NBA新闻整合与检索系统设计文档

程序介绍

设计目的

身处信息时代，数据的搜集、整合和检索是大数据技术的第一步。由于Python这个解释型语言的特性和库的丰富性，很适合用来进行爬虫的编写和作Web服务器的后端。使用不限制第三方库的Python编写数据搜集、整合和检索系统，有助于增加对Python的熟练度，并初步了解爬虫与Web服务器的搭建方法。

本程序的设计分为以下几个模块：(1)数据爬取

从虎扑爬取NBA板块（voice.hupu.com/nba）的**新闻信息**，内容包括新闻的标题、来源、发布时间、正文文本。

从虎扑NBA的球队主页爬取**球队信息**，包括球队名称、所在城市、球员。

(2)Web前端

对爬取的数据进行预处理并储存，以利于在其前端的Web网页上合理展现。

建立Web前端合理的地址和跳转，包含以下页面：

①**球队主页**，包括**球队基本信息**和与**球队的相关新闻**，相关新闻以列表的形式展现标题和部分正文，**标题**是可点击的**超链接**形式，点击后跳转到**新闻详情页。**

②**新闻详情页**，包含新闻的标题、正文、来源和发布时间，其中**正文**中的**球队**和**球员名称**标注上**超链接**，链接指向球队或球员所在**球队主页。**

③**球队热度榜**，显示**球队名**和**相关文章数量**，按相关文章数从高到低**排名**，**球队名**标注**超链接**调转到对应**球队主页。**

新闻列表合理**分页。**

使用CSS等对页面布局进行**美化。**

(3)简易搜索引擎

仿照常规的搜索引擎界面，实现输入**单关键词**，筛选并分页显示标题或正文含有关键词的新闻列表，页面中需显示**查询结果数量**和**查询时间**，搜索结果列表中新闻标题和正文中关键词**高亮**显示，从搜索结果列表中也能**链接**到**新闻详情页**；在单关键词的基础上在同一页面加入**高级搜索**，允许用户输入**多个关键词**或**含多个关键词的句子**，查询与输入内容**相近**的若干条结果，多关键词搜索的文本**不一定要与新闻文本完全匹配。**应当计算查询文本和新闻文本的**匹配程度**，**优先展示匹配程度高的新闻**，匹配程度应当考虑到**不同词的重要程度不同。**

(4)爬虫的Web控制

将爬虫和Web系统集成为一个系统，并增加管理页面，包含**按钮控制系统在后台开始/暂停爬虫与数据预处理工作。**在爬虫进行时，不影响前台**Web页面的访问**，并且新爬取的新闻在**预处理结束后能够更新到Web系统中显示**（更新球队的相关新闻、新闻能被搜索到、更新球队热度榜）。

程序使用方法与功能

程序已在zhengkw.com域名上部署，为了使外来ip能够通过此域名访问Web前端，需要将newsweb/newsweb下settings.py中`ALLOWED_HOSTS = []`改为`ALLOWED_HOSTS = ['zhengkw.com','127.0.0.1']`以设定允许的域名来源。

在newsweb根目录下运行命令行指令`python manage.py runserver (ip:)port`为指定的ip地址和端口开放Web前端访问，若ip留空默认对本机地址127.0.0.1开放，port默认为8080。这里设为0.0.0.0:80，对所有主机开放默认Http的80端口。

