22-Markdown 自动生成文章目录

我们的之前在博客中使用了 Markdown 来为文章提供排版支持。Markdown 在渲染内容的同时还可以自动提取整个内容的目录结构，现在我们来使用 Markdown 为文章自动生成目录。

1、在文中插入目录

先来回顾一下博客的 Post（文章）模型，其中 `body` 是我们存储 Markdown 文本的字段：

【blog/models.py】

```
from django.db import models

class Post(models.Model):
    # Other fields ...
    body = models.TextField()
```

再来回顾一下文章详情页的视图，我们在 `PostDetailView` 中将 `post` 的 `body` 字段中的 Markdown 文本渲染成了 HTML 文本，然后传递给模板显示。注意这里我们使用的是类视图。

【blog/views.py】

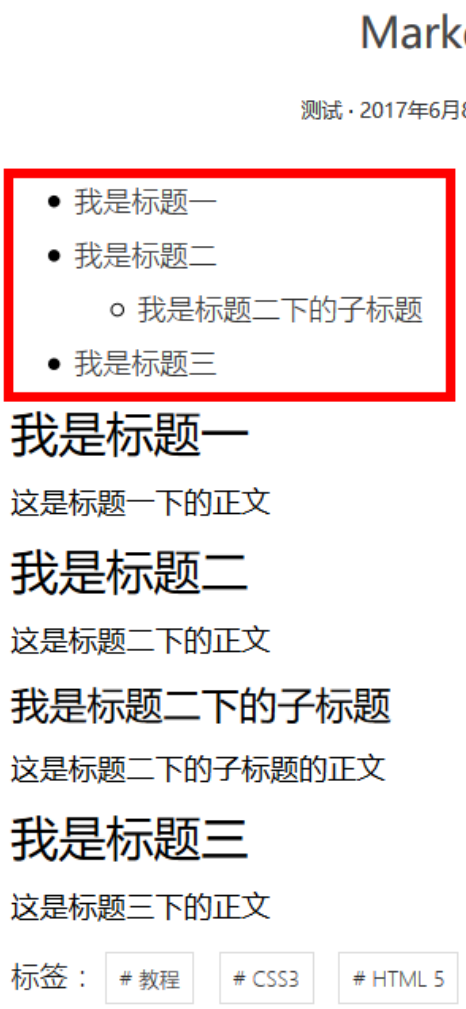
```
class PostDetailView(DetailView):
    # 这些属性的含义和 ListView 是一样的
    model = Post
    template_name = 'blog/detail.html'
    context_object_name = 'post'

    def get(self, request, *args, **kwargs):
        # ...

    def get_object(self, queryset=None):
        # 覆写 get_object 方法的目的是因为需要对 post 的 body 值进行渲染
        post = super(PostDetailView, self).get_object(queryset=None)
        post.body = markdown.markdown(post.body,
                                      extensions=[
                                        'markdown.extensions.extra',
                                        'markdown.extensions.codehilite',
                                        'markdown.extensions.toc',
                                      ])
        return post

    def get_context_data(self, **kwargs):
        # ...
```

看到 `get_object` 方法中的代码，`markdown.markdown()` 方法把 `post.body` 中的 Markdown 文本渲染成了 HTML 文本。同时我们还给该方法提供了一个 `extensions` 的额外参数。其中 `markdown.extensions.codehilite` 是代码高亮拓展，
而 `markdown.extensions.toc` 就是自动生成目录的拓展（这里可以看出我们有先见之明，如果你之前没有添加的话记得现在添加进去）。

在渲染 Markdown 文本时加入了 toc 拓展后，就可以在文中插入目录了。方法是在书写 Markdown 文本时，在你想生成目录的地方插入 [TOC] 标记即可。例如新写一篇 Markdown 博文，其 Markdown 文本内容如下：

```
[TOC]

## 我是标题一

这是标题一下的正文

## 我是标题二

这是标题二下的正文

### 我是标题二下的子标题
这是标题二下的子标题的正文

## 我是标题三
这是标题三下的正文
```

其最终渲染后的效果就是：



原本 [TOC] 标记的地方被内容的目录替换了。

2、在页面的任何地方插入目录

上述方式的一个局限局限性就是只能通过 [TOC] 标记在文章内容中插入目录。如果我想在页面的其它地方，比如侧边栏插入一个目录该怎么做呢？方法其实也很简单，只需要稍微改动一下渲染 Markdown 文本内容的方式即可，具体代码就像这样：

【blog/views.py】

```
class PostDetailView(DetailView):
    # 这些属性的含义和 ListView 是一样的
    model = Post
    template_name = 'blog/detail.html'
    context_object_name = 'post'

    def get(self, request, *args, **kwargs):
        # ...
```

```
    def get_object(self, queryset=None):
        # 覆写 get_object 方法的目的是因为需要对 post 的 body 值进行渲染
        md = markdown.Markdown(extensions=[
            'markdown.extensions.extra',
            'markdown.extensions.codehilite',
            'markdown.extensions.toc',
        ])
        post.body = md.convert(post.body)
        post.toc = md.toc
        return post

    def get_context_data(self, **kwargs):
        # ...
```

