三、各模块详细实现

3.1 Interpreter

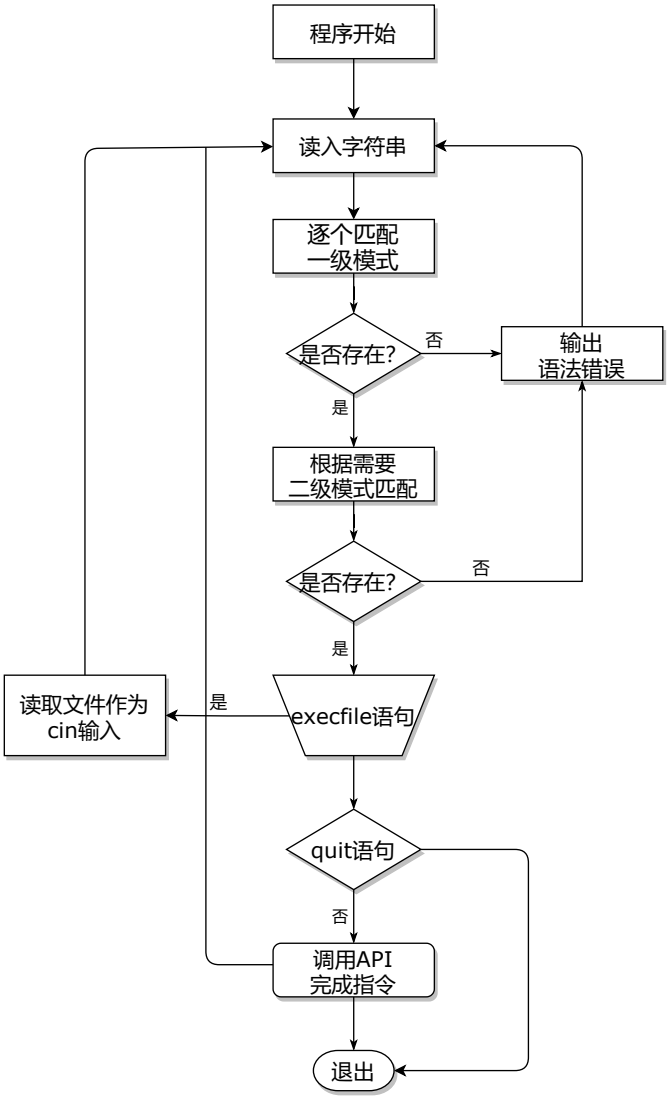
3.1.1、主要函数

Interpreter核心函数与辅助函数如下：

```
int onestep(std::string str, std::vector<std::regex> pattern,
instruction &Instruction);
//对用户输入的语句进行匹配并完成有效字符的分割并得到语句的类型
void loadPattern(std::vector<std::regex> &pattern);
//正则表达式的模式导入
int execfile(std::string filestr);
//专门用于语句execfile

std::vector<std::string> split(std::string str, std::string
pattern);
void strCleaner(std::string &str);
//辅助对于用户输入进行预处理和按某些字符划分
```

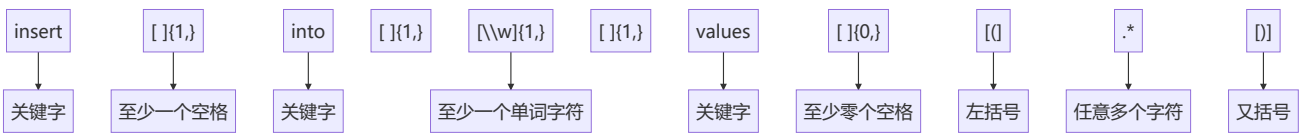
3.1.2 工作流程



3.1.3 实现细节

- 该Interpreter的核心是regex正则表达式的应用，下给出一例说明本模块如何采用正则表达式即本模块使用的pattern判断语句。

```
insert[ ]{1,}into[ ]{1,}([\\w]{1,})[ ]{1,}values[ ]{0,}[(](.*)[)]
```



