文档地址：

中文：<https://www.liujiangblog.com/course/django/95>

英文：<https://docs.djangoproject.com/zh-hans/3.0/topics/db/models/>

“实用方法：”用于举例说明实现指定目标的实用方法。

# Models：

* 一个模型（model）就是一个单独的、确定的数据的信息源，包含了数据的字段和操作方法。
* 通常，每个模型映射为一张数据库中的表。
* 基本的原则如下：
  + 每个模型在Django中的存在形式为一个Python类；
  + 每个模型都是django.db.models.Model的子类；
  + 模型的每个字段（属性）代表数据表的某一列；
  + Django将自动生成一个访问数据库的API。

## 简单示例：Person模型

**from** **django.db** **import** models

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

first\_name = models.CharField(max\_length=30)

last\_name = models.CharField(max\_length=30)

以上代码等价于：

**CREATE** **TABLE** myapp\_person (

"id" serial **NOT** **NULL** **PRIMARY** **KEY**,

"first\_name" varchar(30) **NOT** **NULL**,

"last\_name" varchar(30) **NOT** **NULL**

);

解析：

1. 在models.py文件中导入models类
2. 使用class关键字定义一个模型，即一个模型表现为一个python类
3. python类名称为Person，指明模型名称。
4. 一个model对应数据库中的一个表。但应该注意，虽然在python中的类名为Person，但在数据库中的表名不一定为Person，与model的元数据metadata有关。示例中显示为myapp\_person.
5. (models.Model)表明模型类都是派生于django.db.models.Model的子类。
6. 模型类的属性对应数据表的一个字段，示例中属性first\_name和last\_name对应各自的字段。
7. id字段是Django自动添加的，作为数据表的主键。
8. CharField指明字段的数据类型。
9. (max\_length=30)括号中的参数指明字段的约束。

## 模型的创建与使用

根据Django框架，model是创建在其所属的应用app下的models.py文件中。

在创建model之后，使用model之前，需要在Django的settings文件中注册当前app。

INSTALLED\_APPS = [

*#...*

'myapp',

*#...*

]

完成注册后，每次对模型进行增、删、修改时，请务必执行命令python manage.py migrate，让操作实际应用到数据库上。这里可以选择在执行migrate之前，先执行python manage.py makemigrations让修改动作保存到记录文件中，方便github等工具的使用。

## 模型字段fields

字段在Python中表现为一个类属性，体现了数据表中的一个列。

### 字段类型Field types

Django使用字段类field class来定义数据表中对应字段，包括：

* 指明在数据表中字段的数据类型，如INTEGER, VARCHAR, TEXT
* 指明在HTML中对应的表单标签的类型，如<input type=“text” />
* 在admin后台和自动生成的表单中最小的数据验证需求

