简易key-value型Database设计文档

515030910298

朱伯君

2017.07

一、基本信息

项目名称：简易key-value型数据存储系统

主要数据结构：B-树

开发环境：Visual Studio 2015

测试环境：MacBook Pro

处理器：2.7 GHz Intel Core i5

内存：8 GB 1867 MHz DDR3

概述：实现<key, value>型数据存储系统，支持查找、增添、修改、删除等功能

二、提供接口

只提供<int,string>的存储类型，string的长度不定，但是用户在数据库启动时必须设定string的最大长度。事实上，用户可以根据实际需求将其他类型转化为string类型，并将其存入数据库中。

-创建数据库：DB(const string& pathname);

-新建数据库：newDB(const int& cacheSize, const int& recordSize);

其中cacheSize表示缓存块的大小，recordSize表示每条记录的最大字节数

-从数据库文件中加载配置：loadFromFile();

-从数据库中查找：bool fetchRecord(const KEYTYPE& key, VALUETYPE& value);

若查找成功，会返回true，并设置value的值；若查找失败，会返回false

-从数据库中删除：bool deleteRecord(const KEYTYPE& key);

若删除成功，则返回true；若数据库中没有这条记录，则返回false

-数据库中插入某条数据：bool insertRecord(const KEYTYPE& key, const VALUETYPE& value);

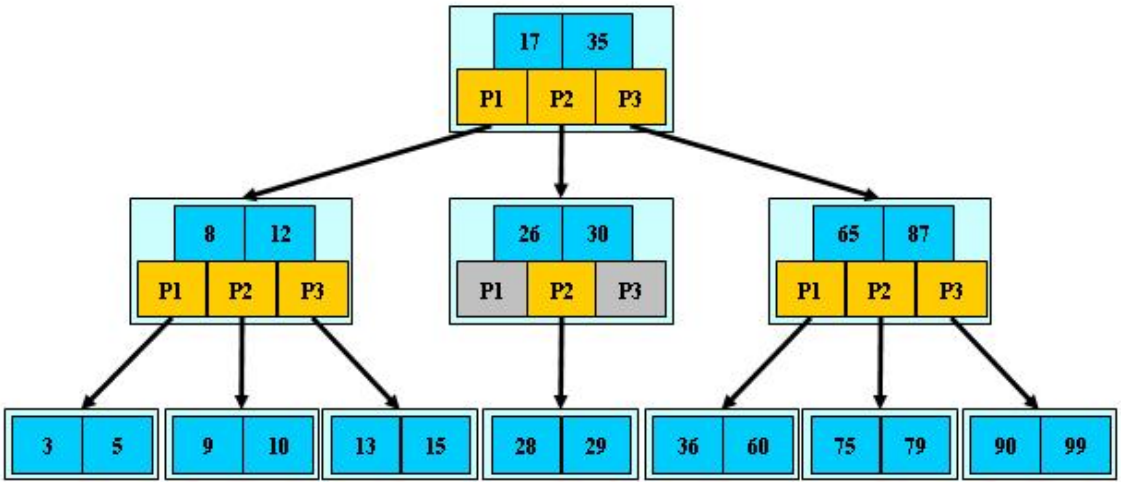
若插入成功，则返回true；若数据库中已经有重复的关键词key的数据，则返回false

-数据库中修改数据：bool modify(const KEYTYPE& key, const VALUETYPE& value);

若修改成功，则返回true；若数据库中没有关键词为key的数据，则返回false

三、功能的实现

-主要的数据机构：B-树



此数据库的数据结构主要是B-树。其中由于数据量不是很大，同时查找迅速，B-树的节点全部在内存中，B-树的每个节点都记有记录所对应的key值以及该记录在数据文件中的偏移量。对于索引文件而言，关闭数据库时，索引文件中首先存入根节点的信息，然后根据根节点的信息递归地载入其他节点（以前序遍历的次序），B-树遍历结束。打开数据库时，根据前序遍历的次序，不断地将索引文件中的信息载入内存当中。

-打开：

若使用newDB(const int& cacheSize, const int& recordSize)的接口，则直接根据缓存大小以及记录的长度等初始化数据库；

若使用loadFromFile()的接口，则从索引文件中初始化B-树，从配置文件中得到数据库原本的缓存大小，记录长度，以及先前删除过程中产生的碎片信息初始化数据库。

