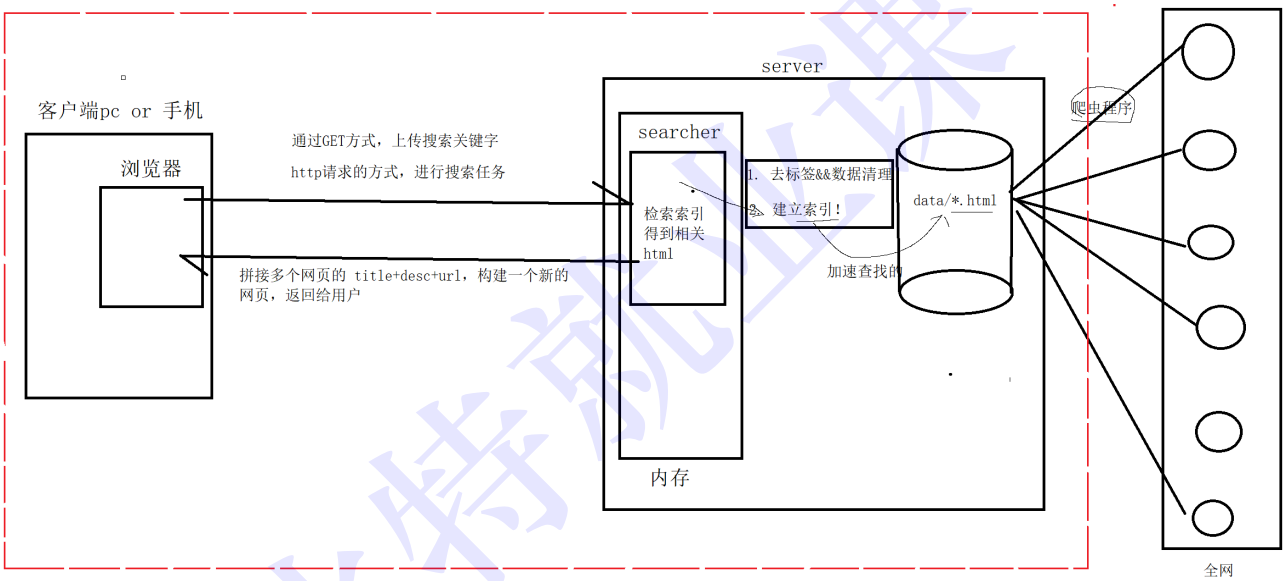
Boost搜索引擎项目

1. 项目的相关背景

> - 公司：百度、搜狗、360搜索、头条新闻客户端 - 我们自己实现是不可能的！
> - 站内搜索：搜索的数据更垂直，数据量其实更小
> - boost的官网是没有站内搜索的，需要我们自己做一个

2. 搜索引擎的相关宏观原理



3. 搜索引擎技术栈和项目环境

- 技术栈: `C/C++ C++11, STL, 准标准库Boost, Jsoncpp, cppjieba, cpp-httplib`, 选学: `html5, css, js、jQuery、Ajax`
- 项目环境: `Centos 7云服务器, vim/gcc(g++)/Makefile`, `vs2019 or vs code`

4. 正排索引 vs 倒排索引 - 搜索引擎具体原理

> - 文档1: 雷军买了四斤小米
> - 文档2: 雷军发布了小米手机

正排索引：就是从文档ID找到文档内容(文档内的关键字)

文档ID	文档内容
1	雷军买了四斤小米
2	雷军发布了小米手机

目标文档进行分词（目的：方便建立倒排索引和查找）：

> - 文档1[雷军买了四斤小米]: 雷军/买/四斤/小米/四斤小米
> - 文档2[雷军发布了小米手机]：雷军/发布/小米/小米手机

停止词：了，的，吗，a，the，一般我们在分词的时候可以不考虑

倒排索引：根据文档内容，分词，整理不重复的各个关键字，对应联系到文档ID的方案

关键字(具有唯一性)	文档ID, weight(权重)
雷军	文档1， 文档2
买	文档1
四斤	文档1
小米	文档1， 文档2
四斤小米	文档1
发布	文档2
小米手机	文档2

模拟一次查找的过程：

用户输入：小米 -> 倒排索引中查找 -> 提取出文档ID(1,2) -> 根据正排索引 -> 找到文档的内容 -> `title+conent (desc) +url` 文档结果进行摘要->构建响应结果

5. 编写数据去标签与数据清洗的模块 `Parser`

```
boost 官网:  https://www.boost.org/
//目前只需要boost_1_78_0/doc/html目录下的html文件，用它来进行建立索引
```

