Django ORM进阶应用

本教程的《Django ORM模块精讲》一节中，我们对 Django 的 ORM 模块进行了详细的介绍，那么 ORM 框架如何配合项目使用呢？本章将围绕 index 应用，首先创建应用所需要的 Models 数据表，之后介绍 ORM API 的相关使用方法，最后讲解如何通过 admin 后台管理系统实现对数据表的增删改查。

由于每一个数据表对应一个 Model 定义，每一个 Model 都是一个 Python 类，所以，Model 之间是可以继承的。Django 规定，所有的 Model 都必须继承自：

django.db.models.Model
可以直接继承，也可以间接继承。Model 中的所有字段都是 `django.db.models.Field` 的子类，Django 会根据 Field 的类型确定数据库表的字段类型。Django 内置了数十种 Field 字段类型。

1. ORM定义项目数据表

下面我们用 ORM 定义 index 应用所需的数据表，首选找到 index 应用下的 models.py 文件，在文件里添加如下代码：

```
class Book(models.Model):  #创建 book 表
    title=models.CharField( max_length=30,unique=True, verbose_name='书名')
    public=models.CharField(max_length=50,verbose_name='出版社')
    price=models.DecimalField(max_digits=7,decimal_places=2,verbose_name='定价')
    def default_price(self):
         return '￥30'
    retail_price=models.DecimalField(max_digits=7,decimal_places=2,verbose_name='零售价',default=default_price)

    def __str__(self):
        return "title:%s pub:%s price:%s" % (self.title, self.public, self.price)

class Author(models.Model): #创建作者表
    name=models.CharField(max_length=30,verbose_name='姓名')
    email=models.EmailField(verbose_name='邮箱')

    def __str__(self):
        return '作者：%s'%(self.name)

class UserInfo(models.Model): #创建用户信息表
    username=models.CharField(max_length=24,verbose_name='用户注册')
    password =models.CharField(max_length=24,verbose_name='密码')
```

通过上述代码，我们定义了一个名叫 Book 的数据表。数据表由以下字段构成书名（title）、出版社（public）、价格（price）、零售价（retail_price），而且对每个字段都做添加了相应的字段属性以及字段选项。

在《Django ORM模块精讲》一节中，我们对常用的字段属性做了介绍，那什么是字段选项呢？它是对字段的进一步说明，也就是字段附加参数信息。允许出现多个字段选项，多个字段选项之间使用“,”隔开即可，如上所示。

2. Filed的通用字段选项

Model 中添加的字段都是 Field 类型的实例，不同的 Field 类型可能会支持不同的字段选项，但是也有很多字段选项是通用的，即可以用在任何一种 Field 类型中。这里介绍一些常用且重要的通用字段选项，它们都有对应的默认值，这些字段选项都是可选的，理解这些有助于更好地使用它们。

1) blank

默认值是 False，设置为 True 时，字段可以为空。设置为 False 时，字段是必须填写的。如果是字符型字段 CharField 和 TextField，它们是用空字符串来存储空值的。

2) unique

默认值是 False，它是一个数据库级别的选项，规定该字段在表中必须是唯一的。

提示：数据库层面对待对待唯一性约束会创建唯一性索引，所以，如果一个字段设置了 unique=True，就不需要对这个字段加上索引选项了。

3) null

默认为 False，如果此选项为 False 建议加入 default 选项来设置默认值。如果设置为 True，表示该列值允许为空。日期型、时间型以及数字型字段不接受空字符串。所以当设置 IntegerField，DateTimeField 型字段可以为空时，需要将 blank 与 null 均设为 True 才可以。

提示：对于 CharFiled 和 TextFiled 这样的字符串类型，null 字段应该设置为 False，如果为 Ture，对于空数据就会有两种概念。

4) db_index

默认值是 False，如果设置为 True，Django 则会为该字段创建数据库索引，如果该字段经常作为查询的条件，那么就需要设置 db_index 选项，从而加快数据的检索速度。

5）db_column

这个选项用于设置数据库表字段的名称。如果没有指定，Django 默认使用 Model 中字段的名字。

6) default

用于给字段设置默认值，该选项可以设置为一个值或者是可以调用对象，但不能是可变对象，不同字段类型默认值也不同，比如 BooleanFiled 布尔类型 default 值为Ture 或者 False。主要的使用场景是当一个字段的值被用户省略时，后台服务器自动为该字段的设置默认值。

7) primary_key

默认值是 False，如果设置为 True，表示该字段为主键，在 Django 中 默认 id 为主键，也就是说即使你的数据表中没有创建 id 字段，Django 也会自动为你创建 id 字段并将其设置为主键。如果你在表中设置了其他字段为主键的时，那么 Django 将取消为 id 字段设置主键。

8) choices

这个选项用于给字段设置可以选择的值。它是一个可迭代对象，即列表或者元组，其中每一个元素都是一个二元组（a，b）的形式，a 是用来选择的对象，b 是对 a 的描述信息。比如我们对某个人性别定义数据表如下所示：

```
# 创建表
class UserInfo(models.Model):
    # 定义chocies参数的对应关系，以元组（或者列表）的形式进行表述：
    choices = (
        (male, '男性'),
        (female, '女性'),
    )
    gender = models.CharField(max_length=2,choices = choices,default='male')
```

