24-Django Haystack 全文检索与关键词高亮

在此之前我们使用了 Django 内置的一些方法实现了一个简单的搜索功能。但这个搜索功能实在过于简单，没有多大的实用性。对于一个搜索引擎来说，至少应该能够根据用户的搜索关键词对搜索结果进行排序以及高亮关键字。现在我们就来使用 django-haystack 实现这些特性。

0、Django Haystack 简介

django-haystack 是一个专门提供搜索功能的 django 第三方应用，它支持 Solr、Elasticsearch、Whoosh、Xapian 等多种搜索引擎，配合著名的中文自然语言处理库 jieba 分词，就可以为我们的博客提供一个效 s 果不错的博客文章搜索系统。

1、安装必要依赖

要使用 django haystack，首先必须安装它，并且安装一些必要的依赖，具体需要安装的依赖有：

- Whoosh。Whoosh 是一个由纯 Python 实现的全文搜索引擎，没有二进制文件等，比较小巧，配置简单方便。
- jieba 中文分词。由于 Whoosh 自带的是英文分词，对中文的分词支持不是太好，所以使用 jieba 替换 Whoosh 的分词组件。

直接使用 pip 安装这些包即可（安装到你使用的虚拟环境下）：

```
pip install whoosh django-haystack jieba
```

2、配置 Haystack

安装好 django haystack 后需要在项目的 settings.py 做一些简单的配置。

首先是把 django haystack 加入到 `INSTALLED_APPS` 选项里：

【blogproject/settings.py】

```
INSTALLED_APPS = [
    'django.contrib.admin',
    # 其它 app...
```

```
    'haystack',
    'blog',
    'comments',
]
```

然后加入如下配置项：

【blogproject/settings.py】

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'blog.whoosh_cn_backend.WhooshEngine',

        'PATH': os.path.join(BASE_DIR, 'whoosh_index'),
    },
}
HAYSTACK_SEARCH_RESULTS_PER_PAGE = 10
HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'
```

HAYSTACK_CONNECTIONS 的 ENGINE 指定了 django haystack 使用的搜索引擎，这里我们使用了 blog.whoosh_cn_backend.WhooshEngine，虽然目前这个引擎还不存在，但我们接下来会创建它。PATH 指定了索引文件（搜索引擎需要建立索引文件）需要存放的位置，我们设置为项目根目录 BASE_DIR 下的 whoosh_index 文件夹（在建立索引是会自动创建）。

HAYSTACK_SEARCH_RESULTS_PER_PAGE 指定如何对搜索结果分页，这里设置为每 10 项结果为一页。

HAYSTACK_SIGNAL_PROCESSOR 指定什么时候更新索引，这里我们使用 haystack.signals.RealtimeSignalProcessor，作用是每当有文章更新时就更新索引。由于博客文章更新不会太频繁，因此实时更新没有问题。

3、处理数据

接下来就要告诉 django haystack 使用哪些数据建立索引以及如何存放索引。如果要对 blog 应用下的数据进行全文检索，做法是在 blog 应用下建立一个 search_indexes.py 文件，写上如下代码：

【blog/search_indexes.py】

```
from haystack import indexes
from .models import Post

class PostIndex(indexes.SearchIndex, indexes.Indexable):
    text = indexes.CharField(document=True, use_template=True)
```

