小白学Flask第十二天| flask-sqlalchemy数据库扩展包(二)

原创JAP君 Python进击者
2019-10-07原文

数据库的基本操作

今天整体的内容比较的简单，就是数据库的简单操作。大家只要记住这些语句就能够好好玩耍flask-sqlalchemy数据库了。

首先给大家一串主代码：

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy



app = Flask(__name__)
```

```
#设置连接数据库的URL
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql://root:mysql@127.0.0.1:3306/Flask_test'

#设置每次请求结束后会自动提交数据库中的改动
app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = True
#查询时会显示原始SQL语句
app.config['SQLALCHEMY_ECHO'] = True
db = SQLAlchemy(app)

class Role(db.Model):
    # 定义表名
    __tablename__ = 'roles'
    # 定义列对象
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)
    us = db.relationship('User', backref='role')

    #repr()方法显示一个可读字符串
    def __repr__(self):
        return 'Role:%s'% self.name
```

```
class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, index=True)
    email = db.Column(db.String(64),unique=True)
    pswd = db.Column(db.String(64))
    role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))

    def __repr__(self):
        return 'User:%s'%self.name
if __name__ == '__main__':
    db.drop_all()
    db.create_all()
    ro1 = Role(name='admin')
    ro2 = Role(name='user')
    db.session.add_all([ro1,ro2])
    db.session.commit()
    us1 =
User(name='wang',email='wang@163.com',pswd='123456',role_id=ro1.
id)
    us2 =
User(name='zhang',email='zhang@189.com',pswd='201512',role_id=ro
2.id)
```

```
    us3 =
User(name='chen',email='chen@126.com',pswd='987654',role_id=ro2.
id)
    us4 =
User(name='zhou',email='zhou@163.com',pswd='456789',role_id=ro1.
id)
    db.session.add_all([us1,us2,us3,us4])
    db.session.commit()
    app.run(debug=True)
```

（以下代码都参考主代码）

1.创建表：

```
db.create_all()
```

我们需要让flask-sqlalchemy数据库根据模型类创建相应的数据库，db.create_all()函数将寻找所以db.Model的子类。

大家可以看到主代码当中我们所创建的两个模型类就是继承自db.Model。

2.删除表：

```
    db.drop_all()
    db.create_all()
```

为什么我要写两行代码呢？因为删除表这种操作只能在你第一次创建表时使用，不然随意使用删除表，你就等着被炒鱿鱼吧。

如果想要更新现有数据库表的结构，可以先删除旧表再重新创建。

3.插入一条数据：

```
ro1 = Role(name='admin')
db.session.add(ro1)
db.session.commit()
# 再 次 插 入 一 条 数 据
ro2 = Role(name='user')
db.session.add(ro2)
db.session.commit()
```

可以看到想要插入一条数据，只需要去创建一个模型类的实例，然后通过db.session.add(实例名称)来进行插入即可，非常的形象简单。

当然，如果我们有一百万条数据，一条一条的插入，那岂不是得插到猴年马月？所以这里也有多条数据同时插入的方法。

4.一次插入多条数据：

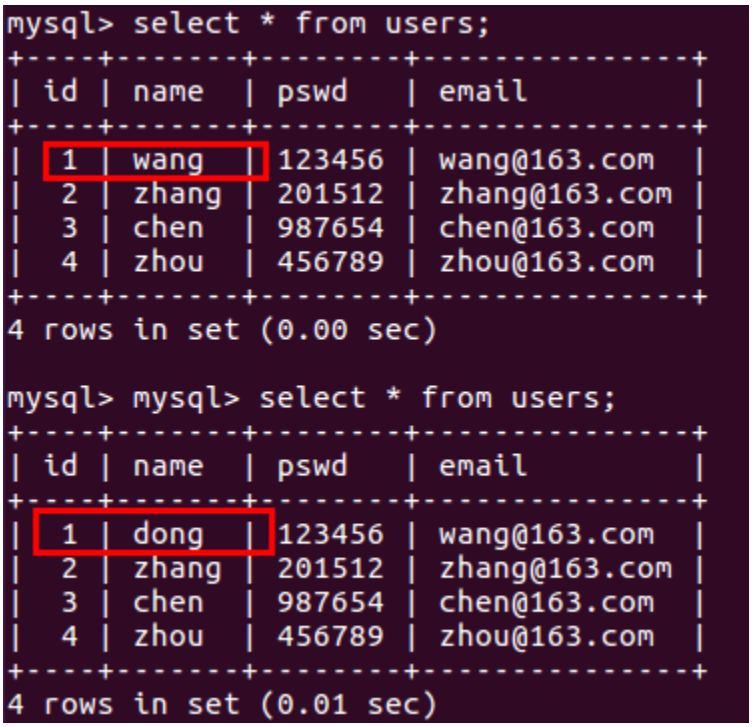
```
us1 =
User(name='wang',email='wang@163.com',pswd='123456',role_id=ro1.id)
us2 =
User(name='zhang',email='zhang@189.com',pswd='201512',role_id=ro2.i
d)
us3 =
User(name='chen',email='chen@126.com',pswd='987654',role_id=ro2.id)
us4 =
User(name='zhou',email='zhou@163.com',pswd='456789',role_id=ro1.id)
```