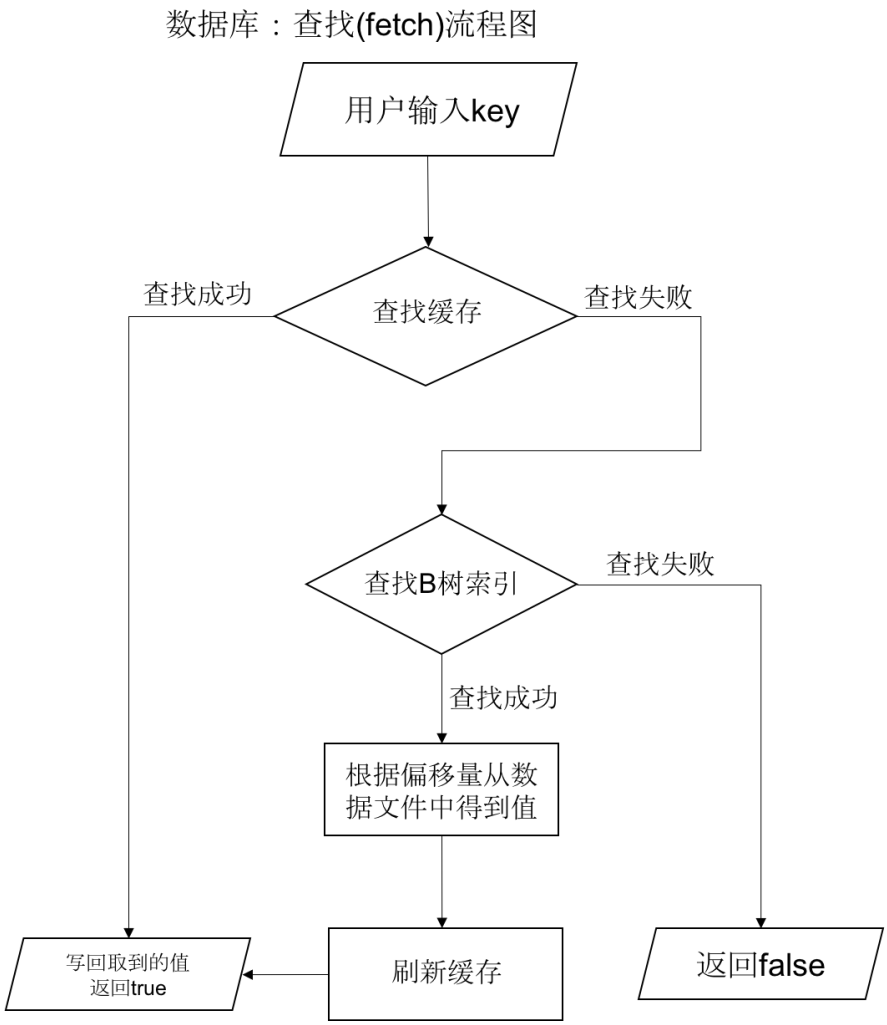
-关闭：

强制刷新缓存缓存，将B-树结构以一定的形式存入索引文件中，关闭文件。

-刷新缓存

当缓存中的元素将要超过缓存的容量时，利用LRU算法驱逐缓存当中最远使用的元素，根据该元素的dirty位判断是否将该元素写回数据文件中。

当从缓存中查找某一元素时，若该元素不存在，则从数据文件中读取。并且利用LRU算法将该元素放置在最先能够查找到的位置。



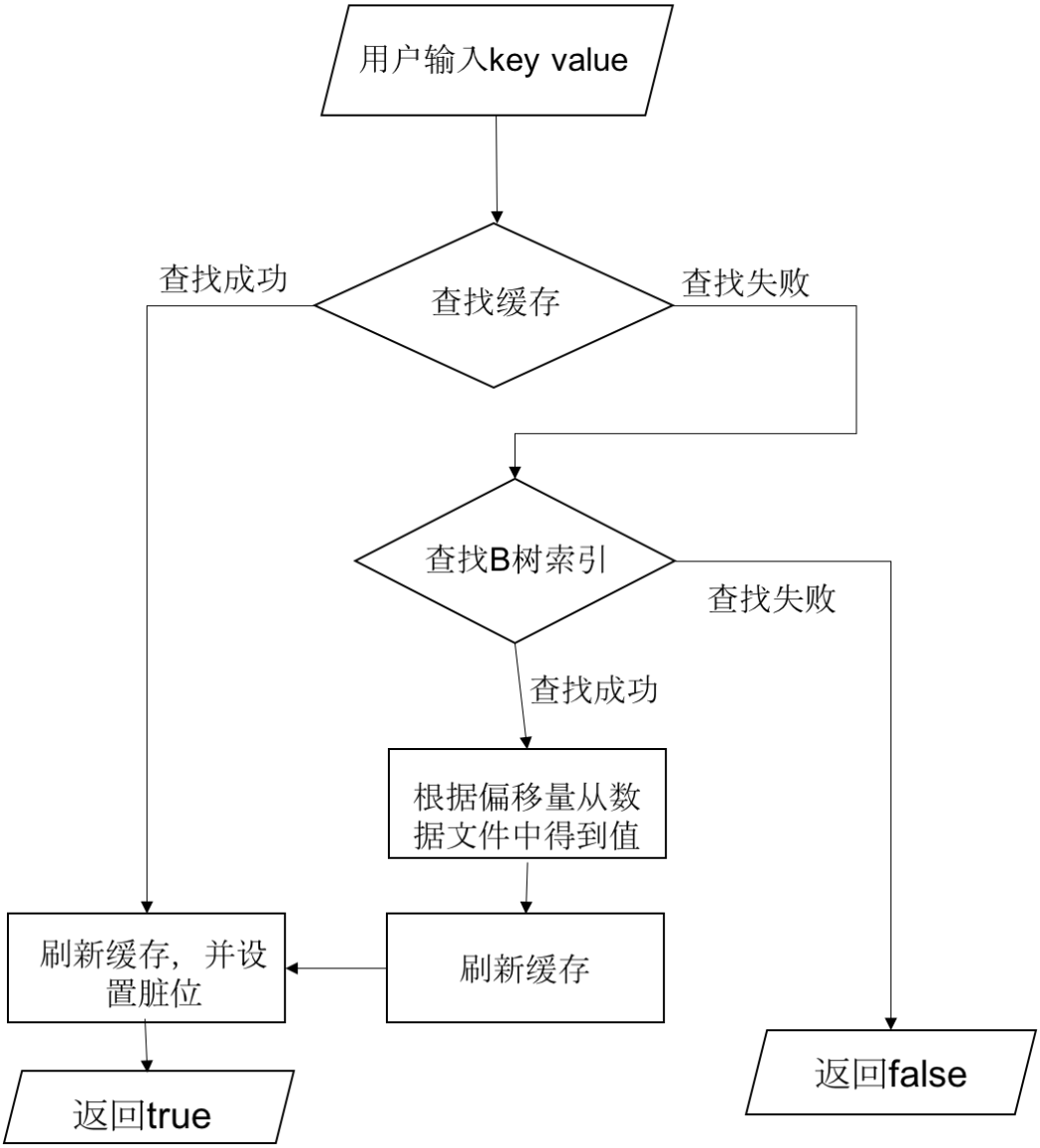
-查找元素：bool fetchRecord(const KEYTYPE& key, VALUETYPE& value);