在newsweb/newsscrapy目录下运行`python run.py`以开启爬虫后台。

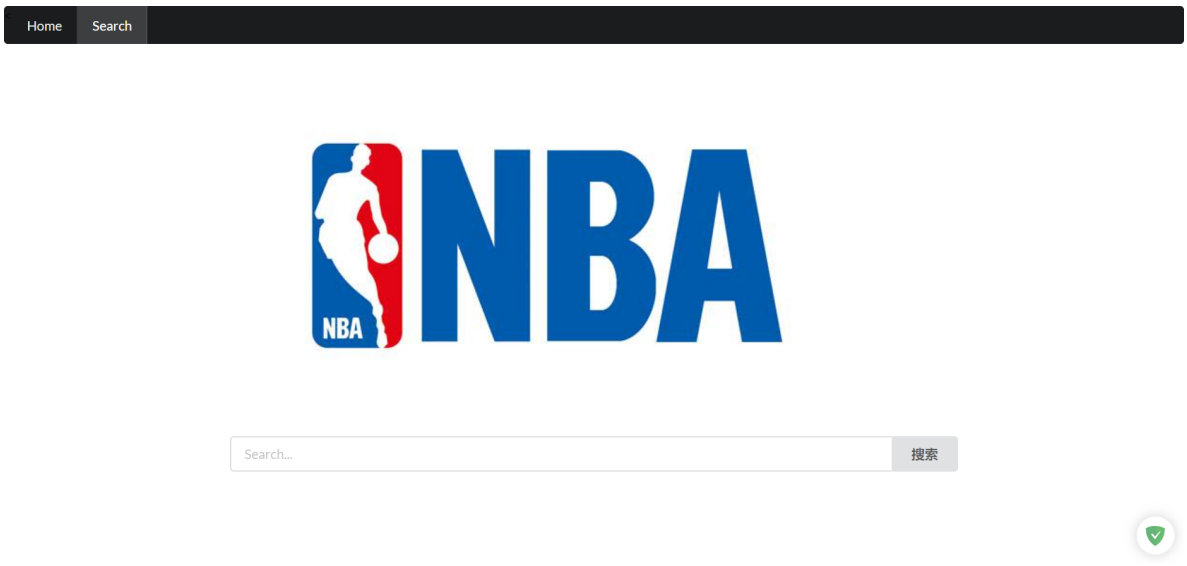
访问zhengkw.com/index，访问Web主页

左侧为新闻列表（按发布时间倒序，每页显示十条新闻），右侧则显示了球队热度榜，最上方有菜单栏。将页面拖至最下方

点击数字或上一页/下一页可跳转分页，不再演示。点击第一条新闻

显示了新闻详情，其中的队名和球员们标注了超链接，点击可跳转到球队主页，不再演示。点击菜单栏的Home返回，点击右端热度榜上排名第一的湖人

左侧显示相关新闻。右侧则是球队基本信息，新闻标题同样可点击跳转。再测试搜索功能，点击上方菜单栏上的Search



此时是搜索内容为空的状态。测试单关键词搜索，输入“詹姆斯”，点击搜索

显示相关新闻数量、搜索用时和分页的搜索结果列表。再测试多关键词搜索，输入“勇士与火箭”

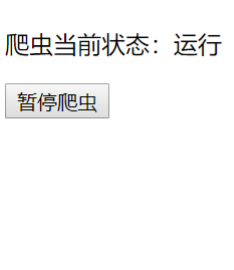
前几条为“勇士”“与”“火箭”三个关键词均包含的新闻。我们到最后一页（第117页）

这时的新闻从含有“火箭”和“与”到只含有“火箭”，相关度达到最低（只含有“与”的新闻由于相关度过低，没有被列入）。我们再搜索“湖人的詹姆斯和勇士球队打篮球”，前几条新闻为

包含了尽量多的关键词。而最后一页（第353页）

只包含了“球队”“的”“和”这样匹配度很低的新闻。

最后时爬虫控制页面，访问zhengkw.com/spider



页面为简单的指示和按钮，点击可控制后台爬虫运行状态。

性能统计

性能状态在上一页已展示完毕。总新闻数量越6000，单关键词搜索时用时为0.001s数量级，多关键词搜索则达到0.01s数量级，关键词越多、句子越长则用时越长。

进入Django的SQLite3数据库后台（zhengkw.com/admin），存储分词的数据库内约27500记录

程序设计

技术基础

本程序在PyCharm中编写测试，共约1000行代码（.py/.html，含自动生成），Python版本为3.7.4。

- Semantic UI
  - Semantic UI是一个优秀的前端开发框架，本程序主要使用了其CSS样式文件和图标。
  - 顶部菜单栏——ui menu,item
  - 页面左右布局——ui grid,XX wide column
    - 主页面、球队主页使用10:2(空白):4的比例设置
    - 搜索页面、新闻详情页使用3(空白):10:3(空白)的比例设置
  - 新闻列表的每一框——ui container segment
  - 水平分割线——ui horizontal divider
  - 分页菜单栏——ui pagination menu,icon item,item
  - 右方表格——ui table
  - 搜索表单——ui action input
- Django
  - 本程序使用了Web框架Django最基础的MTV模式和其自带的SQLite3数据库
    - Model
      - Team包含自增的id字段，name、city、players字段分布储存球队名称、所在城市、球员基本信息，relatedids字段储存相关文章的id
      - Player包含name、team，分布是球员名字和所在球队名称
      - Article包含自增的id，以及title、time、source、content存储文章所有信息
      - Word用于存储倒排索引，word字段是词名，relatedids记录含有此词的文章id
      - Spider用于进行爬虫的Web控制，完成Scrapy和Django两个进程间的通信
    - Template
      - 共5个网页模板，存储在newsweb/newsapp/templates文件夹下，在newsweb/newsweb/Settings.py中添加

