小白学Django第三天| 一文带你快速理解模型Model

原创 JAP君 Python进击者
2020-01-14原文

小白学Django系列：

- 小白学Django第一天| MVC、MVT以及Django的那些事
- 小白学Django第二天| Django原来是这么玩的！
- 日更中...

用最短的时间学最多的知识，本文大约花费7分钟

本文内容：

1. ORM
2. 模型类的设计和表的生成
3. 通过模型类操作数据表
4. 模型类关系和关系查询

1. ORM

在如今很多的框架中，ORM已经应用的非常的广泛，**什么是ORM呢？**

ORM 全拼**Object-Relation Mapping**.

中文意思：**对象-关系 映射**

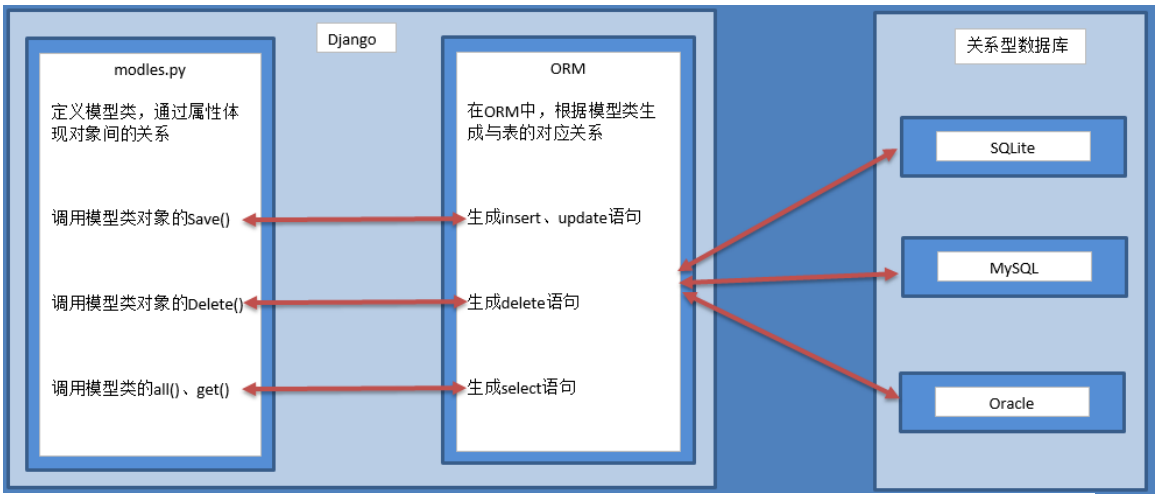
在我们所学的Django中的MVC或MVT中的M就采用了ORM。

它的作用是**实现模型对象到关系型数据库数据的映射**

比如把数据库中每条记录映射为一个模型对象：

id	name	age
1	张三	39
2	李四	40
3	王五	19

图解ORM：



很明显采用ORM模型，**有着很多的优点**。它把面向数据库的编写代码转换成面向对象的编写，而且各种数据操作都转化成类中方法和属性的方法。除此之外，我们不用去写复杂的SQL语句。

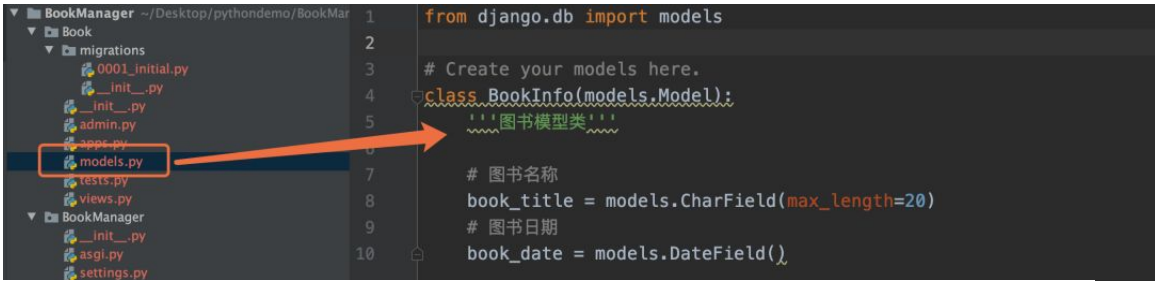
当然除了上面的优点，这种面向对象的写法也让我们忽略了数据库的类型，无论是MYSQL、oracle都可以使用相同的方式，并且我们只需要修改配置文件即可切换数据库类型，不需要改动其他代码。

当然，**缺点也有**。通过这种面向对象的编程难免会比直接SQL语句的性能差一些，这个性能的差值主要是在映射的过程中丧失的。

2. 模型类的设计和表的生成

了解了ORM的含义，我们来体验一下Django框架中是如何具体运用的：

首先编写一个模型类



这里我们有book_title和book_date两个属性。

大家可能会对models的一些方法有疑惑，这里给大家总结了

模型类定义属性：

```
# 书籍信息模型
class BookInfo(models.Model):
    name = models.CharField(max_length=20) #图书名称
```