查找缓存，若存在该记录，则刷新缓存，返回true， 并设置参数value的值；

否则，查找B-树，找到该元素的偏移量，如果偏移量不存在，则返回false；

从数据文件中取出相应的记录，刷新缓存，返回true，并设置参数value的值

数据库：修改(modify)流程图



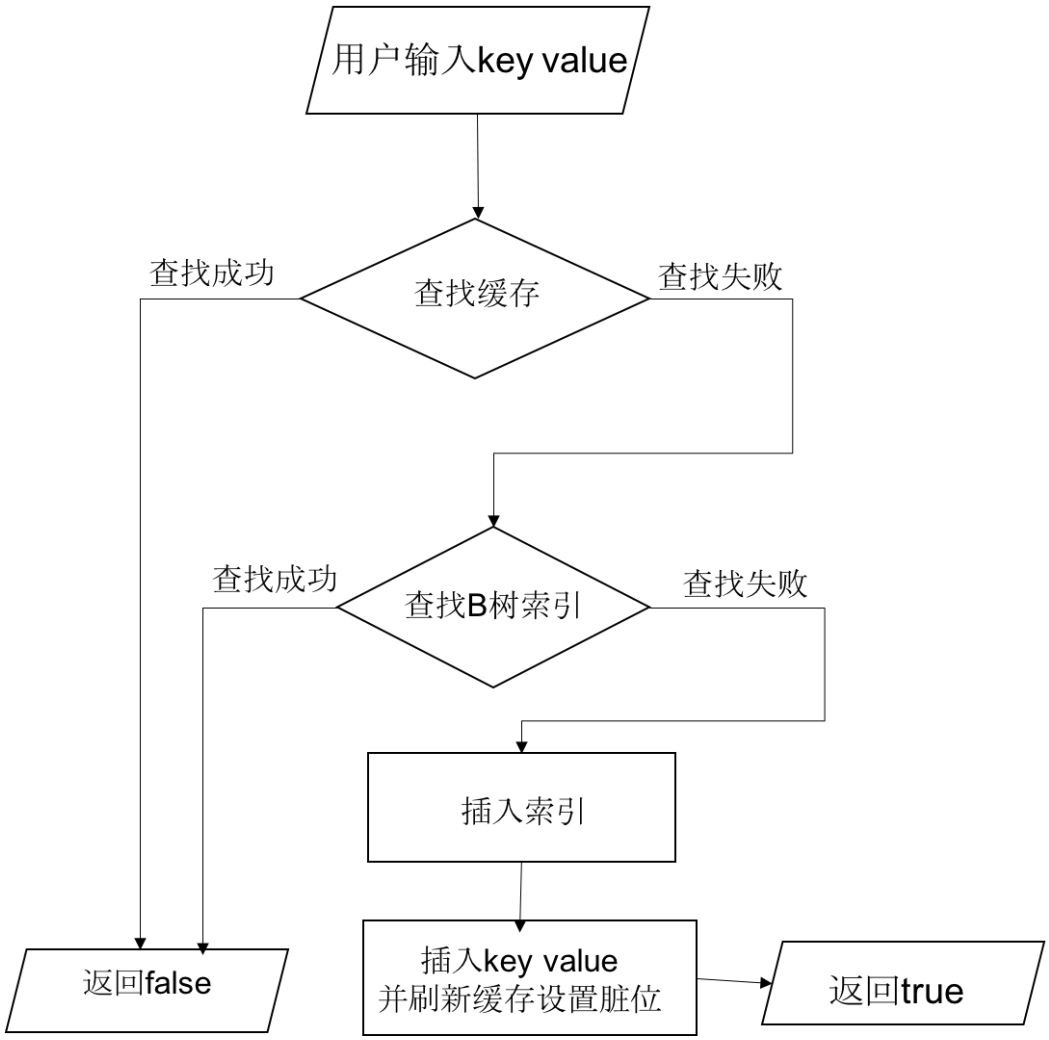
-修改元素：bool modify(const KEYTYPE& key, const VALUETYPE& value);

