

《Django 教程》

- 讲师: 魏明择
- 时间: 2019

目录

- 静态文件
- Django中的应用 - app
 - 创建应用app
 - 应用的分布式路由
 - django.conf.urls 里的include 函数
- 数据库 和 模型
 - Django下使用mysql数据库
- 模型 (Models)
- Python 数据库模型 - Models
 - 字段选项
 - 数据库迁移的错误处理方法
- 数据库的操作(CRUD操作)
 - 管理器对象
 - 创建数据对象
 - Django shell 的使用
 - 查询数据

静态文件

1. 什么是静态文件
 - 不能与服务端做动态交互的文件都是静态文件
 - 如:图片,css,js, 音频,视频,html文件(部分)
2. 静态文件配置
 - 在 settings.py 中配置一下两项内容:
 1. 配置静态文件的访问路径
 - 通过哪个url地址找静态文件
 - STATIC_URL = '/static/'
 - 说明:
 - 指定访问静态文件时是需要通过 /static/xxx或 127.0.0.1:8000/static/xxx
 - xxx 表示具体的静态资源位置
 2. 配置静态文件的存储路径
 - 静态文件在服务器端的保存位置
 - STATICFILES_DIRS=(os.path.join(BASE_DIR,'static'),)
 3. 示例:

```
# file: setting.py
STATICFILES_DIRS = [
```

```
os.path.join(BASE_DIR, "static")
]
```

3. 访问静态文件


1. 使用静态文件的访问路径进行访问

- 访问路径: STATIC_URL=/static/
- 示例:

```


```

2. 通过 {% static %} 标签访问静态文件 {% static %} 表示的就是静态文件访问路径

1. 加载 static {% load static %}
 2. 使用静态资源时 语法:{% static '静态资源路径' %} 
- 示例:
 - 路由配置文件

```
# file: url.py
from . import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^show_image', views.show_image)
]
- 视图函数文件
# file: views.py
from django.shortcuts import render

def show_image(request):
    return render(request, "show_image.html")
```

- 模板文件

```
<html>
<head></head>
<body>
<h1>this is lena!</h1>

<h1>this is templates lena!</h1>
{% load static %}

</body>
</html>
```

- 练习:

1. 127.0.0.1:8000 : 显示首页效果
 2. 127.0.0.1:8000/login : 显示登录页
- 处理好所有的静态文件

Django中的应用 - app

- 应用在Django项目中是一个独立的业务模块,可以包含自己的路由,视图,模板,模型

创建应用app

- 创建步骤
 1. 用manage.py 中的子命令 startapp 创建应用文件夹
 2. 在settings.py 的 INSTALLED_APPS 列表中配置安装此应用
- 创建应用的子命令
 - python3 manage.py startapp 应用名称(必须是标识符命令规则)
 - 如:
 - python3 manage.py startapp music
- Django应用的结构组成
 1. migrations 文件夹
 - 保存数据迁移的中间文件
 2. __init__.py
 - 应用子包的初始化文件
 3. admin.py
 - 应用的后台管理配置文件
 4. apps.py
 - 应用的属性配置文件
 5. models.py
 - 与数据库相关的模型映射类文件
 6. tests.py
 - 应用的单元测试文件
 7. views.py
 - 定义视图处理函数的文件
- 配置安装应用
 - 在 settings.py 中配置应用, 让此应用能和整个项目融为一体

```
# file : settings.py
INSTALLED_APPS = [
    ... ..,
    '自定义应用名称'
```

