搜索引擎需求文档说明书

1.项目开发环境

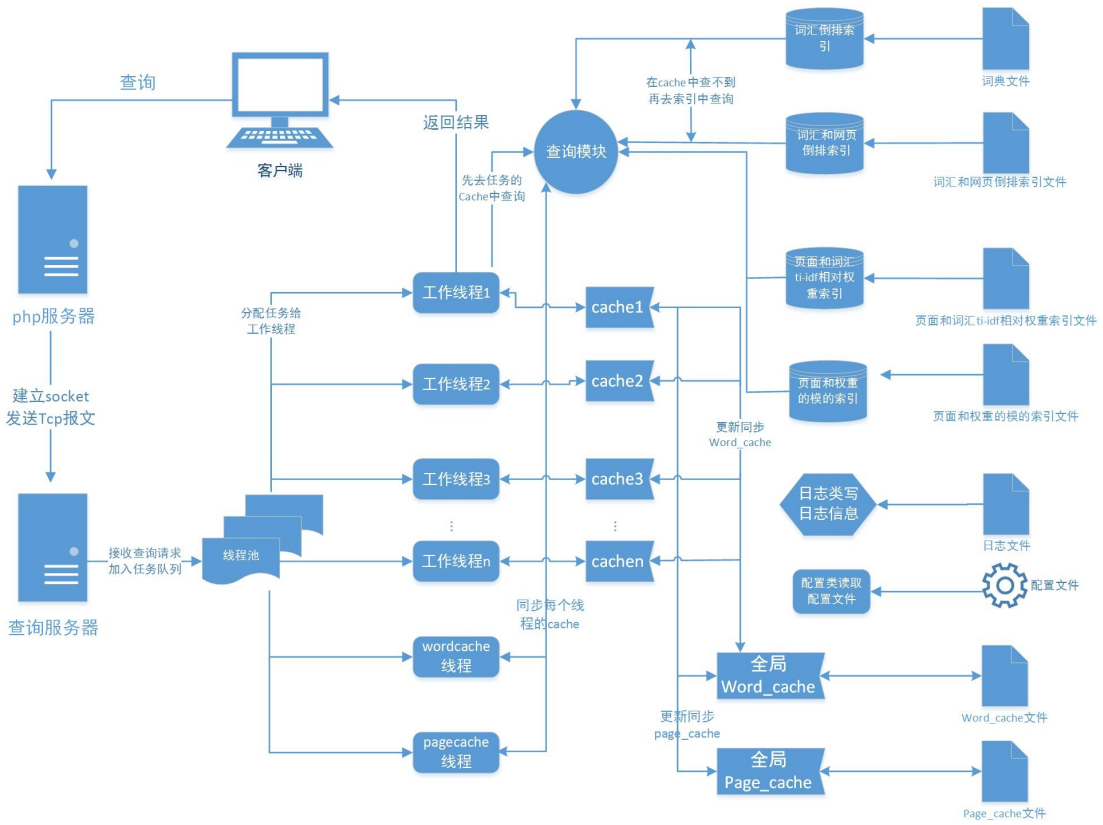
Linux: Ubuntu18.04
G++: Version 4.8.4
Vim: Version 8.0

2.系统目录结构

src/：存放系统的源文件（*.cpp/*.cc） 。
Include/：存放系统的头文件（*.hpp） 。
bin/：存放系统的可执行程序
conf/myconf.conf：存放系统程序中所需的相关配置信息
data/dict.dat: 存放词典
data/dictIndex.dat: 存放单词所在位置的索引库
data/newripepage.dat：存放网页库
data/newoffset.dat：存放网页的偏移库
data/invertIndex.dat：存放倒排索引库
log/:存放日志文件

3. 系统需求

3.1 系统运行过程图



3.2 需求分析

1）客户端发送关键字给服务器，获取相关关键字
2）客户端发送查询关键字给服务器，获取相关网页信息
3）服务器根据关键字，返回给客户端相关关键字
4）服务器根据关键字，返回给客户端相关网页

3.2.1 消息定义

每一条服务器和客户端之间的消息都应该满足以下格式：

消息内容长度(4 个字节)	消息 ID（4 个字节）	消息内容

3.2.2 详细定义

消息 ID	消息内容	发送方向	客户端处理	服务器处理
1	关键字	C->S	无	保存并处理
2	关键字	C->S	无	保存并处理
100	推荐的相关关键字	S->C	获取并展示	无
200	关键字相关网页信息	S->C	获取并展示	无

3.3 系统模块划分

3.3.1 模块一：关键字推荐服务部分

实现该服务，要分为两个部分：离线部分和在线部分。

3.3.1.1 离线部分

1) 创建词典，词典中的每一条记录的格式为：

Word　Frequency

注意：加载候选词文件时，把候选词中的大写全部转成小写，候选词文件的路径通过配置文件读入。

2) 创建索引文件

查询词 query 没有必要与所有的候选词来比较，例如，查询词是 nike，候选词是 appl，这两个词没有任何字符有交集，这种情况没有必要计算编辑距离。如何利用索引来缩小候选词，以达到提高计算性能的目的：当 query 是 nike 的时候，我们只需要查找候选词包含 n 或者 i 或者 k 或者 e 的这些词。

