Django 模型（数据库）

Django 模型是与数据库相关的，与数据库相关的代码一般写在 models.py 中，Django 支持 sqlite3, MySQL, PostgreSQL等数据库，只需要在settings.py中配置即可，不用更改models.py中的代码，丰富的API极大的方便了使用。

**1. 新建项目和应用**

|  |  |
| --- | --- |
| 1  2  3 | django-admin.py startproject learn\_models # 新建一个项目  cd learn\_models # 进入到该项目的文件夹  django-admin.py startapp people # 新建一个 people 应用（app) |

补充：新建app也可以用 python manage.py startapp people, 需要指出的是，django-admin.py 是安装Django后多出的一个命令，并不是运行的当前目录下的django-admin.py（当前目录下也没有），但创建项目会生成一个 manage.py 文件。

那project和app什么关系呢？

一个项目一般包含多个应用，一个应用也可以用在多个项目中。

**2. 添加应用**

将我们新建的应用（people）添加到 settings.py 中的 INSTALLED\_APPS中，也就是告诉Django有这么一个应用。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | INSTALLED\_APPS = (      'django.contrib.admin',      'django.contrib.auth',      'django.contrib.contenttypes',      'django.contrib.sessions',      'django.contrib.messages',      'django.contrib.staticfiles',        'people',  ) |

**3. 修改models.py**

我们打开 **people/models.py**文件，修改其中的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | from django.db import models      class Person(models.Model):      name = models.CharField(max\_length=30)      age = models.IntegerField() |

我们新建了一个Person类，继承自models.Model, 一个人有姓名和年龄。

这里用到了两种**Field，**更多Field类型可以参考教程最后的链接。

**4. 创建数据表**

我们来**同步一下数据库（我们使用默认的数据库 SQLite3，无需配置）**

|  |  |
| --- | --- |
|  | 先 cd 进入 manage.py 所在的那个文件夹下，输入下面的命令  python manage.py makemigrations  python manage.py migrate |

**5. 使用 Django 提供的 QuerySet API**

Django提供了丰富的API, 下面演示如何使用它。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ python manage.py shell    >>> from people.models import Person  >>> Person.objects.create(name="WeizhongTu", age=24)  <Person: Person object>  >>> |

我们新建了一个用户WeizhongTu 那么如何从数据库是查询到它呢？

|  |  |
| --- | --- |
| 1  2  3 | >>> Person.objects.get(name="WeizhongTu")  <Person: Person object>  >>> |

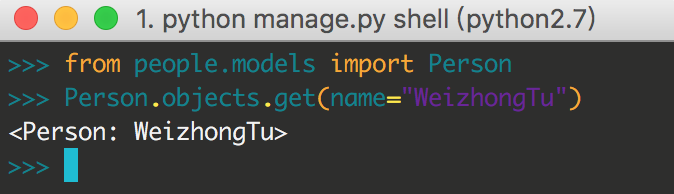
我们用了一个 .objects.get() 方法查询出来符合条件的对象，但是大家注意到了没有，查询结果中显示<Person: **Person object**>，这里并没有显示出与WeizhongTu的相关信息，如果用户多了就无法知道查询出来的到底是谁，查询结果是否正确，我们重新修改一下 people/models.py

name 和 age 等字段中不能有 \_\_（双下划线，因为在Django QuerySet API中有特殊含义（用于关系，包含，不区分大小写，以什么开头或结尾，日期的大于小于，正则等）

也不能有Python中的关键字，name 是合法的，student\_name 也合法，但是student\_\_name不合法，try, class, continue 也不合法，因为它是Python的关键字( import keyword; print(keyword.kwlist) 可以打出所有的关键字)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | from django.db import models      class Person(models.Model):      name = models.CharField(max\_length=30)      age = models.IntegerField()        def \_\_unicode\_\_(self):      # 在Python3中使用 def \_\_str\_\_(self):          return self.name |

按 CTRL + C 退出当前的 Python shell, 重复上面的操作，我们就可以看到:



**新建一个对象的方法有以下几种：**

1. Person.objects.create(name=name,age=age)
2. p = Person(name="WZ", age=23)

p.save()

1. p = Person(name="TWZ")

p.age = 23

p.save()

1. Person.objects.get\_or\_create(name="WZT", age=23)

这种方法是防止重复很好的方法，但是速度要相对慢些，返回一个元组，第一个为Person对象，第二个为True或False, 新建时返回的是True, 已经存在时返回False.

**获取对象有以下方法：**

1. Person.objects.all()
2. Person.objects.all()[:10] 切片操作，获取10个人，不支持负索引，切片可以节约内存
3. Person.objects.get(name=name)

get是用来获取一个对象的，如果需要获取满足条件的一些人，就要用到filter

1. Person.objects.filter(name="abc")  # 等于
2. Person.objects.filter(name\_\_iexact="abc")  # 名称为 abc 但是不区分大小写，可以找到 ABC, Abc, aBC，这些都符合条件
3. Person.objects.filter(name\_\_contains="abc")  # 名称中包含 "abc"的人
4. Person.objects.filter(name\_\_regex="^abc")  # 正则表达式查询
5. Person.objects.filter(name\_\_iregex="^abc")  # 正则表达式不区分大小写

filter是找出满足条件的，当然也有排除符合某条件的

1. Person.objects.exclude(name\_\_contains="WZ")  # 排除包含 WZ 的Person对象
2. Person.objects.filter(name\_\_contains="abc").exclude(age=23)  # 找出名称含有abc, 但是排除年龄是23岁的

