9-支持 Markdown 语法和代码高亮

为了让博客文章具有良好的排版，显示更加丰富的格式，我们使用 Markdown 语法来书写我们的博文。Markdown 是一种 HTML 文本标记语言，只要遵循它约定的语法格式，Markdown 的渲染器就能够把我们写的文章转换为标准的 HTML 文档，从而让我们的文章呈现更加丰富的格式，例如标题、列表、代码块等等 HTML 元素。由于 Markdown 语法简单直观，不用超过 5 分钟就可以掌握常用的标记语法，因此大家青睐使用 Markdown 书写 HTML 文档。下面让我们的博客也支持使用 Markdown 书写。

1、安装 Python Markdown

将 Markdown 格式的文本渲染成标准的 HTML 文档是一个复杂的工作，好在已有好心人帮我们完成了这些工作，我们直接使用即可。首先安装 Markdown，这是一个 Python 第三方库，激活虚拟环境，然后使用命令 `pip install markdown` 安装即可。

2、在 detail 视图中渲染 Markdown

将 Markdown 格式的文本渲染成 HTML 文本非常简单，只需调用这个库的 `markdown` 方法即可。我们书写的博客文章内容存在 `Post` 的 `body` 属性里，回到我们的详情页视图函数，对 `post` 的 `body` 的值做一下渲染，把 Markdown 文本转为 HTML 文本再传递给模板：

【blog/views.py】

```
import markdown
from django.shortcuts import render, get_object_or_404
from .models import Post

def detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    # 相当于对 post.body 多了一个中间步骤，先将 Markdown 格式的文本渲染成 HTML 文本再传递给模板
    post.body = markdown.markdown(post.body,
                                  extensions=[
                                    'markdown.extensions.extra',
                                    'markdown.extensions.codehilite',
                                    'markdown.extensions.toc',
                                  ])
    return render(request, 'blog/detail.html', context={'post': post})
```

这样我们在模板中展示 {{ post.body }} 的时候，就不再是原始的 Markdown 文本了，而是渲染过后的 HTML 文本。注意这里我们给 `markdown` 渲染函数传递了额外的参数 `extensions`，它是对 Markdown 语法的拓展，这里我们使用了三个拓展，分别是 extra、codehilite、toc。extra 本身包含很多拓展，而 codehilite 是语法高亮拓展，这为我们后面的实现代码高亮功能提供基础，而 toc 则允许我们自动生成目录（在以后会介绍）。

来测试一下效果，进入 admin 后台，这次我们发布一篇用 Markdown 语法写的测试文章看看：

```
# 一级标题


## 二级标题


### 三级标题


>引用


- 无序列表

- 无序列表

- 无序列表


[百度的链接](http://baidu.com)


![图片](http://box.bdimg.com/static/fisp_static/common/img/searchbox/logo_news_276_88_1f9876a.png)
```

```python
def detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
        post.body = markdown.markdown(post.body,
extensions=['markdown.extensions.extra',
'markdown.extensions.codehilite',
'markdown.extensions.toc',])
    return render(request, 'blog/detail.html', context={'post':
post})
```

3、safe 标签

我们在发布的文章详情页没有看到预期的效果，而是类似于一堆乱码一样的 HTML 标签，这些标签本应该在浏览器显示它本身的格式，但是 Django 出于安全方面的考虑，任何的 HTML 代码在 Django 的模板中都会被转义（即显示原始的 HTML 代码，而不是经浏览器渲染后的格式）。为了解除转义，只需在模板标签使用 safe 过滤器即可，告诉 Django，这段文本是安全的，你什么也不用做。在模板中找到展示博客文章主体的 {{ post.body }} 部分，为其加上 safe 过滤器，{{ post.body|safe }}，大功告成，这下看到预期效果了。

safe 是 Django 模板系统中的过滤器（Filter），可以简单地把它看成是一种函数，其作用是作用于模板变量，将模板变量的值变为经过滤器处理过后的值。例如这里 {{ post.body|safe }}，本来 {{ post.body }} 经模板系统渲染后应该显示 body 本身的值，但是在后面加上 safe 过滤器后，渲染的值不再是 body 本身的值，而是由 safe 函数处理后返回的值。过滤器的用法是在模板变量后加一个 | 管道符号，再加上过滤器的名称。可以连续使用多个过滤器，例如 {{ var|filter1|filter2 }}。

4、代码高亮

程序员写博客免不了要插入一些代码，Markdown 的语法使我们容易地书写代码块，但是目前来说，显示的代码块里的代码没有任何颜色，很不美观，也难以阅读，要是能够像我们的编辑器里一样让代码高亮就好了。虽然我们在渲染时使用了 codehilite 拓展，但这只是实现代码高亮的第一步，还需要简单的几步才能达到我们的最终目的。

（1）安装 Pygments

首先我们需要安装 Pygments，激活虚拟环境，运行： `pip install Pygments` 安装即可。

搞定了，虽然我们除了安装了一下 Pygments 什么也没做，但 Markdown 使用 Pygments 在后台为我们做了很多事。如果你打开博客详情页，找到一段代码段，在浏览器查看这段代码段的 HTML 源代码，可以发现 Pygments 的工作原理是把代码切分成一个个单词，然后为这些单词添加 css 样式，不同的词应用不同的样式，这样就实现了代码颜色的区分，即高亮了语法。为此，还差最后一步，引入一个样式文件来给这些被添加了样式的单词定义颜色。

（2）引入样式文件

在项目的 blog\static\blog\css\highlights\ 目录下应该能看到很多 .css 样式文件，这些文件是用来提供代码高亮样式的。选择一个你喜欢的样式文件，在 base.html 引

入即可（别忘了使用 static 模板标签）。比如微软的样式 vs.css ，那么在基模板中引入这个文件：

【templates/base.html】

```
...
<link rel="stylesheet" href="{% static 'blog/css/pace.css' %}">
<link rel="stylesheet" href="{% static 'blog/css/custom.css' %}">
...
+ <link rel="stylesheet" href="{% static 'blog/css/highlights/github.css' %}">
```

这里 + 号表示添加这行代码。好了，看看效果，大功告成，终于可以愉快地贴代码了。

注意：如果你按照教程中的方法做完后发现代码依然没有高亮，请依次检查以下步骤：

1. 确保在渲染文本时添加了 `markdown.extensions.codehilite` 拓展，详情见上文。
2. 确保安装了 Pygments。
3. 确保代码块的 Markdown 语法正确，特别是指明该代码块的语言类型，具体请参见上文中 Markdown 的语法示例。
4. 在浏览器端代码块的源代码，看代码是否被 pre 标签包裹，并且代码的每一个单词都被 span 标签包裹，且有一个 class 属性值。如果没有，极有可能是前三步中某个地方出了问题。
5. 确保用于代码高亮的样式文件被正确地引入，具体请参见上文中引入样式文件的讲解。
6. 有些样式文件可能对代码高亮没有作用，首先尝试用 github.css 样式文件做测试。