- Interpreter的实现中采用二级译码的方式，先在一级的pattern中进行匹配，再根据匹配出的结果需要二级pattern匹配如上文中insert类语句的译码对于values内部语句是否符合语法要求在第二级译码中分割并匹配。下给出一个二级匹配例子：createTable语句中对每个字段单独判断

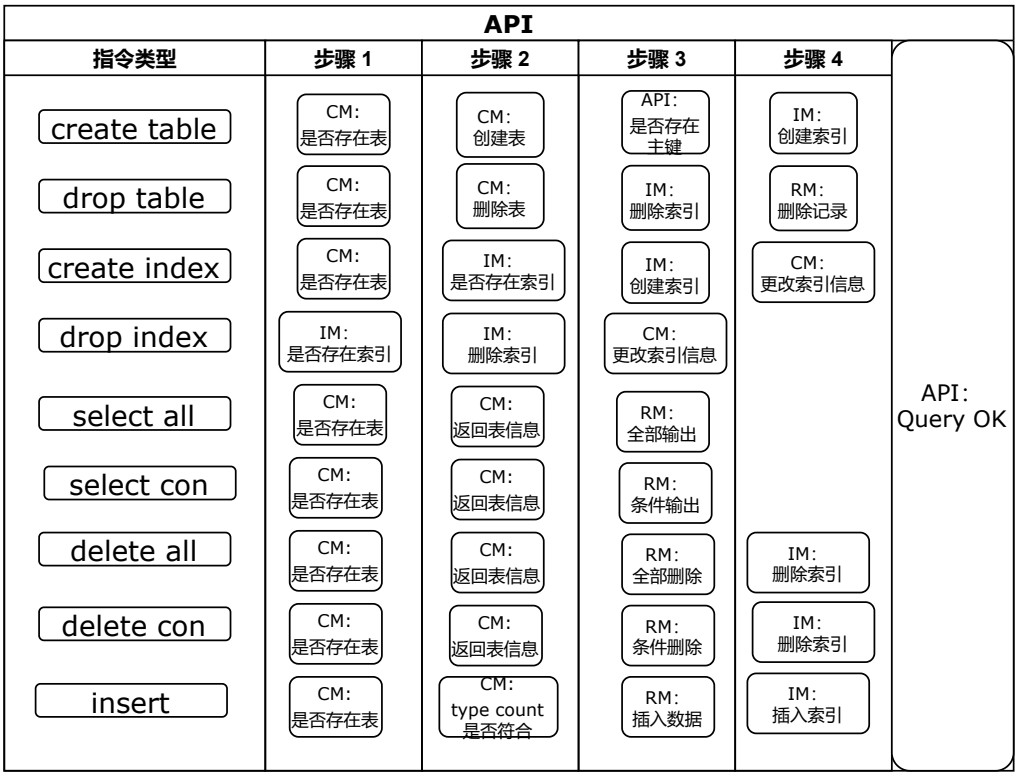
```
"[ ]{0,}(.*)[ ]{1,}(int)[ ]{0,}"//分割int
```

- 实际上传递给API的实际上只是得到的有效字符组成的vector和该指令对应的指令数，API需要再进行译码提取出下层模块可以直接使用的部分。
- 本程序的Interpreter设计与实验指导书有所不同，Interpreter不能调用CatologManager模块提供的接口所以诸如重复建表等判断在该模块是不会返回错误的，该模块只判断语法错误，并且从正确的语句中识别模式并提取有效字符。