n：ipone，iphone

i：ipone，iphone，nike

k：nike，kindle

e：nike，iphone，ipone，kindle

l：loop，appl

中：　中国　中间　　其中

3.3.1.2 在线部分

1. 搭建服务端框架
服务器的搭建，采用之前在课上讲解时封装好的 TcpServer 就可以了，然后等待客户端的连接，接收客户端的查询请求。

2. 获取到客户端传递过来的查询词之后，再从索引之中查找与之相近的候选词，选取到最合适的候选词之后再将其发送给客户端。

- 候选词的选取包括以下几个部分：

1) 实现最小编辑距离核心算法 ---- 计算候选词与查询词的相似度
2) 候选词选择策略：hello　　helli
   a.优先比较最小编辑距离；
   b.在编辑距离相同的条件下，再比较候选词的词频；词频越大优先选择
   c.在词频相同的条件下，按字母表顺序比较候选词；
3) 获取 k 个（例如 3 个或者 5 个）候选词，返回给客户端。（需要使用优先级队列）

- 发送给客户端的数据要采用 JSON 数据格式封包。对于 Json 开源库的选择，可以有多种，比如 jsoncpp、或者 nlohmann/json[位于 github 上]，建议用后者，比前者好用.

3. 功能优化

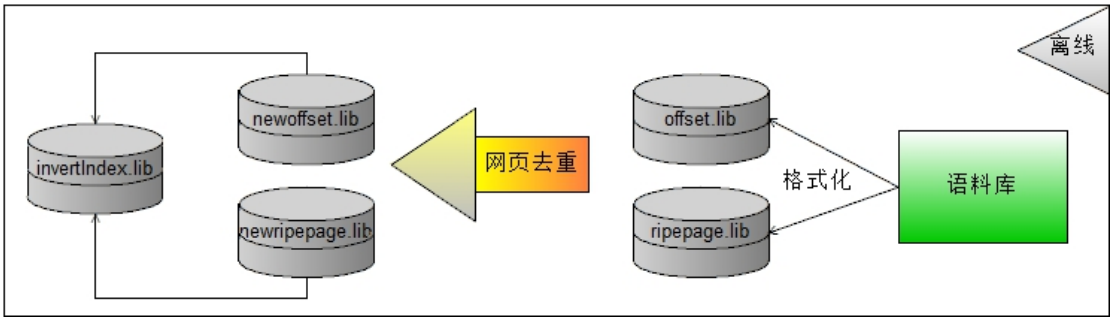
1） 引入缓存系统，有两种方案选择：
   a. 个人设计缓存系统。
   设计思路：
   - 根据线程池中工作线程的数量，每一个工作线程对应一个缓存。
   - 每个缓存内部采用 LRU 算法对冷数据进行淘汰。
   - 当工作一段时间后，每个缓存的内容会不一致，因此需要更新缓存。
   - 更新缓存的操作，交给一个定时器完成，每隔固定的时间，更新缓存数据。

   b. 使用 Redis 来优化查询效率。（使用 Redis++）

2） 处理中文数据（UTF8 编码规则）
   a. 首先需要对中文进行分词，然后再去统计每个词语的词频，建立中文词典和索引，类似英文的处理。
   b. 对中文进行分词，需要使用开源的分词库，如 cppjieba, NLPIR，所以要了解分词库的用法。推荐使用 cppjieba.
   c. **扩展最小编辑距离算法**，让其能对中文进行处理。

3.3.2 模块二：网页查询服务部分

实现该服务，要分为两个大模块：离线部分和在线部分。
离线部分的步骤分为：
（1）建立网页库和网页偏移库；
（2）网页去重；
（3）建立倒排索引，并生成网页去重之后的网页库和网页偏移库；
如下图所示：



在线部分的模块分为：
（1）服务器模块：启动服务器，进入监听状态，等待客户端连接
（2）线程池模块：将客户端的请求封装成任务，交给线程池处理
（3）查询模块：提供网页查询服务
如下图所示：

3.3.2.1 离线部分

1. 建立网页库和网页偏移库。

（1）网页库的格式采用 xml 的形式，细分为<doc>,<id>,<title>,<url>,<content>,具体形式如下：

```
<doc>
    <id>1</docid>
    <url>http://baidu.com/</url>
    <title>博客榜—查看频道“中国互联网观察”</title>
    <content>博客榜—查看频道“中国互联网观察”…</content>
</doc>
<doc>
    <docid>2</docid>
    <url>……</url>
    <title>…. </title>
```

<content>博客榜博客榜—查看频道,随便扯淡吧……. </content>
</doc>
<doc>
<docid>3</docid>
<url>……</url>
<title>…. </title>
<content>博客榜—查看频道,随便扯淡吧……. </content>
</doc>

（2）将所给语料库中的所有文档按照格式，构造成网页库 **ripepage.dat**；
提取网页的标题的策略如下：

1) 如果查找到字符串“【 标 题 】”，则抽取该行的内容为标题；
2) 如果没有找到，则提取文档的第一行为标题

整个文档的全部内容为 content（包括标题）
Url 使用每篇文档所在的绝对路径；然后将所有的字符串拼接起来，构成一篇格式化的文档，例如

```
string fmtTxt = "<doc><docid>" + docid +
                "</docid><url>" + url+
                "</url><title>" + title +
                "</title><content>" + content +
                "</content></doc>";
```