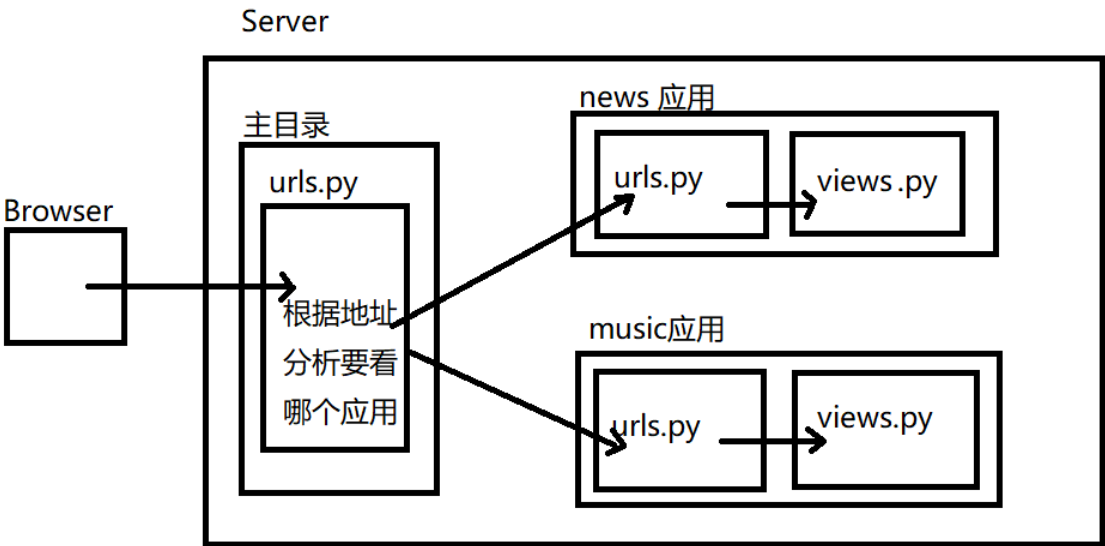
```
]
```

如:

```
INSTALLED_APPS = [  
    # ....  
    'user', # 用户信息模块  
    'music', # 收藏模块  
]
```

应用的分布式路由

- Django中，主文件夹可以不处理用户具体请求的，主文件夹的作用是做项目的初始化以及请求的分发(分布式请求处理)。具体的请求可以由应用来进行处理的
- 如图:



o

django.conf.urls 里的include 函数

- 作用:
 - 用于分发将当前路由转到模块的 urlpatterns 进行分布式处理
- 函数格式
 - include('App名字.url模块名')
- 参数说明:
 - 模块App名字/url模块名.py 文件里必须有urlpatterns 列表
- 使用 include 函数让某个正则匹配后关联分支到某个app示例如下:

```
# file : <项目名>/urls.py
from django.conf.urls import include

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^music/', include('music.urls')),
]
```

```
# file : music/urls.py
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^page1', views.page1),
    url(r'^page2', views.page2),
    url(r'^page3', views.page3),
    # ...
]
```

- 练习:

1. 创建四个应用
 1. 创建 index 应用, 并注册
 2. 创建 sport 应用, 并注册
 3. 创建 news 应用, 并注册
 4. 创建 music 应用, 并注册
2. 创建分布式路由系统

主路由配置只做分发

每个应用中处理具体访问路径和视图

 1. 127.0.0.1:8000/music/index
交给 music 应用中的 index_view() 函数处理
 2. 127.0.0.1:8000/sport/index
交给 sport 应用中的 index_view() 函数处理
 3. 127.0.0.1:8000/news/index
交给 news 应用中的 index_view() 处理处理

数据库 和 模型

Django下使用mysql数据库

1. 安装 pymysql包

- 用作 python 和 mysql 的接口
 - `$ sudo pip3 install pymysql`
- 安装 mysql 客户端(非必须) `$ sudo pip3 install mysqlclient`

2. 创建 和 配置数据库

1. 创建数据库

- 创建 `create database` 数据库名 `default charset utf8 collate utf8_general_ci`;

```
create database mywebdb default charset utf8 collate
utf8_general_ci;
```

2. 数据库的配置

- sqlite 数据库配置

```
# file: settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

- mysql 数据库配置

```
DATABASES = {
    'default' : {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mywebdb', # 数据库名称,需要自己定义
        'USER': 'root',
        'PASSWORD': '123456', # 管理员密码
        'HOST': '127.0.0.1',
        'PORT': 3306,
    }
}
```

3. 关于数据库的SETTING设置

1. ENGINE

- 指定数据库的后端引擎

```
'django.db.backends.mysql'
'django.db.backends.sqlite3'
'django.db.backends.oracle'
'django.db.backends.postgresql'
```

- mysql引擎如下:

- 'django.db.backends.mysql'

2. NAME

- 指定要连接的数据库的名称
- 'NAME': 'mywebdb'

3. USER

- 指定登录到数据库的用户名
- 'USER': 'root'

4. PASSWORD

- 接数据库时使用的密码。
- 'PASSWORD': '123456'

5. HOST

- 连接数据库时使用哪个主机。
- 'HOST': '127.0.0.1'

6. PORT

- 连接数据库时使用的端口。
- 'PORT': '3306'

4. 添加 mysql 支持

- 安装pymysql 模块
 - `$ sudo pip install pymysql`
- 修改项目中__init__.py 加入如下内容来提供pymysql引擎的支持