**2. 获取对象的方法**

|  |  |
| --- | --- |
|  | Person.objects.all() # 查询所有  Person.objects.all()[:10] 切片操作，获取10个人，不支持负索引，切片可以节约内存，不支持负索引  Person.objects.get(name="WeizhongTu") # 名称为 WeizhongTu 的一条，多条会报错    get是用来获取一个对象的，如果需要获取满足条件的一些人，就要用到filter  Person.objects.filter(name="abc") # 等于Person.objects.filter(name\_\_exact="abc")  严格等于 "abc" 的人  Person.objects.filter(name\_\_iexact="abc") # 名称为 abc 但是不区分大小写，  可以找到 ABC, Abc, aBC，这些都符合条件    Person.objects.filter(name\_\_contains="abc") # 名称中包含 "abc"的人  Person.objects.filter(name\_\_icontains="abc") #名称中包含 "abc"，且abc不区分大小写    Person.objects.filter(name\_\_regex="^abc") # 正则表达式查询  Person.objects.filter(name\_\_iregex="^abc")# 正则表达式不区分大小写    # filter是找出满足条件的，当然也有排除符合某条件的  Person.objects.exclude(name\_\_contains="WZ") # 排除包含 WZ 的Person对象  Person.objects.filter(name\_\_contains="abc").exclude(age=23) #  找出名称含有abc, 但是排除年龄是23岁的 |
|  |  |
|  |  |
|  |  |

**3. 删除符合条件的结果**

**和上面类似，得到满足条件的结果，然后 delete 就可以(危险操作，正式场合操作务必谨慎)，比如：**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Person.objects.filter(name\_\_contains="abc").delete() # 删除 名称中包含 "abc"的人    如果写成  people = Person.objects.filter(name\_\_contains="abc")  people.delete()  效果也是一样的，Django实际只执行一条 SQL 语句。 |

**4. 更新某个内容**

**(1) 批量更新，适用于 .all()  .filter()  .exclude() 等后面 (危险操作，正式场合操作务必谨慎)**

|  |  |
| --- | --- |
|  | Person.objects.filter(name\_\_contains="abc").update(name='xxx') #  名称中包含 "abc"的人 都改成 xxx  Person.objects.all().delete() # 删除所有 Person 记录 |

**(2) 单个 object 更新，适合于 .get(), get\_or\_create(), update\_or\_create() 等得到的 obj，和新建很类似。**

|  |  |
| --- | --- |
| 1  2  3  4 | twz = Author.objects.get(name="WeizhongTu")  twz.name="WeizhongTu"  twz.email="tuweizhong@163.com"  twz.save()  # 最后不要忘了保存！！！ |

**5. QuerySet 是可迭代的，比如：**

|  |  |
| --- | --- |
| 1  2  3 | es = Entry.objects.all()  for e in es:      print(e.headline) |

Entry.objects.all() 或者 es 就是 QuerySet 是查询所有的 Entry 条目。

注意事项：

(1). 如果只是检查 Entry 中是否有对象，应该用 Entry.objects.all().**exists()**

**(2). QuerySet 支持切片 Entry.objects.all()[:10] 取出10条，可以节省内存**

**(3). 用 len(es) 可以得到Entry的数量，但是推荐用 Entry.objects.count()来查询数量，后者用的是SQL：SELECT COUNT(\*)**

**(4). list(es) 可以强行将 QuerySet 变成 列表**

**6. QuerySet 查询结果排序**

**作者按照名称排序**

|  |  |
| --- | --- |
|  | Author.objects.all().order\_by('name')  Author.objects.all().order\_by('-name') # 在 column name 前加一个负号，可以实现倒序 |

7**. QuerySet 支持链式查询**

|  |  |
| --- | --- |
|  | Author.objects.filter(name\_\_contains="WeizhongTu").filter(email="123@163.com")  Author.objects.filter(name\_\_contains="Wei").exclude(email="123@163.com")    # 找出名称含有abc, 但是排除年龄是23岁的  Person.objects.filter(name\_\_contains="abc").exclude(age=23) |

8**. QuerySet 不支持负索引**

|  |  |
| --- | --- |
|  | Person.objects.all()[:10] 切片操作，前10条  Person.objects.all()[-10:] 会报错！！！    # 1. 使用 reverse() 解决  Person.objects.all().reverse()[:2] # 最后两条  Person.objects.all().reverse()[0] # 最后一条    # 2. 使用 order\_by，在栏目名（column name）前加一个负号  Author.objects.order\_by('-id')[:20] # id最大的20条 |

9**. QuerySet 重复的问题，使用 .distinct() 去重**

**一般的情况下，QuerySet 中不会出来重复的，重复是很罕见的，但是当跨越多张表进行检索后，结果并到一起，可以会出来重复的值**

|  |  |
| --- | --- |
|  | # 去重方法  People.objects.filter(age=0).values('name').distinct() |

按照name字段去重，如果多个字段则继续在values中加参数。

10**. 聚合查询**

from django.db.models import Avg

from django.db.models import Sum

from django.db.models import Count