查找缓存，若存在该记录，则修改缓存中该记录的值，并将dirty位置为true， 返回true；

否则，查找B-树，找到该元素的偏移量，如果偏移量不存在，则返回false；

从数据库中取出相应的记录，刷新缓存，将缓存中的这条记录修改为新值，将dirty位置为true，返回true

数据库：插入(insert)流程图



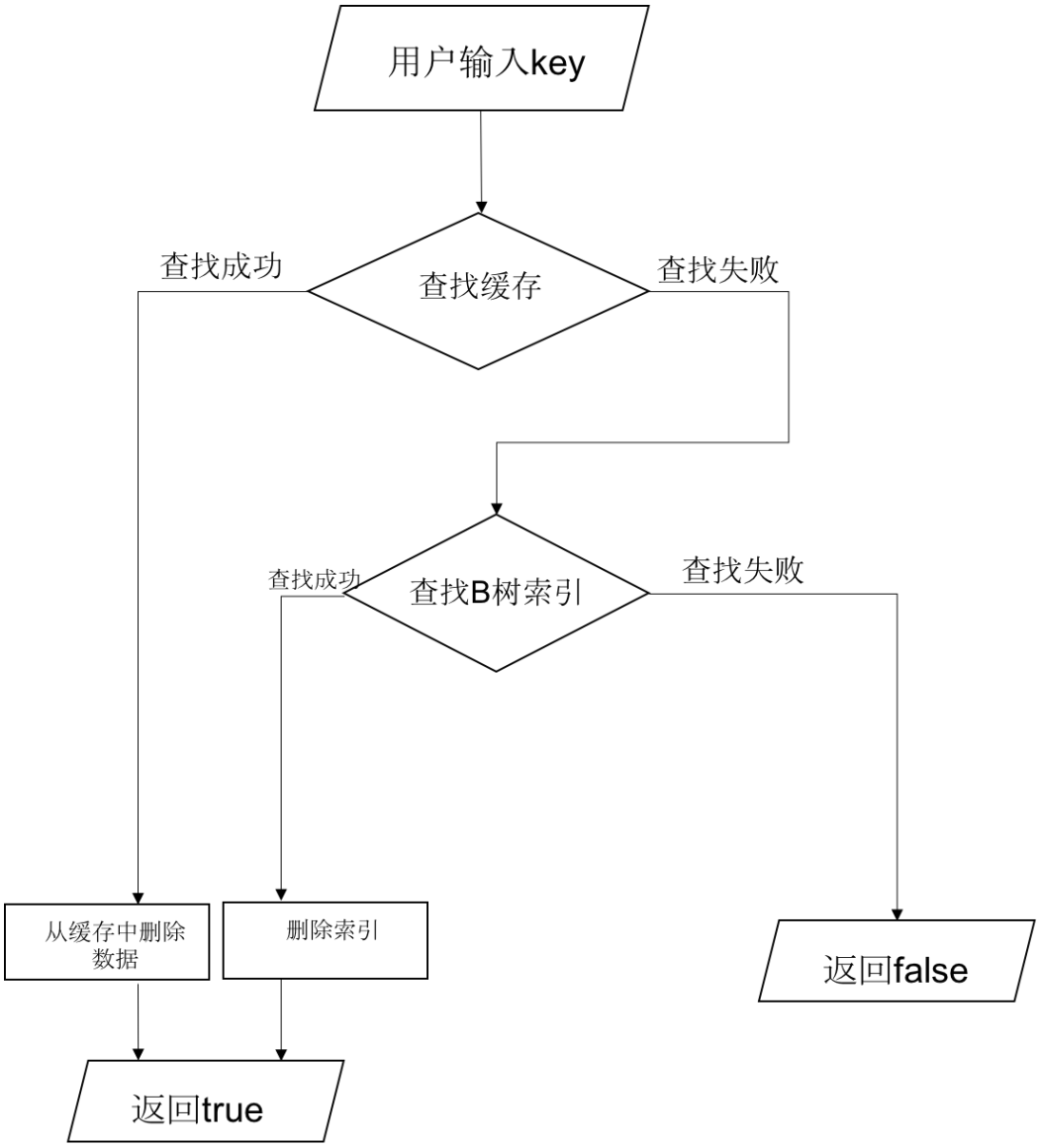
-插入元素：bool insertRecord(const KEYTYPE& key, const VALUETYPE& value);

查找缓存，若存在该记录，则返回false；

否则，查找B-树，找到该元素的偏移量，如果偏移量存在，则返回false；

将记录插入缓存中，并将dirty位置为true，返回true

数据库：删除(delete)流程图



-删除元素：bool insertRecord(const KEYTYPE& key, const VALUETYPE& value);

查找B-树，找到该元素的偏移量，如果偏移量不存在，则返回false；

查找缓存，若缓存中存在该元素的记录，则直接删除该记录

-DB.cpp:

数据库的主体部分之一，主要为缓存以及数据库算法的实现

-BTree.cpp:

数据库的主体部分之一，B-树索引文件类，实现根据索引找到数据在文件中的偏移量，以及索引的增删改查

四、测试以及分析

1. 正确性测试

1.1 100万条数据随机插入，插入后立刻查询，如果和插入的value不同则输出打印在控制台，查询后立即删除，删除后进行同一个key的查询，如果查询非空，则输出在控制台。key和value都用随机工具生成.

