使用SQLAlchemy

502次阅读

数据库表是一个二维表，包含多行多列。把一个表的内容用Python的数据结构表示出来的话，可以用一个list表示多行，list的每一个元素是tuple，表示一行记录，比如，包含id和name的user表：

```
[
    ('1', 'Michael'),
    ('2', 'Bob'),
    ('3', 'Adam')
]
```

Python的DB-API返回的数据结构就是像上面这样表示的。

但是用tuple表示一行很难看出表的结构。如果把一个tuple用class实例来表示，就可以更容易地看出表的结构来：

```
class User(object):
    def __init__(self, id, name):
        self.id = id
        self.name = name

[
    User('1', 'Michael'),
    User('2', 'Bob'),
    User('3', 'Adam')
]
```

这就是传说中的ORM技术：Object-Relational Mapping，把关系数据库的表结构映射到对象上。是不是很简单？

但是由谁来做这个转换呢？所以ORM框架应运而生。

在Python中，最有名的ORM框架是SQLAlchemy。我们来看看SQLAlchemy的用法。

首先通过easy_install或者pip安装SQLAlchemy：

```
$ easy_install sqlalchemy
```

然后，利用上次我们在MySQL的test数据库中创建的user表，用SQLAlchemy来试试：

第一步，导入SQLAlchemy，并初始化DBSession：

```
# 导入:
from sqlalchemy import Column, String, create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# 创建对象的基类:
Base = declarative_base()

# 定义User对象:
class User(Base):
    # 表的名字:
    __tablename__ = 'user'

    # 表的结构:
```

```
    id = Column(String(20), primary_key=True)
    name = Column(String(20))

# 初始化数据库连接:
engine = create_engine('mysql+mysqlconnector://root:password@localhost:3306/test')
# 创建DBSession类型:
DBSession = sessionmaker(bind=engine)
```

以上代码完成SQLAlchemy的初始化和具体每个表的class定义。如果有多个表，就继续定义其他class，例如School：

```
class School(Base):
    __tablename__ = 'school'
    id = ...
    name = ...
```

create_engine()用来初始化数据库连接。SQLAlchemy用一个字符串表示连接信息：

```
'数据库类型+数据库驱动名称://用户名:口令@机器地址:端口号/数据库名'
```

你只需要根据需要替换掉用户名、口令等信息即可。

下面，我们看看如何向数据库表中添加一行记录。

由于有了ORM，我们向数据库表中添加一行记录，可以视为添加一个User对象：

```
# 创建session对象:
session = DBSession()
# 创建新User对象:
new_user = User(id='5', name='Bob')
# 添加到session:
session.add(new_user)
# 提交即保存到数据库:
session.commit()
# 关闭session:
session.close()
```

可见，关键是获取session，然后把对象添加到session，最后提交并关闭。Session对象可视为当前数据库连接。

如何从数据库表中查询数据呢？有了ORM，查询出来的可以不再是tuple，而是User对象。SQLAlchemy提供的查询接口如下：

```
# 创建Session:
session = DBSession()
# 创建Query查询，filter是where条件，最后调用one()返回唯一行，如果调用all()则返回所有行:
user = session.query(User).filter(User.id=='5').one()
# 打印类型和对象的name属性:
print 'type:', type(user)
print 'name:', user.name
# 关闭Session:
session.close()
```

运行结果如下：

```
type: <class '__main__.User'>
name: Bob
```

可见，ORM就是把数据库表的行与相应的对象建立关联，互相转换。

由于关系数据库的多个表还可以用外键实现一对多、多对多等关联，相应地，ORM框架也可以

提供两个对象之间的一对多、多对多等功能。

例如，如果一个User拥有多个Book，就可以定义一对多关系如下：

```
class User(Base):
    __tablename__ = 'user'

    id = Column(String(20), primary_key=True)
    name = Column(String(20))
    # 一对多:
    books = relationship('Book')

class Book(Base):
    __tablename__ = 'book'

    id = Column(String(20), primary_key=True)
    name = Column(String(20))
    # “多”的一方的book表是通过外键关联到user表的:
    user_id = Column(String(20), ForeignKey('user.id'))
```

当我们查询一个User对象时，该对象的books属性将返回一个包含若干个Book对象的list。

小结

ORM框架的作用就是把数据库表的一行记录与一个对象互相做自动转换。

正确使用ORM的前提是了解关系数据库的原理。
