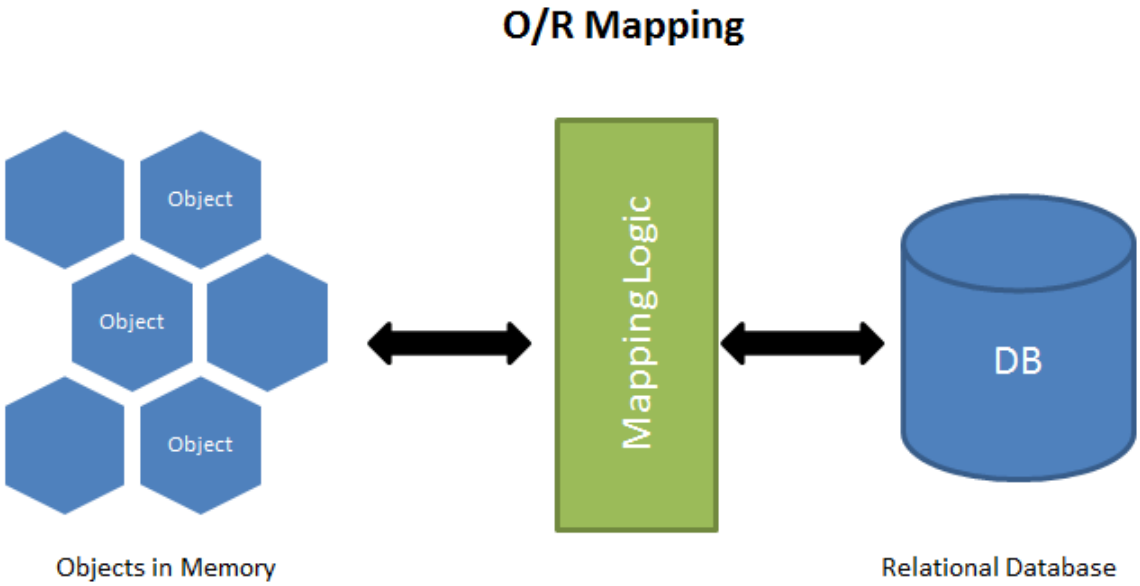
数据库与模板系统

> **Organization**: 千锋教育 Python 教学部
>
> **Date** : 2019-09-08
>
> **Author**: 张旭

一、ORM: 对象关系映射



1. 概述

ORM 全称是：Object Relational Mapping (对象关系映射)。其主要作用是在编程中把面向对象的概念跟数据库中表的概念对应起来。

举例来说就是，我定义一个类，那就对应着一张表，这个类的实例，就对应着表中的一条记录。

面向对象编程把所有实体看成对象（object），关系型数据库则是采用实体之间的关系（relation）连接数据。
很早就有人提出，关系也可以用对象表达，这样的话，就能使用面向对象编程，来操作关系型数据库。

ORM 的优点：

- 数据模型都在一个地方定义，更容易更新和维护，也利于重用代码。
- ORM 有现成的工具，很多功能都可以自动完成，比如数据预处理、事务等等。
- 它迫使你使用 MVC 架构，ORM 就是天然的 Model，最终使代码更清晰。
- 基于 ORM 的业务代码比较简单，代码量少，语义性好，容易理解。
- 你不必编写性能不佳的 SQL。

Python下常用的 ORM 有: Django-ORM、SQLAlchemy、Peewee 等

2. 示例

我们这里使用 SQLAlchemy 来操作数据库，直接看代码:

```
import datetime

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy import Column, String, Integer, Float, Date
from sqlalchemy.ext.declarative import declarative_base

# 建立连接与数据库的连接
engine = create_engine('mysql+pymysql://seamile:54188@localhost:3306/sh1905')
Base = declarative_base(bind=engine)  # 创建模型的基础类
Session = sessionmaker(bind=engine)


class User(Base):
    '''User 模型'''
    __tablename__ = 'user'  # 该模型对应的表名

    id = Column(Integer, primary_key=True)  # 定义 id 字段
    name = Column(String(20), unique=True)  # 定义姓名字段
    birthday = Column(Date)                 # 定义生日字段
    money = Column(Float, default=0.0)      # 定义金钱字段


Base.metadata.create_all(checkfirst=True)  # 创建表

session = Session()
# 定义一些对象
bob = User(name='bob', birthday=datetime.date(1990, 3, 21), money=234)
tom = User(name='tom', birthday=datetime.date(1995, 9, 12))
lucy = User(name='lucy', birthday=datetime.date(1998, 5, 14), money=300)
jam = User(name='jam', birthday=datetime.date(1994, 3, 9), money=58)
alex = User(name='alex', birthday=datetime.date(1992, 3, 17), money=99)
eva = User(name='eva', birthday=datetime.date(1987, 7, 28), money=175)
rob = User(name='rob', birthday=datetime.date(1974, 2, 5), money=274)
ella = User(name='ella', birthday=datetime.date(1999, 5, 26), money=394)

# 增加数据
session.add_all([bob, tom, lucy, jam])  # 在 Session 中记录操作
session.commit()                        # 提交到数据库中执行

# 删除数据
session.delete(jam)  # 记录删除操作
session.commit()     # 提交到数据库中执行

# 修改数据
```

```
tom.money = 270    # 修改数据
session.commit()  # 提交到数据库中执行

# 查询数据
u_query = session.query(User)  # 先定义表的查询对象

# 直接获取主键(ID)为 5 的数据
user = u_query.get(5)
print(user.id, user.name)

# 使用 filter_by 按条件查询
user = u_query.filter_by(id=7).one()
print(user.id, user.name, user.birthday)

# 使用 filter 进行范围查询，并对结果进行排序
users = u_query.filter(User.id>2).order_by('birthday')
for u in users:
    print(u.name, u.birthday, u.money)

# 根据查询结果进行更新
users.update({'money': User.money - 1}, synchronize_session=False)
sessiom.commit()

# 按数量取出数据：limit / offset
users = u_query.limit(3).offset(4)
for u in users:
    print(u.id, u.name)

# 计数
num = u_query.filter(User.money>200).count()
print(num)

# 检查是否存在
exists = q.filter_by(name='Seamile').exists()
result = session.query(exists).scalar()
print(result)
```