提示：由于字段类field class不但在数据库起作用，还在作用于表单，所以可以在设计模型时适当定义，来定义在表单中的类型。

### 内建的字段类型

|  |  |
| --- | --- |
| 类型 | 说明 |
| AutoField | 一个自动增加的整数类型字段。通常你不需要自己编写它，Django会自动帮你添加字段：  id = models.AutoField(primary\_key=True)，这是一个自增字段，从1开始计数。如果你非要自己设置主键，那么请务必将字段设置为primary\_key=True。  Django在一个模型中只允许有一个自增字段，并且该字段必须为主键！ |
| BigAutoField | (1.10新增)64位整数类型自增字段，数字范围更大，从1到9223372036854775807 |
| BigIntegerField | 64位整数字段（看清楚，非自增），类似IntegerField ，-9223372036854775808 到9223372036854775807。在Django的模板表单里体现为一个textinput标签。 |
| BinaryField | 二进制数据类型。使用受限，少用。 |
| BooleanField | 布尔值类型。默认值是None。  在HTML表单中体现为CheckboxInput标签。  如果要接收null值，请使用NullBooleanField。 |
| CharField | 字符串类型。  必须接收一个max\_length参数，表示字符串长度不能超过该值。  默认的表单标签是input text。最常用的filed，没有之一！ |
| CommaSeparatedIntegerField | 逗号分隔的整数类型。必须接收一个max\_length参数。常用于表示较大的金额数目，例如1,000,000元。 |
| DateField | class DateField(auto\_now=False, auto\_now\_add=False, \*\*options)日期类型。一个Python中的datetime.date的实例。   * 在HTML中表现为TextInput标签。 * 在admin后台中，Django会帮你自动添加一个JS的日历表和一个“Today”快捷方式，以及附加的日期合法性验证。   两个重要参数：（参数互斥，不能共存）   * auto\_now:每当对象被保存时将字段设为当前日期，常用于保存最后修改时间。 * auto\_now\_add：每当对象被创建时，设为当前日期，常用于保存创建日期(注意，它是不可修改的)。   设置上面两个参数就相当于给field添加了editable=False和blank=True属性。  如果想具有修改属性，请用default参数。  例子：pub\_time = models.DateField(auto\_now\_add=True)，自动添加发布时间。 |
| DateTimeField | 日期时间类型。  Python的datetime.datetime的实例。与DateField相比就是多了小时、分和秒的显示，其它功能、参数、用法、默认值等等都一样。 |
| DecimalField | 固定精度的十进制小数。相当于Python的Decimal实例。  必须提供两个指定的参数！   * 参数max\_digits：最大的位数，必须大于或等于小数点位数 。 * decimal\_places：小数点位数，精度。   当localize=False时，它在HTML表现为NumberInput标签，否则是text类型。  例子：储存最大不超过999，带有2位小数位精度的数，定义如下：models.DecimalField(..., max\_digits=5, decimal\_places=2)。 |
| DurationField | 持续时间类型。存储一定期间的时间长度。类似Python中的timedelta。  在不同的数据库实现中有不同的表示方法。常用于进行时间之间的加减运算。但是小心了，这里有坑，PostgreSQL等数据库之间有兼容性问题！ |
| EmailField | 邮箱类型，默认max\_length最大长度254位。  使用这个字段的好处是，可以使用DJango内置的EmailValidator进行邮箱地址合法性验证。 |
| FileField | class FileField(upload\_to=None, max\_length=100, \*\*options)上传文件类型。后面“实用方法”中有介绍。 |
| FilePathField | 文件路径类型， |
| FloatField | 浮点数类型，参考整数类型 |
| ImageField | 图像类型， |
| IntegerField | 整数类型，最常用的字段之一。取值范围-2147483648到2147483647。  在HTML中表现为NumberInput标签。 |
| GenericIPAddressField | class GenericIPAddressField(protocol='both', unpack\_ipv4=False, \*\*options)[source],IPV4或者IPV6地址，字符串形式  例如192.0.2.30或者2a02:42fe::4在HTML中表现为TextInput标签。  参数protocol默认值为‘both’，可选‘IPv4’或者‘IPv6’，表示你的IP地址类型。 |
| NullBooleanField | 类似布尔字段，只不过额外允许NULL作为选项之一。 |
| PositiveIntegerField | 正整数字段，包含0,最大2147483647。 |
| PositiveSmallIntegerField | 较小的正整数字段，从0到32767。 |
| SlugField | slug是一个新闻行业的术语。一个slug就是一个某种东西的简短标签，包含字母、数字、下划线或者连接线，通常用于URLs中。  可以设置max\_length参数，默认为50。 |
| SmallIntegerField | 小整数，包含-32768到32767。 |
| TextField | 大量文本内容，  在HTML中表现为Textarea标签，最常用的字段类型之一！  如果你为它设置一个max\_length参数，那么在前端页面中会受到输入字符数量限制，然而在模型和数据库层面却不受影响。只有CharField才能同时作用于两者。 |
| TimeField | 时间字段，Python中datetime.time的实例。接收同DateField一样的参数，只作用于小时、分和秒。 |
| URLField | 一个用于保存URL地址的字符串类型，默认最大长度200。 |
| UUIDField | 用于保存通用唯一识别码（Universally Unique Identifier）的字段。使用Python的UUID类。在PostgreSQL数据库中保存为uuid类型，其它数据库中为char(32)。这个字段是自增主键的最佳替代品。 |

#### ****UUIDField****

数据库无法自己生成uuid，因此需要如下使用default参数

import uuid # Python的内置模块

from django.db import models

class MyUUIDModel(models.Model):

id = models.UUIDField(primary\_key=True, default=uuid.uuid4, editable=False)

# 其它字段

#### 实用方法：如何上传文件和图片

##### 使用FileField字段类型

* 默认情况下，
* 在HTML中，表现为一个ClearableFileInput标签。
* 在数据库内，实际保存的是一个字符串类型，默认最大长度100，可以通过max\_length参数自定义。
* 真实的文件是保存在服务器的文件系统内的。
* 参数：
* upload\_to用于设置上传地址的目录和文件名。

例子：

class MyModel(models.Model):

# 文件被传至`MEDIA\_ROOT/uploads`目录，MEDIA\_ROOT由你在settings文件中设置

upload = models.FileField(upload\_to='uploads/')

* **Django很人性化地帮我们实现了根据日期生成目录的方式！**

例子：

# 被传到`MEDIA\_ROOT/uploads/2015/01/30`目录，增加了一个时间划分

upload = models.FileField(upload\_to='uploads/%Y/%m/%d/')

* **upload\_to参数也可以接收一个回调函数，该函数返回具体的路径字符串**

def user\_directory\_path(instance, filename):