```
import pymysql
pymysql.install_as_MySQLdb()
```

3. 数据库的迁移

- 迁移是Django同步您对模型所做更改（添加字段，删除模型等）到您的数据库模式的方式

1. 生成或更新迁移文件

- 将每个应用下的models.py文件生成一个中间文件,并保存在migrations文件夹中
- `python3 manage.py makemigrations`

2. 执行迁移脚本程序

- 执行迁移程序实现迁移。将每个应用下的migrations目录中的中间文件同步回数据库
- `python3 manage.py migrate`

模型 (Models)

- 模型是提供数据信息的数据库接口。
- 模型是数据的唯一的、确定的信息源。它包含你所储存数据的必要字段和行为。

- 通常，每个模型对应数据库中唯一的一张表。每个模型的实例对应数据表中的一条记录
- 模型说明：
 - 每个模型都是一个Python类，每个模型都是django.db.models.Model的子类。
 - 每一个模型属性都代表数据库中的一个表。
 - 通过所有这一切，Django为你提供一个自动生成的数据库访问API；

Python 数据库模型 - Models

1. ORM框架

- ORM（Object Relationship Mapping）即对象关系映射,它允许你使用类和对象对数据库进行交互（使用类和对象和使用 SQL 一样且更方便各种操作）。
- ORM

Object Relationship Mapping
对象 关系 映射

- 三大特征:
 1. 表 到 类的映射
 2. 数据类型的映射
 3. 关系映射

2. 模型示例:

- 此示例为添加一个 bookstore_book 数据表来存放图书馆中书目信息
- 添加一个 bookstore 的 app

```
$ python3 manage.py startapp bookstore
```

- 添加模型类并注册app

```
# file : bookstore/models.py
from django.db import models

class Book(models.Model):
    title = models.CharField("书名", max_length=50, default='')
    price = models.DecimalField('定价', max_digits=7,
decimal_places=2, default=0.0)
# file : setting.py
INSTALLED_APPS = [
    ...
    'bookstore',
]
```

- 生成迁移脚本文件bookstore/migrations/0001_initial.py并进行迁移


```
$ python3 manage.py makemigrations
Migrations for 'bookstore':
bookstore/migrations/0001_initial.py
    - Create model Book
$ python3 manage.py migrate
Operations to perform:
Apply all migrations: admin, auth, bookstore, contenttypes,
sessions
Running migrations:
Applying bookstore.0001_initial... OK
```

- 查看数据表

```
$ mysql -u root -p
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mygoods |
| mysql |
| mywebdb |
| onlybuyp |
| performance_schema |
| sys |
| test_db |
+-----+
8 rows in set (0.00 sec)

mysql> use mywebdb
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with -A
```

Database changed

```
mysql> show tables;
```

```
+-----+
| Tables_in_mywebdb |
+-----+
| auth_group |
| auth_group_permissions |
| auth_permission |
| auth_user |
| auth_user_groups |
| auth_user_user_permissions |
| bookstore_book |
| django_admin_log |
| django_content_type |
| django_migrations |
| django_session |
+-----+
```

<<== 新加表

```
11 rows in set (0.00 sec)
```

```
mysql> desc bookstore_book;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
title	varchar(50)	NO		NULL	
price	decimal(7,2)	NO		NULL	

```
3 rows in set (0.00 sec)
```

- 表bookstore_book 即为模型 Book 类对应的数据表
 - id 为主键, 当设定主键时会自动添加id字段为主键
 - 如果更新模型类 models.py 中的内容时需要运行 makemigrations 和 migrate 子命名来更新和同步数据库
 - 在开发阶段如果同步数据库出错。用 `sql> drop database 数据库名` 删除数据库后重新迁移数据库
 - 在 xxx_app/migrations/*.py 下的文件是自动生成的迁移脚本文件,可以手动删除且在下一次迁移时自动生成

2. 编写模型类Models

- 模型类需继承自 `django.db.models.Model`

1. Models的语法规范

```
from django.db import models
class CLASSNAME(models.Model):
    NAME = models.FIELD_TYPE(FIELD_OPTIONS)
```

