第124天： Web 开发 Django 模板

原创 極光 Python技术 2月11日

上次为大家介绍了 Django 的模型和自带的管理工具，有了这个工具就可以全自动地根据模型创建后台管理界面，以供网站管理者更方便的管理网站数据。有了网站数据，那怎么样更方便又好看的展示给用户看呢？目前流行的 Web 框架基本都采用 `MVC` 的架构，而 Django 在这个架构基础上做了一点改变，即 `MTV` 框架，这里的 `T` 就是今天我们要讲的 Django 的模板系统（ Template ）。

认识模板

首先模板是一个文本文件，它定义了占位符以及各种用于规范文档该如何显示的各部分基本逻辑，主要用于动态生成 `HTML` ，即模板包含所需 `HTML` 输出的静态部分以及描述动态内容将被插入的一些特殊语法。Django 默认内置了一个模板系统 `DTL` （ Django template language ），在一般项目开发中已足够使用，当然你也可以选择其他第三方模板，一般 Django 项目可以配置一个或多个模板引擎。

使用模板

Django 模板系统其实是 Python 的一个库，只要在代码中引用它，就可以创建并使用模板，但通常情况我们都是把模板和视图结合一起使用的。使用模板一般分两步：

1. 创建一个模板文本，或者直接通过文本路径来创建 `Template` 对象，当然也可以直接在代码中通过字符串创建对象。
2. 调用 `Template` 对象的 `render()` 方法，并传入相应 <K,V> 变量对象，这样我们在模板中配置的变量就会替换为相应的值。

现在我们继续在之前 `TestProject` 项目下的 `polls` 应用里面修改代码，进入到 `polls` 目录下，编辑 `views.py` 文件如下：

```
# views.py

from django.http import HttpResponse
from polls.models import Choice, Question
from django.utils import timezone
# 导入模板相关库
from django.template import Template, Context

# 修改之前 index 页面为模板渲染
def index(request):
    # 定义模板
    t = Template('你好<span style="color:#FF0000">{{ name }}</span>, 这是一个投票页面。')
    # 定义传入对象
    c = Context({'name': '张三'})
    # 返回渲染模板
    return HttpResponse(t.render(c))

#     忽略部分代码
```

```
# ..... 忽略部分代码
```

修改完成并保存，先通过命令 `python manage.py runserver 127.0.0.1:8080` 启动本地 server，然后通过访问 URL `http://127.0.0.1:8080/polls` 就能看到我们刚修改完的页面，展示如下图所示：

模板语法规则

上面我们完成了使用模板来输出数据，从而实现了数据与视图的分离。下面我们再来详细介绍下在模板中常用的语法规则：

1. 条件判断语法 基本语法：

```
{% if 条件 %}
#    满足条件展示内容
{% endif %}

# 或者

{% if 条件1 %}
#    满足条件1展示
{% elif 条件2 %}
#    满足条件2展示
{% else %}
#    不满足条件1条件2的展示
{% endif %}
```

另外，条件判断还支持 and ， or 或者 not 关键字来对多个变量进行判断，模板代码如下：

```
{% if 变量1 and 变量2 %}
#    当变量1或者变量2为 true ，那就展示此处模板内容
{% endif %}
```

现在我们继续在 `TestProject` 项目下的 `polls` 应用里面修改代码，进入到 `polls` 目录下，编辑 `views.py` 文件如下：

```
# views.py

from django.http import HttpResponse
from polls.models import Choice, Question
from django.utils import timezone
# 导入模板相关库
from django.template import Template, Context

# 修改之前 index 页面为模板渲染
```

```
def index(request):
    # 定义模板
    t = Template('{%if name %}你好<span style="color:#FF0000">{{ name }}</span>,{% else %} 你好游
    # 定义传入对象
    c = Context({'name': '张三'})
    # 返回渲染模板
    return HttpResponse(t.render(c))

# …… 忽略部分代码
```

刷新页面，发现跟之前页面没有区别，因为我们这里 `Context` 里传入了 `name` 变量，如果不传 `name` 变量，得到的结果如下图：

2. 循环迭代语法 和大多数语言相同，循环是通过 for 语法实现，每一次循环中，模板系统会渲染在 {% for %} 和 {% endfor %} 之间的所有内容。

```
{% for book in book_list %}
    <li>{{ book.name }}</li>
{% endfor %}

# 或者 可以增加 reversed 标签反向迭代 book_list

{% for book in book_list reversed %}
    <li>{{ book.name }}</li>
{% endfor %}
```

我们继续在 `TestProject` 项目下的 `polls` 应用里面修改代码，编辑 `views.py` 文件如下：

```
# views.py

from django.http import HttpResponse
from polls.models import Choice, Question
from django.utils import timezone
# 导入模板相关库
from django.template import Template, Context

# 修改之前 index 页面为模板渲染
def index(request):
    # 定义模板
    t = Template(' 以下{% for name in name_list %} <li>{{ name }}</li>  {% endfor %}  请选择?')
    # 定义传入对象
    c = Context({'name_list': ('张一','张二','张三')})
    # 返回渲染模板
    return HttpResponse(t.render(c))

# …… 忽略部分代码
```