去标签

```
[whb@VM-0-3-centos boost_searcher]$ touch parser.cc
//原始数据 -> 去标签之后的数据
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>   <!--这是一个标签-->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Chapter 30. Boost.Process</title>
<link rel="stylesheet" href="../../doc/src/boostbook.css" type="text/css">
<meta name="generator" content="DocBook XSL Stylesheets V1.79.1">
<link rel="home" href="index.html" title="The Boost C++ Libraries BoostBook Documentation
Subset">
<link rel="up" href="libraries.html" title="Part I. The Boost C++ Libraries (BoostBook
Subset)">
<link rel="prev" href="poly_collection/acknowledgments.html" title="Acknowledgments">
<link rel="next" href="boost_process/concepts.html" title="Concepts">
```

```
</head>
<body bgcolor="white" text="black" link="#0000FF" vlink="#840084" alink="#0000FF">
<table cellpadding="2" width="100%"><tr>
<td valign="top"><img alt="Boost C++ Libraries" width="277" height="86"
src="../../boost.png"></td>
<td align="center"><a href="../../index.html">Home</a></td>
<td align="center"><a href="../../libs/libraries.htm">Libraries</a></td>
<td align="center"><a href="http://www.boost.org/users/people.html">People</a></td>
<td align="center"><a href="http://www.boost.org/users/faq.html">FAQ</a></td>
<td align="center"><a href="../../more/index.htm">More</a></td>
</tr></table>
.........

// <> : html的标签，这个标签对我们进行搜索是没有价值的，需要去掉这些标签，一般标签都是成对出现的!

[whb@VM-0-3-centos data]$ mkdir raw_html
[whb@VM-0-3-centos data]$ ll
total 20
drwxrwxr-x 60 whb whb 16384 Mar 24 16:49 input     //这里放的是原始的html文档
drwxrwxr-x  2 whb whb  4096 Mar 24 16:56 raw_html  //这是放的是去标签之后的干净文档

[whb@VM-0-3-centos input]$ ls -Rl  | grep -E '*.html' | wc -l
8141

目标：把每个文档都去标签，然后写入到同一个文件中！每个文档内容不需要任何\n！文档和文档之间用 \3 区分
version1:
类似：XXXXXXXXXXXXXXXX\3YYYYYYYYYYYYYYYYYYY\3ZZZZZZZZZZZZZZZZZZZZZZZZZZ\3

采用下面的方案：
version2：写入文件中，一定要考虑下一次在读取的时候，也要方便操作!
类似：title\3content\3url \n title\3content\3url \n title\3content\3url \n ...
方便我们getline(ifsream, line)，直接获取文档的全部内容：title\3content\3url
```

编写parser

```
//代码的基本结构：
#include <iostream>
#include <string>
#include <vector>

//是一个目录，下面放的是所有的html网页
const std::string src_path = "data/input/";
const std::string output = "data/raw_html/raw.txt";

typedef struct DocInfo{
    std::string title;   //文档的标题
    std::string content; //文档内容
    std::string url;     //该文档在官网中的url
}DocInfo_t;

//const &: 输入
//*: 输出
//&: 输入输出
```

```
bool EnumFile(const std::string &src_path, std::vector<std::string> *files_list);
bool ParseHtml(const std::vector<std::string> &files_list, std::vector<DocInfo_t>
*results);
bool SaveHtml(const std::vector<DocInfo_t> &results, const std::string &output);

int main()
{
    std::vector<std::string> files_list;
    //第一步: 递归式的把每个html文件名带路径，保存到files_list中，方便后期进行一个一个的文件进行读取
    if(!EnumFile(src_path, &files_list)){
        std::cerr << "enum file name error!" << std::endl;
        return 1;

    }
    //第二步: 按照files_list读取每个文件的内容，并进行解析
    std::vector<DocInfo_t> results;
    if(!ParseHtml(files_list, &results)){
        std::cerr << "parse html error" << std::endl;
        return 2;
    }

    //第三步: 把解析完毕的各个文件内容，写入到output,按照\3作为每个文档的分割符
    if(!SaveHtml(results, output)){
        std::cerr << "sava html error" << std::endl;
        return 3;
    }


    return 0;
}
bool EnumFile(const std::string &src_path, std::vector<std::string> *files_list)
{
    return true;
}
bool ParseHtml(const std::vector<std::string> &files_list, std::vector<DocInfo_t> *results)
{
    return true;
}
bool SaveHtml(const std::vector<DocInfo_t> &results, const std::string &output)
{
    return true;
}
```

boost 开发库的安装

```
[whb@VM-0-3-centos boost_searcher]$ sudo yum install -y boost-devel //是boost 开发库
```

提取title

```
1: d/i/ratio.html
 1 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"
 2 <html>
 3 <head>
 4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 5 <title>Chapter 35. Boost.Ratio 2.1.0</title>
 6 <link rel="stylesheet" href="../../doc/src/boostbook.css" type="text/css">
 7 <meta name="generator" content="DocBook XSL Stylesheets V1.79.1">
 8 <link rel="home" href="index.html" title="The Boost C++ Libraries BoostBook Documentation Subset">
 9 <link rel="up" href="libraries.html" title="Part I. The Boost C++ Libraries (BoostBook Subset)">
10 <link rel="prev" href="boost_random/history_and_acknowledgements.html" title="History and Acknowledge
11 <link rel="next" href="ratio/users_guide.html" title="User's Guide">
12 </head>
13 <body bgcolor="white" text="black" link="#0000FF" vlink="#840084" alink="#0000FF">
```

```
1: d/i/ratio.html
 1 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"
 2 <html>
 3 <head>
 4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 5 <title>Chapter 35. Boost.Ratio 2.1.0</title>
 6 <link rel="stylesheet" href="../../doc/src/boostbook.css" type="text/css">
 7 <meta name="generator" content="DocBook XSL Stylesheets V1.79.1">
 8 <link rel="home" href="index.html" title="The Boost C++ Libraries BoostBook Documentation Subset">
 9 <link rel="up" href="libraries.html" title="Part I. The Boost C++ Libraries (BoostBook Subset)">
10 <link rel="prev" href="boost_random/history_and_acknowledgements.html" title="History and Acknowledge
11 <link rel="next" href="ratio/users_guide.html" title="User's Guide">
12 </head>
13 <body bgcolor="white" text="black" link="#0000FF" vlink="#840084" alink="#0000FF">
```