#文件上传到MEDIA\_ROOT/user\_<id>/<filename>目录中

return 'user\_{0}/{1}'.format(instance.user.id, filename)

class MyModel(models.Model):

upload = models.FileField(upload\_to=user\_directory\_path)

解析：

* 例子中，user\_directory\_path这种回调函数，必须接收两个参数，然后返回指定格式的字符串，用于指明上传文件的路径。
* 参数instace代表一个定义了FileField的模型的实例，说白了就是当前数据记录。filename是原本的文件名。

##### 使用ImageField字段类

用于保存图像文件的字段。

其基本用法和特性与FileField一样，只不过多了两个属性height和width。

**默认情况下：**

* 在HTML中，表现为一个ClearableFileInput标签。
* 在数据库内，我们实际保存的是一个字符串类型，默认最大长度100，可以通过max\_length参数自定义。
* 真实的图片，保存在服务器的文件系统内。

**参数：**

* height\_field参数：保存有图片高度信息的模型字段名。
* width\_field参数：保存有图片宽度信息的模型字段名。

**python使用：**

* **使用Django的ImageField需要提前安装pillow模块，pip install pillow即可。**

**Django使用步骤：**

1. 设置服务器路径

在settings文件中，配置MEDIA\_ROOT，作为你上传文件在服务器中的基本路径（为了性能考虑，这些文件不会被储存在数据库中）。再配置个MEDIA\_URL，作为公用URL，指向上传文件的基本路径。请确保Web服务器的用户账号对该目录具有写的权限。

1. 创建模型

添加FileField或者ImageField字段到你的模型中，定义好upload\_to参数，文件最终会放在MEDIA\_ROOT目录的“upload\_to”子目录中。

* 实质效果：
* 数据库：只存储指向上传文件路径的字符串而已。
* html中：
* 获取文件：通过url属性，

在Django的模板中方便的访问这些文件。例如，假设你有一个ImageField字段，名叫mug\_shot，那么在Django模板的HTML文件中，可以使用{{ object.mug\_shot.url }}来获取该文件。其中的object用你具体的对象名称代替。

* 获取文件的名称和大小信息：可以通过name和size属性，。

**安全建议：**

无论你如何保存上传的文件，一定要注意他们的内容和格式，避免安全漏洞！务必对所有的上传文件进行安全检查，确保它们不出问题！如果你不加任何检查就盲目的让任何人上传文件到你的服务器文档根目录内，比如上传了一个CGI或者PHP脚本，很可能就会被访问的用户执行，这具有致命的危害。

##### 使用FilePathField字段类

一种用来保存文件路径信息的字段。在数据表内以字符串的形式存在，默认最大长度100，可以通过max\_length参数设置。

**参数：**

* path：必须指定的参数。表示一个系统绝对路径。
* match:可选参数，一个正则表达式，用于过滤文件名。只匹配基本文件名，不匹配路径。例如foo.\*\.txt$，只匹配文件名foo23.txt，不匹配bar.txt与foo23.png。
* recursive:可选参数，只能是True或者False。默认为False。决定是否包含子目录，也就是是否递归的意思。
* allow\_files:可选参数，只能是True或者False。默认为True。决定是否应该将文件名包括在内。它和allow\_folders其中，必须有一个为True。
* allow\_folders： 可选参数，只能是True或者False。默认为False。决定是否应该将目录名包括在内。

**例子：**

FilePathField(path="/home/images", match="foo.\*", recursive=True)

解析：

它只匹配/home/images/foo.png，但不匹配/home/images/foo/bar.png，因为默认情况，只匹配文件名，而不管路径是怎么样的。

### 关系类型字段

#### 多对一（ForeignKey）

外键，多对一的关系。

外键字段类的定义如下：

class ForeignKey(to, on\_delete, \*\*options)[source]

**外键要定义在‘多’的一方，指向“一”的另一方。**

**如果需要在尚未定义的模型上创建关系，则可以使用模型的名称，而不是模型对象本身。**

例子：

class Car(models.Model):

manufacturer = models.ForeignKey(

'Manufacturer',

on\_delete=models.CASCADE,

)

# ...

class Manufacturer(models.Model):

# ...

pass

解析：

* 一个制造商可以生产多辆车。在“多”辆车中定义外键。而设置的外键对象类则是“一”个制造商。
* 注意，这里使用了字符串指向外键所在的模型，即模型的名称。而所指向的模型可以是尚未定义。
* 在实际的数据库后台，Django会为每一个外键添加\_id后缀，并以此创建数据表里的一列。在上面的工厂与车的例子中，Car模型对应的数据表中，会有一列叫做manufacturer\_id。但实际上，在Django代码中你不需要使用这个列名，除非你书写原生的SQL语句，一般我们都直接使用字段名manufacturer。
* 注意，在其他字段类型的第一个参数都是verbose\_name参数，用于设置字段的别名。在使用外键字段类型时，第一个参数则是外键所在的类名。

