针对清华大学新闻网适配的轻量级信息检索系统设计与实现

本系统使用Python 2开发（下面的Python均指Python 2），共分为两个部分，前端和后端。前端主要是显示在终端用户浏览器上的内容（HTML等），后端是本次实现的重点，包括蜘蛛模块、索引模块以及检索模块。本系统还使用了Python的`django`框架来进行`SQLite`数据库（可以较为简便的转换为其他SQL数据库）读写与网页编程。特别地，本系统针对清华大学新闻网进行了适配与优化。

后端服务程序

首先介绍后端的详细设计与实现细节。

蜘蛛模块

蜘蛛模块是一个很形象的叫法，蜘蛛会在网上爬来爬去，蜘蛛模块干的也是相似的事情，由`spider.py`文件实现。

蜘蛛模块使用Python的`urllib`、`urllib2`以及`urlparse`等模块进行网页的下载和URL的解析。下载网页的方法十分简单，设置好相应的请求`Request`，然后调用`urlopen`、`info`及`read`即可。调用`info`是为了获取HTTP响应头中的`Content-Type`（内容类型）字段，从而判断相应的文件是否是HTML文件。

这里需要指出的是，`read`之后需要解码，首先，程序尝试使用UTF-8解码，失败之后再尝试使用GB18030解码，如果继续失败，那么跳过失败的字符。详见`spider.py`中`decode_chinese`函数。

下载并解码之后就可以开始解析，解析的办法有两种。一种是直接使用正则表达式来删除HTML标签，提取出HTML中的文本信息；另外一种是使用Python的模块来解析HTML。

这里，我使用了Python自带的模块`HTMLParser`。这个模块十分轻便，方便使用，用户只需要继承`HTMLParser`类，重写`handle_starttag`、`handle_data`和`handle_endtag`方法即可。我编写的子类是`SpiderParser`，它只提取出标题、文本信息（不包含样式表以及脚本）和超级链接。

提取后，将网页标题以及文本信息存入数据库`search_page`表（对应`models`的`Page`）保存，超级链接经过域名检查（检查域名是否在允许列表里）、合并（如果是相对链接，那么就需要和当前页面URL合并，使用`urlparse`的`urljoin`）后送入队列，待继续处理。

之后，程序从队列中取出下一个待处理的超级链接，重复进行上述操作，实现对网页节点的宽度优先搜索。

另外，在系统管理员停止蜘蛛模块程序的时候，模块会自动保存当前队列的内容，待下一次启动模块继续爬行。模块也会重复读取已保存的页面获得其中的超级链接来初始化队列。

加载已保存的链接由`load_links_from_file`实现，重复爬取页面来获得超级链接由`get_initial_urls`实现。

针对清华大学新闻网的适配

经过观察，我特别对清华大学新闻网新闻页面的格式进行了适配，使用了简单的正则表达式来提取出标题以及新闻时间信息，由`spider_news.py`文件实现。内容由正则表达式提取后，使用了`SpiderParser`来过滤出文本信息。

匹配标题的正则表达式是`\<title\>(.+?)\</title\>`，匹配内容的正则表达式是`\<article(.+?)\>([\s\S]+?)\</article\>`，匹配时间的正则表达式是`\</i\>(\d{4})年(\d{1,2})月(\d{1,2})日\s+?(\d{1,2}):(\d{1,2}):(\d{1,2})[\s ]+?清华新闻网\</div\>`。

索引模块

索引，主要就是对已存入数据库的网页文本进行分词，建立倒排索引，由`index.py`实现。倒排索引存储的是对于每一个词，所有出现这个词的页面的一个唯一标识符（对应于数据库的主键）。

本系统使用Python的`jieba`（结巴中文分词）模块来进行网页文本内容的中文分词，由于使用了SQL数据库而不是文档型数据库，本系统对于每个页面，该页面的每个词语，将页面唯一标识符以及这个词语作为唯一的一行数据保存到数据库`search_word`表（对应`models`的`Word`）。但是，这样会使得`search_word`表行数很多，大约是页面数（以清华大学新闻网为例，大约为40000）乘以平均每页词语数（以清华大学新闻网为例，大约为400）。如使用文档型数据库，如MongoDB，则可以直接将倒排索引存入数据库，词语字段进行唯一索引，页面标识符是一个数组（列表），但`django`对于MongoDB的支持不好，不符合需求。