二、Tornado 的模板系统

模板系统是为了更快速、更方便的生产大量的页面而设计的一套程序。

借助模板系统，我们可以先写好页面大概的样子，然后预留好数据的位置，再然后将我们需要的数据，按照既定规则拼到模板中的指定位置，然后渲染出完整页面。

现代的模板系统已经相当成熟，甚至可以通过 `if...else`、`for` 等语句在模板中写出简单的逻辑控制。

1. 模版与静态文件的路径配置

定义 app 时，在 Application 中定义, 可以是想对路径, 也可以是绝对路径

```
app = Application(
    template_path = 'templates',  # 模版路径
    static_path = 'statics'         # 静态文件路径
)
```

2. 模板中的变量

在模板中，变量和表达式使用 `{{ ... }}` 包围，可以写入任何的 Python 表达式或者变量

```
<!DOCTYPE html>
<html lang='en'>
    <head>
        <meta charset='UTF-8'>
        <title>Templates</title>
    </head>

    <body>
        <div>你好 {{ name }}，欢迎回来！ </div>
        <div>猜一猜，3 x 2 等于几? </div>
        <div>我就不告诉你等于 {{ 3 * 2 }}</div>
    </body>
</html>
```

3. 从 Python 程序中传递参数

```
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        name = 'Tom'
        say = "Hello, world"
        self.render('index.html', name=name, say=say)
```

4. 模板中的 `if...else` 结构

模板中的控制语句使用 `{% ... %}` 包围，如下所示

```
<p>
    根据您的条件我们进行了筛选
    {% if 条件 %}
        <div>第 1 条数据</div>
        <div>第 2 条数据</div>
        <div>。。。</div>
    {% else %}
        <div>抱歉我们没有找到合适的内容</div>
    {% end %}
</p>
```

5. 模板中的 `for` 循环

Python 程序中

```
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        students = ["Lucy", "Tom", "Bob"]
        self.render("student.html", students=students)
```

页面中：

```
<html>
  <head>
    <title>学生信息</title>
  </head>
  <body>
    <ul>
      {% for student in students %}
          <li>{{ student }}</li>
      {% end %}
    </ul>
  </body>
</html>
```

6. 静态文件

1. 先参照第 1 小结添加静态文件的路径配置
2. 页面中静态文件的路径需要以 '/static/' 开头, 后面跟文件路径

   比如，目录结构如下:

```
├── main.py
├── statics
|   └── img
|       └── coder.jpg
└── templates
    └── index.html
```

在模板中使用静态文件时，静态文件的路径这样写:

```
<img class="avatar" src="/static/img/coder.jpg" />
```

7. 模板的继承

网站中，大多数页面都是同样的结构和风格，我们没有必要在所有页面中把相同的样式重复的写很多遍。

Tornado 为我们提供了模板的继承机制，只需要写好父模板，然后让其他模板继承一下即可

父模板文件名经常定义为 "base.html"

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %} 通用标题 {% end %}</title>
</head>
<body>
    {% block body %}{% end %}
</body>
</html>
```

子模板:

```
{% extends "base.html" %}

{% block title %} 子页面的标题 {% end %}

{% block body %}
<div>
    子页面里的内容
    Bala Bala
</div>
{% end %}
```

三、练习

利用 SQLAlchemy 重新创建昨天的表结构，实现下面接口

1. 开发接口，并根据用户传入的 id 显示对应的用户信息
2. 开发接口，并根据用户传入的数值修改用户数据
3. 整体数据在使用模版展现出来，边栏显示学生姓名，点击边栏链接可以显示相应学生信息