10-页面侧边栏：使用自定义模板标签

我们的博客侧边栏有四项内容：最新文章、归档、分类和标签云。这些内容相对比较固定，且在各个页面都会显示，如果像文章列表或者文章详情一样，从视图函数中获取然后传递给模板，则每个页面对应的视图函数里都要写一段获取这些内容的代码，这会导致很多重复代码。更好的解决方案是直接在模板中获取，为此，我们使用 Django 的一个新技术：自定义模板标签来完成任务。

1、使用模板标签的解决思路

我们前面已经接触过一些 Django 内置的模板标签，比如比较简单的 {% static %} 模板标签，这个标签帮助我们在模板中引入静态文件。还有比较复杂的如 {% for %} {% endfor%} 标签。这里 我们希望自己定义一个模板标签，例如名为 `get_recent_posts` 的模板标签，它可以这样工作：我们只要在模板中写入 {% get_recent_posts as recent_post_list %}，那么模板中就会有一个从数据库获取的最新文章列表，并通过 as 语句保存到 `recent_post_list` 模板变量里。这样我们就可以通过 {% for %} {% endfor%} 模板标签来循环这个变量，显示最新文章列表了，这和我们在编写博客首页面视图函数是类似的。首页视图函数中从数据库获取文章列表并保存到 `post_list` 变量，然后把这个 `post_list` 变量传给模板，模板使用 for 模板标签循环这个文章列表变量，从而展示一篇篇文章。这里唯一的不同是我们从数据库获取文章列表的操作不是在视图函数中进行，而是在模板中通过自定义的 {% get_recent_posts %} 模板标签进行。

以上就是解决思路，但模板标签不是我们随意写的，必须遵循 Django 的规范我们才能在 Django 的模板系统中使用自定义的模板标签，下面我们就依照这些规范来实现我们的需求。

2、模板标签目录结构

首先在我们的 blog 应用下创建一个 templatetags 文件夹。然后在这个文件夹下创建一个 __init__.py 文件，使这个文件夹成为一个 Python 包，之后在 templatetags\ 目录下创建一个 blog_tags.py 文件，这个文件存放自定义的模板标签代码。

此时你的目录结构应该是这样的：

```
blog\
  __init__.py
  admin.py
  apps.py
  migrations\
    __init__.py
  models.py
  static\
  templatetags\ 自定义模板标签
    __init__.py
    blog_tags.py
  tests.py
  views.py
```

3、编写模板标签代码

接下来就是编写各个模板标签的代码了，自定义模板标签代码写在 blog_tags.py 文件中。其实模板标签本质上就是一个 Python 函数，因此按照 Python 函数的思路来编写模板标签的代码就可以了，并没有任何新奇的东西或者需要新学习的知识在里面。

（1）最新文章模板标签

打开 blog_tags.py 文件，开始写我们的最新文章模板标签。为了能够通过 {% get_recent_posts %} 的语法在模板中调用这个函数，必须按照 Django 的规定注册这个函数为模板标签，方法如下：

【blog/templatetags/blog_tags.py】

```
from django import template
from ..models import Post

register = template.Library()

@register.simple_tag
def get_recent_posts(num=5):
    return Post.objects.all().order_by('-created_time')[:num]
```

这里我们首先导入 template 这个模块，然后实例化了一个 template.Library 类，并将函数 get_recent_posts 装饰为 register.simple_tag。这样就可以在模板中使用语法 {% get_recent_posts %} 调用这个函数了（**自定义模板标签的步骤**）。

（2）归档模板标签

和最新文章模板标签一样，先写好函数，然后将函数注册为模板标签即可。

【blog/templatetags/blog_tags.py】

```
@register.simple_tag
def archives():
    return Post.objects.dates('created_time', 'month', order='DESC')
```

这里 dates **方法会返回一个列表，列表中的元素为每一篇文章（Post）的创建时间，且是 Python 的** date **对象，精确到月份，降序排列。**接受的三个参数值表明了这些含义，一个是 created_time ，即 Post 的创建时间，month 是精度，order='DESC' 表明降序排列（即离当前越近的时间越排在前面）。例如我们写了 3 篇文章，分别发布于 2017 年 2 月 21 日、2017 年 3 月 25 日、2017 年 3 月 28 日，那么 dates 函数将返回 2017 年 3 月 和 2017 年 2 月这样一个时间列表，且降序排列，从而帮助我们实现按月归档的目的。