1.2 建立数据库，执行老师提供的ppt上面的操作：

(1)向数据库写 nrec 条记录。

(2)通过关键字读回 nrec 条记录。

(3)执行下面的循环 nrec×5 次。

a) 随机读一条记录。

b) 每循环 37 次，随机删除一条记录。

c) 每循环 11 次，随机添加一条记录并读取这条记录。

d) 每循环 17 次，随机替换一条记录为新记录。在连续两次替换中，一次用同样大小的记录替换，一次用比以前更长的 记录替换。

(4) 将此进程写的所有记录删除。每删除一条记录，随机地寻找 10 条记录。

1.3 对数据库进行了随机测试，此测试主要为人工测试，设计出多个测试数据保证代码的每一个片段都能够争取地运行

以上三个测试当中，控制台都无输出异常，说明数据库有很大概率是正确的。

2. 性能测试

2.1 数据量对性能影响

建立数据库，执行老师提供的的ppt上面的操作：

(1) 向数据库写 nrec 条记录。

(2) 通过关键字读回 nrec 条记录。

(3) 执行下面的循环 nrec×5 次。

(a) 随机读一条记录。

(b) 每循环 37 次，随机删除一条记录。

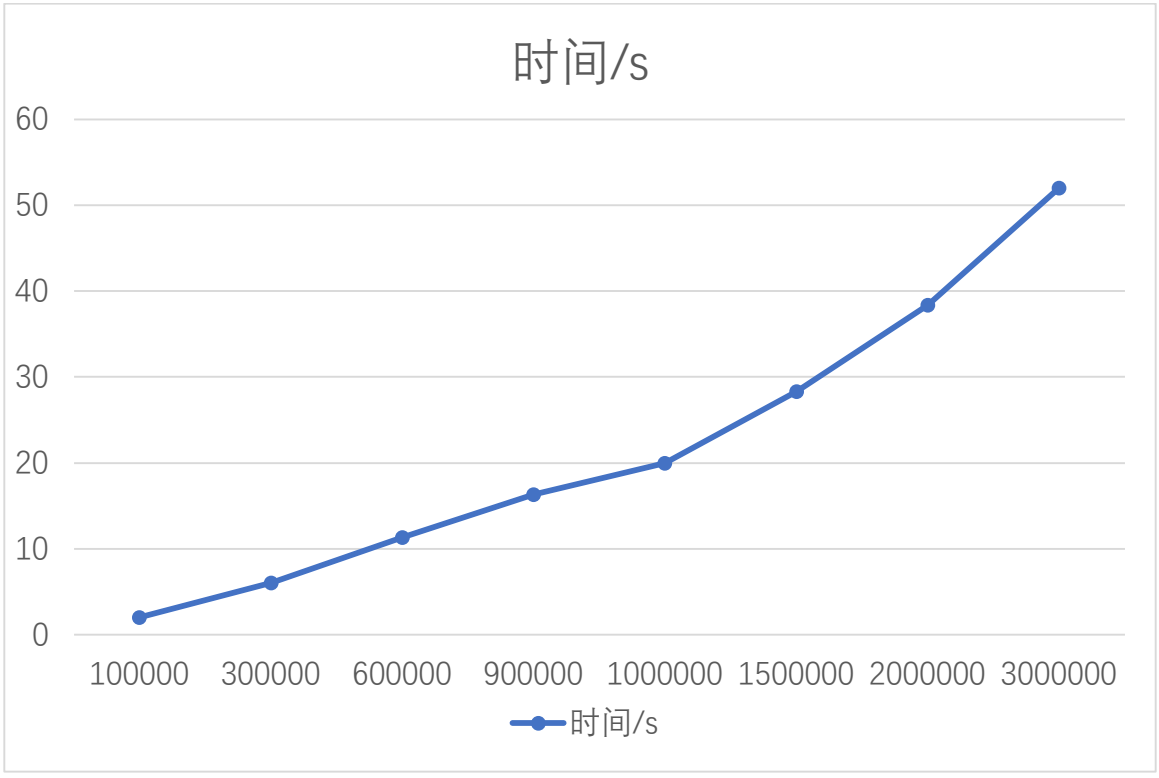
(c) 每循环 11 次，随机添加一条记录并读取这条记录。