**如果要关联的对象在另外一个app中，可以显式的指出：**

class Car(models.Model):

manufacturer = models.ForeignKey(

'production.Manufacturer', # 关键在这里！！

on\_delete=models.CASCADE,

)

**如果要创建一个递归的外键，也就是自己关联自己的的外键，使用下面的方法：**

models.ForeignKey('self', on\_delete=models.CASCADE)

核心在于‘self’这个引用。什么时候需要自己引用自己的外键呢？

例子：典型的例子就是评论系统！一条评论可以被很多人继续评论

class Comment(models.Model):

title = models.CharField(max\_length=128)

text = models.TextField()

parent\_comment = models.ForeignKey('self', on\_delete=models.CASCADE)

# .....

注意上面的外键字段定义的是父评论，而不是子评论。为什么呢？因为外键要放在‘多’的一方！

**参数：**

* 参数to关联的模型：被关联的类
* 参数on\_delete选项，该选项会设置为必填。当指向的外键模型被删除时，所执行的操作。

on\_delete可选项：

* CASCADE，默认值。当外键模型对象被删除时，同时删除匹配的行。
* PROTECT:阻止上面的删除操作，但是弹出ProtectedError异常
* SET\_NULL：将外键字段设为null，只有当字段设置了null=True时，方可使用该值。
* SET\_DEFAULT:将外键字段设为默认值。只有当字段设置了default参数时，方可使用。
* DO\_NOTHING：什么也不做。
* SET()：设置为一个传递给SET()的值或者一个回调函数的返回值。注意大小写。

例子：在关联的对象被删除时，自动设为null

user = models.ForeignKey(

User,

models.SET\_NULL,

blank=True,

null=True,

)

例子：

from django.conf import settings

from django.contrib.auth import get\_user\_model

from django.db import models

def get\_sentinel\_user():

return get\_user\_model().objects.get\_or\_create(username='deleted')[0]

class MyModel(models.Model):

user = models.ForeignKey(

settings.AUTH\_USER\_MODEL,

on\_delete=models.SET(get\_sentinel\_user),

)

在关联对象被删除时，调用函数get\_sentinel\_user()，将user的值设置为deleted。

* 参数limit\_choices\_to：

该参数用于限制外键所能关联的对象，只能用于Django的ModelForm（Django的表单模块）和admin后台，对其它场合无限制功能。其值可以是一个字典、Q对象或者一个返回字典或Q对象的函数调用，

例子：

staff\_member = models.ForeignKey(

User,

on\_delete=models.CASCADE,

limit\_choices\_to={'is\_staff': True},

)

解析：

* 该处参数是一个字典。
* ModelForm的staff\_member字段列表中，只会出现那些is\_staff=True的Users对象，这一功能对于admin后台非常有用。

例子：使用函数定义

def limit\_pub\_date\_choices():

return {'pub\_date\_\_lte': datetime.date.utcnow()}

# ...

limit\_choices\_to = limit\_pub\_date\_choices

# ...

* 参数related\_name：用于关联对象反向引用模型的名称。

以前面车和工厂的例子解释，就是从工厂反向关联到车的关系名称。

例子略。

* 参数related\_query\_name：反向关联查询名。用于从目标模型反向过滤模型对象的名称。

例子略。

* 参数to\_field：关联到外键对象的字段

默认情况下，外键都是关联到被关联对象的主键上（一般为id）。

如果指定这个参数，可以关联到指定的字段上，但是该字段必须具有unique=True属性，也就是具有唯一属性。

* 参数db\_constraint：

默认情况下，这个参数被设为True，表示遵循数据库约束，这也是大多数情况下你的选择。

如果设为False，那么将无法保证数据的完整性和合法性。在下面的场景中，你可能需要将它设置为False：

* 有历史遗留的不合法数据，没办法的选择
* 你正在分割数据表

当它为False，并且你试图访问一个不存在的关系对象时，会抛出DoesNotExist 异常。

#### 多对多（ManyToManyField）

多对多关系在数据库中也是非常常见的关系类型。比如一本书可以有好几个作者，一个作者也可以写好几本书。

多对多的字段可以定义在任何的一方，请尽量定义在符合人们思维习惯的一方，但不要同时都定义

class ManyToManyField(to, \*\*options)[source]

多对多关系需要一个位置参数：关联的对象模型。它的用法和外键多对一基本类似。

**在数据库后台，Django实际上会额外创建一张用于体现多对多关系的中间表**。默认情况下，该表的名称是“多对多字段名+关联对象模型名+一个独一无二的哈希码”，例如‘author\_books\_9cdf4’，当然你也可以通过db\_table选项，自定义表名。