3.2 API

3.2.1 流程图

API作为指令实现的中枢，其工作核心即为调用其他模块接口，故在这里只给出流程图与文字描述，接口详细定义参见其他下层模块。



con: 条件
CM: catalogManager
RM: recordManager
IM: indexManager

该图仅作阐述之用，故判断均假设向可以成功完成语句方向进行，出现错误的具体报错输出请见下层模块。

3.2.2、辅助函数与类简介

由于Interpreter实质上只给出了vector<string>和指令数，对于下层模块来说实质上还需要进一步译码，为此需要一些辅助函数和类，下面予以简介：

- condition类：此类一开始是为将select condition与delete condition的condition封装传入，后来也用作create table传入参数，实现类的复用。

```
class mycondition{
public:
    std::vector<std::string> col;
    mycondition project(std::string colName);
    //投影到某个col为index设计
    bool satify(std::string input,int i);//重载satify函数，判断是否满足
    bool satify(int input,int i);
    bool satify(float input,int i);
    std::vector<std::string> value;
    std::vector<int> op;  //0=  1<> 2<  3>  4<= 5>=
    std::vector<int> type;//0 int 1 string
};
```

- API::getins(instruction ins). 此函数即为进一步译码模块，即对于复杂指令来完成condtiion类的创建和赋值。具体来说为create table, select/delete condition, insert语句(即需要二级译码模块)，需要将那些列、插入元素的值封装进condition中传入下层模块。

3.3 CatalogManager

3.3.1 CatalogManager类

```
class CatalogManager{
public:
    bool createTable(std::string tableName, std::vector<std::string>
colName, std::vector<int> colType, std::vector<std::string>
isUnique,std::string primary);
    bool createIndex(std::string indexName, std::string
tableName,std::string colName);
    bool dropTable(std::string tableName);
    bool dropIndex(std::string indexName, std::string tableName,
std::string colName);
    bool selectCheck(std::string tableName, std::vector<std::string>
colName,std::vector<int> &colOrder,std::vector<std::string>
&colAll);//select or delete check
    bool insertCheck(std::string tableName, std::vector<int>
colType,std::vector<char>&isUnique,
std::vector<std::string>&indexName);
    bool selectall(std::string
tableName,std::vector<std::string>&colAll,std::vector<int>&colType);
//select delete all check
    bool deleteall(std::string tableName);
private:
    int existTableCreate(std::string tableName);
    void getTable(std::string tableName, std::vector<std::string>
&colNow,int block_id);//得到每一列的名字
    void getTable(std::string tableName, std::vector<int>
&typeNow,int block_id);//得到每一列所存储的类型
    void getTable(std::string tableName, std::vector<char>
&isUnique, std::vector<std::string> &indexName,int block_id); //得到
每一列是否unique
    int existTable(std::string tableName);
```

```
    int existIndex(std::string indexName, std::string tableName,
std::string colName,int &table_id);
};
```

3.3.2 成员函数说明

```
bool createTable(string tableName, vector<string> colName,
vector<int> colType, vector<string> isUnique,string primary);
```

- 传入数据表名，列名，列类型，是否唯一、主键，传回是否建表成功

```
bool createIndex(string indexName, string tableName,string colName);
```

- 传入索引名，数据表名，列名，传回是否建索引成功

```
bool dropTable(string tableName);
```

- 传入数据表名，传回是否删除表成功

```
bool dropIndex(string indexName, string tableName, string colName);
```

- 传入索引名，数据表名，列名，传回是否删除索引成功

```
bool selectCheck(string tableName, vector<string>
colName,vector<int> &colOrder,vector<string> &colAll);
//select or delete check
```

- 传入条件删除中的条件列名，返回这些列是否存在，次序以及所有列名供输出之用。

```
bool insertCheck(string tableName, vector<int>
colType,vector<char>&isUnique, vector<string>&indexName);
```

传入插入类型供检查，传出是否允许重复，索引名

```
bool selectall(string tableName,
vector<string>&colAll,vector<int>&colType);
```

- 传出列名，列类型

```
void getTable(string tableName, vector<string> &colNow,int
block_id);
```

- 得到每一列的名字

```
void getTable(string tableName, vector<int> &typeNow,int block_id);
```

