模板

模板用于快速生成动态页面返回给客户端，模板是一个文本，用于分离文档的表现形式和内容。 模板定义了占位符以及各种用于规范文档该如何显示的模板标签。模板通常用于产生HTML，但是Django的模板也能产生任何基于文本格式的文档。模板包含两部分：

- html代码
- 模板标签

一、模板位置

- 在应用中建立templates目录，好处不需要注册，不好的地方，有多个应用的时候不能复用页面
- 第二种是放在工程的目录下，好处是如果有多个应用，可以调用相同的页面，需要注册
    - 需要修改项目的配置文件setings.py

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')], #模板绝对路
径
        'APP_DIRS': True, #是否在应用目录下查找模板文件
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

> Django 模板查找机制： Django 查找模板的过程是在每个 app 的 templates 文件夹中找（而不只是当前 app 中的代码只在当前的 app 的 templates 文件夹中找）。各个 app 的 templates 形成一个文件夹列表，Django 遍历这个列表，一个个文件夹进行查找，当在某一个文件夹找到的时候就停止，所有的都遍历完了还找不到指定的模板的时候就是 Template Not Found （过程类似于Python找包）。这样设计有利当然也有弊，有利是的地方是一个app可以用另一个app的模板文件，弊是有可能会找错了。所以我们使用的时候在 templates 中建立一个 app 同名的文件夹，这样就好了。

二、模板的渲染

2.1 loader加载

好处是可以加载一次模板，然后进行多次渲染

```
from django.template import loader  #导入loader

def index(request):
    temp = loader.get_template('index.html')
    print(temp.__dict__)
    # 渲染模板，生出html源码
    res = temp.render(context={'content':'hello index'})
    print(res)
    return HttpResponse(res)
```

2.2 render

```
from django.shortcuts import render
render(request,templatesname,context=None)
参数:
    request:   请求对象
    templatesname：模板名称
    context：参数字典，必须是字典
```

三、模板语法

django模板中包括两部分：变量和内置标签。变量会在模板渲染时被其值代替，内置标签负责逻辑控制。

3.1 变量

变量在模板中的表示为：{{ 变量名 }}，变量名就是render中context中的键。变量可以基本类型中的数值、字符串、布尔，也可以是字典、对象、列表等。django提供了点号来访问复杂数据结构。

- 列表、元组的元素可以使用索引引用，不能使用负索引，语法：变量.索引
- 字典： 字典变量.key
- 对象： 对象.属性 对象.方法名（方法不能有参数）

当模板系统在变量名中遇到点时，按照以下顺序尝试进行查找:

- 字典类型查找
- 属性查找
- 方法调用
- 列表类型索引

如果模板中引用变量未传值，则会被置为空，不会报错，除非你对其进行了操作。

3.2 过滤器

过滤器是在变量显示之前修改它的值的一个方法，过滤器使用管道符。过滤器可以串联调用

```
{{ 变量|方法 }}
```

常见的过滤器方法:

方法名	作用	示例
default	缺省值	{{ li|default:"缺省值" }}
default_if_none	如果变量是none 则显示缺省值	{{ value|default_if_none:'hello' }}
cut	从字符中删除指定字符	{{ value | cut:' ' }} 删除value所有空格
length	获取字符串或列表的长度	{{ str1|length }}
lower	将所有字母都变为小写	
upper	将所有字母都变为大写	
truncatechars	截取字符串前n个字符	{{ value|truncatechars:9 }}
date	格式化日期字符串	{{ value|date:"Y-m-d H:i:s" }}
add	增加变量的值	{{ num|add:"3" }}
divisibleby	把变量的值除以指定值	{{ value|divisibleby:"3" }}
first	获取列表第一个元素	{{ value|first }}
last	获取列表最后一个元素	
join	将列表内容链接为一个字符串	{{ value|join:'-' }}
autoescape	设置或取消转义	{% autoescape off %}{{ data }}{% endautoescape %}