提取content，本质是进行去标签

```
d/i/ratio.html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Chapter 35. Boost.Ratio 2.1.0</title>
<link rel="stylesheet" href="../../doc/src/boostbook.css" type="text/css">
<meta name="generator" content="DocBook XSL Stylesheets V1.79.1">
<link rel="home" href="index.html" title="The Boost C++ Libraries BoostBook Documentation Subset">
<link rel="up" href="libraries.html" title="Part I. The Boost C++ Libraries (BoostBook Subset)">
<link rel="prev" href="boost_random/history_and_acknowledgements.html" title="History and Acknowledgements">
<link rel="next" href="ratio/users_guide.html" title="User's Guide">
</head>
<body bgcolor="white" text="black" link="#0000FF" vlink="#840084" alink="#0000FF">
<table cellpadding="2" width="100%"><tr>
<td valign="top"><img alt="Boost C++ Libraries" width="277" height="86" src="../../boost.png"></td>
<td align="center"><a href="../../index.html">Home</a></td>
<td align="center"><a href="../../libs/libraries.htm">Libraries</a></td>
<td align="center"><a href="http://www.boost.org/users/people.html">People</a></td>
<td align="center"><a href="http://www.boost.org/users/faq.html">FAQ</a></td>
<td align="center"><a href="../../more/index.htm">More</a></td>
</tr></table>
<hr>
<div class="spirit-nav">
<a accesskey="p" href="boost_random/history_and_acknowledgements.html"><img src="../../doc/src/images/prev.png" alt="Prev"></a><a accesskey
../../doc/src/images/up.png" alt="Up"></a><a accesskey="h" href="index.html"><img src="../../doc/src/images/home.png" alt="Home"></a><a acc
ml"><img src="../../doc/src/images/next.png" alt="Next"></a>
</div>
<div class="chapter">
<div class="titlepage"><div>
<div><h2 class="title">
<a name="ratio"></a>Chapter 35. Boost.Ratio 2.1.0</h2></div>
<div><div class="author"><h3 class="author">
<span class="firstname">Howard</span> <span class="surname">Hinnant</span>
</h3></div></div>
<div><div class="author"><h3 class="author">
<span class="firstname">Beman</span> <span class="surname">Dawes</span>
</h3></div></div>
<div><div class="author"><h3 class="author">
<span class="firstname">Vicente J.</span> <span class="surname">Botet Escriba</span>
RMAL  data/input/ratio.html                                                                    html
```

在进行遍历的时候，只要碰到了 `>`,就意味着，当前的标签被处理完毕. 只要碰到了`<`意味着新的标签开始了

构建URL

boost库的官方文档，和我们下载下来的文档，是有路径的对应关系的

官网URL样例:　　https://www.boost.org/doc/libs/1_78_0/doc/html/accumulators.html

我们下载下来的url样例: boost_1_78_0/doc/html/accumulators.html

我们拷贝到我们项目中的样例: data/input/accumulators.html　//我们把下载下来的boost库 doc/html/* copy data/input/

url_head = "https://www.boost.org/doc/libs/1_78_0/doc/html";
url_tail = [data/input](删除) /accumulators.html -> url_tail = /accumulators.html

url = url_head + url_tail ; 相当于形成了一个官网链接

将解析内容写入文件中

//见代码
采用下面的方案:
version2: 写入文件中，一定要考虑下一次在读取的时候，也要方便操作!
类似: title\3content\3url \n title\3content\3url \n title\3content\3url \n ...
方便我们getline(ifsream, line)，直接获取文档的全部内容: title\3content\3url

6. 编写建立索引的模块 `Index`

```
//inidex.hpp基本结构

#pragma once

#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>

namespace ns_index{

    struct DocInfo{
        std::string title;   //文档的标题
        std::string content; //文档对应的去标签之后的内容
        std::string url;     //官网文档url
        uint64_t doc_id;          //文档的ID，暂时先不做过多理解
    };

    struct InvertedElem{
        uint64_t doc_id;
        std::string word;
        int weight;
    };

    //倒排拉链
    typedef std::vector<InvertedElem> InvertedList;

    class Index{
```

```
        private:
            //正排索引的数据结构用数组，数组的下标天然是文档的ID
            std::vector<DocInfo> forward_index; //正排索引
            //倒排索引一定是一个关键字和一组(个)InvertedElem对应[关键字和倒排拉链的映射关系]
            std::unordered_map<std::string, InvertedList> inverted_index;
        public:
            Index(){}
            ~Index(){}
        public:
            //根据doc_id找到找到文档内容
            DocInfo *GetForwardIndex(uint64_t doc_id)
            {
                return nullptr;
            }
            //根据关键字string，获得倒排拉链
            InvertedList *GetInvertedList(const std::string &word)
            {
                return nullptr;
            }
            //根据去标签，格式化之后的文档，构建正排和倒排索引
            //data/raw_html/raw.txt
            bool BuildIndex(const std::string &input) //parse处理完毕的数据交给我
            {
                return true;
            }
    };
}
```

