

《Django 教程》

- 讲师: 魏明择
- 时间: 2019

目录

- 静态文件
- Django中的应用 - app
 - 什么是应用(app)
 - 创建应用app
 - Django应用的结构组成
- 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
 - 数据库的操作(CRUD操作)
 - 创建数据对象
 - 查询数据

静态文件

1. 什么是静态文件
 - 不能与服务器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:

```
# file: url.py  
from . import views  
  
urlpatterns = [  
    url(r'^admin/', admin.site.urls),  
    url(r'^show_image', views.show_image)  
]  
  
# file: views.py  
from django.shortcuts import render  
  
def show_image(request):  
    return render(request, "show_image.html")
```

```
<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
 3. 127.0.0.1:8000/cart : 显示购物车页
- 处理好所有的静态文件

Django中的应用 - app

什么是应用(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,... ..
- Django中,主文件夹是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).具体的请求是由应用来进行处理的

创建应用app

- 创建应用的指令
 - python3 manage.py startapp 应用名称
 - 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
 - 使用include 函数让某个正则匹配后关联分支到某个app

```

# 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')),
    url(r'^sport/', include('sport.urls')),
    url(r'^news/', include('news.urls')),
]

# file : <App名>/urls.py
from django.conf.urls import url
from . import views

urlpatterns = [
    # 购物车模块用到的路由
    url(r'^page1', views.page1),
    url(r'^page2', views.page2),
    url(r'^page3', views.page3),
    # ...
]

```

- 练习:

1. 创建三个应用
 1. 创建 music 应用, 并注册
 2. 创建 sport 应用, 并注册
 3. 创建 news 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() 函数处理
 2. 127.0.0.1:8000/sport/index
交给 sport 应用中的 index() 函数处理
 3. 127.0.0.1:8000/news/index
交给 news 应用中的 index() 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. 关于数据为的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`
- 3. 查看迁移执行的SQL语句
 - 将 `sqlmigrate`, 显示迁移的sql语句
 - `python3 manage.py sqlmigrate`

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)
```

```

    price = models.DecimalField('定价', max_digits=7,
decimal_places=2)
# 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   | <== 新加表
| django_admin_log |
| django_content_type |
| django_migrations |
| django_session   |
+-----+
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类名(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

- 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
    )
```

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://iyibooks.cn/xx/Django_1.11.6/ref/models/fields.html

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

2. null
 - 如果设置为True,表示该列值允许为空
 - 默认为False
3. default
 - 设置所在列的默认值
4. db_index
 - 如果设置为True,表示为该列增加索引
5. unique
 - 如果设置为True,表示该列的值唯一
6. 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

ex:

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://yiyibooks.cn/xx/Django_1.11.6/ref/models/fields.html?highlight=booleanfield
 - <https://docs.djangoproject.com/zh-hans/2.2/ref/models/fields/#field-attribute-reference>
- 示例:

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

class Book(models.Model):
    title = models.CharField("书名", max_length=50)
    price = models.DecimalField('定价', max_digits=7, decimal_places=2)
    pub_house = models.CharField("出版社", max_length=50, default='清华大学出版社')
    pub_date = models.DateField('出版时间', default='1970-1-1')
    market_price = models.DecimalField('市价', max_digits=7, decimal_places=2, default=9999)
```

```
# file : bookstore/views.py
from django.http import HttpResponse
from . import models

def init_books(request):
    models.Book.objects.create(title='C', price=30, market_price=35, pub_house="清华大学出版社")
    models.Book.objects.create(title='C++', price=40, market_price=45, pub_house="清华大学出版社")
    models.Book.objects.create(title='Java', price=50, market_price=55,
```

```

pub_house="清华大学出版社")
    models.Book.objects.create(title='Python', price=20, market_price=25,
pub_house="清华大学出版社")
    models.Book.objects.create(title='Python3', price=60, market_price=65,
pub_house="清华大学出版社")
    models.Book.objects.create(title='Django', price=70, market_price=75,
pub_house="清华大学出版社")
    models.Book.objects.create(title='JQuery', price=90, market_price=85,
pub_house="机械工业出版社")
    models.Book.objects.create(title='Linux', price=80, market_price=65,
pub_house="机械工业出版社")
    models.Book.objects.create(title='Windows', price=50, market_price=35,
pub_house="机械工业出版社")

    abook = models.Book(title="HTML5", price=90, market_price=105,
pub_date='2019-1-1') # 创建新书
    abook.save() # 保存

    return HttpResponse("初始化成功")

```

- 第一次执行视图views.py 中的init_books()函数时结果

```

mysql> select * from bookstore_book;
+----+-----+-----+-----+-----+-----+
| id | title  | price | market_price | pub_date | pub_house |
+----+-----+-----+-----+-----+-----+
| 1  | C      | 30.00 | 35.00 | 1970-01-01 | 清华大学出版社 |
| 2  | C++    | 40.00 | 45.00 | 1970-01-01 | 清华大学出版社 |
| 3  | Java   | 50.00 | 55.00 | 1970-01-01 | 清华大学出版社 |
| 4  | Python | 20.00 | 25.00 | 1970-01-01 | 清华大学出版社 |
| 5  | Python3 | 60.00 | 65.00 | 1970-01-01 | 清华大学出版社 |
| 6  | Django | 70.00 | 75.00 | 1970-01-01 | 清华大学出版社 |
| 7  | JQuery | 90.00 | 85.00 | 1970-01-01 | 机械工业出版社 |
| 8  | Linux  | 80.00 | 65.00 | 1970-01-01 | 机械工业出版社 |
| 9  | Windows | 50.00 | 35.00 | 1970-01-01 | 机械工业出版社 |
| 10 | HTML5  | 90.00 | 105.00 | 2019-01-01 | 清华大学出版社 |
+----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

```

- 练习:

在 bookstore 应用中添加两个model类

1. Author - 作者
 - 1.name - 姓名,非空,唯一,加索引
 - 2.age - 年龄,非空
 - 3.email - 邮箱,允许为空
2. Book - 图书

- 1.title - 书名,非空,唯一,加索引
- 2.publish_date - 出版时间,非空,加索引

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)
- 数据换的增删改查可以通过模型的管理器实现
 - MyModel.objects 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=值, 属性=值)
 - 返回值: 返回创建好的实体对象
 2. 创建Entry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. 使用字典创建对象,并调用save()进行保存

```
dic = {
    '属性1': '值',
    '属性2': '值2',
}

obj = Entry(**dic)
obj.save()
```

- 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

查询数据

- 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
get()	查询符合条件的单一记录
filter()	查询符合条件的多条记录

方法	说明
<code>exclude()</code>	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
 - `select * from tabel`
- 返回值: QuerySet对象

2. 查询返回指定列

- 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
 - 会将查询出来的数据封装到字典中,再封装到查询集合QuerySet中

3. 查询返回指定列

- 方法: `values_list('列1', '列2')`
- 用法: `Entry.objects.values_list(...)`
- 返回值: QuerySet
 - 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

4. 排序查询

- 方法: `order_by`
- 用法: `Entry.objects.order_by('-列', '列')`
- 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

```
Entry.objects.all()
数据类型: <class 'django.db.models.query.QuerySet'>
结果: <QuerySet [<Author: Author object>, <Author: Author object>]>

Entry.objects.values()
<QuerySet
[
    {'name': '老舍', 'email': 'laoshe@163.com'}, {'name': '巴金',
'email': 'bajin@163.com'}
]>
```

- Django shell 的使用
 - 启动 Django shell

```
$ 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]:
```