

05 Django模型进阶

一. 配置MySQL

1, 安装mysql

2, MySQL驱动

使用mysqlclient

```
pip install mysqlclient
```

(如果上面的命令安装失败, 则尝试使用国内豆瓣源安装:

```
pip install -i https://pypi.douban.com/simple mysqlclient
```

)

(Linux Ubuntu下需要先安装: apt install libmysqld-dev

再安装: apt install libmysqld-dev

)

2, 在Django中配置和使用mysql数据库

使用mysql数据库, settings中配置如下:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'mydb',  
        'USER': 'root',  
        'PASSWORD': '123456',  
        'HOST': '127.0.0.1',  
        'PORT': '3306',  
    }  
}
```

二. 多模块关联关系

多个模块关联

关联分类

- ForeignKey: 一对多, 将字段定义在多的端中
- ManyToManyField: 多对多, 将字段定义在两端的任意一端中
- OneToOneField: 一对一, 将字段定义在任意一端中

一对多关系, 举例说明 (一对一, 多对多类似):

一个班级可以有多个学生, 一个学生只能属于一个班级

```
class Grade(models.Model):  
    name = models.CharField(max_length=20)  
class Student(models.Model):  
    name = models.CharField(max_length=20)  
    grade = models.ForeignKey(Grade, on_delete=)
```

对象的使用：

正向（在Student这边，有grade属性的这一边）：

获取学生所在班级（对象）：`stu.grade`

获取学生所在班级的属性：`stu.grade.name`

反向（在Grade这边）：

获取班级的所有学生（获取Manager对象）：`grade.student_set`

获取班级的所有学生（获取QuerySet查询集）：`grade.student_set.all()`

`filter()`, `get()` 等操作中的使用：

正向（在Student这边，有grade属性的这一边）：

`Student.objects.filter(属性__name='1')`

如：`Student.objects.filter(grade__name='1')`

反向（在Grade这边）：

`Grade.objects.filter(类名小写__id=7)`

如：`Grade.objects.filter(student__id=7)`

三. Model连表结构

一对多：`models.ForeignKey(其他表)`

多对多：`models.ManyToManyField(其他表)`

一对一：`models.OneToOneField(其他表)`

应用场景：

一对多：当一张表中创建一行数据时，有一个单选的下拉框（可以被重复选择）

例如：创建用户信息时候，需要选择一个用户类型【普通用户】【金牌用户】【铂金用户】

多对多：在某表中创建一行数据时，有一个可以多选的下拉框。（猫眼App，淘票票，格拉瓦电影）

例如：创建用户信息，需要为用户指定多个爱好。

一对一：在某表中创建一行数据时，有一个单选的下拉框（下拉框中的内容被用过一次就消失了）

例如：有个身份证表，有个person表。每个人只能有一张身份证，一张身份证也只能对应一个人，这就是一对一关系。

一对多关联

一对多关系，即外键

为什么要用一对多。先来看一个例子。有一个用户信息表，其中有个用户类型字段，存储用户的用户类型。如下：

```
class UserInfo(models.Model):
    username = models.CharField(max_length=32)
    age = models.IntegerField()
    user_type = models.CharField(max_length=10)
```

不使用外键时用户类型存储在每一行数据中。如使用外键则只需要存储关联表的id即可，能够节省大量的存储空间。同时使用外键有利于维持数据完整性和一致性。

当然也有缺点，数据库设计变的更复杂了。每次做DELETE 或者UPDATE都必须考虑外键约束。

刚才的例子使用外键的情况：单独定义一个用户类型表：

```

class UserType(models.Model):
    caption = models.CharField(max_length=32)

class UserInfo(models.Model):
    user_type = models.ForeignKey('UserType')
    username = models.CharField(max_length=32)
    age = models.IntegerField()

```

我们约定：

正向操作：ForeignKey在UserInfo表里，如果根据UserInfo去操作就是正向操作。

反向操作：ForeignKey不在UserType里，如果根据UserType去操作就是反向操作。

一对多的关系的增删改查：

正向操作：

增

1) 创建对象实例，然后调用save方法：

```

obj = UserInfo(name='li', age=44, user_type_id=2)
obj.save()

```

2) 使用create方法

```

UserInfo.objects.create(name='li', age=44, user_type_id=2)

```

3) 使用get_or_create方法，可以防止重复

