**SmartCorrect项目文档**

# 需求分析

本程序对用户输入的拼写错误的单词（如单词”ipone”），通过计算编辑距离的方法返回与原单词相对接近且出现频率较高的单词（对于单词”ipone”，返回”iphone”）

# 程序目录结构

SmartCorrect/

bin/

server

cache/

cache\_a.txt

cache\_b.txt

config/

config.txt

dat/

(包括词库和索引文件等)

Include/

Cache.h CacheManager.h Edist.h NonCopyable.h query.h

SmartCorrect.h Timer.h TimerThread.h tThread.h

lib/

TcpServer/

Condition.h InetAddress.h MutexLock.h net.h NonCopyable.h PollPoller.h PtrVector.h Rio.h Socket.h TcpConnection.h TcpServer.h Thread.h ThreadPool.h

Condition.cpp client.c InetAddress.cpp InetAddress.cpp net.c PollPoller.cpp Socket.cpp TcpConnection.cpp Thread.cpp ThreadPool.cpp

log/

make/

prepare/

src/

Cache.cpp CacheManager.cpp Edist.cpp main.cpp query.cpp SmartCorrect.cpp Timer.cpp TimerThread.cpp tThread.cpp

tests/

# 程序主流程

Client

…………….

Threadn

ThreadPool

Thread1

……………………..

CacheManager

DiskCache

SmartCorrect服务器

“ipone”

CacheWriter

# 用到的主要技术

* 1. 计算编辑距离算法

作为本项目的核心算法，采用动态规划的方法来求单词的编辑距离，递推公式如下：

(本项目单词纠错并不单纯以编辑距离作为唯一的参考标准，而是先筛选出编辑距离在3以内的单词，然后再根据出现的词频进行排序来得到较接近的单词)

* 1. 实现LRU(Least Recent Used) Cache

1. SmartCorrect对象通过 CacheManager来对Cache进行存取，而不是直接对Cache进行存取。
2. Cahce的主要结构是一个map和一个list如下所示std::unordered\_map<std::string,std::pair<std::string,std::list<std::string>::iterator> >cache;和std::list<std::string>list.主结构为一个map通过一个辅助的结构list来实现对其的更新,其联系结构图如下

List

Map

|  |  |
| --- | --- |
| S1 | iterator |
| S2 | iterator |
| S3 | iterator |
| S4 | iterator |
| S5 | iterator |
| S6 | iterator |

|  |
| --- |
| S2 |
| S3 |
| S4 |
| S6 |
| S1 |
| S5 |

只有插入(insertCache)和查询(queryCache)两种操作，其源代码如下

insertCache(const string&linea, const string& value){  
 list.pushback(linea);

list<string>::iterator it = list.end();it--;

cache[linea] = make\_pair(lineb,it);//通过list的iterator来记录使用的时间

if(cache\_size() > Cache\_Size\_){//超出Cache的Size淘汰较长时间未使用的元素

string s = \*list.begin();

list.pop\_front();//在list中弹出

cache.erase(s)//在cache中擦除

}

｝

bool Cache::queryCache(const string& query,string& result){

string lineb;

bool flag = false;

if(cache\_.count(query))

{

lineb = cache\_[query].first;

result = process(lineb);

/\*\* 对list表的更新，将原来的iterator删去，构造新的iterator指向表的尾部 \*\*/

list\_.erase(cache\_[query].second);

list\_.push\_back(query);

list<string>::iterator it = list\_.end();

it--;

cache\_[query] = make\_pair(lineb,it);

flag =true;

}

return flag;

}

1. SmartCorrect对象通过一个定时器线程cache\_writer来将内存中的cache写到磁盘中，将键存到cache\_a文件中，将值存到cache\_b文件中
   1. 建立索引以便减小搜索范围

1.索引作用：建立索引是为了减小每次查询时的范围，即进行搜索剪枝。

2.索引结构与内容：索引文件命名为英语字母和阿拉伯数字的utf8编码，内容存放的是含有该字母或数字的单词以及其词频（如6a.txt存放的是含有字符’j’的单词和）

3.索引的使用方法：在每次查询时将单词中所有字符相对应的索引文件中的做并操作，以这些作为候选词，就不用再以整个字典的词作为候选词了。

4.索引建立过程如下：

1.先将字典”Dic.txt”文件中的单词大写转换成小写，并且将标点符号转换为空格存入”Dic1.txt”中

2.通过map对词频进行记数,word\_count[word]++

3.打开相应的文件控制符

4.将map中的单词及其词频写入到其对应的文件中

* 1. 线程池服务器

TcpServer对象中包装了三个对象:Socket对象、PollPoller对象和ThreadPool对象;其中socket对象用于建立Tcp连接，PollPoller对象用于多路转接实现并发服务器，ThreadPool对象实现线程池

本线程池是基于生产者/消费者问题构造的。ThreadPool对象中包含条件信号量 Condition empty,full和互斥锁MutexLock mutex和一个存放任务的队列std::queue<Task>task\_queue,而Task对象定义：typedef std:: function<void ()>Task。所以task\_queue是一个存放函数对象的队列，在这个模型中线程池对象看做是生产者，而各子线程看作是消费者对task\_queue对象进行互斥访问。

* 1. 输入输出优化

通过加入缓存技术提高读写数据效率，为优化读写效率构造Rio对象，其封装了缓存和一系列优化的读写方法，类声明如下：

class Rio : private NonCopyable

{

public :

explicit Rio(int fd);

ssize\_t readn( char \*usrbuf, size\_t n);//从userbuf中读取n个字符

ssize\_t readline(char \*usrbuf, size\_t maxlen);// 从userbuf中读取一行

ssize\_t writen(const char \*usrbuf, size\_t n);// userbuf中写入n个字符

private :

size\_t rio\_read(char \*buf, int n);

int fd\_;

int left\_;

char \*bufptr\_;

char buffer\_[RIO\_BUFFER];//封装的缓存

};