(d) 每循环 17 次，随机替换一条记录为新记录。在连续两次替换中，一次用同样大小的记录替换，一次用比以前更长的 记录替换。

(4) 将此进程写的所有记录删除。每删除一条记录，随机地寻找 10 条记录。

测试结果如下图所示：

数据分别为100000, 300000, 600000, 900000, 1000000, 1500000, 2000000条数据，测试结果如下图所示：



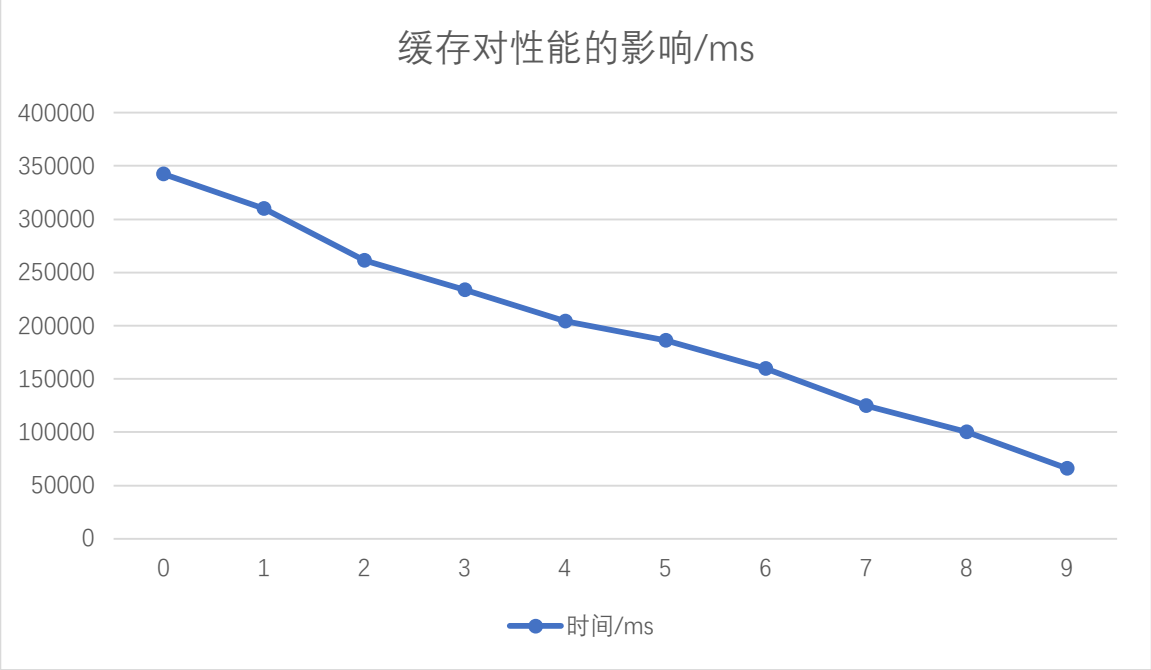
操作所用的时间基本上与数据量呈线性关系。

2.2 缓存大小对性能影响

测试方法：按顺序插入10条数据，然后随机读取10条数据中的任意一条（随机数生成key）一百次，重复循环nrec/10次。

结果如下图表所示：

大小	0	1	2	3	4	5	6	7	8	9
时间/ms	342541	310312	261074	233436	204267	186117	159667	124730	100202	65806



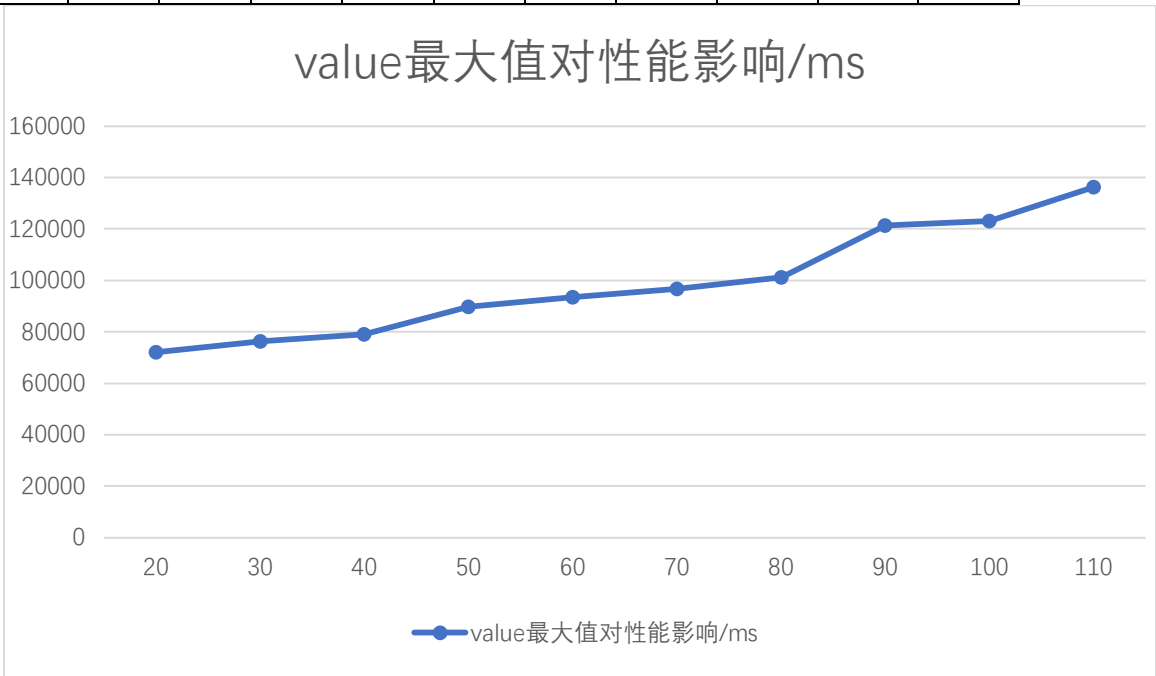
数据的结果符合初期预测，该测试过程模拟了用户查询数据的大部分情况，一定意义下具有普遍性。从此次测试能够看出缓存对于数据库减少I/O操作具有一定的效果，同时LRU算法在此种情况下能够大大地优化数据库的查询操作。

