Django模板

模板和模板引擎

- 模板具有一定的格式或骨架，可以动态的生成HTML
- 模板引擎决定以何种方式组织代码
- 一个项目可以有一个或者是多个模板引擎

DTL

Jinja2

DTL介绍

- DTL（Django Template Language）是django原生的模板系统
- 直到Django1.8, 唯一的模板引擎支持

Jinja2介绍

- 速度更快，Python 的功能齐全的开源模板引擎
- 安装

  >> pip install jinja2

课程介绍

章节概要

- Django模板相关配置
- 模板变量的使用
- 模板标签的使用
- 模板的继承与包含
- 模板过滤器
- 自定义过滤器

课程目标

- 掌握模板引擎的配置
- 理解模板加载的顺序
- 掌握Django模板引擎中的变量、标签的使用
- 掌握模板的继承和包含
- 掌握过滤器、自定义过滤器的使用
- 掌握DTL与Jinja2的使用区别

Django模板相关配置

本节内容

- 掌握 settings.py中模板相关的配置
- 了解 多个模板引擎支持情况下配置
- 掌握 模板的加载顺序

templates相关配置

- 配置示例

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
        # 其它选项配置
        },
    },
]
```

配置选项

- BACKEND——模板引擎配置

  django.template.backends.django.DjangoTemplates

  django.template.backends.jinja2.Jinja2

- DIRS——模板引擎按列表顺序搜索这些目录以查找模板源文件

配置选项

- APP_DIRS——决定模板引擎是否应该进入每个已安装的应用中查找模板

  每种模板引擎后端都定义了一个惯用的名称作为应用内部存放模板的子目录名称

  DTL——templates目录

  Jinja2——jinja2目录

- OPTIONS——其他选项配置

同时支持两种模板引擎

- 添加配置模板引擎支持

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
    },
    {
        'BACKEND': 'django.template.backends.jinja2.Jinja2',
        'DIRS': [os.path.join(BASE_DIR,    'jinja2')],
    }
]
```

同时支持两种模板引擎

- 模板文件查找规则get_template('index.html')

  templates/index.html

  jinja2/index.html

小结

- 模板查找的顺序：

  按顺序查找、先根目录后模块目录

思考

如何在模板中使用Python变量？

模板变量的使用

本节内容

- 掌握 模板语法
- 掌握 Python中简单数据类型的渲染
- 掌握 Python中复杂数据类型的渲染
- 掌握 DTL与Jinja2的使用区别

渲染Python中的变量

- 语法

  {{ variable }}

- 当模版引擎遇到一个变量，它将计算这个变量，然后用结果替换掉它本身

- 变量名称中不能有空格或标点符号，不能以“_”开头

渲染静态图片

- 图片地址的变量

  img_url = ‘/media/images/dog.jpg’

- 模板文件中渲染

渲染Python中的对象

- 语法

  {{ object.attribute }}

- dict类型数据的渲染
- list/tuple类型数据的渲染
- list/tuple嵌套dict复杂类型数据的渲染

DTL与Jinja2的使用区别

- 注意：变量名称中不能有空格或标点符号
- 下面的语法在DTL中不被支持

  {{ object["a.b"] }}
  {{ object["a b"] }}

DTL与Jinja2的使用区别

- 类中的成员方法调用不需要()，也不支持参数传递

```
class A():
    def display():
        return 'hello'


a = A()
{{ a.display }}
```

小结

记住{{ variable }}语法和variable的命名

思考

Django模板引擎for循环有哪些不一样吗?

模板标签的使用

本节内容

- 了解 模板标签的作用
- 掌握 常用的的模板标签
- 掌握 DTL与Jinja2的使用区别

模板标签是什么

- 思考：使用变量时有哪些不方便的地方？
- list/tuple如果没有循环标签，肯定很难办
- 如果能加上if等条件判断就好了
- 页面较多，如果能共用部分页面就好了
- .....

模板标签的使用

- 语法

  {% tag %}

常用的模板标签

- 循环控制
- 条件控制
- 模板注释
- URL解析
- with语句块
- 当前时间显示
- 继承与包含

循环控制

- 语法参考

```
{%  for  item in data_list %}
<li>内容</li>
{% empty %}
<li>暂无内容</li>
{% endfor %}
```

循环控制

- 循环内的变量forloop

变量	描述
forloop.first	如果是第一次迭代，为 True
forloop.last	如果是最后一次迭代，为 True
forloop.counter0	计数器，从0开始
forloop.counter	计数器，从1开始

循环控制

- for循环dict

```
{% for key, value in data.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

- 重复循环(循环中再循环)

```
{% cycle 'row1' 'row2' %}
```

与Jinja2的区别

- 循环变量forloop, Jinja2中为loop
- List为空：{% empty %}，Jinja2中为{% else %}
- 循环中的再循环

  DTL: {% cycle 'odd' 'even’ %}

  Jinja2: {{ loop.cycle('odd', 'even') }}

- DTL不支持continue和break

条件控制

- 语法参考

```
{% if condition_a %}
    满足了A条件
{% elif condition_b %}
    满足了B条件
{% else %}
    都不满足
{% endif %}
```

- ifequal/ ifnotequal

```
{% ifequal a b %}
...
{% endifequal %}
```

- 循环内ifchanged

条件控制

- 其他逻辑控制
  - and, or
  - == , !=
  - > , <
  - >= , <=
  - in, not in
  - is

模板注释

- 添加注释

{# 注释内容 #}

{% comment "简单的描述" %}
    HTML内容 {{ create_date }}
{% endcomment %}

- 与HTML注释的区别

<!-- 注释内容 -->

URL解析

- URL标签的使用

{% url 'url_name' params %}

- static静态文件URL解析

