小白学Django第七天| 模型类Model进阶学习

原创 JAP君 Python进击者
2020-02-22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！
- 小白学Django第三天| 一文带你快速理解模型Model
- 小白学Django第四天| Django后台管理及配置MySQL数据库
- 小白学Django第五天| 视图View的初步使用
- 小白学Django第六天| 一文快速搞懂模板的使用
- Django实战小型图书人物信息网页(MVT的综合运用)
- 持续更新中...

前言

在前面的第三天的学习中，我们了解到了ORM，也学会了模型类的简单设计和表的生成，同时也学会了如何去操作数据表以及相关的查询。之所以写这篇文章是为了将 Model 模型类讲的更加的详细。

模型类--定义属性

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

django
会为表创建**自动增长的主键列**，每个模型只能有**一个主键列**，
如果**使用选项设置某属性为主键列后 django**
不会再创建自动增长的主键列。

默认创建的主键列属性为 id，可以使用 pk 代替，pk 全拼为 primary key。

无论我们是定义什么，肯定有它的一些规则，在属性的命名上，也是如此：

- 不能是 python 的保留关键字。
- 不允许使用连续的下划线，这是由 django 的查询方式决定的，在第 4 节会详细讲解查询。
- 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，语法如下：

```
属性=models.字段类型(选项)
```

字段类型

其实有关字段类型，在第三天的文章中也总结过，这里再次给大家整理，大家可以收藏这篇文章，随时查阅。

- 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)：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。

使用时需要引入 django.db.models 包”

选项

除了字段类型之外，还有约束字段的选项，同样也是有很多种限制，具体可以看下面：

- null：如果为 True，表示允许为空，默认值是 False。
- blank：如果为 True，则该字段允许为空白，默认值是 False。对比：null 是数据库范畴的概念，blank 是表单验证范畴的。
- db_column：字段的名称，如果未指定，则使用属性的名称。
- db_index：若值为 True, 则在表中会为此字段创建索引，默认值是 False。
- default：默认值。
- primary_key：若为 True，则该字段会成为模型的主键字段，默认值是 False，一般作为 AutoField 的选项使用。
- unique：如果为 True, 这个字段在表中必须有唯一值，默认值是 False。

综合演示

这里给出具体的例子，大家可以方便看出相应的用法：

```
from django.db import models

#定义图书模型类BookInfo
class BookInfo(models.Model):
    #btitle = models.CharField(max_length=20)#图书名称
    btitle = models.CharField（max_length=20,
db_column='title')#通过db_column指定btitle对应表格中字段的名字为title
    bpub_date = models.DateField()#发布日期
    bread = models.IntegerField(default=0)#阅读量
    bcomment = models.IntegerField(default=0)#评论量
    isDelete = models.BooleanField(default=False)#逻辑删除

#定义人物模型类HeroInfo
class PeopleInfo(models.Model):
```

```
    pname = models.CharField(max_length=20)#姓名
    pgender = models.BooleanField(default=True)#性别
    isDelete = models.BooleanField(default=False)#逻辑删除
    #pcomment = models.CharField(max_length=200)#描述信息
    pcomment = models.CharField(max_length=200, null=True, blank=False)
#pcomment对应的数据库中的字段可以为空，但通过后台管理页面添加信息时pcomment对应的输入框不能为空
    pbook =
models.ForeignKey('BookInfo')#人物与图书表的关系为一对多，所以属性定义在人物模型类中
```

条件查询--字段查询

所谓字段查询就是实现 sql 语句的 where 功能，调用过滤器 filter()、exclude()、get()。

通过"属性名_id"表示外键对应对象的 id 值。

语法如下：

```
属性名称__比较运算符=值
```

★ 说明：属性名称和比较运算符间使用两个下划线，所以属性名不能包括多个下划线。

条件运算符

1) 查询等

exact：表示判等。

例：查询编号为 1 的图书。

```
list=BookInfo.objects.filter(id__exact=1)
```

可简写为：

```
list=BookInfo.objects.filter(id=1)
```

2) 模糊查询

contains：是否包含。

说明： 如果要包含%无需转义，直接写即可。

例：查询书名包含'传'的图书。

```
list = BookInfo.objects.filter(btitle__contains='传')
```

startswith、endswith： 以指定值开头或结尾。例：查询书名以'部'结尾的图书

```
list = BookInfo.objects.filter(btitle__endswith='部')
```

以上运算符都区分大小写，在这些运算符前加上 i 表示不区分大小写，如 iexact、icontains、istartswith、iendswith.