（3）获取格式化好的文档之后，通过 C++的文件输出流将文档写到文件中去，同时记录下该篇文档在文件的位置信息，即 **offset.dat**。

2. 网页去重

当网页库建立好之后，需要删除网页库中相同的文档，即网页去重。网页去重的方法有两种：
（1）TopK。为每个网页提取网页中的特征码，特征码为该网页中词频（就是一个词在该网页中出现的次数）最高的 10 个词----top10 词；对于每两篇网页，比较 top10 词的交集，如果交集大于等于 8 个，认为它们是互为重复的网页。
在计算 top 词的时候，就需要使用分词程序，即将一篇网页分词，然后统计词频。
互为重复的网页，删除哪一个？比如说，删除 docid 大的，或者删除文档内容少的。

采用此方法，需要先用分词库对文档进行分词，然后去停用词，最后统计每篇文档的词语和词频。可以使用的分词库有：

~~NLPIR(中科院)：http://ictclas.nlpir.org/~~
cppjieba(github)：https://github.com/yanyiwu/cppjieba

（2）Simhash 算法，强烈推荐使用，google 出品。

3. 建立倒排索引

倒排索引：英文原名为 Inverted index，大概因为 Invert 有颠倒的意思，就被翻译成了倒排。一个未经处理的网页库中，一般是以文档 ID 作为索引，以文档内

容作为记录。而 Inverted index 指的是将单词或记录作为索引，将文档 ID 作为记录，这样便可以方便地通过单词或记录查找到其所在的文档。
在此项目中，倒排索引的数据结构采用的是:

unordered_map <string, vector<pair<int, double>>> InvertIndexTable

其中 unordered_map 的 key 为出现在文档中的某个词语，对应的 value 为包含该词语的文档 ID 的集合以及该词语的权重值 w。

倒排索引的建立既是此项目的重点也是难点，而建倒排索引时最难的是每个词语的权重值的计算，它涉及到如下几个概念：

TF : Term Frequency, 某个词在文章中出现的次数；

DF: Document Frequency, 某个词在所有文章中出现的次数，即包含该词语的文档数量；

IDF: Inverse Document Frequency, 逆文档频率，表示该词对于该篇文章的重要性的一个系数，其计算公式为：

$IDF = \log_2(N/(DF+1))$，其中 N 表示文档的总数或网页库的文档数

最后，词语的权重 w 则为：

$$w = TF * IDF$$

可以看到权重系数与一个词在文档中的出现次数成正比，与该词在整个网页库中的出现次数成反比。

而一篇文档包含多个词语 w1,w2,...,wn，还需要对这些词语的权重系数进行归一化处理，其计算公式如下：

$$w' = w /sqrt(w1^2 + w2^2 +...+ wn^2)$$

w'才是需要保存下来的，即倒排索引的数据结构中 InvertIndexTable 的 double 类型所代表的值。此权重系数的算法称为 TF-IDF 算法。

3.3.2.2 在线部分

1. 搭建服务端框架

服务器框架的搭建，与之前的然后等待客户端的连接，接收客户端的查询请求。

2. 提供网页查询服务

查询策略：

（1）在处理查询请求时，对于查询的关键词，将它们视为一篇文档 X，通过 TF-IDF 算法计算出每个关键词的权重系数；将其组成一个向量(x1, x2, ...,xn)，该向量作为基准向量 Base，在下面的(3)中使用；

（2）通过倒排索引表去查找包含所有关键字的网页；只要其中有一个查询词不在索引表中，就认为没有找到相关的网页；

（3）如果找到了网页，则需要对查找到的网页进行排序。排序算法采用余弦相似度。既然查找到的网页都包含查询词，那么获取每个查询词的 w，将它们组成一个向量 Y = (y1,y2, ... yn)，用该向量代表这篇网页，该向量 Y 是网页的特征，然后计算它与 Base 的余弦值，该余弦值代表的就是 Y 与 X 的相似度；那么现在只需要将查找到的所有网页都与 X 进行余弦相似度 cosθ的计算，然后根据

cosθ的大小进行排序，cosθ越大越相似，这样的网页应该出现在前排的位置。
cosθ的计算方法如下：

$$X * Y = (x1 * y1 + x2 * y2 + x3 * y3)$$

$$\cos\theta = (X * Y) / (|X| * |Y|)$$

（4）当找到网页之后，还需要提取每篇网页中的标题和摘要信息，然后将这些信息封装成一个 JSON 字符串，交给服务器框架模块去发送给客户端。要注意的是：摘要信息是根据查询词自动生成的。