```
{% load static %}
<img src="{% static "images/cat.jpg" %}" alt="Cat">
```

与Jinja2的区别

- DTL

  {% url 'url_name' params %}

- Jinja2

  {{ url_for('index') }}

with语句块

- 语法参考

```
{% with alpha=1 beta=2 %}
    ...
{% endwith %}
```

当前时间显示

- 当前时间显示

{% now "jS F Y H:i" %}

DTL与Jinja2的使用区别

- 区别一：循环中的区别
- 区别二：URL解析的区别

小结

DTL与Jinja2大同小异，注意使用区别

思考

模板如何实现像Flask中的继承？

模板的继承与包含

本节内容

- 了解 模板抽象和继承的使用场景
- 掌握 相关的几个标签
- 掌握 DTL与Jinja2的使用区别

使用场景

- 思考：如下场景怎样设计（易维护、可扩展）？

  每个页面都引用了公共的头部，js，css

  有几个页面结构和内容及其相似（如：导航菜单）

模板的抽象和继承

- 步骤一：将可变的部分圈出来(base.html)

```
{% block sidebar %}
    <!--菜单栏的内容 -->
{% endblock %}
```

- 步骤二：继承父模板

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

模板的抽象和继承

- 步骤三：填充新的内容(index.html)

```
{% extends "base.html" %}
{% block sidebar %}
    <!-- 新的菜单栏的内容 -->
{% endblock %}
```

模板的抽象和继承

- 步骤四：复用父模板的内容(可选)

```
{% extends "base.html" %}
{% block sidebar %}
    {{ block.super }}
    <!-- 新的菜单栏的内容 -->
{% endblock %}
```

在模板中添加公共部分

- 步骤一：将可变的部分拆出来(footer.html)

```
<footer>
 这是页脚公共的部分
</footer>
```

在模板中添加公共部分

- 步骤二：将拆出来的部分包进来（index.html)

```
{% extends "base.html" %}
{% block content %}
   <!– 页面主要内容区域-->

  {# 公用的footer #}
  {% include "footer.html" %}
{% endblock %}
```

小结

理解继承的好处，面向对象的思想无处不在

思考

在模板中如何改变变量的值?

模板过滤器

本节内容

- 了解 过滤器的作用
- 掌握 过滤器的使用
- 掌握 Django中常用的过滤器
- 掌握 DTL与Jinja2的使用区别

什么是过滤器

- 思考：我想对变量进行特殊处理后再渲染？

- 举例：将英语单词转换为大写

model => MODEL
view => VIEW

什么是过滤器

- 过滤器语法

{{ value|filter_name: params }}

- 实例：使用过滤器将字母大写

{{ value|upper}}

内置的过滤器

- 默认值显示

{{ value|default:"" }}
{{ value|default_if_none: “无" }}

- 数字四舍五入显示

{{ value | floatformat：3 }}

与Jinja2的区别

- DTL语法

{{ var|default:'我默认没有' }}

- Jinja2语法

{{ var|default('我默认没有') }}

内置的过滤器

- 日期对象格式化

{{ value|date:"D d M Y" }}

- 时间对象格式化

{{ value|time:"H:i" }}

格式化选项传送门>>

内置的过滤器

- 富文本内容转义显示

{{ value|safe }}

- 字符串截取

{{ value|truncatechars:9 }}
{{ value|truncatechars_html:9 }}
{{ value|truncatewords:2 }}

小结

过滤器传参，记住和Jinja2最大的区别

思考

DTL中如何自定义过滤器?

自定义过滤器

本节内容

- 了解 自定义过滤器的使用场景
- 掌握 Django中如何自定义过滤器
- 了解 与Flask中自定义过滤器的区别

使用场景

- 思考：在什么情况下需要自定义过滤器？

自定义过滤器

- 步骤一：在app模块目录下新建包templatetags

```
polls/
    __init__.py
    models.py
    templatetags/
        __init__.py
        poll_extras.py
    views.py
```

自定义过滤器

- 步骤二：实现过滤器poll_extras.py

```
from django import template
register = template.Library()

def warning(value):
    """ 将第一个字符变红 """
    return '<span class="red">' + value[0] +
'</span>+ value[1:]
```

自定义过滤器

- 步骤三：注册过滤器

```
# 方式一：注册过滤器
register.filter('warning', warning)

# 方式二：注册过滤器
@register.filter(name='warning')
def warning(value):
    pass
```

自定义过滤器

- 步骤四：在模板中使用过滤器

```
{% load poll_extras %}
{{ value| warning }}
```

自定义过滤器

- 注意

添加自定义过滤器后记得重启开发服务器

模块需要添加到settings.py中的INSTALLED_APPS内

自定义过滤器

- 要求如下

张三 => 张***
李四 => 李***

Flask中的自定义过滤器

- 方式一：使用装饰器注册

```
@app.template_filter('reverse')
def reverse_filter(s):
    return s[::-1]
```

- 方式二：调用函数注册

```
def reverse_filter(s):
    return s[::-1]
app.jinja_env.filters['reverse'] = reverse_filter
```

小结

看起来容易，自己动手试一下?

课程总结

知识点回顾

重难点

- 模板的配置和渲染原理
- 模板变量、标签的使用
- 模板过滤器、自定义过滤器
- 模板的抽象和继承
- DTL与Jinja2的使用区别