# 模型M

**蓝色是重点 黄色是辅助说明**

## 模型层需求分析

django框架遵循MVT架构，M（即模型层）**用于实现模型类与数据库中的数据表一一对应（即ORM）**。

定义模型类必须知道模型的字段属性和选项。

模型类定义好之后如何实现对数据表操作？其中最关键的是查询数据表。

一个类有属性和方法，而模型类是为了建立与数据表的映射关系，不应在模型类中定义方法，就需要自定义模型类对应的模型管理器类，可将方法定义于其中。

多个数据表之间存在关系，对应的模型类之间也应当表现出相同的关系，常见的有一对多关系。

## 定义模型类

class BookInfo(models.Model):

'''图书模型类，必须继承models.Model'''  
  
 # 主键ID字段 自动生成  
 btitle = models.CharField(max\_length=20) # 字符串类型，指定最大长度  
 bpub\_date = models.DateField() # 日期类型

**class Meta:**

**db\_table = 'df\_user' # 数据表名**

**verbose\_name = '用户'**

**verbose\_name\_plural = verbose\_name # 后台模型类名**

定义好模型类之后，想要在数据库中生成对应字段的表，需要在项目目录

python manage.py makemigrations生成迁移文件，python manage.py migrate执行迁移文件生成表  
 模型改变，需要重新生成和执行迁移文件，如果先前已经执行过迁移文件，之后再添加模型类再执行时，可能会出现一些问题，根据提示解决

## 模型的字段属性和选项

1、模型类属性命名限制：  
 不能时python保留关键字  
 不允许双下划线，这是由django查询方式决定的  
 定义属性时需要指定字段类型，通过字段类型的参数指定选项，语法如下：  
 属性名=models.字段类型(选项)

2、字段类型（13）：使用时需要导入django.db.models包  
 AutoField 自增长的IntegerField，通常不用指定，不指定时django自动创建名为id的自增长属性  
 BooleanField 布尔字段，值为True或False  
 NullBooleanField 支持Null、True、False三种值  
 CharField 字符串，参数max\_length表示最大长度  
 TextField 大文本字段，一般超过4000字  
 IntegerField 整数  
 DecimalField (decimal十进制的)浮点数，参数max\_digits表示总位，decimal\_places表示小数位数  
 FloatField 浮点数，参数同DecimalField，精确度比不上DecimalField，因为是用科学计数表示法存储  
 DateField 日期，参数auto\_now\_add=False默认,为True时该字段为对象 第一次 创建时的当前时间(字段设置的时间)  
 参数auto\_now=False默认，为True时对象属性值 发生改变时 最后修改 的时间（字段更新的时间）  
 不能一起使用  
 TimeField 时间，参数同DateField  
 DataTimeField 日期时间，参数同DateField  
 FileField 上传文件字段  
 ImageField 继承于FileField，对上传的内容进行校验，确保是有效的图片， upload\_to参数决定上传路径

3、选项（7）：实现对字段进行约束  
 default 默认值  
 primary\_key 为True，该字段为主键,默认False，一般作为AutoField的选项使用  
 unique 为True则字段在表中必须有唯一值，默认False  
 db\_index 为True为此字段创建索引，默认False  
 db\_column 字段的名称，如果未指定，则使用属性名称  
 null 为True时字段允许为 空 ，默认False 数据库范畴  
 blank 为True时字段允许为 空白 ，默认False 与后台管理有关，表单范畴

## 数据表操作

增删改由save()完成，重点是查询

1、查询函数

通过 模型类.objects 属性可以调用如下函数，实现对模型类对应数据表的查询  
 1>get 返回表中满足条件的一条且只能有一条数据返回值是一个模型的类对象（一个模型类对象即是表中一行数据——记录）  
 参数写查询条件如id=1，如果查到多条数据，抛出MultipleObjectsReturned,如果**查不到数据，抛出DoesNotExist** 2>all 返回模型类的对应表格中所有数据 返回值是QuerySet类型（查询集） 支持迭代  
 3>filter 返回 满足 条件的数据 返回值是QuerySet类型 参数写查询条件  
 4>exclude 返回 不满足 条件的数据 返回值是QuerySet类型 参数写查询条件  
 5>order\_by 对查询结果排序 返回值是QuerySet类型 参数写根据 哪些 字段进行排序  
 默认从小到大，从大到小用负号- 如('-id')  
 >>>查询集QuerySet可以继续调用上面5个方法

>>>查询集特性：  
 惰性查询：只有在实际使用查询集中数据时 才对数据库查询 如：books=BookInfo.objects.all()此时不会查询,使用books时才查询  
 缓存：第一次查询后，结果被缓存起来，之后再用这个查询集时，用的是缓存中的结果