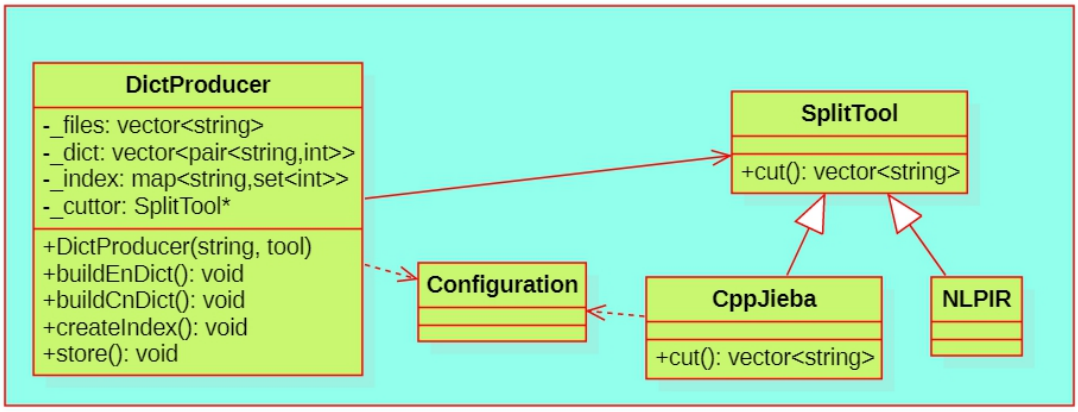
3 优化查询模块，提升网页查询效率：使用 Redis 建立缓存系统

4. 相关类及其说明

红色部分表示其成员函数为 public 权限，绿色部分表示其成员函数为 private 权限

4.1 离线部分

4.1.1 模块一：创建词典和索引文件（针对于关键字推荐）



Class DictProducer(词典创建类)

数据成员：

vector<string> _files;	语料库文件的绝对路径集合
vector<pair<string,int>> _dict;	词典
SplitTool * _splitTool;	分词工具
Map<string, set<int>> _index;	词典索引

主要的函数成员：

DictProducer(const string& dir)	构造函数
DictProducer(const string& dir, SplitTool * splitTool)	构造函数,专为中文处理
void buildEnDdict()	创建英文字典

void buildCnDict()	创建中文字典
void storeDict(const char * filepath)	将词典写入文件
void showFiles()const	查看文件路径，作为测试用
void showDict()const	查看词典，作为测试用
void getFiles()	获取文件的绝对路径
void pushDict(const string & word)	存储某个单词

Class SplitTool(分词工具类)

主要的函数成员：

SplitTool()	构造函数
virtual ~SplitTool()	虚析构函数
virtual vector<string> cut(const string & sentence)=0	分词函数，纯虚函数，提供接口

Class SplitToolCppJieba(分词工具类)

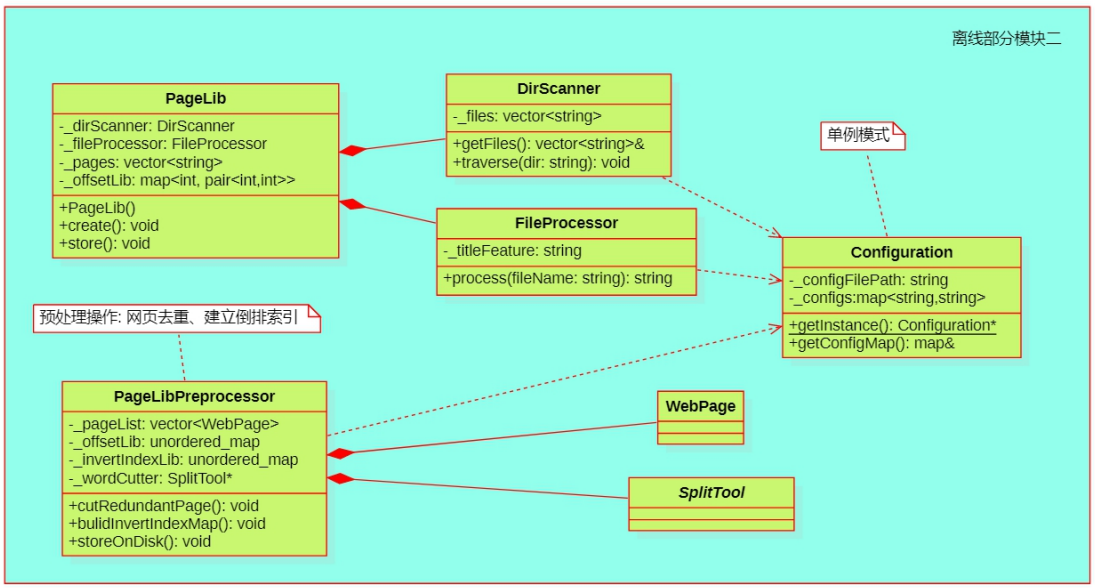
数据成员：

Configuration & _conf;	配置文件路径

主要的函数成员：

SplitToolCppJieba()	构造函数
virtual ~SplitToolCppJieba()	虚析构函数
virtual vector<string> cut(const string & sentence)=0	分词函数，纯虚函数，提供接口

4.1.2 模块二：创建网页库相关（针对于网页查询）



Class PageLib(网页库类)

数据成员：

DirScanner & _dirScanner;	目录扫描对象的引用
vector<string> _files;	存放格式化之后的网页的容器
Map<int, pair<int, int> > _offsetLib;	存放每篇文档在网页库的位置信息

主要的函数成员：

PageLib(Configuration& conf, DirScanner & dirScanner, FileProcessor & fileProcessor)	构造函数
void create()	创建网页库
void store()	存储网页库和位置偏移库

Class Configuration(配置文件类)

数据成员：

string _filepath;	配置文件路径
map<string,string> _configMap;	配置文件内容
set<string> _stopWordList;	停用词词集

主要的函数成员：

Configuration(const string& filepath)	构造函数
Map<string,string> & getConfigMap()	获取存放配置文件内容的 map
Set<string> getStopWordList()	获取停用词词集

Class DirScanner(目录扫描类 -- 递归扫描)

