Django Model三种继承模型详解

在 Django 中每个 Model 都是一个 Pyhton 类，前文之前提到过 Model 继承自 django.db.models.Model。通过类之间的继承 Django 会对自定义的 Model 自动添加了两个属性分别是 id 和 objects。

在 Model 不指定主键的情况下，Django 会通过 AutoFiled 字段类型将 id 设置为默认自增主键。这里就不加赘述了，在本节我们将从另一个属性 objects 讲起，然后再深入了解 Model 的继承模型。

1. objects查询管理器

objects 是 Manager 类的实例对象，被称为查询管理器，是数据库查询的入口。每一个 Django Model 都至少有一个 Manager 实例，可以通过自定义创建 Manager 以实现对数据库的定制访问，这里我们讲到 Manager 类，它也同样定义在 models 模块中，引入方式如下：

django.db.models.Manager

2. Model的继承模型

Django Model 的继承与 Python 类的继承是一样的，只是 Django 要求所有自定义的 Model 都必须继承自 django.db.models.Model。在 Django 中 Model 之间有三种继承模型，它们分别是抽象基类、多表继承以及代理模型。

1) 抽象基类

抽象类继承的作用是将子表中通用的字段聚合在一起，并将这些字段统一定义在抽象基类中，避免于重复定义这些字段。抽象基类的定义通过在模型的 Meta 中定义属性 abstract=True 来实现。示例如下：

```
from django.db import models

class AbstractBase(models.Model):
    id = models.AutoField()
    content = models.CharField(max_length=100)
    username = models.CharField(max_length=80)
```

```
    nowday = models.DateTimeField()
    class Meta:
        abstract = True

class SomeThing(AbstractBase):
    testexams = models.CharField(max_length=50)

class SomeComment(AbstractBase):
    level = models.CharField(max_length=20)
```

本例中 3 个类映射到数据库后，但会被定义为两个数据表。 分别是 SomeThing 与 SomeComment 它们都继承自 AbstractBase，且继承了父表中的所字段值，同时自身又自定义了新的字段。所以，它们对应的字段分别如下所示：

- SomeThing 数据表：有 id、content、username、nowday、testexams 等 5 个字段；
- SomeComment 数据表：有 id、content、username、nowday、level 等 5 个字段。

关于 Model 的元数据继承关系，遵循以下几个规则：

- 抽象基类中定义的元数据，子类中没有定义，子类会继承基类中的元数据；
- 抽象基类中定义的元数据，子类也定义了，子类优先级更高；
- 子类可以定义自己的元数据，即不出现在抽象基类中的元数据。

在定义抽象基类时，需要注意，如果定义了 ForeignKey 或 ManyToManyField 类型的字段，并且设置了 related_name 或者 related_query_name 参数，由于继承关系，子类也会拥有同样的字段，所以，在子类中的反向名称和查询名称是唯一的。

2) 多表继承

这是 Django 支持的第二种继承方式，因为每个类都是一个完整的 model，而不属于抽象基类，所以父 model 和子 Model 都会有数据库表，而且 Django 默认会给和子表和父表之间自动创建一个 OneToOneField 数据表关系，并且该字段将作为子表的主键。示例如下：

```
from django.db import models
class a(A):
testname=models.charFiled(max_length=255,help_text="测试")
```

如果你想指定链接父类的属性名称，你可以创建你自己的 OneToOneField 字段,并且设置 parent_link=True 从而使用该字段链接父类。

多表继承与抽象基类有一个显著的不同点是 Meta 内部类的继承：子类不会继承父类的 Meta 定义。但是，有两个 Meta 元数据项比较

例，它们分别是 ordering 和 get_latest_by，它们是会被子类继承的，所以，如果不想让它们影响子类的行为，应该覆盖这两个元选项。比如父类有了排序设置，而你并不想让子类有任何排序设置，你可以使用如下方式来禁用子类的排序：

```
class ChildModelName(ParentModelName):
    class Meta:
        ordering = []#子表将不会排序
```

3) 代理模型

代理模型用来给父 Model 添加一些方法或者修改其 Meta 选项，但是父 Model 的字段定义不会被修改。我们可以理解为对原父 Model 进行了 Copy，而被 Copy 出来的 Model 就叫做父 Model 的代理模型，但是这个代理模型又有其自己的特点，这相当于 Python 面向对象中的类继承与多态。

这里需要注意的是代理模型不会在数据库中创建新的数据表，它将使用父 Model 的数据表，即对代理模型的 CURD 操作将会作用到原始的 Model 中。

那么如何创建代理模型呢？在 Meta 类中为我们提供了 proxy 选项。在《Django Meta元数据类属性解析》一节我们曾提到过这个选项，将其设置为 True 即表示创建代理模型。我们通过 BookStore 项目示例进行说明：

```
class BookExtend(Book):
    """
    BOOK代理模型
    """
    class Meta:
        ordering=['id'] #定义Meta选项顺序排序按照id字段
        proxy=True #设置代理模型
    def __str__(self):
        return "title:%s pub:%s price:%s" % (self.title, self.pub, self.price) #定义方法
```

添加完上述代码，然后执行数据库迁移操作。通过查看 MySQL 数据库可以看到并没有新建 BookExtend 数据表。如下所示：

```
mysql> show tables;
+----------------------------+
| Tables_in_bookstoredb      |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
| index_author               |
| index_author_books         |
```

```
| index_book                  |
| index_extenduserinfo        |
| index_pubname               |
| index_userinfo              |
+-----------------------------+
16 rows in set (0.00 sec)
```

提示：最后需要注意的是代理只能继承自一个非抽象的基类，并且不能同时继承多个非抽象基类。