djnago模板引擎

Ringo Yungpo Kao

January 01, 2018

# djnago模板引擎

Django 1.8 支持很多模板引擎，但是内置的只有 DTL 和 Jinja2。

## 语法区别

### 不同的地方：

|  |  |  |
| --- | --- | --- |
| 目的 | DTL | Jinja2 |
| 方法调用 | {{ user.get\_favorites }} | {{ user.get\_favorites() }} |
| 过滤器参数 | {{ toppings | join:”, “ }} |
| 空循环 | {% empty %} | {% else %} |
| 循环变量 | {{ forloop }} | {{ loop }} |
| Cycle | {% cycle “odd” “even” %} | {% loop.cycle(“odd”, “even”) %} |

### 相似的地方

|  |  |  |
| --- | --- | --- |
| 目的 | DTL | Jinja2 |
| 条件 | {% if topping==”sprinkles” %} | {% if topping==”sprinkles” %} |
| 条件 | {% elif topping==”fudge” %} | {% elif topping==”fudge” %} |
| 条件 | {% else %} | {% else %} |
| is 操作符 | {% customer is happy %} | {% customer is happy %} |

### 需要换到 Jinja2 吗？

实际上在同一个项目中可以同时使用这两个引擎。DTL 有大量的资源，而 Jinja2 性能更好。

### DTL 优点

* 文档齐全，例子丰富。
* DTL+Django 的项目比 Jinja2+Django 的项目更成熟。
* 大部分第三方应用都使用 DTL。
* 将 DTL 转换成 Jinja2 要花大量时间。

### Jinja2 的优点

* 能独立于 Django 使用。
* Jinja2 的语法更像 Python，因而更直观。
* 更加显式，比如函数调用用括号。
* 逻辑限制更少，可以处理更多逻辑。
* 性能更好。

### 应该用哪个？

* 新用户应该用 DTL。
* 使用了 DTL 的老项目应该保持用 DTL，除了一些需要改善性能的页面。
* 经验丰富的用户应该先尝试两者，再酌情选取。

## 选用 Jinja2 后的注意事项

### CSRF 和 Jinja2

Jinja2 使用 Django CSRF 机制与 DTL 不同。使用如下代码将 CSRF 加入到 Jinja2 的模板中：

<div style="display:none;">  
 <input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf\_token }}">  
</div>

### 在 Jinja2 模板中使用 Tag

目前还不支持 Tag，不过可以：

* 将 Tag 的功能转换成一个函数，然后使用。
* 创建 Jinja2 的扩展。

### 在 Jinja2 模板中使用 Django 风格的模板过滤器

由于 Django 的默认模板过滤器就是函数，因此我们可以定制一个 Jinja2 环境，将它们包含进来：

# core/jinja2.py  
from \_\_future\_\_ impor absolute\_import # Python 2 only  
   
   
from django.contrib.staticfiles.storage import staticfiles\_storage  
from django.core.urlresolvers import reverse  
from django.template import defaultfilters  
   
from jinja2 import Environment  
   
def environment(\*\*options):  
 env = Environment(\*\*options)  
 env.globals.update({  
 'static': staticfiles\_storage.url,  
 'url': reverse,  
 'dj': defaultfilters  
 })  
 return env

上面的设置代码将所有的默认过滤器函数都放在了 dj 中。

以下是在 Jinja2 模板中将 Django 模板过滤器作为函数使用的例子：

<table><tbody>  
{% for purchase in purchase\_list %}  
 <tr>  
 <a href="{{ url("purchase:detail", pk=purchase.pk) }}">  
 {{ purchase.title }}  
 </a>  
 </tr>  
 <tr>{{ dj.date(purchase.created, "SHORT\_DATE\_FORMAT") }}</tr>  
 <tr>{{ dj.floatformat(purchase.amount, 2) }}</tr>  
{% endfor %}  
</tbody></table>

如果不想用这种全局的方式，也可以用 Mixin 的方式，然后在模板中通过 view 变量来访问。

先创建一个 Mixin：

# core/mixins.py  
from django.template import defaultfilters  
   
class DjFilterMixin(object):  
 dj = defaultfilters

然后，当视图继承了该 Mixin 后，就能在 Jinja2 模板中使用了：

<table><tbody>  
{% for purchase in purchase\_list %}  
 <tr>  
 <a href="{{ url("purchase:detail", pk=purchase.pk) }}">  
 {{ purchase.title }}  
 </a>  
 </tr>  
 <!-- Call the django.template.defaultfilters functions from the view -->  
 <tr>{{ view.dj.date(purchase.created, "SHORT\_DATE\_FORMAT") }}</tr>  
 <tr>{{ view.dj.floatformat(purchase.amount, 2) }}</tr>  
{% endfor %}  
</tbody></table>

### Jinja2 模板不会调用 Context Processors 中的函数

context processors 是在 DTL 设置参数 settings.TEMPLATES 中的 context\_processors 项中指定的一系列可调用对象，这些可调用对象接受一个 request 对象为入参，返回一个字典，字典的内容会合并到模板上下文对象中。

要在 Jinja2 中实现类似的功能，应将 context\_processors 中的功能用 middleware 实现。

假设要实现在每个模板中都添加一个广告，原来 DTL 中用 context\_processors 实现如下：

# advertisements/context\_processors.py  
import random  
   
from advertisements.models import Advertisement as Ad  
   
def advertisements(request):  
 count = Advertisement.objects.filter(subject='ice-cream').count()  
 ads = Advertisement.objects.filter(subject='ice-cream')  
 return {'ad': ads[random.randrange(0, count)]}

然后在 base.html 中：

<!-- base.html -->  
...  
<div class="ice-cream-advertisement">  
 <a href="{{ ad.url }}">  
 <img src="{{ ad.image }}" />  
 </a>  
</div>  
...

以上的代码不能在 Jinja2 模板中使用，但可以用 middleware 实现类似功能：

# advertisements/middleware.py  
import random  
   
from advertisements.models import Advertisement as Ad  
   
def AdvertisementMiddleware(object):  
   
 def process\_request(request):  
 count = Advertisement.objects.filter(subject='ice-cream').count()  
 ads = Advertisement.objects.filter(subject='ice-cream')  
 # If necessary, add a context variable to the request object.  
 if not hasattr(request, 'context'):  
 request.context = {}  
 # Don't overwrite the context, instead we build on it.  
 request.context.update({'ad': ads[random.randrange(0, count)]})

然后在 Jinja2 模板中使用：

<!-- base.html -->  
{% set ctx = request.context %}  
...  
<div class="ice-cream-advertisement">  
 <a href="{{ ctx.ad.url }}">  
 <img src="ctx.ad.image.url" />  
 </a>  
</div>  
...

### Jinja2 的 Environment 对象应该认为是静态的

Jinja2 通过 jinja2.Environment 类的实例在模板间共享配置信息、过滤器、全局变量等。当第一个模板加载时，Jinja2 会对它进行初始化，然后保持不变，因而它本质上是一个 static object。