Django1.11.4之高级应用

静态文件

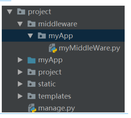
- 配置settings.py文件
  - STATIC_URL = '/static/'
  - STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
- {% load static from staticfiles %}
- <script src="{% static 'myApp/js/bootstrap.min.js' %}"></script>

中间件

- 概述：轻量级，底层的插件，可以介入django的请求和响应
- 本质：一个Python类
- 方法
  - __init__：不需要传参数，服务器响应第一个请求的时候自动调用，用于确定是否启用该中间件
  - process_request(self, request)：在执行视图之前被调用（分配url匹配视图之前），每个请求上都会调用，返回None或者HttpResponse对象
  - process_view(self, request, view_func, view_args, kargs)：在分配url之后，匹配视图之前被调用，每个请求上都会调用，返回None或者HttpResponse对象
  - process_template_response(self, request, response)：在视图刚好执行完后调用，每个请求上都会调用，返回None或者HttpResponse对象
    - 使用render
  - process_response(self, request, response)：所有响应返回浏览器之前调用，所以请求都会调用，返回HttpResponse对象
  - process_exception(self, request, exception)：当视图抛出异常时调用，返回HTTPResponse对象

执行位置

- 自定义中间件
  - 创建目录和python文件




```
class MyMiddleWare(MiddlewareMixin):
    def process_request(self, request):
        print('get参数为', request.GET.get('a'))
```

  - 使用自定义中间件
    - 配置settings.py文件
      - MIDDLEWARE中添加路径：'middleware.myApp.myMiddleWare.MyMiddleWare'

上传图片

- 概述
  - 文件上传时，文件数据存储在request.FILES属性中
  - 注意：form表单上传文件要加 enctype="multipart/form-data
  - 注意：上传文件必须是POST请求
- 存储路径
  - 在static目录下创建upfile目录用于存储接收上传的文件
  - 配置settings文件
    - # 上传文件目录
    - MEDIA_ROOT = os.path.join(BASE_DIR, r'static\upfile')
- 代码示例
  - 页面

```
<form method="post" action="/savefile/" enctype="multipart/form-data">
    {% csrf_token %}
    <input type="file" name="file">
    <input type="submit" name="上传">
</form>
```

```
def upfile(request):
    return render(request, 'myApp/upfile.html' )

import os
from django.conf import settings
def savefile(request):
    # 上传文件不能是get
    if request.method == 'POST':
        f = request.FILES['file']
        # 合成文件在服务器的路径
        filePath = os.path.join(settings.MEDIA_ROOT , f.name)
        with open(filePath, 'wb') as fp:
            #  chunks将文件分段
            for info in f.chunks():
                fp.write(info)
        return HttpResponse('上传成功')
    else:
        return HttpResponse('上传失败')
```

  - 视图

分页

- Paginator对象
  - 创建对象
    - 格式：Paginator(列表, 整数)
    - 返回值：返回的分页对象
  - 属性
    - count：对象总数
    - num_pages：页面总数
    - page_range：页码列表
      - [1,2,3,4,5]
      - 页码从1开始
  - 方法
    - page(num)：获得一个Page对象，如果提供的页码不存在，会抛出"InvalidPage"异常
  - 异常
    - InvalidPage：当向Page()传递的是一个无效的页码时抛出
    - PageNotAnInteger：当向Page()传递的不是一个整数时抛出
    - EmptyPage：当向page()传递一个有效值但是该页面没有数据时抛出
- Page对象
  - 创建对象
    - Paginator对象的page()方法返回得到Page对象
    - 不需要手动创建
  - 属性
    - object_list：当前页面的所有数据的列表
    - number：当前页面的页码值
    - paginator：当前page对象关联的Paginator对象
  - 方法
    - has_next()：判断是否有下一页，如果有返回True
    - has_previous()：判断是否有下一页，如果有返回True
    - has_other_pages()：判断是否有其他也，如果有返回True
    - next_page_number()：返回下一页的页码，如果不存在抛出"InvalidPage"
    - previous_page_number()：返回上一页的页码，如果不存在抛出"InvalidPage"
    - len()：返回当前页的数据个数
- Paginator与Page的关系