**参数说明**：

* related\_name：参考外键的相同参数。
* related\_query\_name：参考外键的相同参数。
* limit\_choices\_to：参考外键的相同参数。但是对于使用through参数自定义中间表的多对多字段无效。
* symmetrical：默认情况下，Django中的多对多关系是对称的。
* through：定义中间表。中间表，存储A表对象与B表对象之间关系的表。

如果你想自定义多对多关系的那张额外的关联表，可以使用这个参数！参数的值为一个中间模型。

通常情况下，这张表在数据库内的结构是这个样子的：

中间表的id列....模型对象的id列.....被关联对象的id列

# 各行数据

如果自定义中间表并添加时间字段，则在数据库内的表结构如下：可用于记录两个对象关系的时间序列。

中间表的id列....模型对象的id列.....被关联对象的id列.....时间对象列

# 各行数据

例子：

from django.db import models

class Person(models.Model):

name = models.CharField(max\_length=50)

class Group(models.Model):

name = models.CharField(max\_length=128)

members = models.ManyToManyField(

Person,

through='Membership', ## 自定义中间表

through\_fields=('group', 'person'),

)

class Membership(models.Model): # 这就是具体的中间表模型

group = models.ForeignKey(Group, on\_delete=models.CASCADE)

person = models.ForeignKey(Person, on\_delete=models.CASCADE)

inviter = models.ForeignKey(

Person,

on\_delete=models.CASCADE,

related\_name="membership\_invites",

)

invite\_reason = models.CharField(max\_length=64)

解析：

上面的代码中，通过class Membership(models.Model)定义了一个新的模型，用来保存Person和Group模型的多对多关系，并且同时增加了‘邀请人’和‘邀请原因’的字段。

* 参数through\_fields：

接着上面的例子。Membership模型中包含两个关联Person的外键，Django无法确定到底使用哪个作为和Group关联的对象。所以，在这个例子中，必须显式的指定through\_fields参数，用于定义关系。

through\_fields参数接收一个二元元组('field1', 'field2')，field1是指向定义有多对多关系的模型的外键字段的名称，这里是Membership中的‘group’字段（注意大小写），另外一个则是指向目标模型的外键字段的名称，这里是Membership中的‘person’，而不是‘inviter’。

再通俗的说，就是through\_fields参数指定从中间表模型Membership中选择哪两个字段，作为关系连接字段。

* 参数db\_table：设置中间表的名称。不指定的话，则使用默认值。
* db\_constraint：参考外键的相同参数。

**ManyToManyField多对多字段不支持Django内置的validators验证功能。**

**null参数对ManyToManyField多对多字段无效！设置null=True毫无意义**

#### 一对一（OneToOneField）

一对一关系类型的定义如下：

class OneToOneField(to, on\_delete, parent\_link=False, \*\*options)[source]

从概念上讲，一对一关系非常类似具有unique=True属性的外键关系，但是反向关联对象只有一个。这种关系类型多数用于当一个模型需要从别的模型扩展而来的情况。

比如，Django自带auth模块的User用户表，如果你想在自己的项目里创建用户模型，又想方便的使用Django的认证功能，那么一个比较好的方案就是在你的用户模型里，使用一对一关系，添加一个与auth模块User模型的关联字段。

* 参数to：关联的模型
* 参数related\_name：使用当前模型的小写名作为默认值

例子：

from django.conf import settings

from django.db import models

# 两个字段都使用一对一关联到了Django内置的auth模块中的User模型

class MySpecialUser(models.Model):

user = models.OneToOneField(

settings.AUTH\_USER\_MODEL,

on\_delete=models.CASCADE,

)

supervisor = models.OneToOneField(

settings.AUTH\_USER\_MODEL,

on\_delete=models.CASCADE,

related\_name='supervisor\_of',

)

这样下来，你的User模型将拥有下面的属性：

>>> user = User.objects.get(pk=1)

>>> hasattr(user, 'myspecialuser')

True

>>> hasattr(user, 'supervisor\_of')

True

OneToOneField一对一关系拥有和多对一外键关系一样的额外可选参数，只是多了一个parent\_link参数。

## 字段参数Field option

特定的的字段类型包含特定的的字段参数，如CharField字段的max\_length参数。

通用字段参数：适用于所有字段

### 常用的通用字段参数：

#### **null**

指定数据库是否允许保存空值，默认为false。

对于保存字符串类型数据的字段，请尽量避免将此参数设为True，那样会导致两种‘没有数据’的情况，一种是NULL，另一种是‘空字符串’。

#### **blank**

True时，字段可以为空。默认False。

和null参数不同的是，null是纯数据库层面的，而blank是验证相关的，它与表单验证是否允许输入框内为空有关，与数据库无关。所以要小心一个null为False，blank为True的字段接收到一个空值可能会出bug或异常。