- 得到每一列所存储的类型

```
void getTable(string tableName, vector<char> &isUnique,
vector<string> &indexName,int block_id);
```

- 得到每一列是否unique

```
int existTable(string tableName);
```

- 判断数据表是否存在

```
int existIndex(string indexName, string tableName, string
colName,int &table_id);
```

- 判断索引是否存在

3.3.3 数据表存储结构

```
    /*
    format: len(4)放总字符数 + len(tablename) tablename
    {
        len(colname)colname+2bytes+1byte(具体含义见下)
    }//遍历所有column

    2字节存储类型
 case: +n char(n)  0 int    -1 float
       '1' 'n'    '0' '1'  '0'  '0'

    1字节存储主键/唯一/索引
    unique primary  index
    'u'    'p'      'i'     'n'
```

```
        若为'i'则+len(indexname)indexname
    */
```

3.3.4 评论

事实上，该模块的实现实在算不上好，最好可以实现一个Attibute类来专门记录数据表的信息，每次调用CataologManager时直接读取Block初始化Attibute数据表，这样上述函数的实现就可以避免冗长的根据数据表的存储结构来访问和修改，而变为访问一个类，这样函数层次也会更加清晰。

3.4 RecordManager

3.4.1 RecordManager类

```
class RecordManager{
public:
    bool createTable(std::string tableName,std::vector<int>
colType);
    bool dropTable(std::string tableName);
    record insertRecord(std::string tableName,
std::vector<std::string> tableValue, std::vector<int>
colType,std::vector<char> isUnique);
    int deleteRecord(std::string tableName, std::vector<int>
block_id, std::vector<int> offset, std::vector<int> colType); //利用
索引删除
    int deleteRecord(std::string tableName, mycondition condition,
std::vector<int> colOrder, std::vector<int> colType);   //硬删
    int deleteAll(std::string tableName, std::vector<int>
colType);//直接重建表格即可
    int select(std::string tableName, std::vector<int> colType,
std::vector<std::string> colName, int mode, mycondition condition,
std::vector<int> colOrder);
    //可能需要更多的mode来支持insert 无B+树Unique primary key 重复的判断
利用不同mode来完成代码的重用
private:
    int recordLength(std::vector<int> colType);
```

```
    std::vector<std::string> getValue(char a[], std::vector<int>
colType);
    int deleteone(char block[], int offset,int length);
    bool satis(std::vector<std::string> values, mycondition
condition, std::vector<int> colOrder);
    bool initialHead(std::string tableName, int &head, int
&sumblock, std::vector<int> &blockRecord); //bool:0 empty
};
```

3.4.2 主要成员函数说明

```
bool createTable(string tableName,vector<int> colType);
```

- 根据type得到一条记录长度,初始化record/tableName

```
bool dropTable(string tableName);
```

- 删除record/tableName

```
int deleteAll(string tableName, vector<int> colType);
```

- 直接重建表格即可

```
record insertRecord(string tableName, vector<string> tableValue,
vector<int> colType,vector<char> isUnique);
```

- 传入需要插入的值，类型，是否允许重复等来完成插入记录操作

```
int deleteRecord(string tableName, vector<int> block_id, vector<int>
offset, vector<int> colType);
```

- 利用索引删除

```
int deleteRecord(string tableName, mycondition condition,
vector<int> colOrder, vector<int> colType);
```

- 遍历根据condition是否满足删除

```
int select(string tableName, vector<int> colType, vector<string>
colName, int mode, mycondition condition, vector<int> colOrder);
```