9) verbose_name

设置此字段在 admin 后台管理系统界面上的显示名称，如果没有设置这个字段，Django 将会直接展示字段名并且将字段中的下划线转变为空格。

3. index应用数据库迁移

上面我们创建好了 index 应用所需的数据表，下一步就是执行数据库的迁移，之后让我们再来看一下又有什么新的变化发生呢？数据库迁移的两个命令分步骤执行，如下所示：

```
python manage.py makemigrations
python manage.py migrate
```

执行完毕后会在 CMD 命令行得到如下输出：

```
C:¥Users¥Administrator¥Book¥BookStore>python manage.py makemigrations
Migrations for 'index':
  index¥migrations¥0001_initial.py
    - Create model Author
    - Create model Book
    - Create model UserInfo

C:¥Users¥Administrator¥Book¥BookStore>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, index, sessions
Running migrations:
  Applying index.0001_initial... OK
```

若迁移过程中出现报错提示，首先检查您的 models.py 文件是否正确书写，除此之外，也可能由于 Django 与 MySQL 版本问题导致报错，此时需要更改 Django 源码包，请参考如下文章《Django数据同步问题解决》。

从上述输出结果可以看出，我们对 index 应用进行了数据库迁移工作，并且在数据库中创建了三张表，分别是 Author、Book、UserInfo。而且在 index 应用下的 migrations 目录下生还成了一个 0001_initial.py 的文件，文件的内容如下：

```
# Generated by Django 2.2.10 on 2020-04-07 13:03

from django.db import migrations, models
import index.models

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Author',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=30, verbose_name='姓名')),
                ('email', models.EmailField(max_length=254, verbose_name='邮箱')),
            ],
        ),
        migrations.CreateModel(
            name='Book',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=30, unique=True, verbose_name='书名')),
                ('pub', models.CharField(max_length=50, verbose_name='出版社')),
                ('price', models.DecimalField(decimal_places=2, max_digits=7, verbose_name='定价')),
                ('retail_price', models.DecimalField(decimal_places=2, default=index.models.Book.default_price, max_digits=7, verbose_name='零售价')),
            ],
        ),
        migrations.CreateModel(
            name='UserInfo',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('Username', models.CharField(max_length=24, verbose_name='用户注册')),
                ('password', models.CharField(max_length=24, verbose_name='密码')),
                ('gender', models.CharField(choices=[('M', '男性'), ('F', '女性')], default='M', max_length=2)),
            ],
        ),
    ]
```

这个迁移文件包含了创建数据表时用到的所有信息，这是一个临时的过度文件，在执行完 makemigrations 命令后生成。我们可以使用如下命令打印迁移文件执行的 SQL 语句：

python manage.py sqlmigrate index 0001_initial

输出如下所示：

```
C:¥Users¥Administrator¥Book¥BookStore>python manage.py sqlmigrate index 0001_initial
BEGIN;
--
-- Create model Author
--
CREATE TABLE `index_author` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `name` varchar(30) NOT NULL, `email` varchar(254) NOT NULL);
--
-- Create model Book
--
```

```
CREATE TABLE `index_book` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `title` varchar(30) NOT NULL UNIQUE, `pub` varchar(50) NOT NULL, `price` numeric(7, 2) NOT NULL, `retail_price` numeric(7, 2) NOT NULL);
--
-- Create model UserInfo
--
CREATE TABLE `index_userinfo` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `Username` varchar(24) NOT NULL, `password` varchar(24) NOT NULL, `gender` varchar(2) NOT NULL);
COMMIT;
```

sqlmigrate 命令后紧跟应用名和文件的名字，这个命令不会实现数据库的最终迁移，它只是将 SQL 语句打印输出到命令行中。 最终我们使用 migrate 命令将 index 应用的模型类（models）转换成数据库中的数据表。

4. 魔术方法__str__

__str__ 方法是 Python 中的 "魔术" 方法，它是为 print 这样的打印函数设计的，它属于 python 的 object 基类的一个方法，也就是说 python 所有的类都有该方法，当然 Django 的 modle 类也有。如果没有这个方法定义，打印对象会显示对象的内存地址，但是这样的显示方式不够友好，且不利于调试，而用 __str__ 方法后，可以在 print 时得到易于人阅读的信息，在如下所示：

```
# 直接print打印
class TestClass:
    def __init__(self):
        self.name = 'testcase'
t = TestClass() #实例化对象
print(t)                # 结果显示：<__main__.TestClass object at 0x8f5c27b42367>

# __str__方法
class TestClass:
    def __init__(self):
        self.name = '小明'
    def __str__(self):
        return self.name
t = TestClass() #实例化对象
print(t)                # 结果显示：小明
```

不仅如此 __str__ 方法在 admin 后台系统也发挥着巨大的作用，它会将函数的返回值作为对象来显示。

本节内容为 index 应用创建了数据表，并且一步步带领大家实现了如何自定义模型类以及完成了数据表的迁移的，而且对产生的迁移文件也做了介绍，在下一节，我们将再次学习 Django 的后台管理系统，看看它是如何配合 ORM 使用的。