《Django 教程》

- 讲师: 魏明择
- 时间: 2019

目录

分页

- 分页是指在web页面有大量数据需要显示时，当一页的内容太多不利于阅读和不利于数据提取的情况下，可以分为多页进行显示。
- Django提供了一些类来帮助你管理分页的数据 — 也就是说，数据被分在不同页面中，并带有“上一页/下一页”链接。
- 这些类位于django/core/paginator.py中。

Paginator对象

- 对象的构造方法
    - Paginator(object_list, per_page)
    - 参数
        - object_list 对象列表
        - per_page 每页数据个数
    - 返回值:
        - 分页对象
- Paginator属性
    - count：对象总数
    - num_pages：页面总数
    - page_range：从1开始的range对象, 用于记录当前面码数
    - per_page 每页个数
- Paginator方法
    - Paginator.page(number)
        - 参数 number为页码信息(从1开始)
        - 返回当前number页对应的页信息
        - 如果提供的页码不存在，抛出InvalidPage异常
- Paginator异常exception
    - InvalidPage：当向page()传入一个无效的页码时抛出
    - PageNotAnInteger：当向page()传入一个不是整数的值时抛出
    - EmptyPage：当向page()提供一个有效值，但是那个页面上没有任何对象时抛出

Page对象

- 创建对象 Paginator对象的page()方法返回Page对象，不需要手动构造
- Page对象属性
    - object_list：当前页上所有对象的列表
    - number：当前页的序号，从1开始
    - paginator：当前page对象相关的Paginator对象
- Page对象方法
    - has_next()：如果有下一页返回True
    - has_previous()：如果有上一页返回True
    - has_other_pages()：如果有上一页或下一页返回True
    - next_page_number()：返回下一页的页码，如果下一页不存在，抛出InvalidPage异常
    - previous_page_number()：返回上一页的页码，如果上一页不存在，抛出InvalidPage异常
    - len()：返回当前页面对象的个数
- 说明:
    - Page 对象是可迭代对象,可以用 for 语句来 访问当前页面中的每个对象
- 参考文档https://docs.djangoproject.com/en/1.11/topics/pagination/
- 分页示例:
    - 视图函数

```
def book(request):
    bks = models.Book.objects.all()
    paginator = Paginator(bks, 10)
    print('当前对象的总个数是:', paginator.count)
    print('当前对象的面码范围是:', paginator.page_range)
    print('总页数是: ', paginator.num_pages)
    print('每页最大个数:', paginator.per_page)

    cur_page = request.GET.get('page', 1)  # 得到默认的当前页
    page = paginator.page(cur_page)
    return render(request, 'bookstore/book.html', locals())
```

    - 模板设计

```
<html>
<head>
    <title>分页显示</title>
</head>
<body>
{% for b in page %}
    <div>{{ b.title }}</div>
{% endfor %}
```

```
{# 分页功能 #}
{# 上一页功能 #}
{% if page.has_previous %}
<a href="{% url "book" %}?page={{ page.previous_page_number }}">上一页
</a>
{% else %}
上一页
{% endif %}

{% for p in paginator.page_range %}
    {% if p == page.number %}
        {{ p }}
    {% else %}
        <a href="{% url "book" %}?page={{ p }}">{{ p }}</a>
    {% endif %}
{% endfor %}

{#下一页功能#}
{% if page.has_next %}
<a href="{% url "book" %}?page={{ page.next_page_number }}">下一页</a>
{% else %}
下一页
{% endif %}
总页数: {{ page.len }}
</body>
</html>
```

day08

文件上传

- 文件上传必须为POST提交方式
- 表单`<form>`中文件上传时必须有带有`enctype="multipart/form-data"` 时才会包含文件内容数据。
- 表单中用`<input type="file" name="xxx">`标签上传文件
    - 名字`xxx`对应`request.FILES['xxx']` 对应的内存缓冲文件流对象。可以能过`request.FILES['xxx']` 返回的对象获取上传文件数据
    - `file=request.FILES['xxx']` file 绑定文件流对象，可以通过文件流对象的如下信息获取文件数据 file.name 文件名 file.file 文件的字节流数据
- 如下上传文件为图片类型，可以用模块类属性定义成models.ImageField类型
    - `image_file = models.ImageField(upload_to='images/')`
    - 注意：如果属性类型为ImageField需要安装包Pilow
    - pip install Pillow==3.4.1
    - 图片存储路径
- 练习: 在项目根目录下创建static文件夹 图片上传后，会被保存到“/static/files/"下 打开settings.py文件，增加MEDIA_ROOT项 MEDIA_ROOT=os.path.join(BASE_DIR,"static/files") 使用django后台管理，遇到