建立正排的基本代码

```
 DocInfo *BuildForwardIndex(const std::string &line)
{
     //1. 解析line，字符串切分
     //line -> 3 string, title, content, url

     std::vector<std::string> results;
     const std::string sep = "\3";   //行内分隔符
     ns_util::StringUtil::CutString(line, &results, sep);
     //ns_util::StringUtil::CutString(line, &results, sep);
     if(results.size() != 3){
         return nullptr;
     }
     //2. 字符串进行填充到DocIinfo
     DocInfo doc;
     doc.title = results[0]; //title
     doc.content = results[1]; //content
     doc.url = results[2];   ///url
     //先进行保存id，在插入，对应的id就是当前doc在vector中的下标!
     doc.doc_id = forward_index.size();
     //3. 插入到正排索引的vector
     forward_index.push_back(std::move(doc)); //doc,html文件内容
     return &forward_index.back();
}
```

建立倒排

```
//原理:
struct InvertedElem{
    uint64_t doc_id;
    std::string word;
    int weight;
};

//倒排拉链
typedef std::vector<InvertedElem> InvertedList;
//倒排索引一定是一个关键字和一组(个)InvertedElem对应[关键字和倒排拉链的映射关系]
std::unordered_map<std::string, InvertedList> inverted_index;

//我们拿到的文档内容
struct DocInfo{
    std::string title;   //文档的标题
    std::string content; //文档对应的去标签之后的内容
    std::string url;     //官网文档url
    uint64_t doc_id;          //文档的ID，暂时先不做过多理解
};
//文档:
title ： 吃葡萄
content: 吃葡萄不吐葡萄皮
url: http://XXXX
doc_id: 123

根据文档内容，形成一个或者多个InvertedElem(倒排拉链)
因为当前我们是一个一个文档进行处理的，一个文档会包含多个"词"， 都应当对应到当前的doc_id

1. 需要对 title && content都要先分词  --使用jieba分词
title: 吃/葡萄/吃葡萄(title_word)
content: 吃/葡萄/不吐/葡萄皮(content_word)

词和文档的相关性（词频：在标题中出现的词，可以认为相关性更高一些，在内容中出现相关性低一些）
2. 词频统计
struct word_cnt{
    title_cnt;
    content_cnt;
}
unordered_map<std::string, word_cnt> word_cnt;
for &word : title_word{
     word_cnt[word].title_cnt++; //吃 (1) /葡萄 (1) /吃葡萄 (1)
}
for &word : content_word {
    word_cnt[word].content_cnt++; //吃 (1) /葡萄 (1) /不吐 (1) /葡萄皮 (1)
}

知道了在文档中，标题和内容每个词出现的次数
3. 自定义相关性
for &word : word_cnt{
    //具体一个词和123文档的对应关系，当有多个不同的词，指向同一个文档的时候，此时该优先显示谁？？相关性!
    struct InvertedElem elem;
```

```
    elem.doc_id = 123;
    elem.word = word.first;
    elem.weight = 10*word.second.title_cnt + word.second.content_cnt ; //相关性,我们这里拍着脑门写了
    inverted_index[word.first].push_back(elem);
}

//jieba的使用--cppjieba
获取链接:  git clone https://gitcode.net/mirrors/yanyiwu/cppjieba.git
如何使用:注意细节,我们需要自己执行:  cd cppjieba; cp -rf deps/limonp include/cppjieba/, 不然会编译报错

[whb@VM-0-3-centos test]$ ll
total 372
-rwxrwxr-x 1 whb whb 366424 Mar 28 12:11 a.out
drwxrwxr-x 8 whb whb   4096 Mar 28 12:01 cppjieba
-rw-rw-r-- 1 whb whb    856 Mar 28 12:11 demo.cpp
lrwxrwxrwx 1 whb whb     13 Mar 28 12:05 dict -> cppjieba/dict
lrwxrwxrwx 1 whb whb     16 Mar 28 12:06 inc -> cppjieba/include
-rw-rw-r-- 1 whb whb    365 Mar 28 10:21 test.cc
[whb@VM-0-3-centos test]$ cat demo.cpp
#include "inc/cppjieba/Jieba.hpp"
#include <iostream>
#include <string>
#include <vector>
using namespace std;

const char* const DICT_PATH = "./dict/jieba.dict.utf8";
const char* const HMM_PATH = "./dict/hmm_model.utf8";
const char* const USER_DICT_PATH = "./dict/user.dict.utf8";
const char* const IDF_PATH = "./dict/idf.utf8";
const char* const STOP_WORD_PATH = "./dict/stop_words.utf8";

int main(int argc, char** argv) {
  cppjieba::Jieba jieba(DICT_PATH,
        HMM_PATH,
        USER_DICT_PATH,
        IDF_PATH,
        STOP_WORD_PATH);
  vector<string> words;
  string s;

  s = "小明硕士毕业于中国科学院计算所，后在日本京都大学深造";
  cout << s << endl;
  cout << "[demo] CutForSearch" << endl;
  jieba.CutForSearch(s, words);
  cout << limonp::Join(words.begin(), words.end(), "/") << endl;

  return EXIT_SUCCESS;
}

//编写倒排索引的代码
//注意:建立倒排索引的时候,要忽略大小写!!
```