- 自定义过滤器

  内置过滤器功能有限，如果不能满足需求，可以自己定义过滤器。

- 在app里创建一个包：templatetags
- 在包里创建一个py文件

```
from django import template
# 实例化自定义过滤器注册对象
register = template.Library()

# name代表在模板中使用的过滤器的名称
@register.filter(name='hello')
def hello(value,arg):
    """
    :param value: 传给hello过滤的值
    :param arg: hello自带的参数
    :return:
    """
    return value + str(arg)

@register.filter('time_ago')
def time_ago(value):
    """
    定义一个距离当前时间多久之前的过滤器
    :param value:
    :return:
    1.如果时间间隔小于1分钟内,那么就显示刚刚
    2.如果时间间隔大于1分钟小于1小时,那么就显示xx分钟前
    3.如果时间间隔大于1小时小于24小时,那么就显示xx小时前
    4.如果时间间隔大于24小时小于30天,那么就显示xx天前
    5.如果时间间隔大于30天,那么就显示具体时间
    """
    if not isinstance(value, datetime.datetime):
        return value
    now = datetime.datetime.now()
    timestamp = (now - value).total_seconds()
    if timestamp < 60:
        return '刚刚'
    elif timestamp >= 60 and timestamp < 60 * 60:
        return '{}分钟前'.format(int(timestamp / 60))
    elif timestamp >= 60 * 60 and timestamp < 60 * 60 * 24:
        return '{}小时前'.format(timestamp / 60 / 60)
    elif timestamp >= 60 * 60 * 24 and timestamp < 60 * 60 * 23 *
30:
        return '{}天前'.format(int(timestamp / 60 / 60 / 24))
```

```
        else:
            return value.strftime('%Y-%m-%d %H:%M')
```

- 在模板中使用

```
{% load customfilter %}  #加载自定义过滤器的模块
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
{{ name |hello:' how are you' }} #使用自定义过滤器
</body>
</html>
```

3.3 内置标签

语法：{% tag %}

1.if标签

```
{% if express1 %}
    # to do
{% elif express2 %}
    # to do
{% else %}
    # to do
{% endif %}
```

- 可以在表达式中使用以下运算符（优先级从高到低）：
  - < >= <= == !=
  - in 、not in
  - is、is not
  - not
  - and
  - or
- 不要在表达式中使用（），可以使用if嵌套实现功能
- 不支持 if 3 < b < 5这种写法

2.for

遍历可迭代对象

```
{% for x in y %}
    ...
{% endfor %}
```

- 反向迭代(reversed)

```
{% for value in c [1,2,3,4,5] reversed %}
    <span>{{ value }}---</span>
{% endfor %}
```

- empty 当可迭代对象为空或不存在时执行，否则不执行

```
{% for value in c  %}
    <span>{{ value }}---</span>
{% empty %}
    数据不存在
{% endfor %}
```

- 字典迭代

```
# e = {'a1':20,'b1':40}
{% for k,v in e.items %}
    <div>{{ k }}---{{ v }}</div>
{% endfor %}
```

- 获取for循环迭代的状态

变量名称	变量说明
forloop.counter	获取迭代的索引 从1开始
forloop.counter0	获取迭代的索引 从0开始
forloop.revcounter	迭代的索引从最大递减到1
forloop.revcounter0	迭代的索引从最大递减到0
forloop.first	是否为第一次迭代
forloop.last	是否为最后一次迭代
forloop.parentloop	获取上层的迭代对象

```
{% for i in c %}
    <li>{{ forloop.first }}</li>
    <li>{{ forloop.last }}</li>
    <li>{{ forloop.counter }}</li>
    <li>{{ forloop.counter0 }}</li>
    <li>{{ forloop.revcounter }}</li>
    <li>{{ forloop.revcounter0 }}</li>
{% endfor %}
```

3. ifequal/ifnotequal

用于判断两个值相等或不等的

```
{% ifequal var var %}
{% endifequal %}

{% ifnotequal var var %}
{% endifnotequal %}
```

4.注释

- 单行注释

```
{# 注释的内容 #}
```

- 多行注释

```
{% comment %}
    ...
{% endcomment %}
```

5.跨站请求伪造 csrf

防止网站受第三方服务器的恶意攻击（确定表单到底是不是本网站的表单传递过来的）。csrf相当于在表达中增加了一个隐藏的input框，用于向服务器提交一个唯一的随机字符串用于服务器验证表单是否是本服务器的表单。