2.3 value最大值对性能影响

本测试采用利用课程项目介绍PPT上面的测试方法，配置为cache大小为2，数据量为400W。

结果如下图所示：

value最大长度(字节)	20	30	40	50	60	70	80	90	100	110
时间/ms	72163	76352	78965	89867	93576	96760	101223	121432	123132	136324



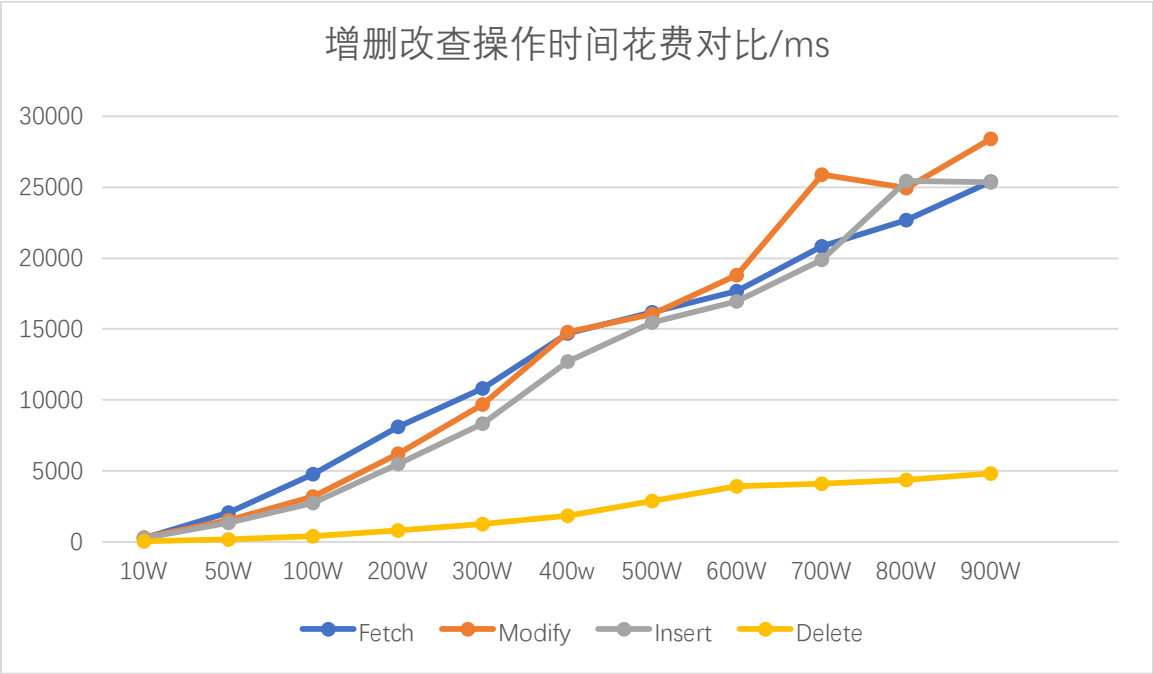
测试出的数据基本符合预期。可以看出value的最大长度越大，数据库的性能越低，这与可能文件I/O的操作中fseek函数有关系。所以在使用此数据库的时候，若不必要，尽量的不增加value的值。

2.4 增删改查操作时间花费对比

测试结果如下图表所示：

操作 / 数据量	查找	修改	插入	删除
10W	284	303	266	32
50W	2071	1534	1375	186
100W	4804	3193	2779	393
200W	8128	6215	5519	827
300W	10839	9702	8355	1272

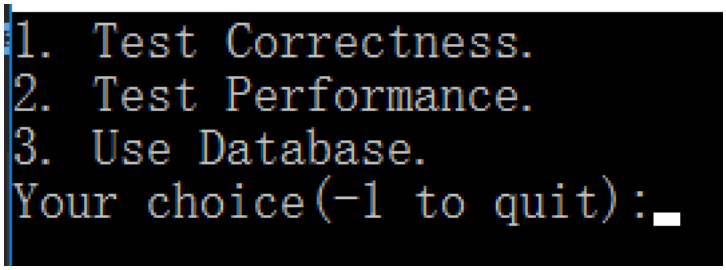
400W	14686	14794	12741	1857
500W	16175	16070	15470	2886
600W	17693	18805	16969	3928
700W	20858	25905	19880	4111
800W	22691	24941	25426	4404
900W	25382	28427	25348	4854



