Django实现自定义标签

通过前面几节的内容，我们对 Django 内置的模板标签与过滤器做了深入的探讨学习。Django 虽然内置了二十多种标签和六十多种过滤器，但是为了给 Web 开发者提供更好使用体验，Django 也提供了自定义标签与过滤器的功能。当内置标签与过滤器满足不了实际业务的需求，那么我们就可以通过自定义的方式去实现，在本节我们将对如何实现自定义标签进行讲解。

1. 如何实现自定义标签

自定义标签可以分为三种类型：简单标签（simple_tag）、引用标签（inclusion_tag）、赋值标签（assignment_tag），在本节我们对它们进行详细的描述。

1) 定义之前的准备工作

Django 为我们提供了自定义的机制，我们可以通过使用 Python 代码来自定义标签来，最后使用{% load %}标签进行加载。但是在自定义标签之前，需要我们做一些准备工作，如下所示：

- 创建专门的应用来装载自定义标签或者在项目原始 app 上进行自定义，在这里我们依旧使用原有的 index 应用；
- 在 index 应用下创建名为 templatetags（名字不能变） 的 Python 包 ，并在包中新建__init__.py 文件作为 ；
- 在新建的 Python 包中新建一个名为 index_tags.py 文件，该文件命名时避免与内置标签与过滤器名字冲突；
- 在 INSTALLED_APPS 列表中注册 app，因为 index 应用之前已经注册，所以就无须操作了，若是新建的 app 就需要注册。

给 index_tags.py 文件命名时，需要注意不能与 Django 内置的标签或者过滤器名字冲突，如同 Python 中命名不可以使用关键字一样，所以我们在命名时应该尽量使用带有下划线的命名方式，这样可以确保名字不冲突。

上述操作完成后，我们就可以使用 {% load index_tags %} 加载自定义标签了， loda 标签将加载指定的的自定义标签，但是在 templatetags 目录中自定义标签或者过滤器的数量是没有限制的，你可以根据自己实际需求进行构建。

提示：{% load xxx%} 将会载入给定模块名下的标签或者过滤器，而不是 app 应用下的中所有标签和过滤器。

2) 模块变量 register

要在模块内自定义标签，该模块必须包含一个名为 register 的模板层变量，且它的值是 template.Library 的实例，所有的标签和过滤器都是在其中注册的。所以我们需要打开 index_tags.py 文件，并在文件顶部加上如下代码：

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

2. 实现自定义简单标签

简单标签通过接收参数，对输入的参数做一些处理并返回结果。如下所示，在 index_tags .py 文件中定义 addstr_tag 标签：

```
#注册自定义简单标签
@register.simple_tag
def addstr_tag(strs):
    return 'Hello'%strs
```

addstr_tag 函数使用 register.simple_tag 进行装饰，目的是能够将 addstr_tag 注册到模板系统中。然后我们就可以使用 {% load %} 加载自定义的标签了，使用如下方式：

{% load index_tags %}

加载之后我们就可以使用我们的自定义标签了，通过举例看一下实际的效果：

```
In [1]: from django.template import Template,Context
In [2]: t=Template("""
   ...: {% load index_tags %}
   ...: {% addstr_tag 'Django BookStore' %}
   ...:
   ...: """)
   ...: t.render(Context())
Out[2]: 'Hello Django BookStore'
```

上述就是一个简单标签的实现过程，自定义不同类型的标签它们的过程是一样的，而且我们还可以通过 name 参数给自定义的标签其别名，这样在使用 load 加载时就可以直接使用别名了，如下所示：

@register.simple_tag(name='abc')

3. 实现**自定**义**引用**标签

这种类型的标签可以对其他模板进行渲染，然后将渲染结果输。下面举例说明这类标签的用法。

1)定义模板文件

在 BookStore/templates 中定义模板文件 inclusion.html ，并在 body 中编写如下代码：

```
<p>{{ hello }}</p>
```

在 index_tags 中自定义引用标签：

```
#注册自定义引用标签
@register.inclusion_tag('inclusion.html',takes_context=True)
#定义函数渲染模板文件 inclusion.html
def add_webname_tag(context,namestr): #使用takes_context=True此时第一个参数必须为context
    return {'hello':'%s %s'%(context['varible'],namestr)}
```

我可以看出，引用标签使用 register.inclusion_tag 来注册，它的第一个参数用来指定要被渲染的模板文件，takes_context=True 参数可以让我们访问模板的当前环境上下文，并将当前环境上下文中的参数和值作为字典传入到函数的 contex 参数中，当使用 take_context=True 时，注册标签函数的第一个参数必需为 context。

通过 Django shell 编写如下代码看一下它是如何应用的，如下所示：

```
In [1]: from django.template import Template,Context
In [2]: t=Template("""
   ...: {% load index_tags %}
   ...: {% add_webname_tag 'C 语言中文网' %}
   ...:
   ...: """)
   ...: t.render(Context({'varible':'Hello'}))
Out[2]: '¥n¥n<!DOCTYPE html>¥n<html lang="en">¥n<head>¥n    <meta charset="UTF-8">¥n    <title>C语言中文网</title>¥n</head>¥n<body>¥n<p>Hello C 语言中文网</p>¥n¥n</body>¥n</html>¥n¥n'
```

从输出的结果可以得出，引用标签对 inclusion.html 模板进行了渲染，将{{ hello }}变量渲染成了 Hello C 语言中文网。

4. 实现**自定**义赋值标签

这个标签类似于简单标签，使用 register.simple_tag 进行注册，但它并不会直接输出结果，而是使用 as 关键字将结果储存在指定的上下文变量中，从而降低了传输上下文的成本。下面在 index_tags.py 中定义test_as_tag 标签，如下所示：

```
#注册自定义赋值标签
@register.simple_tag
def test_as_tag(strs):
    return 'Hello Test Tag-%s'%strs
```

使用自定义赋值标签，实例如下所示：

```
In [1]: from django.template import Template,Context
In [2]: t=Template("""
   ...: {% load index_tags %}
   ...: {% test_as_tag '语言中文网欢迎你' as test %}
   ...: <p>{{ test }}</p>
   ...: """)
   ...: t.render(Context())
Out[2]: '¥n¥n¥n<p>Hello Test Tag-语言中文网欢迎你</p>¥n'
```

自定义标签看似简单，但需要我们灵活的掌握它们，使用最合适的方式，最适用的逻辑让复杂的问题变得简单化。这是一个慢慢锻炼的过程，而理解了它最基本的使用方法，是漫漫征程中的第一步。