ImageField类型的属性会出现一个file框，完成文件上传 手动上传的模板代码

- 上传文件的表单书写方式

```
<!-- file:static/upload.html -->
<html>
<head>
    <title>文件上传</title>
    <meta charset="utf-8">
</head>
<body>
    <h1>上传文件</h1>
    <form method="post" action="/upload" enctype="multipart/form-data">
        <input type="file" name="myfile"/><br>
        <input type="submit" value="上传">
    </form>
</body>
</html>
```

- 在setting.py 中设置一个变量MEDIA_ROOT 用来记录上传文件的位置

```
# file : settings.py
...
STATIC_URL = '/static/'
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
MEDIA_ROOT = os.path.join(BASE_DIR, 'static/files')
...
```

- 在当前项目文件夹下创建 `static/files` 文件夹

```
$ mkdir -p static/files
```

- 添加路由及对应的处理函数

```
# file urls.py
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^upload', views.upload_file)
]
```

- 上传文件的视图处理函数

```python
# file views.py
from django.http import HttpResponse, Http404
from django.conf import settings
import os

# from  django.views.decorators.http import require_POST
# @require_POST
def upload_file(request):
    if request.method == "POST":
        file = request.FILES['myfile']
        print("上传文件名是:", file.name)

        filename =os.path.join(settings.MEDIA_ROOT, file.name)
        with open(filename, 'wb') as f:
            f.write(file.file.read())
            return HttpResponse("接收文件成功")
    raise Http404
```

- 访问地址: http://127.0.0.1:8000/static/upload.html

项目部署

- 项目部置在软件开发完毕后，将开发机器上运行的开发板软件实际安装到服务器上进行长期运行
- 部署要分以下几个步骤进行
    1. 在安装机器上安装和配置同版本的数据库
    2. django 项目迁移(在安装机器上配置与开发环境相同的python版本及依懒的包)
    3. 用 uwsgi 替代`python3 manage.py runserver`方法启动服务器
    4. 配置 nginx 反向代理服务器
    5. 用nginx 配置静态文件路径,解决静态路径问题

1. 安装同版本的数据库
    - 安装步骤略
2. django 项目迁移
    1. 安装python
        - `$ sudo apt install python3`
    2. 安装相同版本的包
        - 导出当前模块数据包的信息:
            - `$ pip3 freeze > package_list.txt`
        - 导入到另一台新主机
            - `$ pip3 install -r package_list.txt`
    3. 将当前项目源代码复制到运程主机上(scp 命令)
        - $ sudo scp -a 当前项目源代码 远程主机地址和文件夹
3.

WSGI Django工作环境部署

- WSGI (Web Server Gateway Interface)Web服务器网关接口，是Python应用程序或框架和Web服务器之间的一种接口，被广泛使用

- 它实现了WSGI协议、http等协议。Nginx中HttpUwsgiModule的作用是与uWSGI服务器进行交换。WSGI是一种Web服务器网关接口。

uWSGI 网关接口配置

- 使用 `python manage.py runserver` 通常只在开发和测试环境中使用。
- 当开发结束后，完善的项目代码需要在一个高效稳定的环境中运行，这时可以使用uWSGI
- uWSGI是WSGI的一种,它可以让Django、Flask等开发的web站点运行其中.
- 安装uWSGI $ sudo pip3 install uwsgi
- 配置uWSGI

```
# file: 项目文件夹/uwsgi.ini  # 如: mysite1/uwsgi.ini
[uwsgi]
# 套接字方式的 IP地址:端口号
# socket=127.0.0.1:8000
# Http通信方式的 IP地址:端口号
http=127.0.0.1:8000
# 项目当前工作目录
chdir=/home/weimz/my_django_project ... 这里需要换为项目地址
# 项目中wsgi.py文件的目录，相对于当前工作目录
wsgi-file=my_django_project/wsgi.py
# 进程个数
process=4
# 每个进程的线程个数
threads=2
# 服务的pid记录文件
pidfile=uwsgi.pid
# 服务的目志文件位置
daemonize=uwsgi.log
```