检索模块

由于使用了`django`框架以及数据库，自然就可以使用`django`框架提供的功能或者编写SQL语句来实现查询功能。

经过测试，`django`框架`models`模块提供的`filter`等方法能够获得正确的结果，但在特定情况下性能不够优秀，所以本系统没有采用这种方法。

本系统使用SQL来查询数据库`search_word`表，性能可以接受。查询前，也对用户输入进行了中文分词。

SQL代码示例如下：

```
SELECT `search_page`.* FROM `search_page`
    INNER JOIN (
        SELECT `page_id` FROM `search_word`
        WHERE `word` IN (?, ..., ?)
        GROUP BY `page_id` HAVING count(`word`) >= ?
        ORDER BY sum(`rank`) DESC
    ) AS `words` ON `search_page`.`id` = `words`.`page_id`
WHERE strftime('%%Y', `update_at`) = ?
LIMIT ?, ?
```

事实上，真正的SQL语句根据用户提供的查询请求构造（构造代码位于`search/views.py`中），用户输入数据通过参数传入。

参数说明：

- 第4行的`(?, ..., ?)`根据用户输入分词结果生成，一个词对应一个参数（问号）。
- 第5行的参数表明用户输入的词语数。
- 第8行是可选的，也可以有别的形式，例如`julianday(datetime('now')) - julianday(`update_at`) <= 31`或`julianday(datetime('now')) - julianday(`update_at`) <= 7`。这三条`WHERE`子句分别筛选出年份（参数即为年份）、近一月以及近一周更新的页面。
- 第9行为分页信息，第一个参数为偏移量，第二个参数为当前页应显示的记录条数。

这条SQL语句首先从`search_word`表中查询出所有符合用户输入的词语的记录，然后对每一个页面（以`page_id`标识）分组统计满足条件的词语数，再返回按照权重排序后的包含所有用户输入的词语（满足条件的词语数等于用户输入的词语数）的页面标识符；随后从`search_page`表中查询出这些页面，然后根据筛选条件（第8行中已说明）筛选出符合用户查找的条件的页面，最后进行分页操作。

这条SQL语句应还可以进一步优化，但是由于本人能力有限，没能继续优化。

前端用户界面

前端界面主要包括主搜索页（首页）、高级搜索页以及搜索结果页。

主搜索页包括搜索框、搜索按钮和高级搜索按钮，单击搜索按钮则转向搜索结果，单击高级搜索按钮转向高级搜索页。

高级搜索页包括搜索框、年份输入框、近一月复选框、近一周复选框以及搜索按钮。

搜索结果页从上到下含有搜索框、搜索按钮、高级搜索按钮、最多10条搜索结果和翻页组件。

为了使得页面更令人赏心悦目，需要精细的设计页面布局以及进行充足的界面美化工作。

这里，我使用了`Bootstrap 3`前端开发工具包、`bootstrap-dialog`组件以及`jQuery` JavaScript库来进行前端开发及页面美化。

`Bootstrap 3`十分易用，只需要修改HTML中少量的代码，修改需要美化的组件的`class`属性，即可在浏览器显示出漂亮的组件。`bootstrap-dialog`组件用来弹出对话框，在显示“网页文本快照”的时候用到。

用户点击“网页文本快照”时，前端会发送获取网页文本的请求到后端，随后后端返回网页文本给前端，前端将其中关键字高亮后显示出来。

优化

搜索结果优化

为了使得更相关的网页更靠前，我在SQL语句加入了第6行的排序操作。每一个页面的每个词都有自己的`rank`字段来表明重要程度，目前，标题中的词语该字段为2，文本中的词语该字段为1。后续可以使用更好的算法，比如TF-IDF提取关键词，提升关键词的重要程度。

然而，这个排序会使得查询速度变慢，所以需要进行进一步的用户体验的优化。

用户体验优化

为了让用户有更好的搜索体验，让用户感觉搜索结果返回的时间更短，我在搜索结果页加入了对于下一页的`prefetch`和`prerender`标签，提示浏览器预加载、渲染下一页。这样，在用户浏览完毕当前页面的时候，下一页已经准备好，用户可以瞬间访问。这为用户节省了时间，也为后端赢得了更多的用于检索结果的时间，多出来的时间可以为用户提供更优质的结果。

另外，我也在内存中缓存了用户输入过的关键词，其他用户再次查询的时候即可直接从缓存中取出，节约了查询数据库的时间。