3) 空查询

isnull：是否为null。

例：查询书名不为空的图书。

```
list = BookInfo.objects.filter(btitle__isnull=False)
```

4) 范围查询

in：是否包含在范围内。

例：查询编号为1或3或5的图书

```
list = BookInfo.objects.filter(id__in=[1, 3, 5])
```

5) 比较查询

gt、gte、lt、lte：大于、大于等于、小于、小于等于。

例：查询编号大于3的图书

```
list = BookInfo.objects.filter(id__gt=3)
```

不等于的运算符，使用exclude()过滤器。

例：查询编号不等于3的图书

```
list = BookInfo.objects.exclude(id=3)
```

6) 日期查询

year、month、day、week_day、hour、minute、second：对日期时间类型的属性进行运算。

例：查询1980年发表的图书。

```
list = BookInfo.objects.filter(bpub_date__year=1980)
```

例：查询1980年1月1日后发表的图书。

```
list = BookInfo.objects.filter(bpub_date__gt=date(1990, 1, 1))
```

F对象

之前的查询都是对象的属性与常量值比较，两个属性怎么比较呢？

答：使用F对象，被定义在django.db.models中。

语法如下：

```
F(属性名)
```

例：查询阅读量大于等于评论量的图书。

```
from django.db.models import F
...
list = BookInfo.objects.filter(bread__gte=F('bcomment'))
```

bread_gte后的gte表示大于等于的关系。

可以在F对象上使用算数运算。

例：查询阅读量大于2倍评论量的图书。

```
list = BookInfo.objects.filter(bread__gt=F('bcomment') * 2)
```

Q对象

多个过滤器逐个调用表示逻辑与关系，同sql语句中where部分的and关键字。

例：查询阅读量大于20，并且编号小于3的图书。

```
list=BookInfo.objects.filter(bread__gt=20,id__lt=3)
或
list=BookInfo.objects.filter(bread__gt=20).filter(id__lt=3)
```

如果需要实现逻辑或or的查询，需要使用Q()对象结合|运算符，Q对象被义在django.db.models中。

语法如下：

```
Q(属性名__运算符=值)
```

例：查询阅读量大于20的图书，改写为Q对象如下。

```
from django.db.models import Q
...
list = BookInfo.objects.filter(Q(bread__gt=20))
```

Q对象可以使用&、|连接，&表示逻辑与，|表示逻辑或。

例：查询阅读量大于20，或编号小于3的图书，只能使用Q对象实现

```
list = BookInfo.objects.filter(Q(bread__gt=20) | Q(pk__lt=3))
```

Q对象前可以使用~操作符，表示非not。

例：查询编号不等于3的图书。

```
list = BookInfo.objects.filter(~Q(pk=3))
```

聚合函数

使用aggregate()过滤器调用聚合函数。聚合函数包括：**Avg，Count，Max，Min，Sum，** 被定义在django.db.models中。

例：查询图书的总阅读量。

```
from django.db.models import Sum
...
list = BookInfo.objects.aggregate(Sum('bread'))
```

注意aggregate的返回值是一个字典类型，格式如下：

```
{'聚合类小写__属性名':值}
```

如:{'sum__bread':3}
使用count时一般不使用aggregate()过滤器。

例：查询图书总数。

```
list = BookInfo.objects.count()
```

注意count函数的返回值是一个数字。

查询集

查询集表示从数据库中获取的对象集合，在管理器上调用某些过滤器方法会返回查询集，查询集可以含有零个、一个或多个过滤器。过滤器基于所给的参数限制查询的结果，从Sql的角度，查询集和select语句等价，过滤器像where和limit子句。

返回查询集的过滤器如下：

- all()：返回所有数据。
- filter()：返回满足条件的数据。
- exclude()：返回满足条件之外的数据，相当于sql语句中where部分的not关键字。
- order_by()：对结果进行排序。

返回单个值的过滤器如下：

- get()：返回单个满足条件的对象如果未找到会引发"模型类.DoesNotExist"异常。如果多条被返回，会引发"模型类.MultipleObjectsReturned"异常。
- count()：返回当前查询结果的总条数。
- aggregate()：聚合，返回一个字典。

判断某一个查询集中是否有数据：

- exists()：判断查询集中是否有数据，如果有则返回True，没有则返回False。

两大特性

查询集有两个特性，如下：

1. 惰性执行：创建查询集不会访问数据库，直到调用数据时，才会访问数据库，调用数据的情况包括迭代、序列化、与if合用。
2. 缓存：使用同一个查询集，第一次使用时会发生数据库的查询，然后把结果缓存下来，再次使用这个查询集时会使用缓存的数据。

查询集的缓存

每个查询集都包含一个缓存来最小化对数据库的访问。

在新建的查询集中，缓存为空，首次对查询集求值时，会发生数据库查询，django会将查询的结果存在查询集的缓存中，并返回请求的结果，接下来对查询集求值将重用缓存中的结果。

第一种情况： 如下是两个查询集，无法重用缓存，每次查询都会与数据库进行一次交互，增加了数据库的负载。

```
from booktest.models import BookInfo
[book.id for book in BookInfo.objects.all()]
[book.id for book in BookInfo.objects.all()]
```

（注意：这里使用了两个查询）

第二种情况： 经过存储后，可以重用查询集，第二次使用缓存中的数据。

```
list=BookInfo.objects.all()
[book.id for book in list]
[book.id for book in list]
```

