《Django Web框架教学笔记》

- 讲师: 魏明择
- 时间: 2019

目录

中间件 Middleware

- 中间件是 Django 请求/响应处理的钩子框架。它是一个轻量级的、低级的“插件”系统，用于全局改变 Django 的输入或输出。
- 每个中间件组件负责做一些特定的功能。例如，Django 包含一个中间件组件 AuthenticationMiddleware，它使用会话将用户与请求关联起来。
- 他的文档解释了中间件是如何工作的，如何激活中间件，以及如何编写自己的中间件。Django 具有一些内置的中间件，你可以直接使用。它们被记录在 built-in middleware reference 中。
- 中间件类:
  - 中间件类须继承自 `django.utils.deprecation.MiddlewareMixin` 类
  - 中间件类须实现下列五个方法中的一个或多个:
    - `def process_request(self, request):` 执行视图之前被调用，在每个请求上调用，返回None或HttpResponse对象
    - `def process_view(self, request, callback, callback_args, callback_kwargs):` 调用视图之前被调用，在每个请求上调用，返回None或HttpResponse对象
    - `def process_response(self, request, response):` 所有响应返回浏览器之前被调用，在每个请求上调用，返回HttpResponse对象
    - `def process_exception(self, request, exception):` 当处理过程中抛出异常时调用，返回一个HttpResponse对象
    - `def process_template_response(self, request, response):` 在视图刚好执行完毕之后被调用，在每个请求上调用，返回实现了render方法的响应对象
  - 注： 中间件中的大多数方法在返回None时表示忽略当前操作进入下一项事件，当返回HttpResponese对象时表示此请求结果，直接返回给客户端
- 编写中间件类:

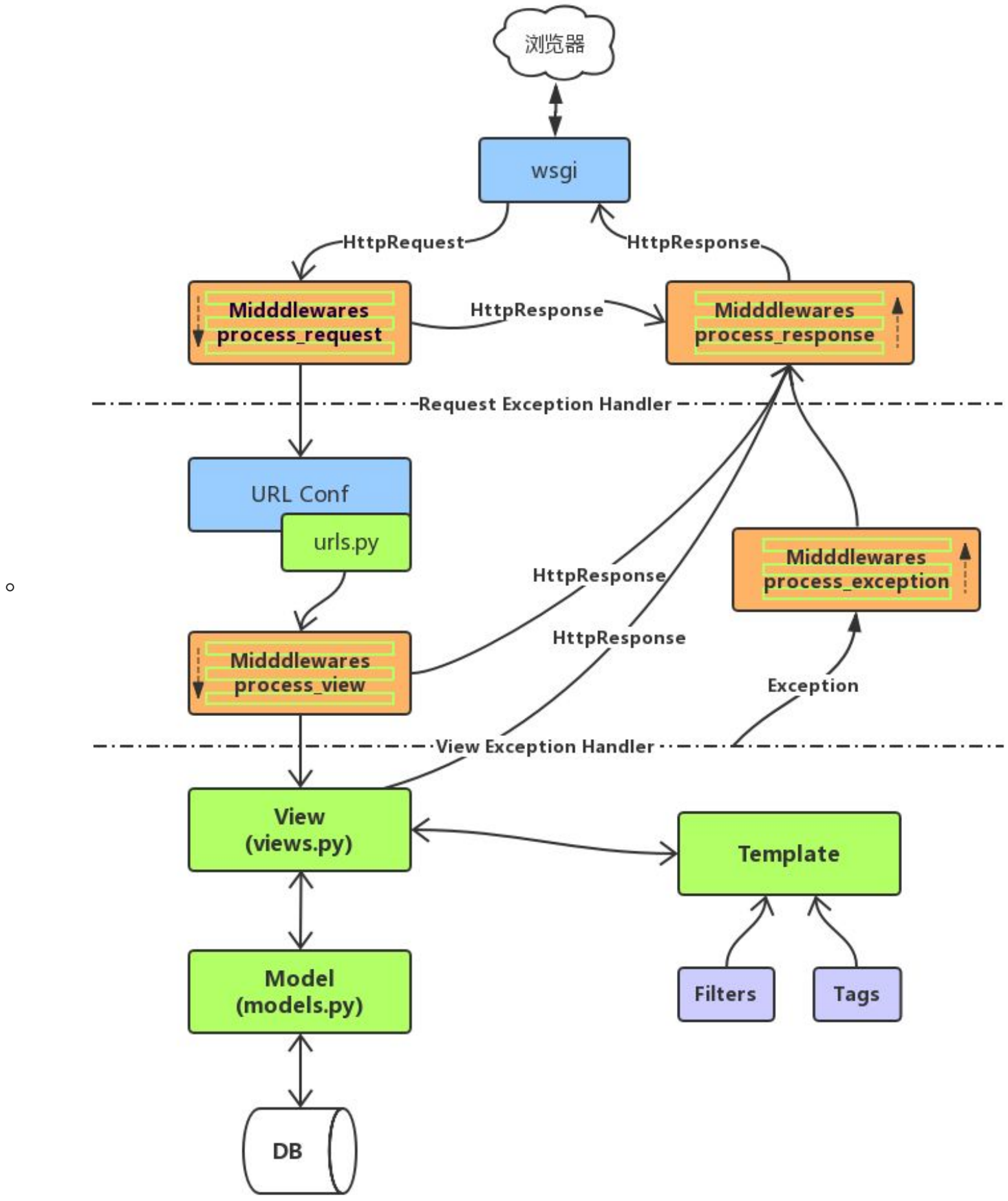
```
# file : middleware/mymiddleware.py
from django.http import HttpResponse, Http404
from django.utils.deprecation import MiddlewareMixin

class MyMiddleWare(MiddlewareMixin):
    def process_request(self, request):
        print("中间件方法 process_request 被调用")

    def process_view(self, request, callback, callback_args, callback_kwargs):
        print("中间件方法 process_view 被调用")

    def process_response(self, request, response):
        print("中间件方法 process_response 被调用")
        return response

    def process_exception(self, request, exception):
        print("中间件方法 process_exception 被调用")

    def process_template_response(self, request, response):
        print("中间件方法 process_template_response 被调用")
```

```
        return response
```