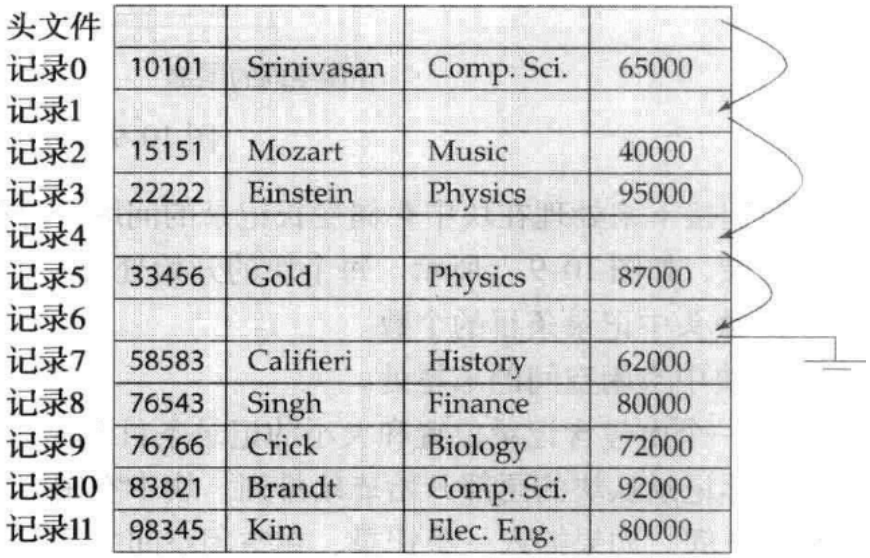
- mode 0: select all
- mode 1: select condition

```
 int deleteone(char block[], int offset,int length);
```

- 删除给定字符数组中offset处长为length记录

3.4.3 记录存储结构

- 由于本程序存储的都是定长记录，所以采取课本中“空闲链表”的存储结构，这样可以避免频繁插入删除下文件空间的浪费，具体存储方式采用**课本图片**说明。



头文件				
记录0	10101	Srinivasan	Comp. Sci.	65000
记录1				
记录2	15151	Mozart	Music	40000
记录3	22222	Einstein	Physics	95000
记录4				
记录5	33456	Gold	Physics	87000
记录6				
记录7	58583	Califieri	History	62000
记录8	76543	Singh	Finance	80000
记录9	76766	Crick	Biology	72000
记录10	83821	Brandt	Comp. Sci.	92000
记录11	98345	Kim	Elec. Eng.	80000

3.4.4 实现细节

- 记录传输中全部以string存储，再加上type才可以真正判断该列对应多少字节以及读取出的字节如何转变回真实的值。
- 空闲链表的维护与链表基本上无异，区别在于删除记录相当于链表的插入，指针在文件中实际上以偏移代替。

