Django自定义过滤器

学习完《Django实现自定义标签》后，我们对于模板语言实现自定义标签的过程有了深入的了解。自定义模板过滤器与自定义标签有着异曲同工之处，所以有了前面知识的铺垫，本节知识也很好理解。

首先自定义过滤器与自定义标签需要做同样的准备工作，即模板层变量 register 和 app 应用注册到 INSTALLED_APPS 列表中，如果大家如果忘了相应的过程，可以参见《Django实现自定义标签》一节。在本节我们将重点讲解如何实现自定义一个过滤器，主要包括注册过滤器的与编写过滤器函数。准备工作就不在详细赘述，我们直接进入正题。

1.实现自定义过滤器

1)自定义过滤器替换指定字符串

在 index_tags.py 文件中创建一个 hello_my_filter 过滤器，并使用 @register.filter对此过滤器进行注册，代码如下所示：

```
@register.filter
def hello_my_filter(value):
    return value.replace('django','Python')
```

使用 Django shell 测试自定义过滤器：

```
from django.template import Template,Context
t=Template("""
...: {% load index_tags %}
...: <h1>:{{ Web|hello_my_filter }}</h1>
...: """)
...: t.render(Context({'Web':'Web django Django'}))
'¥n¥n<h1>:Web Python Django</h1>¥n'
```

从输出结果可以看出，自定义过滤器实现了字符串的替换功能，将原来的 django 替换为了 Python。

2)自定义过滤器实现列表排序

同样在 index_tags.py 文件中定义 sorted_filter 过滤器，在自定义过滤器中同样也可以使用 name 属性，如下所示：

```
@register.filter(name='prefix')#使用name参数指定别名
def sorted_filter(value):
    return sorted(value)
```

然后在执行以下代码，测试过滤器是否实现了我们预期的功能，结果如下所示：

```
from django.template import Template,Context
t=Template("""
   ...: {% load index_tags %}
   ...: <p>:{{ num|prefix }}</p>
   ...: """)
   ...: t.render(Context({'num':[1,4,2]}))
'¥n¥n<h1>:[1, 2, 4]</h1>¥n'
```

本节通过讲解自定义过滤器的两个简单实例，相信大家对于自定义过滤器有了进一步的认识。自定义过滤器在实际的开发工作中，也同样需要灵活掌握，有时自定义一个简单的过滤器，就可能帮助我们省去许多不必要的代码，从而优化了代码的整洁性，也提升了开发者的工作效率。