(type(user.articles)) #<class 'sqlalchemy.orm.dynamic.AppenderQuery'>  print(user.articles)  #辨析 AppenderQuery 和 Query  **from** sqlalchemy.orm.query **import** Query  **def** oper3():  user = session.query(User)  print(type(user)) #<class 'sqlalchemy.orm.query.Query'>  print(user) #sql语句  #有2层意思  #1.是一个Query对象。可以调用Query对象的方法  #2.是一个AppenderQuery对象。可以继续追加数据进去  **def** oper4():  user = session.query(User).first()#可以调用Query对象的方法  print(type(user))  print(user.articles.filter(Article.id>=50).all())  # article = Article(title='title 100')  # user.articles.append(article)#2.是一个AppenderQuery对象。可以继续追加数据进去  # session.commit()  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  # add\_data()  # oper1()  # oper2()  # oper3()  oper4() |