#### **choices**

指定一组可选值。

* 指定的值不一定是列表或元组，只需要是一个序列对象sequence object。
* 指定的值必须是固定的，如果需要使用动态值，最好使用外键。
* 在HTML上的选择框标签（select box）。
* 通常是指定一个二维元组，第一个元素表示存在数据库内真实的值，第二个HTML上显示的内容。最好将选项定义在类里，并取一个直观的名字：

from django.db import models

class Student(models.Model):

FRESHMAN = 'FR'

SOPHOMORE = 'SO'

JUNIOR = 'JR'

SENIOR = 'SR'

YEAR\_IN\_SCHOOL\_CHOICES = (

(FRESHMAN, 'Freshman'),

(SOPHOMORE, 'Sophomore'),

(JUNIOR, 'Junior'),

(SENIOR, 'Senior'),

)

year\_in\_school = models.CharField(

max\_length=2,

choices=YEAR\_IN\_SCHOOL\_CHOICES,

default=FRESHMAN,

)

* 要获取一个choices的第二元素的值，可以使用get\_<FieldName>\_display().<FieldName>是字段名。

例子：

model：

from django.db import models

class Person(models.Model):

SHIRT\_SIZES = (

('S', 'Small'),

('M', 'Medium'),

('L', 'Large'),

)

name = models.CharField(max\_length=60)

shirt\_size = models.CharField(max\_length=1, choices=SHIRT\_SIZES)

python命令：

>>> p = Person(name="Fred Flintstone", shirt\_size="L")

>>> p.save()

>>> p.shirt\_size

'L'

>>> p.get\_shirt\_size\_display()

'Large'

* 可以直接指定一个枚举类。

**from** **django.utils.translation** **import** gettext\_lazy **as** \_

**class** **Student**(models.Model):

**class** **YearInSchool**(models.TextChoices):

FRESHMAN = 'FR', \_('Freshman')

SOPHOMORE = 'SO', \_('Sophomore')

JUNIOR = 'JR', \_('Junior')

SENIOR = 'SR', \_('Senior')

GRADUATE = 'GR', \_('Graduate')

year\_in\_school = models.CharField(

max\_length=2,

choices=YearInSchool.choices,

default=YearInSchool.FRESHMAN,

)

* 可以指定一个命名组，来组织化选项：

MEDIA\_CHOICES = [

('Audio', (

('vinyl', 'Vinyl'),

('cd', 'CD'),

)

),

('Video', (

('vhs', 'VHS Tape'),

('dvd', 'DVD'),

)

),

('unknown', 'Unknown'),

]

解析：

'Audio'是第一个元组的命名，以下如此类推。

unknown元组则不适用命名。

#### **default**

字段的默认值。

可以是值或者一个可调用对象。如果是可调用对象，那么每次创建新对象时都会调用。

设置的默认值不能是一个可变对象，比如列表、集合等等。

lambda匿名函数也不可用于default的调用对象，因为匿名函数不能被migrations序列化。

注意：在某种原因不明的情况下将default设置为None，可能会引发intergyerror：not null constraint failed，即非空约束失败异常，导致python manage.py migrate失败，此时可将None改为False或其它的值，只要不是None就行。

#### **help\_text**

额外显示在表单部件上的帮助文本。使用时请注意转义为纯文本，防止脚本攻击。

#### **primary\_key**

指定当前列为主键。

* 如果你为某个字段设置了primary\_key=True，则当前字段变为主键，并关闭Django自动生成id主键的功能。
* 如果你没有给模型的任何字段设置这个参数为True，Django将自动创建一个AutoField自增字段，名为‘id’，并设置为主键。也就是id = models.AutoField(primary\_key=True)。
* **primary\_key=True隐含null=False和unique=True的意思。一个模型中只能有一个主键字段！**
* 主键字段不可修改，如果你给某个对象的主键赋个新值实际上是创建一个新对象，并不会修改原来的对象：

from django.db import models

class Fruit(models.Model):

name = models.CharField(max\_length=100, primary\_key=True)

###############

>>> fruit = Fruit.objects.create(name='Apple')

>>> fruit.name = 'Pear'

>>> fruit.save()

>>> Fruit.objects.values\_list('name', flat=True)

['Apple', 'Pear']

#### unique

设为True时，在整个数据表内该字段的数据不可重复。

注意：对于ManyToManyField和OneToOneField关系类型，该参数无效。

注意： 当unique=True时，db\_index参数无须设置，因为unqiue隐含了索引。

注意：自1.11版本后，unique参数可以用于FileField字段。

### 多见的通用字段参数：

#### verbose\_name

为字段设置一个人类可读，更加直观的别名。

* 对于每一个字段类型，除了ForeignKey、ManyToManyField和OneToOneField这三个特殊的关系类型，其第一可选位置参数都是verbose\_name。

first\_name = models.CharField("person's first name", max\_length=30)