7. 编写搜索引擎模块 `Searcher`

基本代码结构

```
#include "index.hpp"

namespace ns_searcher{
    class Searcher{
        private:
            ns_index::Index *index; //供系统进行查找的索引
        public:
            Searcher(){}
            ~Searcher(){}
        public:
            void InitSearcher(const std::string &input)
            {
                //1. 获取或者创建index对象
                //2. 根据index对象建立索引
            }
            //query: 搜索关键字
            //json_string: 返回给用户浏览器的搜索结果
            void Search(const std::string &query, std::string *json_string)
            {
                //1.[分词]:对我们的query进行按照searcher的要求进行分词
                //2.[触发]:就是根据分词的各个"词", 进行index查找
                //3.[合并排序]: 汇总查找结果, 按照相关性(weight)降序排序
                //4.[构建]:根据查找出来的结果, 构建json串 -- jsoncpp
            }
    };
}
```

关键字(具有唯一性)	文档ID, weight(权重)
雷军	文档1， 文档2
买	文档1
四斤	文档1
小米	文档1， 文档2
四斤小米	文档1
发布	文档2
小米手机	文档2

搜索：雷军小米 -> 雷军、小米->查倒排->两个倒排拉链（文档1，文档2，文档1、文档2）

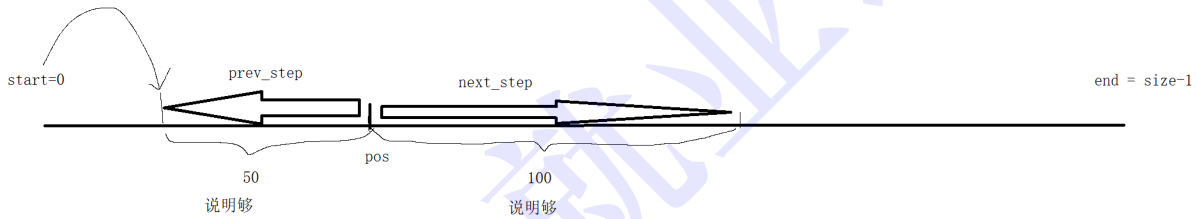
安装 `jsoncpp`

```
[whb@VM-0-3-centos boost_searcher]$ sudo yum install -y jsoncpp-devel
[sudo] password for whb:
Loaded plugins: aliases, auto-update-debuginfo, fastestmirror, protectbase
Repository epel is listed more than once in the configuration
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * epel-debuginfo: mirrors.tuna.tsinghua.edu.cn
 * extras: mirrors.aliyun.com
 * updates: mirrors.aliyun.com
0 packages excluded due to repository protections
Package jsoncpp-devel-0.10.5-2.el7.x86_64 already installed and latest version
//我已经安装了
```

具体实现代码

```
//见项目代码
```

获取摘要




```
//2. 获取start, end
std::size_t start = 0;
std::size_t end = html_content.size() - 1;
//如果之前有50+字符, 就更新开始位置
if(pos - prev_step > start) start = pos - prev_step;
if(pos + next_step < end) end = pos + next_step;
```

关于调试

把整个文件读到内存

先拿到标题，取到了标题。

对整个文件进行去标签，其中是包括标签的！！！！

实际如果一个词在title中出现，一定会被当标题 和 当内容分别被统计一次！！！

8. 编写`http_server`模块

cpp-httplib库：https://gitee.com/zhangkt1995/cpp-httplib?_from=gitee_search

注意：cpp-httplib在使用的时候需要使用较新版本的gcc，centos 7下默认gcc 4.8.5

升级gcc

```
[whb@VM-0-3-centos boost_searcher]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --
infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-
bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-
zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --
enable-linker-build-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-
c++,java,fortran,ada,go,lto --enable-plugin --enable-initfini-array --disable-libgcj --
with-isl=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/isl-install --
with-cloog=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-install -
-enable-gnu-indirect-function --with-tune=generic --with-arch_32=x86-64 --build=x86_64-
redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-44) (GCC)

用老的编译器，要么编译不通过，要么直接运行报错

搜索：scl gcc devsettool升级gcc
//安装scl
[whb@VM-0-3-centos boost_searcher]$ sudo yum install centos-release-scl scl-utils-build
//安装新版本gcc
[whb@VM-0-3-centos boost_searcher]$ sudo yum install -y devtoolset-7-gcc devtoolset-7-gcc-
c++
[whb@VM-0-3-centos boost_searcher]$ ls /opt/rh/
//启动：  细节，命令行启动只能在本会话有效
[whb@VM-0-3-centos boost_searcher]$ scl enable devtoolset-7 bash
[whb@VM-0-3-centos boost_searcher]$ gcc -v
//可选：如果想每次登陆的时候，都是较新的gcc
[whb@VM-0-3-centos boost_searcher]$ cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH

#每次启动的时候，都会执行这个scl命令
```

```
scl enable devtoolset-7 bash
```