刷新页面，这里在 `Context` 里传入了 `name_list` 列表变量，使用循环迭代语法，得到的结果如下图：

3. 特殊比较语法

```
# ifequal/ifnotequal 语法用于更直接的比较两个变量是否相等
{% ifequal user 'admin' %}
    <h1>这是一个管理员页面！</h1>
{% endifequal %}

# 或者

{% ifnotequal user 'admin' %}
    <h1>这是是一个游客页面！</h1>
{% endifnotequal %}

# 和条件判断语法类似， {% ifequal %} 支持可选的 {% else%} 标签
{% ifequal user 'admin' %}
    <h1>这是一个管理员页面！</h1>
{% else %}
    <h1>这是是一个游客页面！</h1>
{% endifequal %}
```

我们继续在 `TestProject` 项目下的 `polls` 应用里面修改代码，编辑 `views.py` 文件如下：

```
# views.py

from django.http import HttpResponse
from polls.models import Choice, Question
from django.utils import timezone
# 导入模板相关库
from django.template import Template, Context

# 修改之前 index 页面为模板渲染
def index(request):
    # 定义模板
    t = Template('{% ifequal type 1 %}你好<span style="color:#FF0000"> 管理员</span>,这是个管理页面
    # 定义传入对象
    c = Context({'type': 1})
    # 返回渲染模板
    return HttpResponse(t.render(c))

# …… 忽略部分代码
```

刷新页面，这里在 `Context` 里传入了 `type` 变量，当变量传入1时，得到管理页面，如果不是1则还是原来的页面， `type` 传入 1时，得到的结果如下图：

4. 模板过滤器

模板过滤器可以在变量被显示前修改它，过滤器使用管道字符。另外过滤管道可以被套接，即一个过滤器管道的输出又可以作为下一个管道的输入。

```
# {{ book_name }} 变量被过滤器 lower 处理后，文档大写转换文本为小写。
 {{ book_name|lower }}

# 将变量第一个元素转化为大写字母
 {{ book_name|first|upper }}

# 将截取变量 book_name 的前10位字符
{{ book_name|truncatewords: "10" }}

```

当然除了上面三个，还有很多其他过滤器就不再一一介绍，如果有需要可自行搜索，总之可以看出过滤器大大方便我们在模板中的灵活操作。

5. 模板继承

模板继承就是说可以通过这种方式实现模板的复用，既然需要复用，那我们就需要先创建一个能被复用的模板文件。首先在项目根目录下，创建 `templates` 目录用来存放模板文件，并在该目录下创建 `common.html` 文件用于复用，编写如下代码：

```
<html>
  <head>
      <title>公共页面</title>
  </head>

  <body>
      <h1>你好<span style="color:#FF0000">{{ name }}</span>,这是一个公共页面!</h1>
      {% block mainbody %}
      <p>原始页</p>
      {% endblock %}
  </body>
</html>
```

然后再创建 `index.html` 模板文件，并在文件中编写代码，实现对 `common.html` 模板文件的利用。

```
  {% extends "common.html" %}

  {% block mainbody %}
  <p>这里继承了 common.html 模板! </p>
  {% endblock %}
```

最后修改项目目录下，`TestProject/settings.py` 配置文件，在配置文件找到 `TEMPLATES` 对象，并在修改其中 `DIRS` 的配置路径如下：

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR+"/templates",],     # 修改模板配置路径
        'APP_DIRS': True,
        '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',
            ],
        },
    },
]
```

好了，现在模板配置完成，下一步要修改我们的渲染页面，再次打开 `polls` 应用下的 `views.py`,把主页代码注释，然后重新编写主页代码如下：

```
from django.shortcuts import render
from django.http import HttpResponse
from polls.models import Choice, Question
from django.utils import timezone
from django.template import Template, Context

# 注释原代码
#def index(request):
#    return HttpResponse("你好，这是一个投票页面。")
#    t = Template('你好<span style="color:#FF0000">{{ name }}</span>, 这是一个投票页面。')
#    c = Context({'name': '张三'})
#    return HttpResponse(t.render(c))

# 增加新代码
def index(request):
    context = {}
    context['name'] = '张三'
    # 返回模板渲染后的主页界面
    return render(request, 'index.html', context)
```

修改完成，最后通过访问 URL `http://127.0.0.1:8080/polls` 就能看到我们刚修改完的页面，展示如下图所示：

这样就能更好的实现模板的复用了。

6. 模板包含

这个就是通过 {% include %} 语法标签，将一个模板引入另一个或多个模板的内容，代码如下：

```
{% include "common.html" %}

```

总结

本文为大家介绍了 Django Template 模板基础，可以通过应用的模板实现数据和展示的分离，这样我们可以对前端进行更灵活的展示操作。当然只是简单介绍了下模板，它还有更复杂的使用方式，以后会为大家深入介绍。

参考

Django 中文官网：https://docs.djangoproject.com/zh-hans/2.2

> 示例代码：https://github.com/JustDoPython/python-100-day

系列文章

第123天：Web 开发 Django 管理工具
第122天：Flask 单元测试
第121天：机器学习之决策树
从 0 学习 Python 0 - 120 大合集总结

PS： 公号内回复 ：Python，即可进入Python 新手学习交流群，一起**100天计划！**

-END-

Python 技术

关于 Python 都在这里