2. CLASSNAME

- 实体类名,表名组成一部分,建议类名首字母大写
- 默认表名组成规范:
 - 应用名称_classname

3. NAME

- 类属性名,映射回数据库就是字段名

4. FIELD_TYPE

- 字段类型:映射到表中的字段的类型

3. FIELD_TYPE 类型及含义

1. BooleanField()

- 数据库类型:tinyint(1)
- 编程语言中:使用True或False来表示值
- 在数据库中:使用1或0来表示具体的值

2. CharField()

- 数据库类型:varchar
- 注意:
 - 必须要指定max_length参数值

3. DateField()

- 数据库类型:date
- 作用:表示日期
- 编程语言中:使用字符串来表示具体值
- 参数:
 - DateField.auto_now: 每次保存对象时, 自动设置该字段为当前时间(取值:True/False)。
 - DateField.auto_now_add: 当对象第一次被创建时自动设置当前时间(取值:True/False)。
 - DateField.default: 设置当前时间(取值:字符串格式时间如: '2019-6-1')。
 - 以上三个参数只能多选一

4. DateTimeField()

- 数据库类型:datetime(6)
- 作用:表示日期和时间
- auto_now_add=True

5. DecimalField()

- 数据库类型:decimal(x,y)
- 编程语言中:使用小数表示该列的值
- 在数据库中:使用小数
- 参数:
 - DecimalField.max_digits: 位数总数, 包括小数点后的位数。该值必须大于等于 decimal_places.
 - DecimalField.decimal_places: 小数点后的数字数量
- 示例:

```
money=models.DecimalField(  
    max_digits=7,  
    decimal_places=2,  
    default=0.0  
)
```

6. FloatField()

- 数据库类型:double
- 编程语言中和数据库中都使用小数表示值

7. EmailField()

- 数据库类型:varchar
- 编程语言和数据库中使用字符串

8. IntegerField()

- 数据库类型:int
- 编程语言和数据库中使用整数

9. URLField()

- 数据库类型:varchar(200)
- 编程语言和数据库中使用字符串

10. ImageField()

- 数据库类型:varchar(100)
- 作用:在数据库中为了保存图片的路径
- 编程语言和数据库中使用字符串
- 示例:

```
image=models.ImageField(
    upload_to="static/images"
)
```

- upload_to:指定图片的上传路径 在后台上传时会自动的将文件保存在指定的目录下

11. TextField()

- 数据库类型:longtext
- 作用:表示不定长的字符数据
- 参考文档 <https://docs.djangoproject.com/en/1.11/ref/models/fields/#field-types>

字段选项

4. FIELD_OPTIONS

- 字段选项, 指定创建的列的额外的信息
- 允许出现多个字段选项,多个选项之间使用,隔开

1. primary_key

- 如果设置为True,表示该列为主键,如果指定一个字段为主键, 则此数据库表不会创建id字段

2. blank

- 设置为True时, 字段可以为空。设置为False时, 字段是必须填写的。字符型字段CharField和TextField是用空字符串来存储空值的。默认值是False。

3. null

- 如果设置为True,表示该列值允许为空。日期型、时间型和数字型字段不接受空字符串。所以设置IntegerField, DateTimeField型字段可以为空时, 需要将blank, null均设为True。
- 默认为False,如果此选项为False建议加入default选项来设置默认值

4. default

- 设置所在列的默认值,如果字段选项null=False建议添加此项

5. db_index

- 如果设置为True,表示为该列增加索引

6. unique

- 如果设置为True,表示该字段在数据库中的值必须是唯一(不能重复出现的)

7. db_column

- 指定列的名称,如果不指定的话则采用属性名作为列名

8. verbose_name

- 设置此字段在admin界面上的显示名称。

◦ 示例:

```
# 创建一个属性,表示用户名称,长度30个字符,必须是唯一的,不能为空,添加索引

name = models.CharField(max_length=30, unique=True, null=False,
db_index=True)
```

• 文档参见:

- <https://docs.djangoproject.com/en/1.11/ref/models/fields/#field-options>

• 示例:

数据库迁移的错误处理方法

- 当执行 `$ python3 manage.py makemigrations` 出现如下迁移错误时的处理方法

◦ 错误信息

```
$ python3 manage.py makemigrations
You are trying to change the nullable field 'title' on book to
non-nullable without a default; we can't do that (the database
needs something to populate existing rows).
Please select a fix:
1) Provide a one-off default now (will be set on all existing
rows with a null value for this column)
2) Ignore for now, and let me handle existing rows with NULL
myself (e.g. because you added a RunPython or RunSQL operation to
```

```
handle NULL values in a previous data migration)
3) Quit, and let me add a default in models.py
Select an option:
```