```
    def get_model(self):
        return Post

    def index_queryset(self, using=None):
        return self.get_model().objects.all()
```

这是 django haystack 的规定：要相对某个 app 下的数据进行全文检索，就要在该 app 下创建一个 search_indexes.py 文件，然后创建一个 XXIndex 类（XX 为含有被检索数据的模型，如这里的 Post），并且继承 SearchIndex（注意不是 SearchField) 和 Indexable。

为什么要创建索引？索引就像是一本书的目录，可以为读者提供更快速的导航与查找。在这里也是同样的道理，当数据量非常大的时候，若要从这些数据里找出所有的满足搜索条件的几乎是不太可能的，将会给服务器带来极大的负担。所以我们需要为指定的数据添加一个索引（目录），在这里是为 Post 创建一个索引，索引的实现细节是我们不需要关心的，我们只关心为哪些字段创建索引，如何指定。

每个索引里面必须有且只能有一个字段为 document=True，这代表 django haystack 和搜索引擎将使用此字段的内容作为索引进行检索(primary field)。注意，如果使用一个字段设置了 document=True，则一般约定此字段名为 text，这是在 SearchIndex 类里面一贯的命名，以防止后台混乱，当然名字你也可以随便改，不过不建议改。

并且，haystack 提供了 use_template=True 在 text 字段中，这样就允许我们使用数据模板去建立搜索引擎索引的文件，说得通俗点就是索引里面需要存放一些什么东西，例如 Post 的 title 字段，这样我们可以通过 title 内容来检索 Post 数据了。举个例子，假如你搜索 Python ，那么就可以检索出 title 中含有 Python 的 Post 了.数据模板的路径为 templates/search/indexes/youapp/<model_name>_text.txt（例如 templates/search/indexes/blog/post_text.txt），其内容为：

【templates/search/indexes/blog/post_text.txt】

```
{{ object.title }}
{{ object.body }}
```

这个数据模板的作用是对 Post.title、Post.body 这两个字段建立索引，当检索的时候会对这两个字段做全文检索匹配，然后将匹配的结果排序后作为搜索结果返回。

4、配置 URL

接下来就是配置 URL，搜索的视图函数和 URL 模式 django haystack 都已经帮我们写好了（直接 include(haystack.urls)，相当于 haystack 这个应用已经隐式地创建好了，且自带 urls.py 文件），只需要项目的 urls.py 中包含它：

【blogproject/urls.py】

```
urlpatterns = [
    # 其它...
    url(r'^search/', include('haystack.urls')),
]
```

另外在此之前我们也为自己写的搜索视图配置了 URL，把那个 URL 删掉，以免冲突：

【blog/urls.py】

```
# url(r'^search/$', views.search, name='search'),
```

5、修改搜索表单

修改一下搜索表单，让它提交数据到 django haystack 搜索视图对应的 URL：

【templates/base.html】

```
<form role="search" method="get" id="searchform" action="{% url 'haystack_search' %}">
  <input type="search" name="q" placeholder="搜索" required>
  <button type="submit"><span class="ion-ios-search-strong"></span></button>
</form>
```

主要是把表单的 action 属性改为 {% url 'haystack_search' %}

6、创建搜索结果页面

haystack_search 视图函数会将搜索结果传递给模板 search/search.html，因此创建这个模板文件，对搜索结果进行渲染：

【templates/search/search.html】

```
{% extends 'base.html' %}
{% load highlight %}

{% block main %}
    {% if query %}
        {% for result in page.object_list %}
            <article class="post post-{{ result.object.pk }}">
                <header class="entry-header">
                    <h1 class="entry-title">
                        <a href="{{ result.object.get_absolute_url }}">{% highlight result.object.title with query %}</a>
                    </h1>
                    <div class="entry-meta">
```