* 如果没指定这个参数，Django会利用字段的属性名自动创建它，并将下划线转换为空格。

first\_name = models.CharField(max\_length=30) # first name

* 对于外键、多对多和一对一字字段，由于第一个参数需要用来指定关联的模型，因此必须用关键字参数verbose\_name来明确指定。

poll = models.ForeignKey(

Poll,

on\_delete=models.CASCADE,

verbose\_name="the related poll",

)

sites = models.ManyToManyField(Site, verbose\_name="list of sites")

place = models.OneToOneField(

Place,

on\_delete=models.CASCADE,

verbose\_name="related place",

)

#### unique\_for\_date

日期唯一。可能不太好理解。

举个栗子，如果你有一个名叫title的字段，并设置了参数unique\_for\_date="pub\_date"，那么Django将不允许有两个模型对象具备同样的title和pub\_date。有点类似联合约束。

#### editable

如果设为False，那么当前字段将不会在admin后台或者其它的ModelForm表单中显示，同时还会被模型验证功能跳过。参数默认值为True。

#### error\_messages

用于自定义错误信息。参数接收字典类型的值。字典的键可以是null、 blank、 invalid、 invalid\_choice、 unique和unique\_for\_date其中的一个。

#### db\_column

该参数用于定义当前字段在数据表内的列名。如果未指定，Django将使用字段名作为列名。

#### db\_index

该参数接收布尔值。如果为True，数据库将为该字段创建索引。

#### editable

如果设为False，那么当前字段将不会在admin后台或者其它的ModelForm表单中显示，同时还会被模型验证功能跳过。参数默认值为True。

#### validators

运行在该字段上的验证器的列表。

#### unique\_for\_month

同上，只是月份唯一。

#### unique\_for\_year

同上，只是年份唯一。

## 元数据Meta

模型的元数据，指的是“除了字段外的所有内容”，例如排序方式、数据库表名、人类可读的单数或者复数名等等。所有的这些都是非必须的，甚至元数据本身对模型也是非必须的。但是，我要说但是，有些元数据选项能给予你极大的帮助，在实际使用中具有重要的作用，是实际应用的‘必须’。

想在模型中增加元数据，方法很简单，在模型类中添加一个子类，名字是固定的Meta，然后在这个Meta类下面增加各种元数据选项或者说设置项。

例子：

from django.db import models

class Ox(models.Model):

horn\_length = models.IntegerField()

class Meta: # 注意，是模型的子类，要缩进！

ordering = ["horn\_length"]

verbose\_name\_plural = "oxen"

解析：

为模型Ox增加了两个元数据‘ordering’和‘verbose\_name\_plural’，分别表示排序和复数名

* **强调：每个模型都可以有自己的元数据类，每个元数据类也只对自己所在模型起作用。**

### 元数据选项：

#### abstract

如果abstract=True，那么模型会被认为是一个抽象模型。抽象模型本身不实际生成数据库表，而是作为其它模型的父类，被继承使用。具体内容可以参考Django模型的继承。

#### app\_label

如果定义了模型的app没有在INSTALLED\_APPS中注册，则必须通过此元选项声明它属于哪个app，例如：

app\_label = 'myapp'

#### base\_manager\_name

自定义模型的\_base\_manager管理器的名字。模型管理器是Django为模型提供的API所在。

#### db\_table

指定在数据库中，当前模型生成的数据表的表名。比如：

db\_table = 'my\_freinds'

#### db\_tablespace

自定义数据库表空间的名字。默认值是工程的DEFAULT\_TABLESPACE设置。

#### default\_manager\_name

自定义模型的\_default\_manager管理器的名字。

#### default\_related\_name

默认情况下，从一个模型反向关联设置有关系字段的源模型，我们使用<model\_name>\_set，也就是源模型的名字+下划线+set。

这个元数据选项可以让你自定义反向关系名，同时也影响反向查询关系名！

#### get\_latest\_by

Django管理器给我们提供有latest()和earliest()方法，分别表示获取最近一个和最前一个数据对象。但是，如何来判断最近一个和最前面一个呢？也就是根据什么来排序呢？

get\_latest\_by元数据选项帮你解决这个问题，它可以指定一个类似 DateField、DateTimeField或者IntegerField这种可以排序的字段，作为latest()和earliest()方法的排序依据，从而得出最近一个或最前面一个对象。

例子：

get\_latest\_by = "order\_date"

#### managed

不常用。略

#### order\_with\_respect\_to

其用途是根据指定的字段进行排序，通常用于关系字段。

例子：

from django.db import models

class Question(models.Model):

text = models.TextField()

# ...

class Answer(models.Model):