安装 cpp-httplib

最新的cpp-httplib在使用的时候，如果gcc不是特别新的话有可能会有运行时错误的问题

建议：cpp-httplib 0.7.15

下载zip安装包，上传到服务器即可

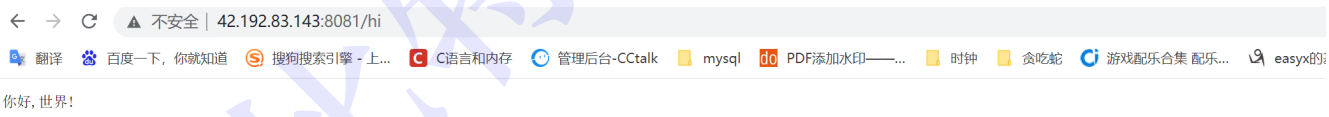
基本使用测试

```
[whb@VM-0-3-centos boost_searcher]$ cat http_server.cc
//#include "searcher.hpp"
#include "cpp-httplib/httplib.h"

int main()
{
    httplib::Server svr;

    svr.Get("/hi", [](const httplib::Request &req, httplib::Response &rsp){
            rsp.set_content("你好,世界!", "text/plain; charset=utf-8");
            });

    svr.listen("0.0.0.0", 8081);
    return 0;
}
```




```
[whb@VM-0-3-centos boost_searcher]$ cat http_server.cc
//#include "searcher.hpp"
#include "cpp-httplib/httplib.h"

const std::string root_path = "./wwwroot";

int main()
{
    httplib::Server svr;

    svr.set_base_dir(root_path.c_str());
    svr.Get("/hi", [](const httplib::Request &req, httplib::Response &rsp){
            rsp.set_content("你好,世界!", "text/plain; charset=utf-8");
            });

    svr.listen("0.0.0.0", 8081);
    return 0;
```

```
}
[whb@VM-0-3-centos boost_searcher]$ tree wwwroot/
wwwroot/
`-- index.html

0 directories, 1 file
[whb@VM-0-3-centos boost_searcher]$ cat wwwroot/index.html
<!DOCTYPE html>

<html>
<head>
    <meta charset="UTF-8">
    <title>for test</title>
</head>
<body>
    <h1>你好,世界</h1>
    <p>这是一个httplib的测试网页</p>
</body>
</html>
```

你好,世界

这是一个httplib的测试网页

正式编写 `httplib` 对应的调用

```
[whb@VM-0-3-centos boost_searcher]$ cat http_server.cc
#include "cpp-httplib/httplib.h"
#include "searcher.hpp"

const std::string input = "data/raw_html/raw.txt";
const std::string root_path = "./wwwroot";

int main()
{
    ns_searcher::Searcher search;
    search.InitSearcher(input);

    httplib::Server svr;
    svr.set_base_dir(root_path.c_str());
    svr.Get("/s", [&search](const httplib::Request &req, httplib::Response &rsp){
            if(!req.has_param("word")){
                rsp.set_content("必须要有搜索关键字!", "text/plain; charset=utf-8");
                return;
            }
            std::string word = req.get_param_value("word");
```

```
            std::cout << "用户在搜索: " << word << std::endl;
            std::string json_string;
            search.Search(word, &json_string);
            rsp.set_content(json_string, "application/json");
            //rsp.set_content("你好,世界!", "text/plain; charset=utf-8");
            });

    svr.listen("0.0.0.0", 8081);
    return 0;
}
```

9. 编写前端模块

了解vscode

编辑器

下载

官网: https://code.visualstudio.com/
下载慢: https://www.zhihu.com/search?type=content&q=vscode%20%E4%B8%8B%E8%BD%BD%E6%85%A2

安装插件

```
Chinese (Simplified) (简体中文) Language Pack for Visual Studio Code
Open in Browser
```