- 翻译为中文如下:

```
$ python3 manage.py makemigrations
您试图将图书上的可空字段“title”更改为非空字段(没有默认值);我们不能这样做(数据库需要填充现有行)。
请选择修复:
1) 现在提供一次性默认值(将对所有现有行设置此列的空值)
2) 暂时忽略, 让我自己处理空值的现有行(例如, 因为您在以前的数据迁移中添加了RunPython或RunSQL操作来处理空值)
3) 退出, 让我在models.py中添加一个默认值
选择一个选项:
```

- 错误原因

- 当将如下代码

```
class Book(models.Model):
    title = models.CharField("书名", max_length=50, null=True)
```

- 去掉 null=True 改为如下内容时会出现上述错误

```
class Book(models.Model):
    title = models.CharField("书名", max_length=50)
```

- 原理是 此数据库的title 字段由原来的可以为NULL改为非NULL状态,意味着原来这个字段可以不填值, 现在改为必须填定一个值, 那填什么值呢? 此时必须添加一个缺省值。

- 处理方法:

1. 选择1 手动给出一个缺省值, 在生成 bookstore/migrations/000x_auto_xxxxxxx_xxxx.py 文件时自动将输入的值添加到default参数中
2. 暂时忽略, 以后用其它的命令处理缺省值问题(不推荐)
3. 退出当前生成迁移文件的过程, 自己去修改models.py, 新增加一个default=XXX 的缺省值(推荐使用)

- 数据库的迁移文件混乱的解决办法

1. 删除 所有 migrations 里所有的 000?_XXXX.py (__init__.py除外)
2. 删除 数据表
 - sql> drop database mywebdb;
3. 重新创建 数据表
 - sql> create database mywebdb default charset...;
4. 重新生成migrations里所有的 000?_XXXX.py

- python3 manage.py makemigrations
- 5. 重新更新数据库
- python3 manage.py migrate

数据库的操作(CRUD操作)

- CRUD是指在做计算处理时的增加(Create)、读取查询(Read)、更新(Update)和删除>Delete)

管理器对象

- 每个继承自 `models.Model` 的模型类，都会有一个 `objects` 对象被同样继承下来。这个对象叫管理器对象
- 数据库的增删改查可以通过模型的管理器实现

```
class Entry(models.Model):  
    ...  
    Entry.objects.create(...) # 是管理器对象
```

创建数据对象

- Django 使用一种直观的方式把数据库表中的数据表示成Python 对象
 - 创建数据中每一条记录就是创建一个数据对象
1. `Entry.objects.create(属性1=值1, 属性2=值1,...)`
 - 成功: 返回创建好的实体对象
 - 失败: 抛出异常
 2. 创建 `Entry` 实体对象,并调用 `save()` 进行保存

```
obj = Entry(属性=值, 属性=值)  
obj.属性=值  
obj.save()  
无返回值, 保存成功后, obj 会被重新赋值
```

3. 示例1:

```
try:  
    abook = Book.objects.create(title='Python', pub='清华大学出版社')  
    print(abook)  
except:  
    print('创建对象失败')
```

4. 示例2:

```
try:
    abook = Book(title='Python', pub='清华大学出版社')
    abook.save
    print(abook)
except:
    print('创建对象失败')
```

5. 示例3:

```
try:
    abook = Book()
    abook.title='Python'
    abook.pub='清华大学出版社'
    abook.save
    print(abook)
except:
    print('创建对象失败')
```

◦ 练习:

- 使用以上三种方式,分别向Book和Publisher表中各增加三条数据

Django shell 的使用

- 在Django提供了一个交互式的操作项目叫 **Django Shell** 它能够在交互模式用项目工程的代码执行相应的操作
- 利用 Django Shell 可以代替编写View的代码来进行直接操作
- 在Django Shell 下只能进行简单的操作, 不能运行远程调式
- 启动 Django shell 显示IPython风格的交互界面如下:

```
$ python3 manage.py shell
manage.py shell
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
In [2]: from bookstore import models
In [3]: models.Book.objects.create(title="Python")
Out[3]: <Book: Book object>
In [4]: book = models.Book.objects.create(title='C++')
In [4]: print(book)
Book object
```