- 注册中间件:

```
# file : settings.py
MIDDLEWARE = [
    ...
    'middleware.mymiddleware.MyMiddleWare',
]
```

- 中间件的执行过程



- 练习
  - 用中间件实现强制某个IP地址只能向/test 发送 5 次GET请求
  - 提示:
    - request.META['REMOTE_ADDR'] 可以得到远程客户端的IP地址
    - request.path_info 可以得到客户端访问的GET请求路由信息
  - 答案:

```
from django.http import HttpResponse, Http404
from django.utils.deprecation import MiddlewareMixin
```

```python
import re
class VisitLimit(MiddlewareMixin):
    '''此中间件限制一个IP地址对应的访问/user/login 的次数不能改过10次,超过后禁止使用'''
    visit_times = {}  # 此字典用于记录客户端IP地址有访问次数
    def process_request(self, request):
        ip_address = request.META['REMOTE_ADDR']  # 得到IP地址
        if not re.match('^/test', request.path_info):
            return
        times = self.visit_times.get(ip_address, 0)
        print("IP:", ip_address, '已经访问过', times, '次!:', request.path_info)
        self.visit_times[ip_address] = times + 1
        if times < 5:
            return

        return HttpResponse('你已经访问过' + str(times) + '次，您被禁止了')
```

跨站请求伪造保护 CSRF

- 跨站请求伪造攻击
  - 某些恶意网站上包含链接、表单按钮或者JavaScript，它们会利用登录过的用户在浏览器中的认证信息试图在你的网站上完成某些操作，这就是跨站请求伪造(CSRF，即Cross-Site Request Forgey)。
- 说明:
  - CSRF中间件和模板标签提供对跨站请求伪造简单易用的防护。
- 作用:
  - 不让其它表单提交到此 Django 服务器
- 解决方案:
  1. 取消 csrf 验证(不推荐)
     - 删除 settings.py 中 MIDDLEWARE 中的 `django.middleware.csrf.CsrfViewMiddleware` 的中间件
  2. 通过验证 csrf_token 验证