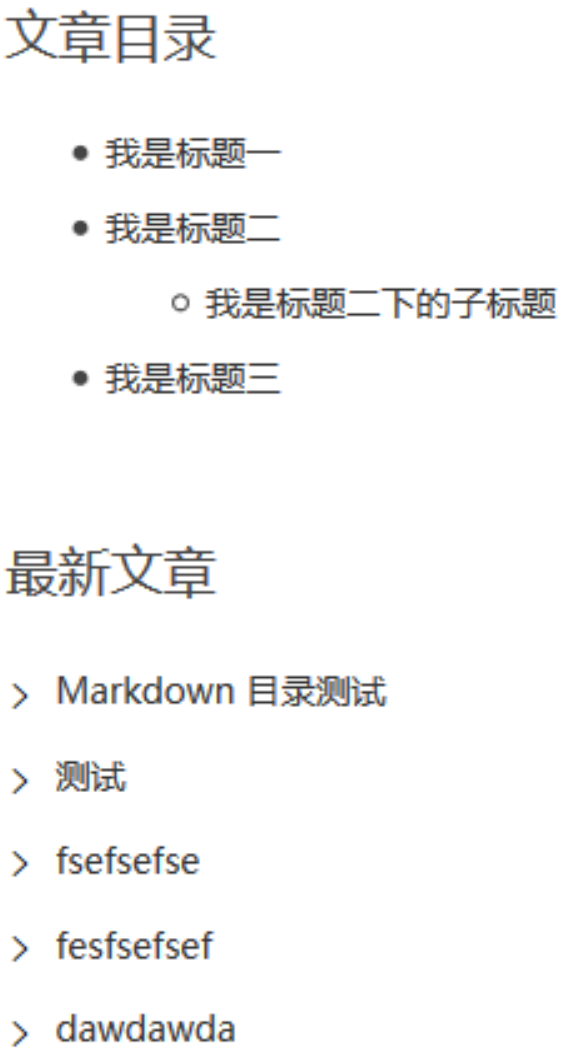
和之前的代码不同，在 get_object 方法中我们没有直接用 markdown.markdown() 方法来渲染 post.body 中的内容，而是先实例化了一个 markdown.Markdown 类 md，和 markdown.markdown() 方法一样，也传入了 extensions 参数（可以用 Markdown 类也可以用 markdown 方法）。接着我们便使用该实例的 convert 方法将 post.body 中的 Markdown 文本渲染成 HTML 文本。而一旦调用该方法后，实例 md 就会多出一个 toc 属性（**注意！调用 md 的属性 convert 方法之后，会改变 md 自身！md 会多出一个 toc 属性！**），这个属性的值就是内容的目录，我们把 md.toc 的值赋给 post.toc 属性（要注意这个 post 实例本身是没有 md 属性的，我们给它动态添加了 md 属性，这就是 Python 动态语言的好处，不然这里还真不知道该怎么把 toc 的值传给模板）。

接下来就在博客文章详情页的文章目录侧边栏渲染文章的目录吧！删掉 detail.html 最末尾占位用的目录内容，替换成如下代码：

【templates/blog/detail.html】

```
{% block toc %}
    <div class="widget widget-content">
        <h3 class="widget-title">文章目录</h3>
        {{ post.toc|safe }}
    </div>
{% endblock toc %}
```

即使用模板变量标签 {{ post.toc }} 显示模板变量的值，注意 post.toc 实际是一段 HTML 代码，我们知道 Django 会对模板中的 HTML 代码进行转义，所以要使用 safe 标签防止 Django 对其转义。其最终渲染后的效果就是：



3、美化标题的锚点 URL

文章内容的标题被设置了锚点，点击目录中的某个标题，页面就会跳到该文章内容中标题所在的位置，这时候浏览器的 URL 显示的值可能不太美观，比如像下面的样子：

http://127.0.0.1:8000/post/8/#_1

http://127.0.0.1:8000/post/8/#_3

`#_1` 就是锚点，Markdown 在设置锚点时利用的是标题的值，由于通常我们的标题都是中文，Markdown 没法处理，所以它就忽略的标题的值，而是简单地在后面加了个 `_1` 这样的锚点值。为了解决这一个问题，我们需要修改一下传给 `extentions` 的参数，其具体做法如下：

【blog/views.py】

```python
from django.utils.text import slugify
from markdown.extensions.toc import TocExtension

class PostDetailView(DetailView):
    # 这些属性的含义和 ListView 是一样的
    model = Post
    template_name = 'blog/detail.html'
    context_object_name = 'post'

    def get(self, request, *args, **kwargs):
        # ...

    def get_object(self, queryset=None):
        # 覆写 get_object 方法的目的是因为需要对 post 的 body 值进行渲染
        md = markdown.Markdown(extensions=[
            'markdown.extensions.extra',
            'markdown.extensions.codehilite',
            # 记得在顶部引入 TocExtension 和 slugify
            TocExtension(slugify=slugify),
        ])
        post.body = md.convert(post.body)
        post.toc = md.toc
        return post

    def get_context_data(self, **kwargs):
        # ...
```

和之前不同的是，extensions 中的 toc 拓展不再是字符串 markdown.extensions.toc ，而是 **TocExtension 的实例**。TocExtension 在实例化时其 slugify 参数可以接受一个函数作为参数，这个函数将被用于处理标题的锚点值。Markdown 内置的处理方法不能处理中文标题，所以我们使用了 django.utils.text 中的 slugify 方法，该方法可以很好地处理中文。

这时候标题的锚点 URL 变得好看多了。

http://127.0.0.1:8000/post/8/#我是标题一

http://127.0.0.1:8000/post/8/#我是标题二下的子标题