（3）分类模板标签

过程还是一样，先写好函数，然后将函数注册为模板标签。注意分类模板标签函数中使用到了 Category 类，其定义在 blog.models.py 文件中，使用前记得先导入它，否则会报错。

【blog/templatetags/blog_tags.py】

```
from ..models import Post, Category

@register.simple_tag
def get_categories():
    return Category.objects.all()
```

尽管侧边栏有 4 项内容（还有一个标签云），但是这里我们只实现最新文章、归档和分类数据的显示，还有一个标签云没有实现。因为标签云的实现稍有一点不同，所以将在后面专门介绍。

4、使用自定义的模板标签

打开 base.html，为了使用模板标签，我们首先需要在模板中导入存放这些模板标签的模块，这里是 blog_tags.py 模块。当时我们为了使用 static 模板标签时曾经导入过 {% load staticfiles %}，这次在 {% load staticfiles %} 下再导入 blog_tags：

【templates/base.html】

```
{% load staticfiles %}
{% load blog_tags %}
<!DOCTYPE html>
<html>
...
</html>
```

然后找到最新文章列表处，把里面的列表修改一下：

【templates/base.html】

```
<div class="widget widget-recent-posts">
 <h3 class="widget-title">最新文章</h3>
 {% get_recent_posts as recent_post_list %}
 <ul>
  {% for post in recent_post_list %}
  <li>
   <a href="{{ post.get_absolute_url }}">{{ post.title }}</a>
  </li>
  {% empty %}
  暂无文章！
  {% endfor %}
 </ul>
</div>
```

这里我们通过使用 `get_recent_posts` 模板标签获取到最新文章列表，然后我们通过 as 语法（Django 模板系统的语法）将获取的文章列表保存进了 `recent_post_list` 模板变量中，之后就可以通过 for 循环来循环显示文章列表数据了，这和我们在写首页视图时是一样的。

然后是归档部分：

【templates/base.html】

```
<div class="widget widget-archives">
 <h3 class="widget-title">归档</h3>
 {% archives as date_list %}
 <ul>
  {% for date in date_list %}
  <li>
   <a href="#">{{ date.year }} 年 {{ date.month }} 月</a>
  </li>
  {% empty %}
  暂无归档！
  {% endfor %}
 </ul>
</div>
```

同样，这里我们调用 `archives` 模板标签自动获取一个已发表文章的日期列表，精确到月份，降序排列，然后通过 as 语法将其保存在 `date_list` 模板变量里。由于日期列表中的元素为 Python 的 `date` 对象，因此可以通过其 `year` 和 `month` 属性分别获取年和月的信息，`<a href="#">{{ date.year }} 年 {{ date.month }} 月</a>` 反应了这个事实。

分类部分也一样：

【templates/base.html】

```
<div class="widget widget-category">
 <h3 class="widget-title">分类</h3>
 {% get_categories as category_list %}
 <ul>
  {% for category in category_list %}
  <li>
   <a href="#">{{ category.name }} <span class="post-count">(13)</span></a>
  </li>
  {% empty %}
  暂无分类！
  {% endfor %}
 </ul>
</div>
```

`<span class="post-count">(13)</span>` 显示的是该分类下的文章数目，这个特性会在接下来的教程中讲解如何实现，目前暂时用占位数据代替吧。

现在运行开发服务器，可以看到侧边栏显示的数据已经不再是之前的占位数据，而是我们保存在数据库中的数据了。

注意：如果你按照教程的步骤做完后发现报错，请按以下顺序检查。

1. 检查目录结构是否正确。确保 templatetags\ 位于 blog\ 目录下，且目录名必须为 templatetags。具体请对照上文给出的目录结构。
2. 确保 templatetags\ 目录下有 __init__.py 文件。
3. 确保使用的 Django 版本不小于 1.9。
4. 确保通过 `register = template.Library()` 和 `@register.simple_tag` 装饰器将函数装饰为一个模板标签。
5. 确保在使用模板标签以前导入了 blog_tags，即 {% load blog_tags %}。注意要在使用任何 blog_tags 下的模板标签以前导入它。
6. 确保模板标签的语法使用正确，即 {% load blog_tags %}，注意 { 和 % 以及 % 和 } 之间没有任何空格。