- 代码示例
  - 配置路由
    - url(r'^studentpage/(\d+)/$', views.studentpage),
  - 编写视图

```
from django.core.paginator import Paginator
def studentpage(request, num):
    allList = Students.stuObj.all()
    paginator = Paginator(allList, 6)
    page = paginator.page(num)
    return render(request, 'myApp/studentpage.html', {'students':page})
```

  - 编写页面

```
<ul>
    {% for stu in students %}
    <li>
        {{stu.name}} -- {{stu.grade.name}}
    </li>
    {% endfor %}
</ul>
<ul>
    {% for index in students.paginator.page_range %}
        {% if index == students.number %}
            <li>{{index}}</li>
        {% else %}
            <li>
                <a href="/studentpage/{{index}}/">{{index}}</a>
            </li>
        {% endif %}
    {% endfor%}
</ul>
```

ajax

需要动态生成，请求json数据

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script type="text/javascript" src="/static/myApp/js/jquery-1.11.3.min.js"></script>
</head>
<body>
    <h1>学生信息列表</h1>
    <button id="btn">显示学生信息</button>
    <script type="text/javascript" src="/static/myApp/js/zcz.js"></script>
</body>
```

```
$(document).ready(function () {
    $("#btn").bind("click", function () {
        $.ajaxf({
            type: "get",
            url: "/studentsinfo",
            dataType: "json",
            success:function (data, status) {
                console.log(data)
                var d = data['data']
                for (var i = 0; i<d.length; i++){
                    document.write('<p>'+d[i][0]+'</p>')
                }

            }
        })
    });
});
```

```
def ajaxstudents(request):
    return render(request, 'myApp/ajaxstudents.html')

def studentsinfo(request):
    stus = Students.stuObj.all()
    list = []
    for stu in stus:
        list.append([stu.name, stu.age])
    return JsonResponse({'data': list})
```

富文本

- pip install django-tinymce
- 在站点中使用
  - 配置settings.py文件
    - INSTALLED_APPS中添加tinymce
    - # 富文本
    - TINYMCE_DEFAULT_CONFIG = {
      - 'theme': 'advanced',
      - 'width': 600,
      - 'height': 400,
      - }
  - 创建一个模型类

```
from tinymce.models import HTMLField

class Text(models.Model):
    str = HTMLField()
```

  - 配置站点：admin.site.register(models.Text)
- 在自定义视图中使用

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>富文本编辑器</title>
    <script type="text/javascript" src="/static/tiny_mce/tiny_mce.js" />/tiny_mce/
    <script type="text/javascript" />
        tinyMCE.init({
            mode : "textareas",
            theme : "advanced",
            width : "600",
            height : "400",
        })
    </script>
</head>
<body>
    <form>
        <textarea name="str">yes is a good man</textarea>
    </form>
</body>
</html>
```

celery

- 文档http://docs.jinkan.org/docs/celery
- 问题
  - 用户发起request，并且要等待response返回，但是在视图中有一些耗时的操作，导致用户可能会等待很长时间
  - 网站每隔一段时间要同步一次数据，但是http请求时需要触发的
- 解决：celery
  - 将耗时的操作放到celery中执行
  - 使用celery定时执行
- celery
  - 任务task：本质是一个Python函数，将耗时操作封装成一个函数
  - 队列：将要执行的任务放到队列中
  - 工人：负责执行队列中的任务
  - 代理broker：负责调度，在部署环境中使用redis
- 安装
  - pip install celery
  - pip install celery-with-redis
  - pip install django-celery
- 配置settings.py
  - INSTALLED_APPS：添加djcelery

```
# celery 配置
import djcelery
BROKER_BACKEND = 'redis'  # 指定消息通道
BROKER_URL = 'redis://localhost:6379/0;redis://:password@localhost:6379/0'
CELERY_RESULT_BACKEND = 'redis'
```

- 在应用目录下创建task.py文件
- 迁移，生成celery需要的数据库表：python manage.py migrate
- 在工程目录下的project目录下创建celery.py

```
from __future__ import absolute_import

import os
from celery import Celery,
from django.conf import settings

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'zhihu_home.settings')

app = Celery('portal')

app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)

@app.task(bind=True)
def debug_task(self):
    print('Request:{0}'.format(self.request))
```

- 在工程目录下的__init__.py文件中添加：from celery import app as celery_app