```
            <span class="post-category">
              <a href="{% url 'blog:category' result.object.category.pk %}">
                {{ result.object.category.name }}</a></span>
              <span class="post-date"><a href="#">
                <time class="entry-date" datetime="{{ result.object.created_time }}">
                  {{ result.object.created_time }}</time></a></span>
              <span class="post-author"><a href="#">{{ result.object.author }}</a></span>
              <span class="comments-link">
              <a href="{{ result.object.get_absolute_url }}#comment-area">
                {{ result.object.comment_set.count }} 评论</a></span>
              <span class="views-count"><a
                  href="{{ result.object.get_absolute_url }}">{{ result.object.views }} 阅读
</a></span>
            </div>
          </header>
          <div class="entry-content clearfix">
            <p>{% highlight result.object.body with query %}</p>
            <div class="read-more cl-effect-14">
              <a href="{{ result.object.get_absolute_url }}" class="more-link">继续阅读 <span
                  class="meta-nav">→</span></a>
            </div>
          </div>
        </article>
      {% empty %}
        <div class="no-post">没有搜索到你想要的结果！</div>
      {% endfor %}
      {% if page.has_previous or page.has_next %}
        <div>
          {% if page.has_previous %}
            <a href="?q={{ query }}&amp;page={{ page.previous_page_number }}">{%
endif %}&laquo; Previous
          {% if page.has_previous %}</a>{% endif %}
          |
          {% if page.has_next %}<a
href="?q={{ query }}&amp;page={{ page.next_page_number }}">{% endif %}Next
          &raquo;{% if page.has_next %}</a>{% endif %}
        </div>
      {% endif %}
    {% else %}
      请输入搜索关键词，例如 django
    {% endif %}
{% endblock main %}
```

这个模板基本和 blog/index.html 一样，只是由于 haystack 对搜索结果做了分页，传给模板的变量是一个 page 对象，所以我们**从 page 中取出这一页对应的搜索结果，然后对其循环显示，即 {% for result in page.object_list %}**。另外要取得 Post（文章）以显示文章的数据如标题、正文，需要从 result 的 object 属性中获取。`query` 变量的值即为用户搜索的关键词。

7、高亮关键词

注意到百度的搜索结果页面，含有用户搜索的关键词的地方都是被标红的，在 django haystack 中实现这个效果也非常简单，只需要使用 {% highlight %} 模板标签即可，其用法如下：

```
# 使用默认值
{% highlight result.summary with query %}

# 这里我们为 {{ result.summary }} 里所有的 {{ query }} 指定了一个<div></div>标签，并且将 class 设置为 highlight_me_please，这样就可以自己通过 CSS 为{{ query }}添加高亮效果了，怎么样，是不是很科学呢
{% highlight result.summary with query html_tag "div" css_class "highlight_me_please" %}

# 可以 max_length 限制最终{{ result.summary }} 被高亮处理后的长度
{% highlight result.summary with query max_length 40 %}
```

在博客文章搜索页中我们对 title 和 body 做了高亮处理：{% highlight result.object.title with query %}，{% highlight result.object.body with query %}。高亮处理的原理其实就是给文本中的关键字包上一个 span 标签并且为其添加 highlighted 样式（当然你也可以修改这个默认行为，具体参见上边给出的用法）。因此我们还要给 highlighted 类指定样式，在 base.html 中添加即可：

【templates/base.html】

```
<head>
  <title>Black &amp;amp; White</title>
  ...
  <style>
    span.highlighted {
      color: red;
    }
  </style>
  ...
</head>
```

8、修改搜索引擎为中文分词

我们使用 Whoosh 作为搜索引擎，但在 django haystack 中为 Whoosh 指定的分词器是英文分词器，可能会使得搜索结果不理想，我们把这个分词器替换成 jieba 中文分词器。从你安装的 site-packages 目录的 haystack 目录中把 haystack/backends/whoosh_backend.py 文件拷贝到 blog/ 下，重命名为 whoosh_cn_backend.py（之前我们在 settings.py 中 的 `HAYSTACK_CONNECTIONS` 指定的就是这个文件），然后找到如下一行代码：

【blog/ whoosh_cn_backend.py】

```
schema_fields[field_class.index_fieldname] = TEXT(stored=True, analyzer=StemmingAnalyzer(), field_boost=field_class.boost, sortable=True)
```

将其中的 analyzer 改为 ChineseAnalyzer，当然为了使用它，你需要在文件顶部引入：from jieba.analyse import ChineseAnalyzer。

【blog/ whoosh_cn_backend.py】

```
from jieba.analyse import ChineseAnalyzer

...
#注意先找到这个再修改，而不是直接添加
schema_fields[field_class.index_fieldname] = TEXT(stored=True,
analyzer=ChineseAnalyzer(),field_boost=field_class.boost, sortable=True)
```

9、建立索引文件

最后一步就是建立索引文件了，运行命令 `python manage.py rebuild_index` 就可以建立索引文件了。

来看看搜搜效果吧，效果还不错。