     需要在表单中增加一个标签

     ```
     {% csrf_token %}
     ```

Django中的forms模块

- 在Django中提供了 forms 模块,用forms 模块可以自动生成form内部的表单控件,同时在服务器端可以用对象的形式接收并操作客户端表单元素，并能对表单的数据进行服务器端验证
- forms模块的作用
  - 可以用类来描述表单内部的控件，最终生成 HTML 格式的表单内容
  - 可以用 forms 模块进行表单验证

用form生成表单

- Form表单类示意：
  - Django Form 实现 Form 表单

    ```python
    class MySearch(forms.Form):
        input_text = forms.CharField(label = '请输入内容')
    ```

  - 此 form 表单生成的代码

    ```
    <label for="id_input_text">请输入内容:</label>
    <input type="text" name="input_text" id="id_input_text" />
    ```

- 使用 forms 模块的步骤
  1. 在应用中创建 forms.py
  2. 导入 django 提供的 forms 模模
     - `from django import forms`
  3. 创建一个表单类，并在表单类内添加相应的控件

     ```
         class Form表单类名(forms.Form):
             表单元素 = forms.Field类型(参数...)
             ...
     ```

4. 创建表对象生成表单

```
    form1 =  FormName()
    html = form1.as_p()  # html 绑定表单内部的input标签...
```

5. 利用Form 类型的对象自动成表单内容

```
    {{ form1.as_p }}
```

3. forms.Form 示例:
    1. 创建表单类

```
from django import forms

class RegForm(forms.Form):
    username = forms.CharField(max_length=30, label='请输入用户名')
    password = forms.CharField(max_length=30, label='请输入密码')
    password2 = forms.CharField(max_length=20, label='请再次输入密码')
    ... ...
```

    4. 字段参数
        1. label
            - 控件前的文本
        2. widget
            - 指定小部件
        3. initial
            - 控件的初始值(主要针对文本框类型)
        4. required
            - 是否为必填项，值为(True/False)，默认为True
5. 在模板中解析form对象
    1. 方法
        1. 需要自定义
        2. 表单中的按钮需要自定义
    2. 解析form
        - 在 视图中创建form对象并发送到模板中解析.
        - 示例

```
form = XXXForm()
return render(request,'xx.html',locals())
```

        1. 手动解析

```
{% for field in form %}
    field : 表示的是form对象中的每个属性(控件)
    {{field.label}} : 表示的是label参数值
    {{field}} : 表示的就是控件
{% endfor %}
```

    3. 自动解析
        1. {{form.as_p}} `将 form 中的每个属性(控件/文本)都使用p标记包裹起来再显示`
        2. {{form.as_ul}}

```
将 form 中的每个属性(控件/文本)都使用li标记包裹起来再显示
注意:必须手动提供ol 或 ul 标记
```

        3. {{form.as_table}}

```
将 form 中的每个属性(控件/文本)都使用tr标记包裹起来再显示
注意:必须手动提供table标记
```

    4. 通过 forms 对象获取表单数据
        1. 通过 forms.Form 子类的构造器来接收 post 数据
            - form = XXXForm(request.POST)

2. 必须是 form 通过验证后,才能取值
    - form.is_valid()
        - 返回True:通过验证,可以取值
        - 返回False:暂未通过验证,则不能取值
3. 通过 form.cleaned_data 字典的属性接收数据
    - form.cleaned_data : dict 类型

6. Field 内置小部件 - widget
    1. 什么是小部件
        - 表示的是生成到网页上的控件以及一些其他的html属性

```
message=forms.CharField(widget=forms.Textarea)
upwd=forms.CharField(widget=forms.PasswordInput)
```

    2. 常用的小部件类型

widget名称	对应和type类值
TextInput	type='text'
PasswordInput	type='password'
NumberInput	type="number"
EmailInput	type="email"
URLInput	type="url"
HiddenInput	type="hidden"
CheckboxInput	type="checkbox"
CheckboxSelectMultiple	type="checkbox"
RadioSelect	type="radio"
Textarea	textarea标记
Select	select标记
SelectMultiple	select multiple 标记