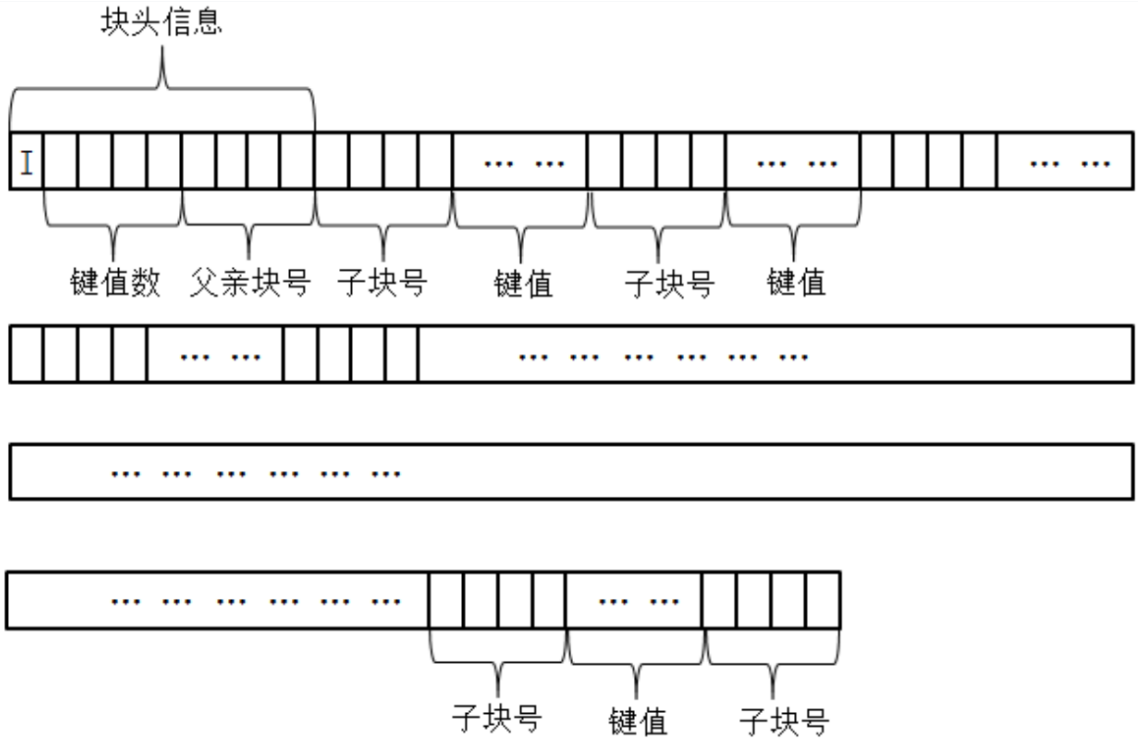
3.5 IndexManager

3.5.1 IndexManager类

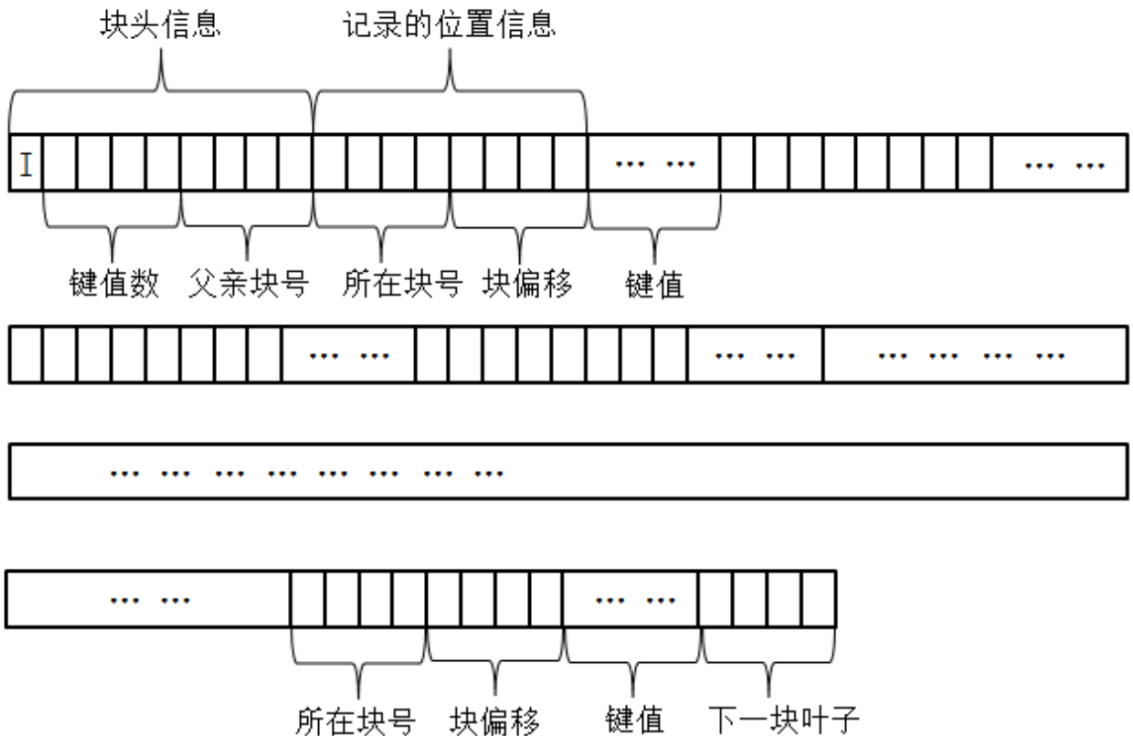
3.5.2 BPlusTree 存储结点结构

本程序采取存储方式以及图片均来源于助教所分享的IndexManager设计报告

- 中间块存储格式



- 叶子节点存储格式



3.6 BufferManager

3.6.1 实现功能

BufferManager模块主要提供对于以Block(4KB)为单位的与硬盘交互方法，并实现LRU替换算法。更高的模块通过调用BufferManager模块提供的接口获得指定目录下给定的某一块内容，并在修改之后通过setdirty, flush让BufferManager写回硬盘。

3.6.2 BufferManager类

```
class BufferManager{
public:
    BufferManager();
    Block Page[BLOCK_NUM];
    int inputBlock(std::string content, std::string fileName, int
block_id);
    //核心函数：将上层模块所需要的指定块读入，返回读入到的Page的id
    int findBlock(std::string content, std::string fileName,int
block_id);//找到一个要写到指定目录的块
    int output(int buffer_id, std::string path, int block_id);
    void setdirty(int i) { dirty.push_back(i);};
```

```
    void clear(std::string content, std::string fileName, int
block_id);
    int flush();
    //核心函数：将所有脏块写回硬盘
private:
    LRU myLRU; //用于缓冲区替换策略
    std::vector<int> dirty;//记录所有脏块
};
```

其中，Block类为集成了一个4KB字符串的类，其定义如下：

```
class Block{//用block来表述一个块
public:
    void setpin(int i) { pin = i; };
    int getpin() { return pin; };
    char pool[BLOCK_SIZE];//实际上到该字符串数组
    void prtpool(); //调试
    bool exist(std::string content, std::string fileName, int
block_id);
    void setFile(std::string content, std::string fileName,int