- 练习:


```
在 bookstore/models.py 应用中添加两个model类
1. Book - 图书
    1. title - CharField 书名,非空,唯一
    2. pub - CharField 出版社,字符串,非空
    3. price - 图书价格,,
    4. market_price - 图书零售价
2. Author - 作者
    1. name - CharField 姓名,非空,唯一,加索引
    2. age - IntegerField, 年龄,非空, 缺省值为1
    3. email - EmailField, 邮箱,允许为空
```

◦ 然后用Django Shell 添加如下数据

■ 图书信息

书名	定价	零售价	出版社
Python	20.00	25.00	清华大学出版社
Python3	60.00	65.00	清华大学出版社
Django	70.00	75.00	清华大学出版社
JQuery	90.00	85.00	机械工业出版社
Linux	80.00	65.00	机械工业出版社
Windows	50.00	35.00	机械工业出版社
HTML5	90.00	105.00	清华大学出版社

■ 作者信息:

姓名	年龄	邮箱
王老师	28	wangwc@tedu.cn
吕老师	31	lvze@tedu.cn
祁老师	30	qitx@tedu.cn

查询数据

- 数据库的查询需要使用管理器对象进行
- 通过 Entry.objects 管理器方法调用查询接口

方法	说明
all()	查询全部记录,返回QuerySet查询对象
get()	查询符合条件的单一记录
filter()	查询符合条件的多条记录
exclude()	查询符合条件之外的全部记录

...

1. all()方法

- 方法: all()
- 用法: Entry.objects.all()
- 作用: 查询Entry实体中所有的数据
 - 等同于
 - select * from tabel
- 返回值: QuerySet容器对象,内部存放Entry 实例
- 示例:

```
from bookstore import models
books = models.Book.object.all()
for book in books:
    print("书名", book.title, '出版社:', book.pub)
```

2. 在模型类中定义 def __str__(self): 方法可以将自定义默认的字符串

```
class Book(models.Model):
    title = ...
    def __str__(self):
        return "书名: %s, 出版社: %s, 定价: %s" % (self.title, self.pub, self.price)
```

3. 查询返回指定列(字典表示)

- 方法: values('列1', '列2')
- 用法: Entry.objects.values(...)
- 作用: 查询部分列的数据并返回
 - select 列1,列2 from xxx
- 返回值: QuerySet
 - 返回查询结果容器, 容器内存字典, 每个字典代表一条数据,
 - 格式为: {'列1': 值1, '列2': 值2}
- 示例:

```
from bookstore import models
books = models.Book.objects.values("title", "pub")
for book in books:
    print("书名", book["title"], '出版社:', book['pub'])
    print("book=", book)
```

4. 查询返回指定列 (元组表示)

- 方法: values_list('列1', '列2')
- 用法: Entry.objects.values_list(...)
- 作用:

- 返回元组形式的查询结果
- 返回值: QuerySet容器对象,内部存放 元组
 - 会将查询出来的数据封装到元组中,再封装到查询集合QuerySet中
- 示例:

```
from bookstore import models
books = models.Book.objects.values_list("title", "pub")
for book in books:
    print("book=", book)  # ('Python', '清华大学出版社')...
```

5. 排序查询

- 方法:order_by
- 用法:Entry.objects.order_by('-列','列')
- 作用:
 - 与all()方法不同, 它会用SQL 语句的ORDER BY 子句对查询结果进行根据某个字段选择性的进行排序
- 说明:
 - 默认是按照升序排序,降序排序则需要在列前增加'-'表示
- 示例:

```
from bookstore import models
books = models.Book.objects.order_by("price")
for book in books:
    print("书名:", book.title, '价格:', book.price)
```

6. 根据条件查询多条记录

- 方法: filter(条件)
- 语法:

```
Entry.objects.filter(属性1=值1, 属性2=值2)
```

- 返回值:
 - QuerySet容器对象,内部存放 Entry 实例
- 说明:
 - 当多个属性在一起时为与关系, 即当`Books.objects.filter(price=20, pub="清华大学出版社")` 返回价格为20 且 出版社为"清华大学出版社"的全部图书
- 示例:

```
# 查询书中出版社为"清华大学出版社"的图书
from bookstore import models
books = models.Book.objects.filter(pub="清华大学出版社")
for book in books:
    print("书名:", book.title)
```

2. 查询Author实体中id为1并且isActive为True的

- authors=Author.objects.filter(id=1,isActive=True)