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
```

```
        'DIRS': [os.path.join(BASE_DIR,
'templates').replace('\\' ,'/')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

- 完成模板加载。之后在View中使用模板templates/xx.html可直接调用`return render(request, "xx.html", context)`完成渲染。
- 在newsweb/newsapp/static文件夹下存储了模板要用到的CSS和图片资源文件。在newsweb/newsweb/Settings.py中添加

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (
# os.path.join(BASE_DIR, 'static')
os.path.join(os.path.dirname(__file__),
'../static/').replace('\\', '/'),
)
```

- 之后在模板文件头部添加`{% load staticfiles %}`，便可使用`{% static'filename' %}`调用static文件夹下的资源文件。

- View
  - Viem层设置了6种URL路由
  - /index——重定向到/index/1
  - /index/page_num——访问主页面，其中新闻列表为page_num页
  - /team/nid/page_num——编号为nid的球队，第page_num页
  - /news/nid——第nid条新闻
  - /search——搜索页面，使用GET获取搜索内容和页码
  - /spider——爬虫控制页面

- Scrapy
  - Scrapy是一个方便的爬虫框架
    - 本程序使用了其Spider-Pipeline结构爬取球员数据和新闻数据。
    - 在Spider中可以使用xpath结点路径定位，本程序使用了css标签属性定位的方法。
    - 在Pipeline中对数据进行入库和预处理操作（分词后入库）。
    - 在Scrapy中可直接使用Django的Model层，方法是在newsweb/newsscrapy/Setting.py中添加

```
import django
sys.path.append(os.path.dirname(os.path.abspath('.')))
os.environ['DJANGO_SETTINGS_MODULE'] = 'newsweb.settings'    # 项目
名.settings
django.setup()
```

        - 之后在Pipeline中from newsapp(Django app名称).models import xx即可
    - 在Scapy的Web控制上，本程序使用Django的Model层数据实现进程间通信。
        - newsweb/newssrapy/newscrapy/spiders文件夹下存储了用于新闻爬取的newsspider。
        - 在newsweb/newssrapy目录下的命令行中键入scrapy crawl newsspider可启动newsspider，每个newsspider运行的一个轮次只顺序爬取1-10页的新闻一遍。而newsspider内部根据Django数据库中的标识，判断爬虫运行情况，若运行则进行文章的解析预处理入库操作。
        - newsweb/newsscrapy/run.py是爬虫的后台控制端，它会等待每一个轮次运行完成，等待一分钟运行下一个轮次，循环往复。

- jieba
    - jieba是优秀的中文分词库，是本程序搜索功能的顶梁柱。本程序主要使用了其中的两个功能
        - 分词。在文章的预处理中，先对其标题和正文分词。再建立倒排索引（将与某词相关的文章id列表使用JSON保存至数据库）。其中分词使用了`jieba.cut_for_search()`函数，这是一种专为搜索引擎服务的较为细致的分词。本程序的匹配程度没有考虑词在文章中的出现次数，如“詹姆斯与篮球”，即使“与”在文章中出现100次，仍没有一个“詹姆斯”或“篮球”有效力。
        - TF-IDF。TD-IDF是一种用于信息检索与数据挖掘的常用加权技术，可用于句段中每个词语的重要性评估。使用`jieba.analyse.extract_tags()`函数可调用jieba内置的TF-IDF权重打分系统。本程序的对于每篇文章的匹配程度评分是简单的每个在文章中出现的关键词的权重相加。

细节处理

- 列表排序
    - 有时列表的元素不是简单的可比较对象，可能是元组，可能是Model。要根据元组的某个分量，或根据Model的某个字段排序，可结合`sorted()`函数与lambda表达式
    - 根据文章的发布时间倒序排列（结合Python的列表推导）

        ```
        articles = sorted([ars[0] for ars in
        [Article.objects.all().filter(id=id) for id in ids]], key=lambda x:
        x.time, reverse=True)
        ```

    - 根据元组列表中元组的第一个分量倒序排列

        ```
        relist = sorted(relist, key=lambda x: x[1], reverse=True)
        ```

- 元组列表的遍历
    - 在Template中的语句只支持简单的if/for遍历Model.field等，并不能进行运算和[]等符号的支持。对于元组列表的遍历，如

        ```
        rank = []
        for team in Team.objects.all():
            idjson = team.relatedids
            ids = json.loads(idjson)
            rank.append((len(ids), team.name, team.id))
        rank = sorted(rank, reverse=True)
        ```

    - 将rank通过context传入模板后可以使用如下方法遍历

