04 Django模型基础

一. models 字段类型

```
# 概述
  django根据属性的类型确定以下信息
    ·当前选择的数据库支持字段的类型
    ·渲染管理表单时使用的默认html控件
    ·在管理站点最低限度的验证
  django会为表增加自动增长的主键列，每个模型只能有一个主键列，如果使用选项设置某属性为主键列后，
      则django不会再生成默认的主键列

# 属性命名限制
  ·遵循标识符规则
  ·由于django的查询方式，不允许使用连续的下划线
      定义属性时，需要字段类型，字段类型被定义在django.db.models.fields目录下，为了方便使用，
        被导入到django.db.models中

# 使用方式
  ·导入from django.db import models
  ·通过models.Field创建字段类型的对象，赋值给属性

# 逻辑删除和物理删除
  对于重要数据都做逻辑删除，不做物理删除，实现方法是定义is_delete属性，类型为BooleanField，
    默认值为False
  is_delete = models.BooleanField(default=False)

# 常用字段类型：
  ·AutoField
  ·一个根据实际ID自动增长的IntegerField，通常不指定，如果不指定，主键字段id将自动添加到模型中

  ·CharField(max_length=字符长度)
  ·字符串，默认的表单样式是 Input

  ·TextField
  ·大文本字段，一般超过4000使用，默认的表单控件是Textarea

  ·IntegerField
  ·整数

  ·DecimalField(max_digits=None, decimal_places=None)
  ·使用python的Decimal实例表示的十进制浮点数
  ·参数说明
    ·DecimalField.max_digits
      ·位数总数
    ·DecimalField.decimal_places
      ·小数点后的数字位数
```

```
·FloatField
·用Python的float实例来表示的浮点数

·BooleanField
·True/False 字段，此字段的默认表单控制是CheckboxInput

·DateField([auto_now=False, auto_now_add=False])
·使用Python的datetime.date实例表示的日期
·参数说明
  ·DateField.auto_now
    ·每次保存对象时，自动设置该字段为当前时间，用于"最后一次修改"的时间戳，它总是使用当前日期，
        默认为false
  ·DateField.auto_now_add
    ·当对象第一次被创建时自动设置当前时间，用于创建的时间戳，它总是使用创建时的日期，
          默认为false
  注意：auto_now_add, auto_now, and default 这些设置是相互排斥的，
        他们之间的任何组合将会发生错误的结果

·TimeField
·使用Python的datetime.time实例表示的时间，参数同DateField

·DateTimeField
·使用Python的datetime.datetime实例表示的日期和时间，参数同DateField

·FileField
·一个上传文件的字段

·ImageField
·继承了FileField的所有属性和方法，但对上传的对象进行校验，确保它是个有效的image
    需要安装Pillow：  "pip install Pillow"
```

二. 常用字段参数

```
# 常用字段选项（通过字段选项，可以实现对字段的约束）：
  1、 null=True
      数据库中字段是否可以为空
  2、 blank=True
      django的 Admin 中添加数据时是否可允许空值

  一般null=True & blank=True 搭配着用，出现null=True就用上blank=True

  3、 primary_key = True
      主键，对AutoField设置主键后，就会代替原来的自增 id 列
  4、 auto_now 和 auto_now_add
      auto_now    自动创建---无论添加或修改，都是当前操作的时间
```

```
    auto_now_add  自动创建---永远是创建时的时间
5、 choices   (后台admin下拉菜单)
    USER_TYPE_LIST = (
        (1, '超级用户'),
        (2, '普通用户'),
    )
    user_type =  models.IntegerField(choices=USER_TYPE_LIST,
                                     default=1,
                                     verbose_name='用户类型')
6、 max_length 最大长度
7、 default    默认值
8、 verbose_name    Admin（后台显示的名称）中字段的显示名称
9、 name|db_column    数据库中的字段名称
10、unique=True    不允许重复
11、db_index = True    数据库索引,例如：如果你想通过name查询的更快的话，给他设置为索引即可
12、editable=True  在Admin里是否可编辑，不可编辑则不显示
  13、设置表名
    class Meta:
        db_table = 'person'
```

三. models基本操作

一般的数据库操作流程：

1. 创建数据库，设计表结构和字段
2. 连接Mysql数据库，并编写数据访问层代码
3. 业务逻辑层去调用数据访问层执行数据库操作

Django通过Model操作数据库，不管你数据库的类型是MySql或者Sqlite，Django自动帮你生成相应数据库类型的SQL语句，所以不需要关注SQL语句和类型，对数据的操作Django帮我们自动完成。只要会写Model就可以了。

django使用对象关系映射（Object Relational Mapping，简称ORM）框架去操控数据库。

ORM(Object Relational Mapping)对象关系映射，是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。