7. 小部件的使用
    1. 继承自forms.Form
        1. 基本版
            1. 语法

```
属性 = forms.CharField() #无预选值使用
    text,password,email,url,textarea,checkbox
属性 = forms.ChoiceField() #有预选值使用
    checkbox,radio,select

属性 = forms.CharField(
    label='xxx',
    widget=forms.小部件类型
)
```

            2. 示例:

```
upwd = forms.CharField(
    label='用户密码',
    widget=forms.PasswordInput
)

message = forms.CharField(
    label='评论内容',
    widget=forms.Textarea
)
```

- 文档参见https://docs.djangoproject.com/en/1.11/topics/forms/

Django之form表单验证

- django form 提供表单和字段验证
- 当在创建有不同的多个表单需要提交的网站时，用表单验证比较方便验证的封装
- 当调用form.is_valid() 返回True表示当前表单合法，当返回False说明表单验证出现问题
- 验证步骤:
    1. 先对form.XXXField() 参数值进行验证，比如:min_length,max_length,如果不符合form.is_valid()返回False
    2. 对各自from.clean_zzz属性名(self): 方法对相应属性进行验证,如果验证失败form.is_valid()返回False
    3. 调胳form.clean(self): 对表单的整体结构进行验证，如果验证失败form.is_valid()返回False
    4. 以上验证都成功 form.is_valid()返回True
- 验证方法:
    - def clean_xxx属性(self):
        - 验证失败必须抛出forms.ValidationError
        - 验证成功必须返回xxx属性的值
    - def clean(self):
        - 验证失败必须抛出forms.ValidationError
        - 验证成功必须返回 self.cleaned_data
- 文档参见https://docs.djangoproject.com/en/1.11/ref/forms/validation/
- 验证示例

```
from django import forms
class RegisterForm(forms.Form):
    username = forms.CharField(label='用户名')
    password = forms.CharField(label='请输入密码', widget=forms.PasswordInput)
    password2 = forms.CharField(label='再次输入新密码', widget=forms.PasswordInput)

    def clean(self):
        pwd1 = self.cleaned_data['password']
        pwd2 = self.cleaned_data['password2']
        if pwd1 != pwd2:
            raise forms.ValidationError('两次密码不一致!')
        return self.cleaned_data  # 必须返回cleaned_data

    def clean_username(self):
        username = self.cleaned_data['username']
        if len(username) < 6:
            raise forms.ValidationError("用户名太短")
        return username
```

分页

- 分页是指在web页面有大量数据需要显示，为了阅读方便在每个页页中只显示部分数据。
- 好处:
    1. 方便阅读
    2. 减少数据提取量，减轻服务器压力。
- Django提供了Paginator类可以方便的实现分页功能
- Paginator类位于`django.core.paginator` 模块中。

Paginator对象

- 对象的构造方法
    - Paginator(object_list, per_page)
    - 参数
        - object_list 需要分类数据的对象列表
        - per_page 每页数据个数
    - 返回值:
        - 分页对象
- Paginator属性
    - count：需要分类数据的对象总数
    - num_pages：分页后的页面总数
    - page_range：从1开始的range对象, 用于记录当前面码数
    - per_page 每页数据的个数

- 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：当前页上所有数据对象的列表
  - 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

```
from django.core.paginator import Paginator
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>
```

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。可通能过`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
- 上传文件的表单书写方式

```
<!-- file: index/templates/index/upload.html -->
<html>
<head>
    <meta charset="utf-8">
    <title>文件上传</title>
</head>
<body>
    <h3>上传文件</h3>
    <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
...
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_view)
]
```

- 上传文件的视图处理函数

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

def upload_view(request):
    if request.method == 'GET':
        return render(request, 'index/upload.html')
```

```
    elif request.method == "POST":
        a_file = request.FILES['myfile']
        print("上传文件名是:", a_file.name)

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

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