```
db.session.add_all([us1,us2,us3,us4])
db.session.commit()
```

通过把模型类实例放入到list当中，然后用add_all函数插入。

5.更新数据：

```
user                        =                       User.query.first()
user.name                               =                           'dong'
db.session.commit()
User.query.first()
```



大家可以从表中体会到上面的语句是啥意思了。

6.删除数据：

```
user                        =                        User.query.first()
db.session.delete(user)
db.session.commit()
```

关于查询数据，这里有几个知识点：

7.精准查询

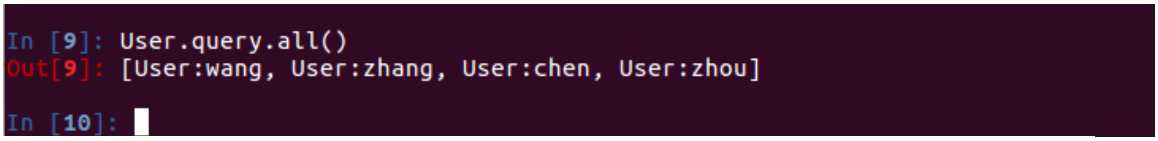
返回名字等于wang的所有人：

```
User.query.filter_by(name='wang').all()
```

返回查询到的第一个对象：

```
User.query.first()
```

返回查询到的所有对象：




```
User.query.all()
```

filter模糊查询，返回名字结尾字符为g的所有数据：

```
In [12]: User.query.filter(User.name.endswith('g')).all()
Out[12]: [User:wang, User:zhang]

In [13]:
```

```
User.query.filter(User.name.endswith('g')).all()
```

get()，参数为主键，如果主键不存在没有返回内容：

```
User.query.get()
```

逻辑非，返回名字不等于wang的所有数据：

```
In [13]: User.query.filter(User.name!='wang').all()
Out[13]: [User:zhang, User:chen, User:zhou]

In [14]:
```

```
User.query.filter(User.name!='wang').all()
```

逻辑与，需要导入and，返回and()条件满足的所有数据：

```
In [4]: User.query.filter(and_(User.name!='wang',User.email.endswith('163.com'))).all()
Out[4]: [User:zhou]

In [5]:
```

```
from sqlalchemy import and_
User.query.filter(and_(User.name!='wang',User.email.endswith('163.com'))).all()
```

逻辑或，需要导入or_：

```
In [8]: User.query.filter(or_(User.name!='wang',User.email.endswith('163.com'))).all()
Out[8]: [User:wang, User:zhang, User:chen, User:zhou]
```

```
from sqlalchemy import or_
User.query.filter(or_(User.name!='wang',User.email.endswith('163.com'))).all()
```

not_ 相当于取反：

```
In [13]: User.query.filter(not_(User.name=='chen')).all()
Out[13]: [User:wang, User:zhang, User:zhou]

In [14]:
```

```
from sqlalchemy import not_
User.query.filter(not_(User.name=='chen')).all()
```

点再看、转发、赞赏。就是给我最大的动力！

Flask系列文章：

持续更新中...

精选留言

暂无...