- uWSGI的运行管理
  - 启动 uwsgi

```
$ cd 项目文件夹
$ sudo uwsgi --ini 项目文件夹/uwsgi.ini
```

  - 停止 uwsgi

```
$ cd 项目文件夹
$ sudo uwsgi --stop uwsgi.pid
```

  - 说明:

        - 当uwsgi 启动后,当前django项目的程序已变成后台守护进程,在关闭当前终端时此进程也不会停止。

- 测试:
    - 在浏览器端输入http://127.0.0.1:8000 进行测试
    - 注意，此时端口号为8000

nginx 反向代理配置

- Nginx是轻量级的高性能Web服务器，提供了诸如HTTP代理和反向代理、负载均衡、缓存等一系列重要特性，在实践之中使用广泛。
- C语言编写，执行效率高
- nginx 作用
    - 负载均衡， 多台服务器轮流处理请求
    - 反向代理
- 原理:
    - 客户端请求nginx,再由nginx 请求 uwsgi, 运行django下的python代码
- ubuntu 下 nginx 安装 $ sudo apt install nginx
- nginx 配置
    - 修改nginx 的配置文件 /etc/nginx/sites-available/default

```
# 在server节点下添加新的location项，指向uwsgi的ip与端口。
server {
    ...
    location / {
        uwsgi_pass 127.0.0.1:8000;  # 重定向到127.0.0.1的8000端口
        include /etc/nginx/uwsgi_params; # 将所有的参数转到uwsgi下
    }
    ...
}
```

- 启动 nginx
    - `$ sudo /etc/init.d/nginx start`
    - 或
    - `$ sudo service nginx restart`
- 查看nginx进程
    - `$ ps aux | grep nginx`
    - 或
    - `$ sudo /etc/init.d/nginx status`

    - 或
    - `$ sudo service nginx status`

- 停止nginx

    - `$ sudo /etc/init.d/nginx stop`
    - 或
    - `$ sudo service nginx stop`

- 重启nginx

    - `$ sudo /etc/init.d/nginx restart`
    - 或
    - `$ sudo service nginx restart`

- 修改uWSGI配置

    - 修改`项目文件夹/uwsgi.ini`下的Http通信方式改为socket通信方式,如:

```
[uwsgi]
# 去掉如下
# http=127.0.0.1:8000
# 改为
socket=127.0.0.1:8000
```

    - 重启uWSGI服务

```
$ sudo uwsgi --stop uwsgi.pid
$ sudo uwsgi --ini 项目文件夹/uwsgi.ini
```

- 测试:

    - 在浏览器端输入http://127.0.0.1 进行测试
    - 注意，此时端口号为80(nginx默认值)

nginx 配置静态文件路径

- 解决静态路径问题

```
# file : /etc/nginx/sites-available/default
# 新添加location /static 路由配置，重定向到指定的绝对路径
server {
    ...
    location /static {
        # root static文件夹所在的绝对路径,如:
        root /home/weimz/my_django_project; # 重定向/static请求的路径，这
里改为你项目的文件夹
```

```
    }
    ...
}
```

- 修改配置文件后需要重新启动 nginx 服务

404 界面

- 在模板文件夹内添加 404.html 模版，当响应返回HttpResponseNotFound 或 raise Http404时将会被显示
- 404.html 仅在发布版中(即setting.py 中的 DEBUG=False时) 才起作用
- 当向应处理函数触发Http404异常时就会跳转到404界面

```
from django.http import Http404
def xxx_view(request):
    raise Http404  # 直接返回404
```