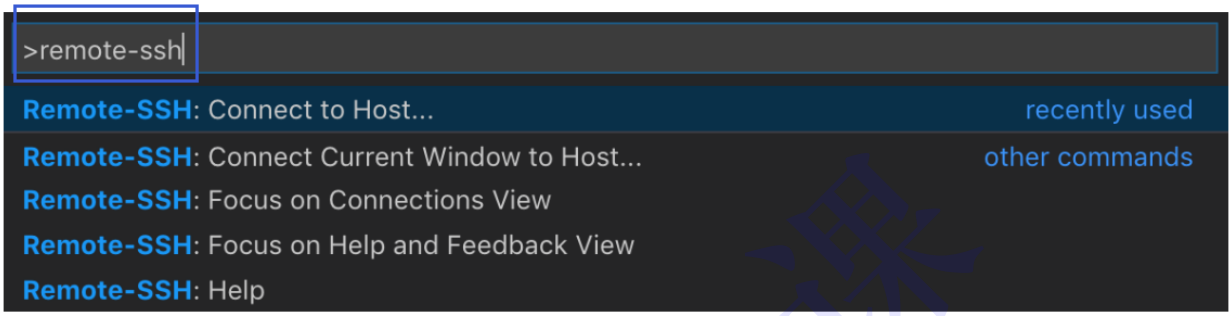
远程链接Linux

安装插件: Remote SSH

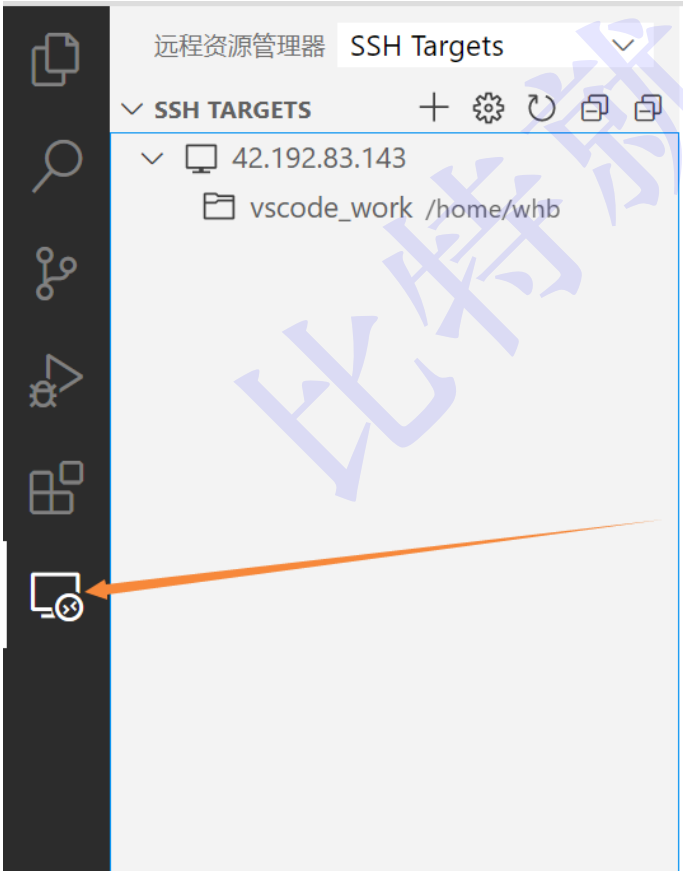
Follow the step-by-step tutorial or if you have a simple SSH host setup, connect to it as follows:

1. Press `F1` and run the **Remote-SSH: Open SSH Host...** command.
2. Enter your user and host/IP in the following format in the input box that appears and press enter: `user@host-or-ip` or `user@domain@host-or-ip`
3. If prompted, enter your password (but we suggest setting up key based authentication).
4. After you are connected, use **File > Open Folder** to open a folder on the host.

You can press `F1` to bring up the Command Palette and type in `Remote-SSH` for a full list of available commands.



You can also click on the Remote "Quick Access" status bar item in the lower left corner to get a list of the most common commands.



编写前端代码

```
1. !tab
2. 由标签构成：单标签，双标签
```

dex.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>boost 搜索引擎</title>
</head>
<body>
    <h1>hello ,欢迎来到我的搜索引擎</h1>
</body>
</html>
```

了解 `html, css, js`

```
html: 是网页的骨骼 -- 负责网页结构
css: 网页的皮肉 -- 负责网页美观的
js (javascript) : 网页的灵魂---负责动态效果，和前后端交互

教程： https://www.w3school.com.cn/
```

编写 `html`

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>boost 搜索引擎</title>
</head>
<body>
    <div class="container">
        <div class="search">
            <input type="text" value="输入搜索关键字...">
            <button>搜索一下</button>
        </div>
        <div class="result">
            <div class="item">
                <a href="#">这是标题</a>
                <p>这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要</p>
                <i>https://search.gitee.com/?skin=rec&type=repository&q=cpp-httplib</i>
            </div>
            <div class="item">
                <a href="#">这是标题</a>
                <p>这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要</p>
                <i>https://search.gitee.com/?skin=rec&type=repository&q=cpp-httplib</i>
            </div>
            <div class="item">
                <a href="#">这是标题</a>
                <p>这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要</p>
                <i>https://search.gitee.com/?skin=rec&type=repository&q=cpp-httplib</i>
            </div>
            <div class="item">
                <a href="#">这是标题</a>
                <p>这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要</p>
                <i>https://search.gitee.com/?skin=rec&type=repository&q=cpp-httplib</i>
            </div>
            <div class="item">
                <a href="#">这是标题</a>
                <p>这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要这是摘要</p>
                <i>https://search.gitee.com/?skin=rec&type=repository&q=cpp-httplib</i>
            </div>
        </div>
    </div>
</body>
</html>
```