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

# ...

class Meta:

order\_with\_respect\_to = 'question'

解析：

在Answer模型中设置了order\_with\_respect\_to = 'question'。Django会自动提供两个API方法，get\_RELATED\_order()和set\_RELATED\_order()，其中的RELATED用小写的模型名代替。

假设现在有一个Question对象，它关联着多个Answer对象，下面的操作返回包含关联的Anser对象的主键的列表[1,2,3]：

>>> question = Question.objects.get(id=1)

>>> question.get\_answer\_order()

[1, 2, 3]

#### ordering：常用

最常用的元数据之一！

用于指定该模型生成的所有对象的排序方式，接收一个字段名组成的元组或列表。默认按升序排列，如果在字段名前加上字符“-”则表示按降序排列，如果使用字符问号“？”表示随机排列。

例子：

ordering = ['pub\_date'] # 表示按'pub\_date'字段进行升序排列

ordering = ['-pub\_date'] # 表示按'pub\_date'字段进行降序排列

ordering = ['-pub\_date', 'author'] # 表示先按'pub\_date'字段进行降序排列，再按`author`字段进行升序排列。

#### permissions

该元数据用于当创建对象时增加额外的权限。它接收一个所有元素都是二元元组的列表或元组，每个元素都是(权限代码, 直观的权限名称)的格式。比如下面的例子：

permissions = (("can\_deliver\_pizzas", "可以送披萨"),)

#### default\_permissions

Django默认给所有的模型设置('add', 'change', 'delete')的权限，也就是增删改。你可以自定义这个选项，比如设置为一个空列表，表示你不需要默认的权限，但是这一操作必须在执行migrate命令之前。

#### proxy

如果设置了proxy = True，表示使用代理模式的模型继承方式。具体内容与abstract选项一样，参考模型继承章节。

#### required\_db\_features

声明模型依赖的数据库功能。比如['gis\_enabled']，表示模型的建立依赖GIS功能。

#### required\_db\_vendor

声明模型支持的数据库。Django默认支持sqlite, postgresql, mysql, oracle。

#### select\_on\_save

略

#### indexes

接收一个应用在当前模型上的索引列表。

例子：

from django.db import models

class Customer(models.Model):

first\_name = models.CharField(max\_length=100)

last\_name = models.CharField(max\_length=100)

class Meta:

indexes = [

models.Index(fields=['last\_name', 'first\_name']),

models.Index(fields=['first\_name'], name='first\_name\_idx'),

]

#### unique\_together：常用

这个元数据是非常重要的一个！它等同于数据库的联合约束！

例子：

假设有一张用户表，保存有用户的姓名、出生日期、性别和籍贯等等信息。要求是所有的用户唯一不重复，可现在有好几个叫“张伟”的，如何区别它们呢？（不要和我说主键唯一，这里讨论的不是这个问题）

我们可以设置不能有两个用户在同一个地方同一时刻出生并且都叫“张伟”，使用这种联合约束，保证数据库能不能重复添加用户（也不要和我谈小概率问题）。在Django的模型中，如何实现这种约束呢？使用unique\_together，也就是联合唯一！

unique\_together = (('name', 'birth\_day', 'address'),)

这样，哪怕有两个在同一天出生的张伟，但他们的籍贯不同，也就是两个不同的用户。一旦三者都相同，则会被Django拒绝创建。这一元数据经常被用在admin后台，并且强制应用于数据库层面。

unique\_together接收一个二维的元组((xx,xx,xx,...),(),(),()...)，每一个元素都是一个元组，表示一组联合唯一约束，可以同时设置多组约束。

为了方便，对于只有一组约束的情况下，可以简单地使用一维元素，例如：

unique\_together = ('name', 'birth\_day', 'address')

联合唯一无法作用于普通的多对多字段。

#### verbose\_name：常用

最常用的元数据之一！用于设置模型对象的直观、人类可读的名称。可以用中文。例如：

verbose\_name = "story"

verbose\_name = "披萨"

如果你不指定它，那么Django会使用小写的模型名作为默认值。

注意，不是在数据库中的数据表表名，只是指定一个别名，提高可读性。指定在数据库中的表名需要使用db\_table.

#### verbose\_name\_plural

英语有单数和复数形式。这个就是模型对象的复数名，比如“apples”。因为我们中文通常不区分单复数，所以保持和verbose\_name一致也可以。

verbose\_name\_plural = "stories"

verbose\_name\_plural = "披萨"

如果不指定该选项，那么默认的复数名字是verbose\_name加上‘s’

#### label

前面介绍的元数据都是可修改和设置的，但还有两个只读的元数据，label就是其中之一。

label等同于app\_label.object\_name。例如polls.Question，polls是应用名，Question是模型名。

#### label\_lower

同上，不过是小写的模型名。

# END