Django QuerySets：真他妈棒？

Django的QuerySets酷毙了！

在本文中我将解释一下QuerySets是什么，它是如何工作的（如果你对它已经熟悉了，你可以直接跳到第二部分），我认为如果可以的话你应该总是返回QuerySets对象，下面让我来谈谈如何做。

QuerySets很酷

QuerySet，本质上是一个给定的模型的对象列表。我说“列表”而不是“组”或更正式的“集合”因为它是有序的。事实上，你可能已经熟悉如何获得QuerySets，因为这就是你调用variousBook.objects.XXX()方法后得到的对象。例如，考虑下面的语句：

```
Book.objects.all()
```

all()返回的就是Book实例的一个QuerySet，它正好包括allBookinstances，下面的其他调用你可能已经知道：

```
# Return all books published since 1990
Book.objects.filter(year_published__gt=1990)

# Return all books *not* written by Richard Dawkins
Book.objects.exclude(author='Richard Dawkins')

# Return all books, ordered by author name, then
# chronologically, with the newer ones first.
Book.objects.order_by('author', '-year_published')
```

关于 QuerySet s最酷的是，由于这些函数操作、返回的都是一个QuerySet，你可以把他们链起来：

```
# Return all book published after 1990, except for
# ones written by Richard Dawkins. Order them by
# author name, then chronologically, with the newer
# ones first.
Book.objects.filter(year_published__gt=1990) \
            .exclude(author='Richard Dawkins') \
            .order_by('author', '-year_published')
```

而且这并不是全部的，它更快！

> 在内部，一个QuerySet可以被构造、过滤、切片及像普通变量那样在没有实际数据库查询的情况下随便传递，在评估处理完QuerySet前不产生数据库活动。

所有我们确认了QuerySets很酷，不是么？

尽可能的返回QuerySets

我最近曾在一个Django应用中用一个模型来表示树（数据结构，不是圣诞装饰）。这意味着每一个实例在树上都有一个指向它父节点的链接。它看起来像这样：

```
01 class Node(models.Model):
02     parent = models.ForeignKey(to='self', null=True, blank=True)
03     value = models.IntegerField()
04 
05     def __unicode__(self):
06         return 'Node #{}'.format(self.id)
07 
08     def get_ancestors(self):
09         if self.parent is None:
10             return []
11         return [self.parent] + self.parent.get_ancestors()
```

这工作的相当好。麻烦的是，我不得不添加另一种方法，get_larger_ancestors，它应该返回所有值大于当前节点的的父节点。这是我能实现这个：

```
1 def get_larger_ancestors(self):
2         ancestors = self.get_ancestors()
3         return [node for node in ancestors if node.value > self.value]
```

问题是，我基本上会在名单上审查两次——Django一次，我自己一次。这让我考虑到-如果get_ancestors返回QuerySet而不是列表会怎样呢？我可以这样做：

```
1 def get_larger_ancestors(self):
2         return self.get_ancestors().filter(value__gt=self.value)
```

很简单，这里更重要的是我没有遍历对象。我可以对get_larger_ancestors的返回使用任何我想使用的过滤器，而且感到安全——我不会得到一个未知大小的对象列表。这样的主要优势是我一直使用相同的查询接口。当用户得到了一大堆的对象，我们不知道他想怎样对它们进行切片分块。而返回QuerySet对象时我保证用户知道如何处理它。

但如何实现get_ancestorsto返回一个QuerySet呢？这是一个小技巧。用一条简单的查询收集我们需要的数据是不可能的，使用任何预定数量的查询也是不可能的。我们要找的法则是动态的，选择的实现看起来很像它现在的样子，下面就是选择，一个更好的实现：

```
01 class Node(models.Model):
02     parent = models.ForeignKey(to='self', null=True, blank=True)
03     value = models.IntegerField()
04 
```

```
    def __unicode__(self):
        return 'Node #{}'.format(self.id)

    def get_ancestors(self):
        if self.parent is None:
            return Node.objects.none()
        return
Node.objects.filter(pk=self.parent.pk) | self.parent.get_ancestors()

    def get_larger_ancestors(self):
        return self.get_ancestors().filter(value__gt=self.value)
```

稍停一会，沉淀一下，马上说出细节。

我想说的是，不论什么时候你返回一系列对象——你应该总是返回一个QuerySet替代。这样做将允许用户使用一种简单、熟悉、具备更好性能的方法自由过滤、剪接和排序结果。

（从一个侧面说get_ancestors查询了数据库，因为我使用了递归的self.parent。这里有一个额外的数据库执行——当实际检测结果时执行了这个函数，未来又执行了另外一次。当我们在数据库查询上使用更多的过滤器或进行高耗内存的操作时我们得到了性能的提升。这里的例子展示了如何将平常的操作转换成QuerySets）。

常见的QuerySet操作

所以，执行简单查询时返回一个QuerySet很简单。当我们想实现复杂一点的东西，我们需要执行相关操作（也包括一些助手函数）。下面是些小窍门（作为练习，试着理解我get_larger_ancestors的实现）。

- **联合** - QuerySet的联合运算符是|，处理复制时管道“symbol.qs1 | qs2”返回所有来自qs1和qs2项目的QuerySet（都在QuerySet的项目将只在结果中出现一次）。
- **交集** - 交集没有特殊的操作，因为你已经知道怎么去做。 像filter等链接函数在原始的QuerySet和新过滤器之前起了交集的作用。
- **差分** - 差分(数学上写为qs1 \ qs2)代表所有在qs1而不在qs2中的项目。请注意，此操作是不对称的（相对于以前的操作）。Python中恐怕没有内置的方式，但你可以这样做：qs1.exclude(pk__in=qs2)
- **从空开始** - 开起来没有用处但实际并非如此，正如上面例子所展示的。很多时候，当我们动态建立一个QuerySet联合时，我们需要从一个空列表开始，这是获取它的方法:MyModel.objects.none().

本文地址：http://www.oschina.net/translate/django-querysets-fucking-awesome-yes
原文地址：http://blog.amir.rachum.com/post/55290451179/django-querysets-fucking-awesome-yes