编写 `css`

设置样式的本质：找到要设置的标签，设置它的属性
1．选择特定的标签：类选择器，标签选择器，复合选择器
2．设置指定标签的属性：见代码

```
    <style>
        /* 去掉网页中的所有的默认内外边距，html的盒子模型 */
        * {
            /* 设置外边距 */
            margin: 0;
            /* 设置内边距 */
            padding: 0;
        }
        /* 将我们的body内的内容100%和html的呈现吻合 */
        html,
        body {
            height: 100%;
        }
        /* 类选择器.container */
        .container {
            /* 设置div的宽度 */
            width: 800px;
            /* 通过设置外边距达到居中对齐的目的 */
            margin: 0px auto;
            /* 设置外边距的上边距，保持元素和网页的上部距离 */
            margin-top: 15px;
        }
        /* 复合选择器，选中container 下的 search */
        .container .search {
            /* 宽度与父标签保持一致 */
            width: 100%;
            /* 高度设置为52px */
            height: 52px;
        }
        /* 先选中input标签， 直接设置标签的属性，先要选中， input：标签选择器*/
        /* input在进行高度设置的时候，没有考虑边框的问题 */
        .container .search input {
            /* 设置left浮动 */
            float: left;
            width: 600px;
            height: 50px;
            /* 设置边框属性：边框的宽度，样式，颜色 */
            border: 1px solid black;
            /* 去掉input输入框的有边框 */
            border-right: none;
            /* 设置内边距，默认文字不要和左侧边框紧挨着 */
            padding-left: 10px;
            /* 设置input内部的字体的颜色和样式 */
            color: #ccc;
            font-size: 15px;
        }
        /* 先选中button标签， 直接设置标签的属性，先要选中， button：标签选择器*/
        .container .search button {
```

```
            /* 设置left浮动 */
            float: left;
            width: 150px;
            height: 52px;
            /* 设置button的背景颜色，#4e6ef2 */
            background-color: #4e6ef2;
            /* 设置button中的字体颜色 */
            color: #FFF;
            /* 设置字体的大小 */
            font-size: 19px;
            font-family:Georgia, 'Times New Roman', Times, serif;
        }
        .container .result {
            width: 100%;
        }
        .container .result .item {
            margin-top: 15px;
        }

        .container .result .item a {
            /* 设置为块级元素，单独站一行 */
            display: block;
            /* a标签的下划线去掉 */
            text-decoration: none;
            /* 设置a标签中的文字的字体大小 */
            font-size: 20px;
            /* 设置字体的颜色 */
            color: #4e6ef2;
        }
        .container .result .item a:hover {
            /*设置鼠标放在a之上的动态效果*/
            text-decoration: underline;
        }
        .container .result .item p {
            margin-top: 5px;
            font-size: 16px;
            font-family:'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans
Unicode', Geneva, Verdana, sans-serif;
        }

        .container .result .item i{
            /* 设置为块级元素，单独站一行 */
            display: block;
            /* 取消斜体风格 */
            font-style: normal;
            color: green;
        }
    </style>
```

编写 `js`

```
如果直接使用原生的js成本会比较高（xmlhttprequest），我们推荐使用JQuery.
JQuery CDN: https://www.jq22.com/cdn/
```

```
<script>
    function Search(){
        // 是浏览器的一个弹出框
        // alert("hello js!");
        // 1. 提取数据，$可以理解成就是JQuery的别称
        let query = $(".container .search input").val();
        console.log("query = " + query); //console是浏览器的对话框，可以用来进行查看js数据

        //2. 发起http请求,ajax: 属于一个和后端进行数据交互的函数，JQuery中的
        $.ajax({
            type: "GET",
            url: "/s?word=" + query,
            success: function(data){
                console.log(data);
                BuildHtml(data);
            }
        });
    }

    function BuildHtml(data){
        // 获取html中的result标签
        let result_lable = $(".container .result");
        // 清空历史搜索结果
        result_lable.empty();

        for( let elem of data){
            // console.log(elem.title);
            // console.log(elem.url);
            let a_lable = $("<a>", {
                text: elem.title,
                href: elem.url,
                // 跳转到新的页面
                target: "_blank"
            });
            let p_lable = $("<p>", {
                text: elem.desc
            });
            let i_lable = $("<i>", {
                text: elem.url
            });
            let div_lable = $("<div>", {
                class: "item"
            });
            a_lable.appendTo(div_lable);
            p_lable.appendTo(div_lable);
            i_lable.appendTo(div_lable);
            div_lable.appendTo(result_lable);
        }
    }
</script>
```

后续

存在搜到重复内容

修正之后

添加日志

```
#pragma once

#include <iostream>
#include <string>
#include <ctime>

#define NORMAL  1
#define WARNING 2
#define DEBUG   3
#define FATAL   4
```

```
#define LOG(LEVEL, MESSAGE) log(#LEVEL, MESSAGE, __FILE__, __LINE__)

void log(std::string level, std::string message, std::string file, int line)
{
    std::cout << "[" << level << "]" << "[" << time(nullptr) << "]" << "[" << message <<
"]" << "[" << file << " : " << line << "]" << std::endl;
}
```

部署服务到`linux`上

```
[whb@VM-0-3-centos boost_searcher]$ nohup ./http_server > log/log.txt 2>&1 &
[1] 26890
```

结项总结

项目扩展方向

```
1. 建立整站搜索
2. 设计一个在线更新的方案，信号，爬虫，完成整个服务器的设计
3. 不使用组件，而是自己设计一下对应的各种方案（有时间，有精力）
4. 在我们的搜索引擎中，添加竞价排名(强烈推荐)
5. 热次统计，智能显示搜索关键词（字典树，优先级队列）(比较推荐)
6. 设置登陆注册，引入对mysql的使用(比较推荐的)
```

备注

项目源代码地址： https://gitee.com/whb-helloworld/search-engine-online-sharing