测试出的数据基本符合预期。虚拟机条件不是很稳定，测试的出的数据可能存在误差，当数据量增加时，操作花费的时间反而下降。从测试的数据以及图中可以看出，对数据的查询、插入、修改所花费的时间大值相当，而删除所花费的时间明显少于其他三个操作。这是因为删除操作未涉及到磁盘的I/O操作，完全在内存当中进行，所以操作所花费的时间少于其他三种操作。

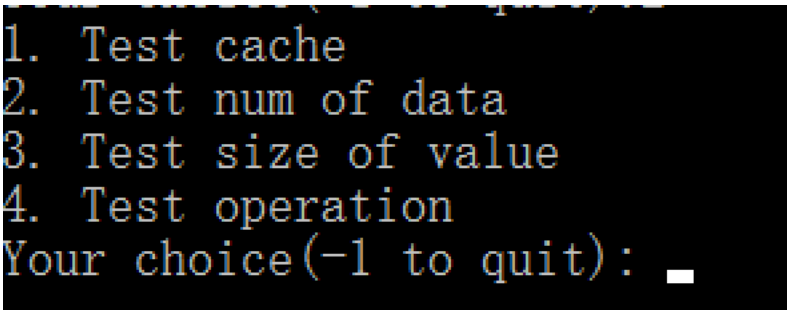
五、数据库界面以及使用方法

打开数据库，看到的是以下界面，用户可以通过三个选项来测试数据库的正确性，测试数据库的性能以及使用数据库的增删改查功能。



1. 若用户选择测试正确性，程序将自动调用测试正确性的函数

2. 若用户选择测试数据库的性能，则程序出现以下界面：



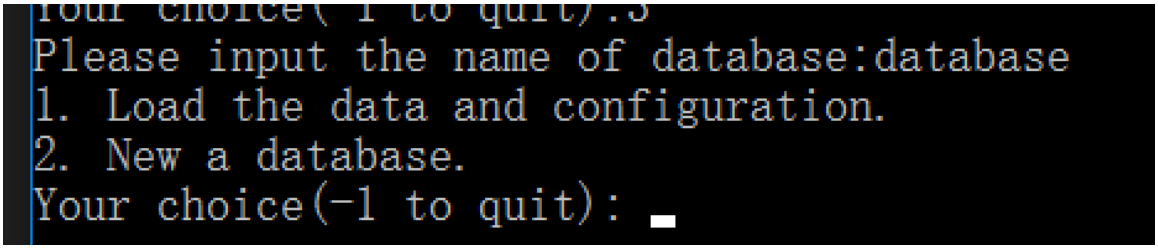
   2.1 测试cache的性能

   2.2 测试数据的数量对数据库性能的影响

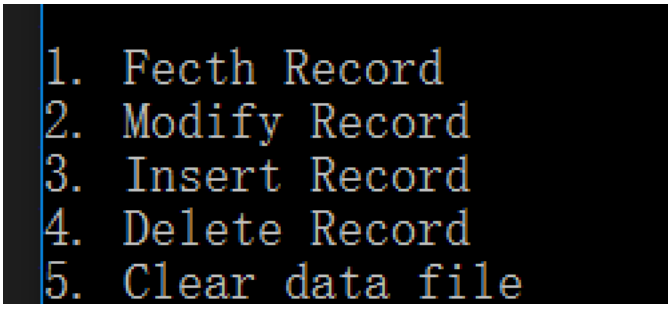
   2.3 测试record的最大长度对数据库性能的影响

   2.4 测试增删改查操作性能上的差别

3. 若用户选择使用数据库，则用户需要输入数据库的名字，选择载入数据库原有的配置或者新建一个数据库：



即可进入到操作数据库的界面。用户不仅进行增删改查的操作，还可以清空数据文件，清空数据文件后，用户需要退出，然后新建数据库，才能够达到清空数据库的操作。



六、总结

-该数据库具有不定长value，schema可拓展性的功能，同时利用了缓存大大地提高增加了数据库性能，用户可以自行定制数据库缓存的大小，以及value值的最大长度

-本数据库对数据的查找操作基本上是根据key获取相应的value，而对于功能较完善的数据库，可能还存在范围查找，排序等功能。该数据库没有实现此功能。由于B-树的查找深度不超过B+树，故简单的根据value查找key的功能，B-树的数据结构要比B+树好得多，也是此次数据库课程项目中我采用B-树的数据结构的原因；

-内存中的比较算法，采用了线性比较的方法，可以采取更加高效的方法实现比较算法，降低时间复杂度；

-本数据库对于缓存算法的设计还存在着不足之处，测试结果表明，该数据库缓存的算法对降低磁盘的I/O操作的次数还不够，所以可能缓存的算法还存在着很大的改进之处；

-此次课程项目的编码中，采用了面向对象与面向接口的编程，即接口与实现分离，有利于后期软件的维护。