数据成员：

vector<string> _files;	存放每个语料文件的绝对路径

主要的函数成员：

DirScanner()	构造函数
void operator()()	重载函数调用运算符，调用 traverse 函数
vector<string> files()	返回_vecFilesfiles 的引用
void traverse(const string &dirname)	获取某一目录下的所有文件

Class PageLibPreprocessor(网页库预处理类)

数据成员：

WordSegmentation _jieba;	分词对象
vector <WebPage> _pageLib;	网页库的容器对象
unordered_map<int, pair<int, int> > _offsetLib;	网页偏移库对象
unordered_map<string, vector<pair<int, double> > > _invertIndexTable;	倒排索引表对象

主要的函数成员：

PageLibPreprocessor(Configuration& conf)	构造函数
void doProcess()	执行预处理
void readInfoFromFile()	根据配置信息读取网页库和位置偏移库的内容
void cutRedundantPages()	对冗余的网页进行去重
Void buildInvertIndexTable()	创建倒排索引表
Void storeOnDisk()	将经过预处理之后的网页库、位置偏移库和倒排索引表写回到磁盘上

Class WebPage(网页类)

数据成员：

const static int TOPK_NUMBER = 20;	
string _doc;	整篇文档，包含 xml 在内
Int _docId;	文档 id
String _docTitle;	文档标题
String _docUrl;	文档 URL
String _docContent;	文档内容
String _docSummary;	文档摘要，需自动生成，不是固定的
vector<string> _topWords;	词频最高的前 20 个词
map<string,int> _wordsMap;	保存每篇文档的所有词语和词频，不包括停用词

主要的函数成员：

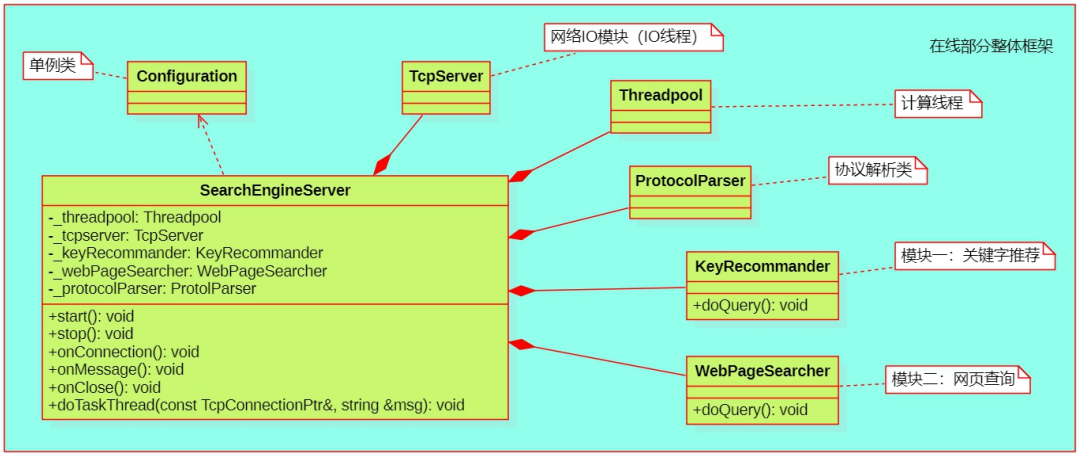
WebPage(string & doc, Configuration & config, WordSegmentation & jieba)	构造函数
int getDocId()	获取文档的 docid
string getDoc()	获取文档
map<string, int> & getWordsMap()	获取文档的词频统计 map
void processDoc(const string & doc, Configuration & config, WordSegmentation&)	对格式化文档进行处理
void calcTopK(vector<string> & wordsVec, int k, set<string> & stopWordList)	求取文档的 topk 词集

友元函数：

friend bool operator==(const WebPage & lhs, const WebPage & rhs)	判断两篇文档是否相等
friend bool operator < (const WebPage & lhs, const WebPage & rhs)	对文档按 Docid 进行排序

4.2 在线部分

4.2.0 在线部分整体架构



4.2.1 基础类--配置文件类

Class Configuration(配置文件类)

数据成员：

string　filepath_;	配置文件路径
map<string,string> configMap_;	配置文件内容
set<string> stopWordList_;	停用词词集

主要的函数成员：

Configuration(const string& filepath)	构造函数
Map<string,string> & getConfigMap()	获取存放配置文件内容的 map
Set<string> getStopWordList()	获取停用词词集

4.2.2 基础类--线程池相关

Class Mutex（互斥变量类）

数据成员：

pthread_mutex_t _mutex;	互斥变量

函数成员：

MutexLock()	构造函数
~MutexLock()	析构函数
void lock()	锁函数

void unlock()	解锁函数
Pthread_mutex_t * getMutexPtr()	获取原生互斥锁的地址，由 Condition 对象进行调用

Class Condition(条件变量类)

数据成员：

CMutex& _mutex	词典
pthread_cond_t _cond	本类实例

函数成员：

Condition(MutexLock & mutex)	构造函数
~Condition()	析构函数
void wait()	等待
void notify()	唤醒
void notifyall()	唤醒全部

Class TaskQueue（任务队列类）

数据成员：

MutexLock _mutex;	互斥锁对象
Condition _notEmpty;	缓冲区有数据的条件变量
Condition _notFull;	缓冲区没有放满数据的条件变量
typedef std::function<void()> Task;	任务的回调函数作为接口
size_t _size;	队列大小
queue<Task> _que;	队列
bool _flag;	标志位，用于退出 pop 函数