（注意：这里只使用了一次查询）

限制查询集

可以对查询集进行取下标或切片操作，等同于sql中的limit和offset子句。

注意：不支持负数索引。”

对查询集进行切片后返回一个新的查询集，不会立即执行查询。

如果获取一个对象，直接使用[0]，等同于[0:1].get()，但是如果没有数据，[0]引发IndexError异常，[0:1].get()如果没有数据引发DoesNotExist异常。

小例子：获取第1、2项

```
list=BookInfo.objects.all()[0:2]
```

模型类关系

在数据库中，数据表之间可能是存在一定关系的，例如一对多，多对一，多对多，一对一。在模型类中也有着相应的关系，怎么来实现？看下面。

关系字段类型

关系型数据库的关系包括三种类型：

- ForeignKey：一对多，将字段定义在多的一端中。
- ManyToManyField：多对多，将字段定义在任意一端中。
- OneToOneField：一对一，将字段定义在任意一端中。
- 可以维护递归的关联关系，使用'self'指定，详见"自关联"。

在我们之前的文章中是使用过ForeignKey的关系，也就是一对多的关系。

1）一对多

这里我就不过多阐述，点后面的文章链接可以跳转过去一对多的案例

2）多对多

我们下面设计一个新闻类和新闻类型类，一个新闻类型下可以用很多条新闻，一条新闻也可能归属于多种新闻类型。

```
class TypeInfo(models.Model):
    tname = models.CharField(max_length=20) #新闻类别

class NewsInfo(models.Model):
    ntitle = models.CharField(max_length=60) #新闻标题
    ncontent = models.TextField() #新闻内容
    npub_date = models.DateTimeField(auto_now_add=True) #新闻发布时间
    ntype = models.ManyToManyField('TypeInfo')
#通过ManyToManyField建立TypeInfo类和NewsInfo类之间多对多的关系
```

通过对象执行关联查询

在定义模型类时，可以指定三种关联关系，最常用的是一对多关系，如前面文章中的"图书-
人物"就为一对多关系，接下来进入shell练习关系的查询。

由一到多的访问语法：

一对应的模型类对象.多对应的模型类名小写_set 例：

```
b = BookInfo.objects.get(id=1)
b.peopleinfo_set.all()
```

由多到一的访问语法：

多对应的模型类对象.多对应的模型类中的关系类属性名 例：

```
p = PeopleInfo.objects.get(id=1)
p.hbook
```

访问一对应的模型类关联对象的id语法：

多对应的模型类对象.关联类属性_id 例：

```
p = PeopleInfo.objects.get(id=1)
p.book_id
```

例：查询编号为1的图书。

```
book=BookInfo.objects.get(pk=1)
```

例：获得book图书的所有人物。

```
book.peopleinfo_set.all()
```

通过模型类执行关联查询

由多模型类条件查询一模型类数据：

语法如下：

```
关联模型类名小写__属性名__条件运算符=值
```

如果没有"__运算符"部分，表示等于，结果和sql中的inner join相同。

由一模型类条件查询多模型类数据：语法如下：

```
一模型类关联属性名__一模型类属性名__条件运算符=值
```

例：查询书名为“天龙八部”的所有人物。

```
list = PeopleInfo.objects.filter(pbook__btitle='天龙八部')
```

自关联

对于地区信息、分类信息等数据，表结构非常类似，每个表的数据量十分有限，**为了充分利用数据表的大量数据存储功能，可以设计成一张表，内部的关系字段指向本表的主键，** 这就是自关联的表结构。

这里给大家简单写一个案例：

打开booktest/models.py文件，定义AreaInfo类。

```
#定义地区模型类，存储省、市、区县信息
class AreaInfo(models.Model):
    atitle=models.CharField(max_length=30)#名称
    aParent=models.ForeignKey('self',null=True,blank=True)#关系
```

说明：关系属性使用self指向本类，要求null和blank允许为空，因为一级数据是没有父级的。”

booktest/views.py文件，定义视图area。

```
from booktest.models import AreaInfo
...
#查询广州市的信息
def area(request):
    area = AreaInfo.objects.get(pk=440100)
    return render(request, 'booktest/area.html', {'area': area})
```

在templates/booktest目录下，新建area.html文件。

```
<html>
<head>
    <title>地区</title>
</head>
<body>
当前地区：{{area.atitle}}
<hr/>
上级地区：{{area.aParent.atitle}}
<hr/>
下级地区：
<ul>
    {%for a in area.areainfo_set.all%}
    <li>{{a.atitle}}</li>
    {%endfor%}
</ul>
</body>
</html>
```

随后进行相关的Django配置，运行服务器

可以看到运行结果：

当前地区：广州市

上级地区：广东省

下级地区：

- 荔湾区
- 越秀区
- 海珠区
- 天河区
- 白云区
- 黄埔区
- 番禺区
- 花都区
- 南沙区
- 萝岗区
- 增城市
- 从化市

原创不易，点个在看吧！

精选留言

暂无...