model relationship API.md

One To One

[django official doc]

- 一对一关系： 使用关键字`OneToOneField`
  `A` can optionally be `B`
  e.g. a `place` can optionally be a `restaruant`
  在这种情况下，将`OneToOneField A`放在`B`的model中，使`A` 成为 `B` 的reference
  model 代码:

```
from django.db import models

class A(models.Model):
    some_Afeild_1 = models.CharField(max_length=50)
  some_Afield_2 = models.CharField(max_length=80)

class B(models.Model):
    a = models.OneToOneField( # 放入对A的reference
        A,
        on_delete=models.CASCADE, # "there are 6 behaviours to adopt when the referenced object is deleted. It is not specific to django, this is an SQL standard."
        primary_key=True, # 此项将A 对象设为 B的pk--B对象将不具有单独的id
    )
    some_Bfield_1 = models.BooleanField(default=False)
    some_Bfield_2 = models.BooleanField(default=False)
```

- 当创造对象时：
  - 被依赖者（此例中为`A`）可以被独立创造（不需要传入`依赖者`作为参数）
  - 依赖者（此例中为`B`）被创造时需传入其`依赖对象`
    - （注意：传入时，该依赖对象必须已被`save()`进数据库
    - 当其依赖对象设置为nullable时可以不传 （？？此项存疑）

- 双向Access：
  可以`A.b`；也可以`B.a`
  - 注意：为防止A（B）没有关联的B（A）导致报错，有两种办法：

```
from django.core.exceptions import ObjectDoesNotExist
try:
    p2.restaurant
except ObjectDoesNotExist:
    print("There is no restaurant here.")
```

```
hasattr(p2, 'restaurant') # 返回 true/ false
```

- 更换关系对象：
  - 当依赖者（B）更换关系对象（A）时，`save()`操作将导致向数据库中存入新的条目（因为依赖对象为其pk）
  - 当被依赖者（A）更换关系对象（B）时，`save()`操作不会向数据库中存入新的条目
- lookups across relationships:透过关系对象的特性对本体进行filter

```
Restaurant.objects.get(place=p1)
Restaurant.objects.get(place__pk=1)
Restaurant.objects.filter(place__name__startswith="Demon")
Restaurant.objects.exclude(place__address__contains="Ashland")
```

也可以反过来查询：

```
Place.objects.get(pk=1)
Place.objects.get(restaurant__place=p1)
Place.objects.get(restaurant=r)
Place.objects.get(restaurant__place__name__startswith="Demon")
```

Many To One

django official doc

- 多对一关系：使用关键字`ForeignKey`
- 定义“多对一”关系：
  - 一个作者（A，单端）可以有多个文章（B，多端）
  - 但一个文章（B，多端）只能有一个作者（A，单端）

model代码：

```
from django.db import models

class A(models.Model): # 单端
    some_Afield_1 = models.CharField(max_length=30)
    some_Afield_2 = models.CharField(max_length=30)
    email = models.EmailField()

class B(models.Model): # 多端
    some_Bfield_1 = models.CharField(max_length=100)
    some_Bfield_2 = models.DateField()
    b = models.ForeignKey(A, on_delete=models.CASCADE) # 将"单端"放入多端
```

- 双向access:
  - 单端access多端：A.b_set (需要用filter查找具体对象)
  - 多端access单端：B.a
- 使用add 添加：A.b_set.add()
  - 创造多端的时候必须传入其单端（必须已经save()到DB中的）
  - 创造单端的时候不必使用多端
  - 将一个多端添加到不同的单端上：该多端将不再在原来的单端上
- 使用create一步添加：
  - 单端创造多端：A.b_set.create(..some B params... ！！注意这里不用再重复输入单端A)

    - （存疑）多端创造单端：B.a.create()
- on_delete默认设置为CASCADE：单端 删除将导致其所有多端都被删除
- lookups across relationships:

  Query the waiters:

```
Waiter.objects.filter(restaurant__place=p1)
Waiter.objects.filter(restaurant__place__name__startswith="Demon")
```

Many To Many

[django official doc]

- 多对多关系：A可以有多个B；B也可以有多个A
- 使用关键字：`ManyToManyField`，将B（A）的reference放入A（B）中；

  model代码

```
from django.db import models

class A(models.Model):
    some_Afield_1 = models.CharField(max_length=30)

    class Meta:
        ordering = ['some_Afield_1']    # 定义该表按some_Afield_1字段排序

    def __str__(self):
        return self.some_Afield_1

class B(models.Model):
    some_Bfield_1 = models.CharField(max_length=100)
    a = models.ManyToManyField(A)  # 将对A的reference放入B中

    class Meta:
        ordering = ['some_Bfield_1'] # 定义该表按some_Bfield_1字段排序

    def __str__(self):
        return self.some_Bfield_1
```

- 使用add 添加：A.b_set.add() ; B.a.add()
    - 添加的对象必须已经save()到数据库中
    - 重复添加同一对象不会导致重复
- 使用create一步创造+添加：
    - 省去了save()操作
- ondelete默认为不会删除
- 可以使用 [lookups across relationships]

```
Publication.objects.filter(article__headline__startswith="NASA")
Article.objects.filter(publications__title__startswith="Science")
```

- 双向access: A.b_set() ； B.a

- 移除：A.b_set().remove() ； B.a.remove()