- 总结语法：属性名 = models.字段类型(选项)
  - 定义属性时需要指定字段类型，通过字段类型的参数指定选项

属性名相关注意事项：

- 不允许使用python的保留关键字
- 不允许使用mysql的保留关键字
- 不允许使用连续的下划线，因为Django的查询语法就是连续的下划线

字段类型

提示：Django根据属性的类型确定以下信息：

- 当前选择的数据库支持字段的类型
- 渲染管理表单时使用的默认html控件
- 在管理站点最低限度的验证
- 使用时需要引入from django.db import models包

- AutoField：自动增长的IntegerField，通常不用指定

- 不指定时Django会自动创建属性名为id的自动增长属性

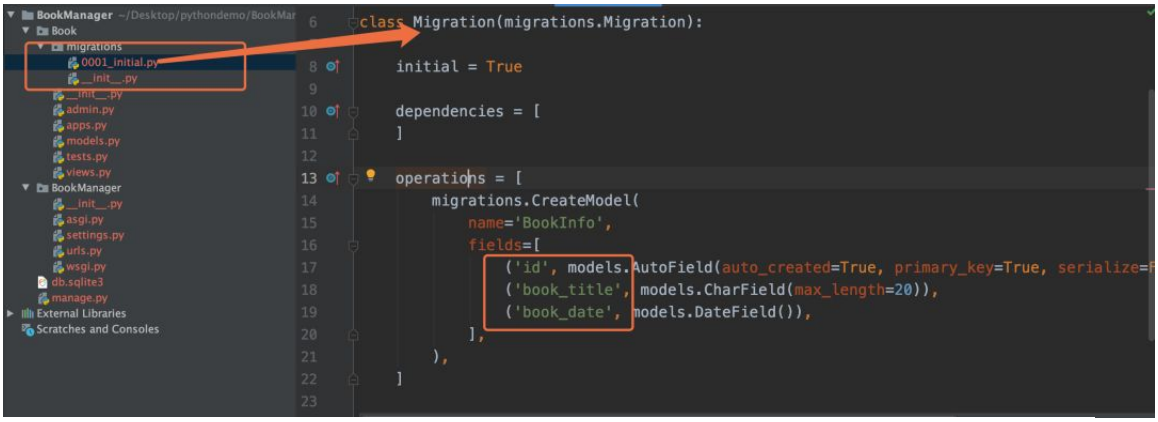
- BooleanField：布尔字段，值为True或False
- NullBooleanField：支持Null、True、False三种值
- CharField(max_length=字符长度)：字符串
  - 参数max_length表示最大字符个数
- TextField：大文本字段，一般超过4000个字符时使用
- IntegerField：整数
- DecimalField(max_digits=None, decimal_places=None)：可以指定精度的十进制浮点数
  - 参数max_digits表示总位数
  - 参数decimal_places表示小数位数
- FloatField：浮点数
- DateField[auto_now=False, auto_now_add=False])：日期
  - 参数auto_now表示每次保存对象时，自动设置该字段为当前时间，用于"最后一次修改"的时间戳，它总是使用当前日期，默认为false
  - 参数auto_now_add表示当对象第一次被创建时自动设置当前时间，用于创建的时间戳，它总是使用当前日期，默认为false
  - 参数auto_now_add和auto_now是相互排斥的，组合将会发生错误
- TimeField：时间，参数同DateField
- DateTimeField：日期时间，参数同DateField
- FileField：上传文件字段
- ImageField：继承于FileField，对上传的内容进行校验，确保是有效的图片

编写完我们的模型类之后，我们需要生成迁移文件：

```
python manage.py makemigrations
```

```
kulsdeMacBook-Pro:BookManager kuls$ python manage.py makemigrations
Migrations for 'Book':
  Book/migrations/0001_initial.py
    - Create model BookInfo
```