函数成员：

TaskQueue(int)	构造函数
void empty()	判断队列是否为空
void full()	判断队列是否已满
void push(Task)	往缓冲区中添加任务
Task pop()	从缓冲区中获取任务
void wakeup()	唤醒_notEmpty 条件变量

Class Thread（线程类）

数据成员：

pthread_t _pthId;	Linux 下的线程类型
bool _isRunning;	记录线程是否正在运行
typedef function<void()> ThreadCallback;	重定义回调函数的标签
ThreadCallback _cb;	执行任务的函数对象

函数成员：

Thread(ThreadCallback cb)	构造函数
~Thread()	析构函数
void start()	线程开始执行
void join()	等待线程执行完毕

static void * threadFunc(void*arg);	线程的函数执行体

Class Threadpool（线程池类）

数据成员：

int _threadNum;	Linux 下的线程类型
vector<Thread *> _vecThreads;	线程对象的容器
int _bufSize;	缓冲区大小
Buffer _buf;	缓冲区对象
typedef function<void()> Task;	重定义回调函数的标签

函数成员：

Threadpool(int threadNum, int bufSize)	构造函数
void start()	线程池开始执行
void stop()	停止线程池
void addTask(Task)	往线程池中添加任务
void threadFunc()	线程池中每个线程的函数执行体
Task getTask()	从缓冲区中获取任务

4.2.3 基础类--网络编程相关

Class InetAddress(网络地址类)

数据成员：

struct sockaddr_in _addr;	Linux 下的 sockaddr_in 类型

函数成员：

InetAddress(unsigned short)	构造函数
InetAddress(const char * ip, unsigned short)	构造函数
InetAddress(const struct sockaddr_in & addr)	构造函数
string ip() const	从网络地址获取点分十进制 ip
unsigned short port() const	从网络地址获取端口号

Class Socket(网络套接字类)

数据成员：

int _sockfd;	Linux 下的 sockaddr_in 类型

函数成员：

Socket()	构造函数
Socket(int sockfd)	构造函数
void nonblock()	设置 fd 为非阻塞模式
void shutdownWrite()	关闭套接字的写端
int fd()	返回_sockfd

Class SocketIO(网络 IO 类)

数据成员：

int _sockfd;	Linux 下的 sockaddr_in 类型

函数成员：

SocketIO(int sockfd)	构造函数
size_t readline(char * buf, size_t max)	从对端读取 1 行数据
size_t readn(char * buf, size_t count)	从对端读取 count 个字节的数据
size_t writen(const char * buf, size_t count)	从本地发送数据
size_t recvPeek(char * buf, size_t count)	查看内核缓冲区，并获取数据

Class Acceptor(接收器类)

数据成员：

Socket _listensock;	服务器监听 Socket 对象
InetAddress _addr;	服务器网络地址

函数成员：

Acceptor(int fd, const InetAddress &)	构造函数
void ready()	服务器监听准备
int accept()	接收新连接
void setReuseAddr(bool on)	设置服务器网络地址可重用
size_t setReusePort(bool on)	设置服务器网络端口可重用
void bind()	绑定网络地址
void listen()	进行监听

Class TcpConnection(网络连接类)

Typedef std::shared_ptr<TcpConnection> TcpConnectionPtr;

数据成员：

Socket _sockfd;	
SocketIO _sockIO;	
const InetAddress _localAddr;	
const InetAddress _peerAddr;	
bool _isShutdownWrite;	
EpollPoller * _loop;	保存 EpollPoller 对象的指针
typedef function<void (const TcpConnectionPtr &) TcpConnectionCallback;	
TcpConnectionCallback _onConnectionCb;	
TcpConnectionCallback _onMessageCb;	
TcpConnectionCallback _onCloseCb;	

函数成员：

TcpConnection(int sockfd)	构造函数
~TcpConnection()	析构函数
string receive()	接收数据
void send(const string & msg)	发送数据
void sendInLoop(const string & msg)	将数据交给 **IO 线程**发送
void sendAndClose(const string &msg)	发送数据，并关闭连接，针对网页服务
void shutdown()	关闭连接
void setConnectionCallback(TcpConnectionCallback cb)	设置回调函数
void setMessageCallback(TcpConnectionCallback cb)	设置回调函数
void setCloseCallback(TcpConnectionCallback cb)	设置回调函数
void handleConnectionCallback()	调用相应的回调函数
void handleMessageCallback()	调用相应的回调函数
void handleCloseCallback()	调用相应的回调函数
string toString()	返回连接的字符串表示

Class EventLoop(epoll 封装类)

数据成员：

int _epollfd;	Epoll 实例的文件描述符
int _eventfd;	eventfd,用于线程之间的通信
int _listenfd;	服务器监听文件描述符
bool _isLooping;	标记是否进行循环
typedef vector<struct epoll_event> EventList;	
EventList _eventList;	存储触发事件的 fd
Typedef map<int, TcpConnetionPtr> ConnectionMap;	
ConnectionMap _connMap;	保存所有已建立的连接
typedef TcpConnection::TcpConnectionCallback EpollCallback;	
EpollCallback _onConnectionCb;	回调函数，传递给 TcpConnection 对象
EpollCallback _onMessageCb;	回调函数，传递给 TcpConnection 对象
EpollCallback _onCloseCb;	回调函数，传递给 TcpConnection 对象
typedef std::function<void()> Functor;	函数回调重定义
Vector<Functor> _pendingFunctors;	需要延迟执行的回调函数