增删改查

```
# ORM:
    模型    <=>     表
    类结构   ->    表结构
    对象     ->    表的一条数据
    类属性   ->    表的字段

# models基本操作
# 增:
  1）创建对象实例，然后调用save方法：
        obj = Author()
```

```
        obj.first_name = 'zhang'
          obj.last_name = 'san'
        obj.save()
```

2）创建对象并初始化,再调用save方法：

```
        obj = Author(first_name='zhang', last_name='san')
        obj.save()
```

3）使用create方法

```
        Author.objects.create(first_name='li', last_name='si')
```

4）使用get_or_create方法，可以防止重复

```
        Author.objects.get_or_create(first_name='zhang', last_name='san')
```

```
# 删：
```

使用Queryset的delete方法：

```
    # 删除指定条件的数据
    Author.objects.filter(first_name='zhang').delete()
    # 删除所有数据
    Author.objects.all().delete()
```

注意： objects不能直接调用delete方法。

使用模型对象的delete方法：

```
    obj = Author.objects.get(id=5)
    obj.delete()
```

```
# 改：
```

```
Author.objects.filter(last_name='dfdf').update(last_name='san')
```

模型没有定义update方法，直接给字段赋值，并调用save，能实现update的功能，比如：

```
    obj = Author.objects.get(id=3)
    obj.first_name = 'zhang'
    obj.save()
```

save更新时会更新所有字段。如果只想更新某个字段，减少数据库操作，可以这么做：

```
    obj.first_name = 'li'
    obj.save(update_fields=['first_name'])
```

```
# 查：
```

get()：获取单条数据：

```
        Author.objects.get(id=123)
```

如果没有找到符合条件的对象，会引发模型类.DoesNotExist异常

如果找到多个，会引发模型类.MultipleObjectsReturned 异常

first()：返回查询集(QuerySet)中的第一个对象

last()：返回查询集中的最后一个对象

count()：返回当前查询集中的对象个数

exists()：判断查询集中是否有数据，如果有数据返回True没有反之

all()：获取全部数据：

```
        Author.objects.all()
```

values()：获取指定列的值，可以传多个参数！返回包含字典的列表（保存了字段名和对应的值）

```
    Author.objects.all().values('password')
```

values_list()：获取指定列的值，可以传多个参数！返回包含元组列表（只保存值）

```
        Author.objects.all().values_list('password')
```

进阶操作：

```
# 获取个数
Author.objects.filter(name='seven').count()

Author.objects.filter(id__gt=1)      # 获取id大于1的值
  # select * from Author where id > 1

Author.objects.filter(id__gte=1)     # 获取id大于或等于1的值
  # select * from Author where id >= 1

Author.objects.filter(id__lt=10)     # 获取id小于10的值
  # select * from Author where id < 10

Author.objects.filter(id__lte=10)    # 获取id小于或等于10的值
  # select * from Author where id <= 10

Author.objects.filter(id__lt=10, id__gt=1)   # 获取id大于1 且 小于10的值
  # select * from Author where id < 10 and id > 1

Author.objects.filter(id__in=[11, 22, 33])   # 获取id在11、22、33中的数据
  # select * from Author where id in (11,22,33)

Author.objects.exclude(id__in=[11, 22, 33])  # not in
  # select * from Author where id not in (11,22,33)

  Author.objects.filter(name__contains="ven")  # contains（和数据库中like语法相同）

  # select * from Author where name like '%ven%'

Author.objects.filter(name__icontains="ven") # icontains大小写不敏感

  Author.objects.filter(name__regex="^ven")  # 正则匹配
  Author.objects.filter(name__iregex="^ven") # 正则匹配,忽略大小写
Author.objects.filter(age__range=[10, 20])   # 范围bettwen and

    # startswith, istartswith, endswith, iendswith:
    #  以什么开始，以什么结束，和上面一样带i的是大小写不敏感的，  其实不带i的也忽略大小写

Author.objects.filter(name='seven').order_by('id')    # asc升序
Author.objects.filter(name='seven').order_by('-id')   # desc降序

Author.objects.all()[10:20]  # 切片，取所有数据的10条到20条，分页的时候用的到，
                             # 下标从0开始，不能为负数，可以实现分页

    # 手动分页
    page 页码
    per_page 每页数量 =5
    第1页(page=1)：0-4   => [0:5]
    第2页(page=2)：5-9    => [5:10]
```

```
        第3页(page=3): 10-14  => [10:15]
        第4页(page=4): 15-19  => [15:20]
            ...
        每一页数据范围:
          [(page-1)*per_page: page*per_page]

# 聚合
     使用aggregate()函数返回聚合函数的值
    Avg: 平均值
    Count: 数量
    Max: 最大
    Min: 最小
    Sum: 求和
     from django.db.models import Count, Min, Max, Sum
   Author.objects.aggregate(Max('age'))
```