执行完后，我们会发现migration文件夹中多出了几个文件：



并且在图中我标记的文件里有着迁移过后所产生我们刚写的模型类所对应的迁移类。

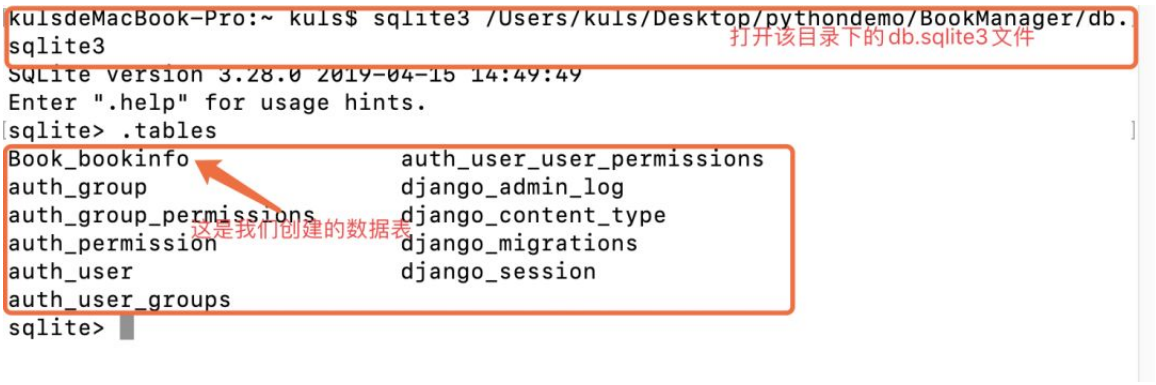
生成了迁移文件之后，我们还需要执行迁移文件，这样才会和我们的数据库映射起来（Django默认配置sqlite数据库，所以我们暂时使用sqlite，之后我们会配置成mysql）

执行迁移文件生成表：

python manage.py migrate

```
kulsdeMacBook-Pro:BookManager kuls$ python manage.py migrate
Operations to perform:
  Apply all migrations: Book, admin, auth, contenttypes, sessions
Running migrations:
  Applying Book.0001_initial... OK
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
kulsdeMacBook-Pro:BookManager kuls$
```

执行完成后，我们会发现我们的项目目录下产生了一个db.sqlite3的文件，这个文件就是sqlite数据库文件，我们通过命令行把这个文件打开



通过查看表的列属性

```
sqlite> PRAGMA table_info(Book_bookinfo);
cid|name|type|notnull|dflt_value|pk
0|id|integer|1||1
1|book_title|varchar(20)|1||0
2|book_date|date|1||0
sqlite>
```

可以发现成功创建了我们模型类相对应的表。

3.通过模型类操作数据表

我们上面模型类和数据表都创建完了，**接下来我们该怎么去添加、修改表中的数据呢？**

早在最前面就跟大家说了ORM，也就是说我们可以直接通过对象来对数据库中的数据进行操作。

我们这里进入项目的shell命令中来给大家讲解操作数据库

执行完毕以后，我们去查看我们的sqlite数据库

	id	book_title	book_date
1	1	Django从入门到放弃	2020-01-01

可以看到我们通过对象储存的数据已经保存至数据库了。

（有关于操作数据库，知识点比较多，我会专门写一篇文章来详细说明）

4.模型类关系和关系查询

```
class BookInfo(models.Model):
    '''图书模型类'''

    # 图书名称
    book_title = models.CharField(max_length=20)
    # 图书日期
    book_date = models.DateField()


class People(models.Model):
    '''人物模型类'''
    name = models.CharField(max_length=20)

    age = models.IntegerField()
    # 设置外键,
    # 在django2.0后，定义外键和一对一关系的时候需要加on_delete选项，此参数为了避免两个表里的数据不一致问题，
    # 不然会报错:
    # TypeError: __init__() missing 1 required positional argument: 'on_delete'
    book_people = models.ForeignKey('BookInfo',on_delete=models.CASCADE)
```

大家可以看到上图，我们在之前的模型类BookInfo基础上，加了一个People模型类，因为在一本书中可能会出现很多人物。由此我们会有一个一对多的关系，这种关系是怎么建立的，大家可以看到最后一句代码 **ForeignKey()**。

我们编写好两个模型类后，同样的步骤给People生成迁移文件，执行迁移文件创建表。

为了给大家演示，我们再次进入shell当中操作：

```
>>> from Book.models import BookInfo,People
>>> b = BookInfo()
>>> b = "西游记"
>>> from Book.models import BookInfo,People
>>> b = BookInfo()
>>> b.book_title = "西游记"
>>> from datetime import date
>>> b.book_date = date(1986,1,1)
>>> b.save()
>>> p = People()
>>> p.name = "孙悟空"
>>> p.age = 25
>>> p.book_people = b
>>> p.save()
```

执行完后，我们来看看两个表之间的数据

	id	book_title	book_date
1	1	Django从入门到放弃	2020-01-01
2	2	西游记	1986-01-01

	id	name	age	book_people_id
1	1	孙悟空	25	2

可以看到，这两张表已经连接起来了

既然已经连接起来，我们自然是可以互相访问数据了

通过人物查找书籍：

```
>>> p.book_people.book_title
'西游记'
```

查找书籍中的所有人物：

```
>>> b.people_set.all()
<QuerySet [<People: People object (1)>]>
```

公众号发送“领取资料”

可以领取我给你们最新整理的自学资料

原创不易，在看、转发！

往期系列：

Flask基础系列文章大全【JAVAandPython君出品】

精选留言

暂无...