```
{% for num,name,id in rank %}
…
{% endfor %}
```

- 分页菜单栏的生成和分页页码范围
  - 后端：使用Paginator结合简单的逻辑分支确定每页的内容和下方可选的页码范围。

```
paginator = Paginator(articles, 10) #每页10条新闻
page_num = int(pn)
try:
    current_list = paginator.page(page_num)
except:
    raise Http404("Page not found")
if paginator.num_pages > 12:  # 如果分页的数目大于12
    if page_num - 5 < 1:  # 你输入的值
        pageRange = range(1, 12)  # 按钮数
    elif page_num + 5 > paginator.num_pages:  # 按钮数加5大于分页数
        pageRange = range(page_num - 10, page_num + 1)  # 显示的按钮
数
    else:
        pageRange = range(page_num - 5, page_num + 6)  # range求的是
按钮数   如果你的按钮数小于分页数 那么就按照正常的分页数目来显示
else:
    pageRange = range(1, paginator.num_pages + 1)  # 正常分配
```

  - 前端：上一页和下一页，循环生成页面按钮，并使用Paginator分页得到的列表自带函数动态判断上一页/下一页的disable状态。

```
<div class="ui pagination menu">
{% if current_list.has_previous %}
    <a href="{% url 'index' current_list.previous_page_number %}"
class="icon item">
        <i class="left arrow icon"></i>
    </a>
{% else %}
    <a class="disabled icon item">
        <i class="left arrow icon"></i>
    </a>
{% endif %}
{% for i in pageRange %}
{% if current_num == i %}
    <a href="{% url 'index' i %}" class="active item">
        {{i}}
    </a>
{% else %}
    <a href="{% url 'index' i %}" class="item">
        {{i}}
    </a>
{% endif %}
{% endfor %}
{% if current_list.has_next %}
    <a href="{% url 'index' current_list.next_page_number %}"
class="icon item">
        <i class="right arrow icon"></i>
    </a>
{% else %}
```

```
    <a class="disabled icon item">
        <i class="right arrow icon"></i>
    </a>
{% endif %}
</div>
```

- 时间日期转换
  - models.DateTimeField()与Python中的datetime库是对应的关系，存储日期和时间信息。使用这个数据类型方便进行排序等操作。
  - 将表示日期时间的文本转化为datetime类型可使用datetime库中的`datetime.strptime()`函数

    ```
    dt = datetime.datetime.strptime(response.css("a.time
    span::text").extract_first("1000-01-01
    00:00:00").replace("\r","").replace("\n","").strip(),"%Y-%m-%d
    %H:%M:%S")
    ```

  - 将datetime类型格式化为文本可使用datetime类型的`strftime()`函数

    ```
    time = article.time.strftime("%Y-%m-%d %H:%M:%S")
    ```

- 正文截取
  - Django自带stringfilter truncateword，如在模板中使用`{{ content| truncateword:'150' }}`会自动限制词数不超过150，并将之后的文本用"..."显示。但这个过滤器只能用于英文文本的截取。
  - 为实现中文段落的截取，实现了文本过滤器truncatehanzi，在newsweb/newsapp/templatetags/filter.py中。文件结构为

    ```
    # coding=utf8
    from django.template import Library
    from django.template.defaultfilters import stringfilter

    register = Library()

    @stringfilter
    def truncatehanzi(value, arg):
        """
        Truncates a string after a certain number of words including
        alphanumeric and CJK characters.
        Argument: Number of words to truncate after.
        """
        …
    register.filter('truncatehanzi', truncatehanzi)
    ```

  - 这样便完成了过滤器标签的注册。在模板头部使用`{% load filter %}`，正文部分使用变量`{{ content| truncatehanzi:'150' }}`会自动显示截取后的文本内容。
- 搜索结果关键字高亮
  - 搜索结果的高亮同样由templatetags来完成，相关文件包括newsweb/newsapp/templatetags/highlight.py和newsweb/newsapp/newshighlight.py。这两个文件主要借鉴了Haystack（一个全文搜索插件，之前本打算使用Whoosh+Haystack完成高级搜索功能，奈何并不能复现TF-IDF中对关键字重要性的评估）。

- 在模板头部使用`{% load highlight%}`，并在需要高亮的部分使用`{% highlight content with query %}`，其中content是期望被渲染的内容，query是搜索内容，可自动完成截取+高亮的工作。高亮实际是在关键词两侧加入了span标签，需要用CSS自定义样式，如`<style>span.highlighted { color: red; }</style>`。