函数成员：

EventLoop(int listenfd)	构造函数
~EventLoop()	析构函数

void loop()	执行事件循环
void unloop()	退出事件循环
void runInLoop(const Functor & cb)	向 IO 线程发送数据
void wakeup()	激活_eventfd(执行写操作)
void handleRead()	处理_eventfd(执行读操作)
void doPendingFunctors()	执行回调函数
void setConnectionCallback(EpollCallback cb)	设置回调函数
void setMessageCallback(EpollCallback cb)	设置回调函数
void setCloseCallback(EpollCallback cb)	设置回调函数
void waitEpollfd()	执行事件循环,由 loop 调用
void handleConnection()	处理新连接
void handleMessage(int peerfd)	处理旧连接（信息）

Class TcpServer(Tcp 服务器类)

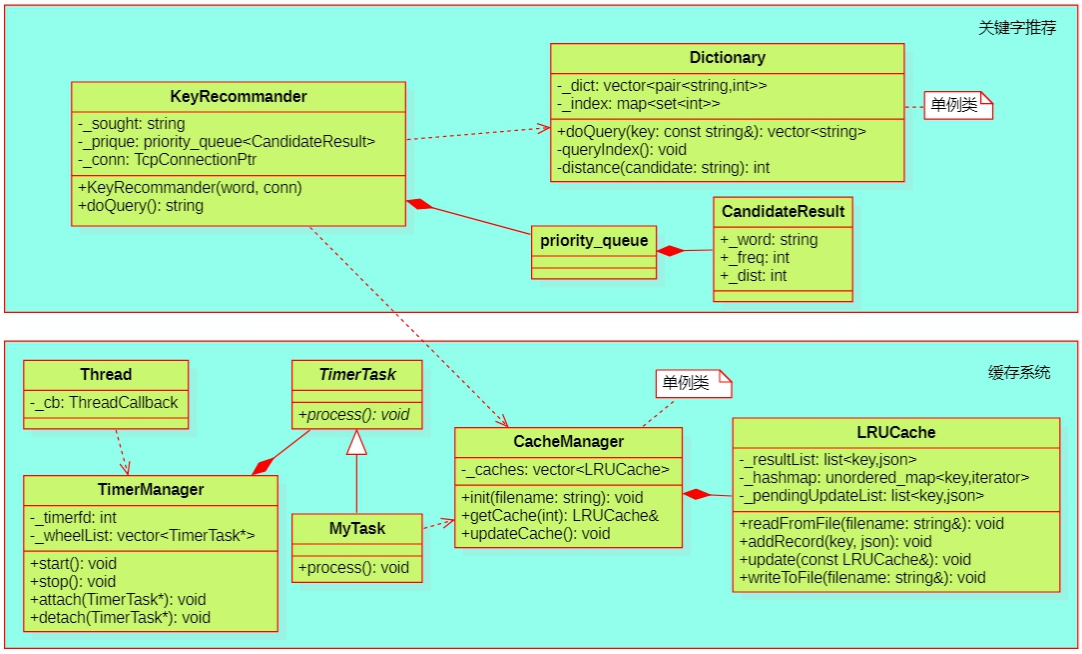
数据成员：

Acceptor _acceptor;	连接器对象
EventLoop _epollfd;	EventLoop 对象
typedef TcpConnection::TcpConnectionCallback TcpServerCallback;	

函数成员：

TcpServer(unsigned short port)	构造函数
TcpServer(const char * ip,unsigned short port)	构造函数
void start()	开始服务
void stop()	停止服务
void setConnectionCallback(TcpServerCallback cb)	设置回调函数
void setMessageCallback(TcpServerCallback cb)	设置回调函数
void setCloseCallback(TcpServerCallback cb)	设置回调函数

4.2.4 模块一：关键字推荐



Class Dictionary（词典类）

数据成员：

vector<pair<string,int>> _dict;	词典
map<string,set<int>> _indexTable;	索引表

函数成员：

static Dictionary * createInstance()	静态函数
void init (const string & dictpath)	通过词典文件路径初始化词典
vector<pair<string, int> > & getDict()	获取词典
map<string, set<int> > & getIndexTable()	获取索引表
string doQuery(string word)	执行查询

Class KeyRecommander(关键字推荐执行类)

数据成员：

string _queryWord;	等查询的单词
TcpConnectionPtr _conn;	与客户端进行连接的文件描述符
priority_queue<MyResult, vector<MyResult>, MyCompare> _resultQue;	保存候选结果集的优先级队列

主要的函数成员：

KeyRecommander(string&query, const TcpConnectionPtr&)	构造函数
void execute()	执行查询
void queryIndexTable()	查询索引
void statistic(set<int> & iset)	进行计算
int distance(const string & rhs)	计算最小编辑距离
void response()	响应客户端的请求

Class LRUCache（缓存类）

数据成员：

unorderd_map<string, iterator> _hashMap;	采用 hashTable 进行查找
list<string, string> _resultsList;	保存键值对
list<string, string> _pendingUpdateList;	等待更新的节点信息
int _capacity;	缓存节点的容量