```

UserInfo.objects.get_or_create(name='li', age=55, user_type_id=2)

```

4) 使用字典。

```

dic = {'name': 'zhangsan', 'age': 18, 'user_type_id': 3}

```

```

UserInfo.objects.create(**dic)

```

5) 通过对象添加

```

usertype = UserType.objects.get(id=1)

```

```

UserInfo.objects.create(name='li', age=55, user_type=usertype)

```

删

和普通模式一样删除即可。如：

```

UserInfo.objects.filter(id=1).delete()

```

改

和普通模式一样修改即可。如：

```

UserInfo.objects.filter(id=2).update(user_type_id=4)

```

查

正向查找所有用户类型为钻石用户的用户，使用双下划线：

```

users = UserInfo.objects.filter(user_type__caption__contains='钻石')

```

正向获取关联表中的属性可以直接使用点.语法，比如：

获取users查询集中第一个用户的caption：

```

users[0].user_type.caption

```

反向操作：

增（一般使用正向增即可）

通过usertype来创建userinfo

1) 通过userinfo_set的create方法

#获取usertype实例

```

ut = UserType.objects.get(id=2)
#创建userinfo
ut.userinfo_set.create(name='smith',age=33)

```

删

删除操作可以在定义外键关系的时候，通过on_delete参数来配置删除时做的操作。

on_delete参数主要有以下几个可选值：

models.CASCADE 表示级联删除，即删除UserType时，
相关联的UserInfo也会被删除。

models.PROTECT 保护模式，阻止级联删除。

models.SET_NULL 置空模式，设为null，null=True参数必须具备

models.SET_DEFAULT 置默认值 设为默认值，default参数必须具备

models.SET() 删除的时候重新动态指向一个实体访问对应元素，可传函数

models.DO_NOTHING 什么也不做。

注意：修改on_delete参数之后需要重新同步数据库，如果使用

改

和普通模式一样，不会影响级联表。

查

通过usertype对象来查用户类型为1的用户有哪些

```

obj=UserType.objects.get(id=1)
obj.userinfo_set.all()

```

可以通过在定义foreignkey时指定related_name来修改默认的用户info_set，
比如指定related_name为info

```

user_type = models.ForeignKey('UserType', related_name='info')

```

指定related_name之后，反向查的时候就变成了：

```

obj.info.all()

```

获取用户类型为1且用户名为shuaige的用户

```

obj.info.filter(username='shuaige')

```

外键关系中，django自动给usertype加了一个叫做userinfo的属性。使用双下划线，
可以通过userinfo提供的信息来查usertype（了解）

```

user_type_obj = UserType.objects.get(userinfo__username='zs')

```

多对多关联

多对多关系

针对多对多关系django会自动创建第三张表。也可以通过through参数指定第三张表。

用户和组是典型的多对多关系：

```

class Group(models.Model):
    name = models.CharField(max_length=20)

    def __str__(self):
        return self.name

class User(models.Model):

```

```
name = models.CharField(max_length=64)
password = models.CharField(max_length=64)
groups = models.ManyToManyField(Group)
```

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

操作:

增:

先分别创建user和group, 再使用add关联

```
u = User(name='aa', password='123')
u.save()
```

```
g = Group(name='g5')
```

```
g.save()
```

通过Manager对象使用add()方法

```
u.groups.add(g) 或 g.user_set.add(u)
```

删:

和一对多类似, 删除user或group会级联删除user_groups表中的关联数据

改:

和一对多类似, 只修改当前表

查:

正向:

查询id=2的用户所在的所有组group

```
u = User.objects.get(id=2)
```

```
u.groups.all()
```

反向:

查询id=1的组中包含的所有用户

```
g = Group.objects.get(id=1)
```

```
g.user_set.all()
```

一对一关联

一对一关系

一对一不是数据库的一个连表操作, 而是Django独有的一个连表操作。一对一关系相当于是特殊的一对多关系, 只是相当于加了unique=True。

一个人只能有一张身份证, 一张身份证对应一个人, 是一个典型的一对一关系。

```
class IdCard(models.Model):
```

```
    idnum = models.IntegerField()
```

```
    def __str__(self):
```

```
        return str(self.idnum)
```

```
class Person(models.Model):
```

```
    idcard = models.OneToOneField(IdCard)
```

```
    name = models.CharField(max_length=20)
```

```
    def __str__(self):
```

```
        return self.name
```

一对一关系比较简单。两种表互相都有对方。比如:

```
>>> lisi = Person.objects.get(id=3)
>>> lisi.idcard
<IdCard: 123456>
>>> ids = IdCard.objects.get(id=3)
>>> ids.person
<Person: lisi>
```