Django QuerySet查询API攻略

Django 经过 API 查询，从数据库中查询出来的返回结果一般是一个集合，这个集合叫做 QuerySet，它可以包含一个、多个或者零个 Model 实例。在上一节中《Django查询数据库操作详解》 我们介绍了单例查询以及原生 SQL 语句查询，在本节将介绍 Django 中返回 QuerySet 的查询 API。我们要学会重点掌握常用 API，比如 all、filter、exclude、order_by 等方法。

QuerySet 可以被构造，过滤，切片，做为参数传递，这些行为都不会对数据库进行操作。只要你查询的时候才真正的操作数据库。QuerySet 有个很重要的特性是惰性加载，当我们使用它的时候，它才会去访问数据库。如果每一次都去查询数据库，则很容易造成性能的浪费。

1. 数据查询常用API

1) all方法获取所有数据记录
查询管理器 Manager 提供的 all 方法可以获取 Model 的所有数据实例，使用方式如下：

books=Book.objects.all()
上面将查询结果赋值给了 books 变量，由于 QuerySet 是惰性加载，所以此时该语句并不会访问数据库，只有当 我们使用 books 的时候，才会访问数据库，获取相应的查询结果。我们可以通过 QuerySet 的 query 属性查看生成的 SQL 语句。如下所示：

print(Book.objects.all().query)
打印结果如下所示：

```
print(Book.objects.all().query)
SELECT `index_book`.`id`, `index_book`.`title`, `index_book`.`price`, `index_book`.`retail_price`, `index_book`.`pub_id` FROM `index_book`
```

2) filter方法条件查询多条记录
filter方法的语法格式如下：

MyModel.objects.filter(属性1=值1, 属性2=值2)
filter 方法返回一个新的 QuerySet ，它包含了与筛选条件相匹配的所有对象。这些筛选条件，通过参数的形式传递给 filter 方法。filter 方法会将传递的参数转换成 WHERE 子句实现过滤查询。若是多个条件那么它们之间属于 and 关系。在没有查询条件的情况下，它的与 all 方法是等价的。

Django 为了更好实现条件查询，提供了条件查询关键字这些关键词用来修饰字段的，例如大于、小于、大于等于、是否包含等，使用方法是字段名后加上双下划线，再加上条件查询关键字，每个查询关键字都有其独立的查询功能。我们把这些查询关键字，也称做“查询谓词”。如下面表格对常用查询谓词进行了部分列举：

Django查询谓词表

条件查询关键字	**代表含义**	**SQL语句**
__gt	大于	select * from index_book where id>3;
__gte	大于等于	select * from index_book where ids>=3;
__lt	小于	select * from index_book where id<3;

__lte	小于等于	select * from index_book where id<=3;
__exact	等于	select * from index_book where id=4;
__iexact	忽略大小写的等于	select * from index_book where name like 'flask';
__in	是否在集合中	select * from index_book where id in (2,8,11) ;
__contains	是否包含...	select * from index_book where title like "%T%";
__startswith	以...开头	select * from index_book where title like "P%";

使用实例说明：

```
Author.objects.filter(name__icontains='l')#icontains不对大小写敏感
# 等同于SQL语句
select * from author where name like '%l%'
# 查找价格在某一区间内的所有书籍
Book.objects.filter(price__range=(20,45))
# 等同于SQL语句
select price where Book between 20 and 45;
```

3) exclude方法反向过滤

它与 filter 方法正好正反，相当于在 filter 方法前面加上一个 NOT，即过滤出来的结果是不满足条件的数据记录。同样它也返回一个新的 QuerySet。fillter 表示正向过滤，exclude 表示反向过滤。示例如下：

```
In [2]: Book.objects.exclude(price__lt=40)
#返回的结果是价格大于40的书籍
Out[2]: <QuerySet [<Book: Book object (1)>, <Book: Book object (3)>, <Book: Book object (4)>, <Book: Book object (5)>]>
```

4) order_by实现自定义排序

order_by 方法可以实现自定义排序，同时也可以指定多个排序字段。order_by方法在执行之前，会清除它之前的所有排序，它和内部类 Meta 提供的 ordering 元数据项起到一样的作用，我们可以二者选择其一来实现字段的排序，如果是字符串将按照字母进行排序，如果是数字将按照大小进行排序。示例如下：

```
In [3]: Book.objects.order_by("-title","price")#按照title逆序排序后再按正序排列
Out[3]: <QuerySet [<Book: Book object (6)>, <Book: Book object (2)>, <Book: Book object (1)>, <Book: Book object (3)>, <Book: Book object (5)>, <Book: Book object (4)>]>
```

order_by 使用时有两种场景需要注意：

- order_by 接收的参数为 ？时，这代表随机排序，但是这种方式会消耗资源性能没并且很慢，不建议使用。
- order_by不接受参数的情况下，将不会有任何排序规则，默认的规则也不会有。

5) reverse方法逆序记录获取

它的使用方法和 all 方法相似，只不过返回的结果是逆序的数据记录，可以理解成逆序的查询方法。

6) values方法获取字典结果

values 方法返回字典，字典中的键对应 Model 的字段名。可以给 values 方法传递参数，用于限制 select 查询范围，如果指定，则查询结果包含 Model 的所有字段。

```
In [8]: Book.objects.values("id","title")#字典的键对应字段名，与其对应的值组成键值对
Out[8]: <QuerySet [{'id': 4, 'title': 'Django'}, {'id': 5, 'title': 'Flask'}, {'id': 3, 'title': 'Java'}, {'id': 1, 'title': 'Python'}, {'id': 2, 'title': 'Redis'}, {'id': 6, 'title': 'Tornado'}]>
```

7) values_list方法获取元组结果

它和 values 方法用法相似，但是它的返回结果是列表，其中每一个元素都是一个元组，而非字典。用法如下所示：

```
In [10]: Book.objects.values_list("id","title","price")
Out[10]: <QuerySet [(1, 'Python', Decimal('59.00')), (2, 'Redis', Decimal('25.00')), (3, 'Java', Decimal('45.00')), (4, 'Django', Decimal('65.00')), (5, 'Flask', Decimal('45.00')), (6, 'Tornado', Decimal('35.00'))]>
```

8) QuerySet切片使用

前面我们介绍 QuerySet 是一个可迭代的对象，同时支持索引和切片，并且执行切片后会返回一个新的 QuerySet。我们可以使用切片来限制返回的数据结果，但是这里需要注意的是，QuerySet 不支持末尾切片，即索引值不允许为负数。示例如下：

```
Book.objects.all()[:3] #返回新的QuerySet对象
<QuerySet [<Book: Book object (1)>, <Book: Book object (2)>, <Book: Book object (3)>]>
Book.objects.all()[1:5:2]#带步长的切片，将返回Model实例列表
[<Book: Book object (2)>, <Book: Book object (4)>]
```

切片后得到的 QuerySet 不能再执行其他操作，比如字段排序、过滤等。

利用 Python 的数组切片语法将 QuerySet 切成指定长度。这等价于 SQL 的 LIMIT 和 OFFSET 子句

在本节我们详细介绍了 QuerySet 的查询 API，这些 API 较为常见。当然还有一些 API，不过它们使用频率相对较低，在 Django 官方网站提供了详细的 《QuerySet API 文档》，如果大家感兴趣，可以自行学习。