使用：

settings.py

```
MIDDLEWARE = [
    'django.middleware.csrf.CsrfViewMiddleware',
]
```

表单里

```
<form action="" method="post">
    {% csrf_token %}
    <input type="text" name="username">
    <p><input type="submit"></p>
</form>
```

- 全站禁用csrf

```
#在settings中设置
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    #'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

- 局部禁用csrf

```
#在不想检验csrf的视图函数前添加装饰器@csrf_exempt。
from django.views.decorators.csrf import csrf_exempt,csrf_protect

@csrf_exempt
def csrf1(request):
    pass
```

- ajax验证csrf

```
Ajax提交数据时候，携带CSRF：
a. 放置在data中携带

<form method="POST" action="/csrf1.html">
    {% csrf_token %}
    <input id="username" type="text" name="username" />
    <input type="submit" value="提交"/>
    <a onclick="submitForm();">Ajax提交</a>
</form>
<script src="/static/jquery-1.12.4.js"></script>
<script>
    function submitForm(){
        var csrf = $('input[name="csrfmiddlewaretoken"]').val();
        var user = $('#user').val();
        $.ajax({
            url: '/csrf1.html',
            type: 'POST',
            data: { "user":user,'csrfmiddlewaretoken': csrf},
            success:function(arg){
                console.log(arg);
            }
        })
    }

</script>
```

注意：

csrf的意义在于 给每一个表单都设置一个唯一的csrf的值 并且cookie也存储一份 当提交表单过来的时候 判断cookie中的值 和csrf_token中的值 是否都为本网站生成的 如果验证通过则提交 否则 403

6.模板导入标签（ include）

可以把指定html文件代码导入到当前文件，实现模板代码的复用/重用。语法格式：

```
{% include '路径/xxx.html' %}
```

7. url标签

在模板中url标签可用于反向解析

```
<h2><a href="{% url 'App:index' %}">动态生成路由地址不带参的跳转</a>
</h2>
<h2><a href="{% url 'App:args1' 1 2 %}">动态生成路由地址带参的跳转</a>
</h2>
<h2><a href="{% url 'App:args1' num1=1 num2=2 %}">动态生成路由地址带关
键字参数的跳转</a></h2>
```

四、模板继承

在整个网站中，如何减少共用页面区域（比如站点导航）所引起的重复和冗余代码？Django 解决此类问题的首选方法是使用一种优雅的策略—— 模板继承 。

本质上来说，模板继承就是先构造一个基础框架模板，而后在其子模板中对它所包含站点公用部分和定义块进 行重载。

- {% extends %} 继承父模板
- {% block %} 子模板可以重载这部分内容。
- {{ block.super }}调用父模板的代码

使用继承的一种常见方式是下面的三层法:

- 创建base.html模板，在其中定义站点的主要外观感受。这些都是不常修改甚至从不修改的部分。
- 为每种类型的页面创建独立的模板，例如论坛页面或者图片库。这些模板拓展相应的区域模板。
- 自己的页面继承自模板，覆盖父模板中指定block

注意事项：

- 如果在模板中使用 {% extends %} ，必须保证其为模板中的第一个模板标记。否则，模板继承将不起作 用。

- 一般来说，基础模板中的 {% block %} 标签越多越好。
- 如果发觉自己在多个模板之间有重复代码，你应该考虑将该代码放置到父模板的某个 {% block %} 中。
- 不在同一个模板中定义多个同名的 {% block %} 。
- 多数情况下， {% extends %} 的参数应该是字符，但是如果直到运行时方能确定父模板名称，这个参数也 可以是个变量。

五、静态资源配置

什么是静态资源：css、js、images 需要从外部导入的资源

5.1创建static文件夹（通常放在根目录下）

5.2需要在settings注册

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
```

5.3在模板中使用静态资源

```
 {% load static %}  #放置到模板开头
 <img src="/static/img/img.jpeg" alt="">  #硬编码
 <img src="{% static 'img/img.jpeg' %}" alt="">  #动态写法，建议用这种
```