函数成员：

LRUCache(int num = 1000)	构造函数
LRUCache(const Cache & cache)	构造函数
void addElement(const string &key, const string & value)	往缓存中添加数据
void readFromFile(const string & filename)	从文件中读取缓存信息
void writeToFile(const string & filename)	将缓存信息写入到文件中
void update(const Cache & rhs)	更新缓存信息
list<string,string> & getPendingUpdateList()	获取待更新的节点 List

Class CacheManager（缓存管理类）

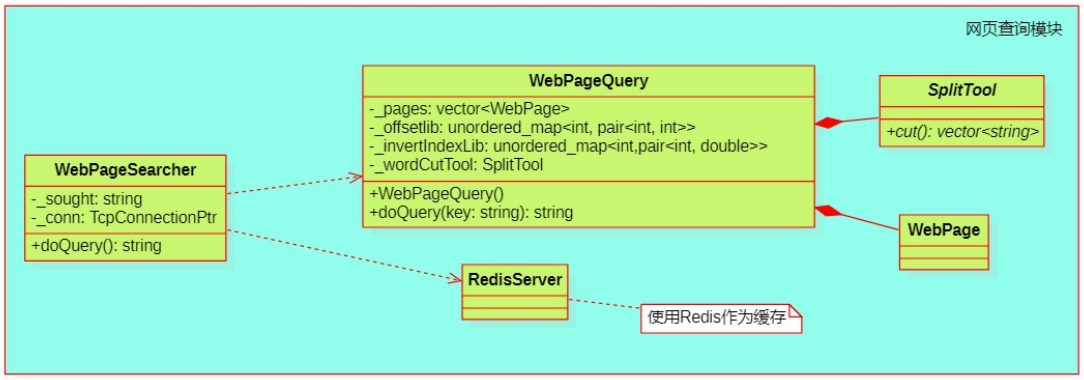
数据成员：

vector<Cache> _cacheList;	缓存的数量与线程个数一致

函数成员：

void initCache(size_t, const string &filename)	从磁盘文件中读取缓存信息
LRUCache & getCache(size_t idx)	获取某个缓存
void periodicUpdateCaches()	定时更新所有的缓存

4.2.5 模块二：网页查询



Class WebPageSearcher(网页搜索类)

数据成员：

string _sought;	查询关键词
TcpConnectionPtr _conn;	TcpConnection 对象

主要的函数成员：

WebPageSearcher(string keys, Const TcpConnectionPtr & conn)	构造函数
void doQuery()	执行查询

Class WebPage(网页类)

数据成员：

const static int TOPK_NUMBER = 20;	
Int _docId;	文档 id
String _docTitle;	文档标题
String _docUrl;	文档 URL
String _docContent;	文档内容
String _docSummary;	文档摘要，需自动生成，不是固定的
vector<string> _topWords;	词频最高的前 20 个词
Map<string,int> _wordsMap;	保存每篇文档的所有词语和词频，不包括停用词

主要的函数成员：

WebPage(string & doc, Configuration & config, WordSegmentation & jieba)	构造函数
int getDocId()	获取文档的 docid
string getTile()	获取文档的标题
String summary (const vector<string> &queryWords)	

map<string, int> & getWordsMap()	获取文档的词频统计 map
void processDoc(const string & doc, Configuration & config, WordSegmentation&)	对格式化文档进行处理
void calcTopK(vector<string> & wordsVec, int k, set<string> & stopWordList)	求取文档的 topk 词集

友元函数：

friend bool operator==(const WebPage & lhs, const WebPage & rhs)	判断两篇文档是否相等
friend bool operator < (const WebPage & lhs, const WebPage & rhs)	对文档按 Docid 进行排序

Class WebPageQuery(网页查询类)

数据成员：

WordSegmentation _jieba;	Jieba 分词库对象
unordered_map<int, WebPage> _pageLib;	网页库
unordered_map<int, pair<int, int> > _offsetLib;	偏移库
unordered_map<string, set<pair<int, double>>> _invertIndexTable;	倒排索引表

函数成员：

WebPageSearcher()	构造函数
String doQuery(const string & str)	执行查询，返回结果
void loadLibrary()	加载库文件
vector<double> getQueryWordsWeightVector(vector<string> & queryWords)	计算查询词的权重值
bool executeQuery(const vector<string> & queryWords, vector<pair<int,vector<double> > > &resultVec)	执行查询
string createJson(vector<int> & docIdVec, const vector<string> & queryWords)	
string returnNoAnswer()	

4.2.6 整体架构类

Class SearchEngineServer(查询服务器类)

数据成员：

TcpServer _tcpServer;	TCP 通信模块
Threadpool _pool;	线程池

KeyRecommander _keyRecommander;	关键字推荐模块
WebPageSearcher _webSearcher;	网页查询模块
ProtocolParser _protocolParser;	协议解析模块

函数成员：

SearchEngineServer(const string & ip, short port)	构造函数
void start()	开始提供服务
void onConnection(const TcpConnectionPtr & conn)	被注册回调函数，提供给 TcpServer 使用
void onMessage(const TcpConnectionPtr & conn)	被注册回调函数，提供给 TcpServer 使用
void onClose(const TcpConnectionPtr & conn)	被注册回调函数，提供给 TcpServer 使用
void doTaskThread(const TcpConnectionPtr & conn, const string & msg)	该方法由线程池的某一个线程执行

Class ProtocolParser(协议解析类)

数据成员：

主要的函数成员：