2、条件查询（6）

条件格式：模型属性名\_\_条件名=值（双下划线）  
 a)判等 条件名：exact  
 BookInfo.objects.get(id=1)  
 BookInfo.objects.get(id\_\_exact=1)  
 b)模糊查询：  
 1>包含: contains  
 BookInfo.objects.filter(btitle\_\_contains='传')  
 2>以...结尾/开头：endswith/startswith  
 BookInfo.objects.filter(btitle\_\_endswith='部')  
 c)空查询: isnull 值为True/False  
 BookInfo.objects.filter(btitle\_\_isnull=False)  
 d)范围查询：in 列表/元组  
 BookInfo.objects.filter(id\_\_in=[1,3,5])  
 e)比较查询：qt/lt/qte/lte  
 BookInfo.objects.filter(id\_\_qt=3)  
 f)日期查询：  
 年 BookInfo.objects.filter(bpub\_date\_\_year=1980)  
 年月日date(1980,1,1)之后 BookInfo.objects.filter(bpub\_date\_\_qt=date(1980,1,1))

1. Q对象  
    用于查询时条件之间的逻辑关系 not and or 可用Q对象进行 &|~操作  
    使用前导入from django.db.models import Q  
    BookInfo.objects.filter(id\_\_qt=3,bread\_\_qt=30)等同(Q(id\_\_qt=3)&Q(bread\_\_qt=30))

BookInfo.objects.filter(~Q(id=3))  
   
4、F对象  
 用于类属性之间的比较  
 from django.db.models import F  
 查询图书 阅读量大于评论量 BookInfo.objects.filter(bread\_\_qt=F('comment'))  
 阅读量大于2倍评论量

(bread\_\_qt=F('comment')\*2)  
  
5、聚合函数 数据库中：sum count avg max min  
 用aggregate函数使用聚合， 返回值是一个 字典  
 from django.db.models import Sum,Count,Max,Min,Avg  
 BookInfo.objects.all().aggregate(Count('id')) 必须写具体属性，不能是\*  
 count函数 返回一个数字  
 BookInfo.objects.all().count()

## 模型管理器对象

1、BookInfo.objects.all()-->objects是一个什么东西？  
 objects是一个django自动生成的管理器对象，通过管理器可以实现对数据的查询  
 objects是models.Manager类的一个对象，自定义管理器之后django不再生成默认管理器对象  
2、自定义模型类对应的管理器类，继承models.Manager类，再在具体的模型类中定义一个自定义管理器类的对象  
 1>改变查询结果集：  
 class BookInfo(models.Model):  
 book = models.Manager() # 仅仅这样没什么用  
 之后查询通过BookInfo.book.all()  
  
 class BookInfoManager(models.Manager):  
 # 改变查询的结果集  
 def all(self):  
 调用父类all，获取所有数据  
 books = super().all() # QuerySet  
 books = books.filter(isDlete=False) # 这样就只是查出删除标识为False的数据  
  
 class BookInfo(models.Model):  
 book1 = BookInfoManager()  
  
 2>添加额外的方法：  
 class BookInfo(models.Model):  
  
 @classmethod  
 def create\_book(cls, btitle, bpub\_date):  
 obj = cls() # 因为是类方法，cls是类名，用于创建实例对象  
 obj.btitle = btitle  
 obj.bpub\_date = bpub\_date  
 obj.save()  
 return obj # 可以继续调用对象.属性查看  
 >>>出现的问题：模型类是为了与数据表相对应，不应该添加太多方法  
  
 class BookInfoManager(models.Manager):  
 封装函数，操作模型类对象的数据表  
 def create\_book(self, btitle, bpub\_date):  
 # book = BookInfo()  
 model\_class = self.model # 获取所在的模型类 解耦作用 模型类名改变，这里也不用改变  
 book = model\_class()  
 book.btitle = btitle  
 book.bpub\_date = bpub\_date  
 book.save()  
 return book  
 >>>其实管理器类中已经定义有creat()函数，只不过参数必须是关键字参数

模型管理器类中查询时

try:

user = super().get(......)

except DoesNotExist as e:

user = None

## 模型类之间的关系

**一对多关系（重点）**  
 如图书类--英雄类  
 models.ForeignKey() 定义在多的类中  
  
**多对多关系**  
 新闻类--新闻类型类  
 models.ManyToManyField() 定义在哪个类中都可以  
  
**一对一关系**  
 员工基本信息类--员工工号  
 models.OneToOneField() 定义在哪个类中都可以

关联查询

1、通过对象查询（一对多）  
 1> 一类 多类 多类中建立关联的类属性叫关联属性  
 查询id=1的图书关联的英雄的信息（一查多）  
 b=BookInfo.object.get(id=1)  
 b.heroinfo\_set.all()  
 查询id=1的英雄关联的图书信息（多查一）  
 h=HeroInfo.object.get(id=1)  
 h.hbook （直接用关联属性）  
 2>多查一：多类的对象.关联属性  
 一查多：一类的对象.多类名小写\_set.all()  
  
2、通过模型 类 实现关联查询  
 查询图书信息，要求图书关联的英雄的描述包含‘八’  
 BookInfo.objects.filter(heroinfo\_\_hcomment\_\_contains='八') 双下划线  
 查询书名为‘天龙八部’的所有英雄  
 HeroInfo.objects.filter(hbook\_\_btitle='天龙八部')  
 通过多类的条件查询一类：（多查一）  
 一类名.objects.filter(多类名小写\_\_多类属性名\_\_条件名)  
 通过一类的条件查询多类：（一查多）  
 多类名.objects.filter(关联属性\_\_一类属性名\_\_条件名)

关联属性确定唯一实例对象（即一行数据）

自关联是一种特殊的一对多，在一张表里面