

Enhancing the Search Functionality of Typecho

Jianlin Su

January 9, 2018

Scientific Space is a blog built using the Typecho program. The sidebar provides a search function; however, Typecho's built-in search is merely a string-based exact match search. Consequently, many reasonable queries fail to yield results. For example, queries like "2018 astronomical phenomena" or "new word algorithm" return nothing because the articles do not contain those exact strings.

Thus, the idea of strengthening the search function emerged, a suggestion previously made by some readers as well. Over the past few days, I conducted some research. I initially planned to use the Whoosh library in Python to build a full-text search engine, but I abandoned that idea as the workload for integration and subsequent maintenance seemed too high. Later, I thought about enhancing Typecho's own search capabilities. With the help of a senior colleague at my company, I completed this improvement.

Since this improvement was implemented by directly modifying Typecho's source files, it might be overwritten if Typecho is upgraded. Therefore, I am recording it here as a memo.

1 Exploration

By searching on GitHub, I discovered that Typecho's search functionality is implemented in `var/Widget/Archive.php`, specifically around lines 1185–1192:

```
if (!$hasPushed) {
    $searchQuery = '%' . str_replace(' ', '%', $keywords) . '%';
    /** Search cannot access private protected archives */

    $select->where('table.contents.password IS NULL')
    ->where('table.contents.title LIKE ? OR table.contents.text
        LIKE ?', $searchQuery, $searchQuery)
    ->where('table.contents.type = ?', 'post');
}
```

As can be seen, search results are returned by matching keywords in SQL, where % is the SQL wildcard character. We also find that if our input query contains spaces, those spaces are replaced by wildcards, making the search slightly more flexible.

Therefore, a natural idea is that regardless of whether the query contains spaces, we manually perform word segmentation on the query and then connect the segmented results with wildcards. This allows for more flexible searching even without spaces. This was indeed the first approach I practiced. However, the problem with this method is that even after segmentation, all words must match for a result to appear. If a single word has never appeared in the blog, no match will be found. To improve this, we need to consider an approach where each word is a candidate rather than a requirement.

2 Practice

To achieve the aforementioned goal, I wrote an HTTP interface in Python and deployed it on the server. This interface is responsible for word segmentation and generating SQL statements. I then replaced `$keywords = $this->request->filter('url', 'search')->keywords;` with `$keywords = $this->request->keywords;` and modified the code above as follows:

```
if (!$hasPushed) {
    $url = 'http://127.0.0.1:7777/token?text=' . $keywords;
    $url = str_replace(' ', '%20', $url);
    $searchQuery = file_get_contents($url);

    /** Use simple full match when the interface fails */
    if (!$searchQuery) {
        $searchQuery = 'SIGN(INSTR(table.contents.title, " ' .
            $keywords . '"))';
        $searchQuery = $searchQuery . ' + SIGN(INSTR(table.
            contents.text, " ' . $keywords . '"))';
    }

    /** Search cannot access private protected archives */
    $select->where('table.contents.password IS NULL')
    ->where($searchQuery . ' > 0')
    ->where('table.contents.type = ?', 'post')
    ->order($searchQuery, Typecho_Db::SORT_DESC);
}
```

The interface at `http://127.0.0.1:7777/token?text=` is a Python program:

```
# -*- coding: utf-8 -*-

import bottle
import jieba
jieba.initialize()

def convert(s):
    ws = jieba.cut(s)
    search = []
    for i in ws:
        search.append('2*SIGN(INSTR(table.contents.title, "%s"))'%i)
        search.append('SIGN(INSTR(table.contents.text, "%s"))'%i)
    return '(%s)%( ' + '.join(search))

@bottle.route('/token', method='GET')
def token_home():
    text = bottle.request.GET.get('text')
    if not text:
        text = ''
    return convert(text)

if __name__ == '__main__':
    bottle.run(host='0.0.0.0', port=7777, server='gunicorn')
```

This interface returns the scoring part of the SQL statement. The specific algorithm is: first, perform word segmentation; if an article title contains a word, add 2 points; if the article content contains the word, add 1 point. Finally, calculate a total score. For the functions used, such as `SIGN` and `INSTR`, you can find details via a search engine. I highly recommend using the `bottle` library; it is very convenient for writing lightweight HTTP interfaces.

Another necessary modification: because we modified the PHP section to use `order($searchQuery, Typecho_Db::SORT_DESC)`; to sort by score in descending order, this will not take effect immediately. Typecho defaults to sorting everything by time in descending order. Therefore, we must also modify lines 1396–1397 of the same file. Change the original:

```
$select->order('table.contents.created', Typecho_Db::SORT_DESC)
->page($this->_currentPage, $this->parameter->pageSize);
```

to:

```
if (strpos($select, 'INSTR') === false) {
    $select->page($this->_currentPage, $this->parameter->
        pageSize)
    ->order('table.contents.created', Typecho_Db::SORT_DESC);
} else {
    $select->page($this->_currentPage, $this->parameter->
        pageSize);
}
```

The logic here is to check if the statement is a search query. If it is, do not sort by time; otherwise, sort by time. Simply removing the time-based sorting is not an option because this line also handles the homepage output, which must be sorted chronologically.

3 Conclusion

Why use this combined Python and PHP solution instead of writing it purely in PHP? It is true that a pure PHP version is possible, and there is indeed a PHP version of the Jieba segmentation library. However, the main issue is that I do not know PHP! Furthermore, the PHP version of Jieba requires additional configuration, which is somewhat cumbersome. Using Python is much simpler for me; if any improvements are needed, I can just modify the Python script.

Finally, some users might worry about whether such a brute-force solution presents efficiency issues. In fact, if there were hundreds of thousands of articles, this approach would certainly have serious efficiency problems. However, for a blog with only a few hundred articles, this is not a concern.

Finally, I can search more freely! I welcome any further suggestions.

Reprinting notice: Please include the original link when reprinting: <https://kexue.fm/archives/4797>

For more details on reprinting, please refer to: *Scientific Space FAQ*