block_id);
    bool inputFile(std::string content, std::string fileName, int
block_id);
    //核心函数：读入文件
    bool outputFile(std::string content, std::string fileName, int
block_id);
    //核心函数：写回文件
    bool outputFile();
private:
    ......
};
```

另外，具体给出哪一个Page_id由LRU模块决定：

```
class LRU{
public:
    LRU();
    int insert(std::string path);
    //核心函数：判断需要读入的块是否存在，存在则返回id，否则读入/替换
    int exist(std::string path) { return (myhash.find(path) ==
myhash.end()) ? -1 : *myhash.find(path)->second; };
private:
......
};
```

3.6.3 实现细节

- 读入和写回文件实质上是利用C语言文件操作相关函数，具体来说是fopen, fread, fwrite, fseek等。文件的存储以字节为单位。
- LRU算法的实现采用C++STL中的list和map，通过对每一个路径+块号映射到list中的元素，再通过对链表本身的插入和删除来实现前移，替换功能。需要注意的是，map实现底层为红黑树，所以复杂度为O(n)。

附录

```
/*********************************************
#程序中所用正则表达式的pattern与instruction.number
#0-10为1级译码
#11-17 createtable列识别
#18-23 condition op识别
*********************************************/
    temp.assign("quit"); //0
    temp.assign("create[ ]{1,}table[ ]{1,}([\\w]{1,})[ ]{0,}[(](.*)
[)]"); //1
    temp.assign("drop[ ]{1,}table[ ]{1,}([\\w]{1,})"); //2
    temp.assign("create[ ]{1,}index[ ]{1,}([\\w]{1,})[ ]{1,}on[ ]
{1,}([\\w]{1,})[ (]{0,}([\\w]{1,})[ ]{0,}[)]"); //3
    temp.assign("drop[ ]{1,}index[ ]{1,}([\\w]{1,})"); //4
    temp.assign("select[ ]{1,}[*][ ]{1,}from[ ]{1,}([\\w]{1,})");
//5
```

```
    temp.assign("select[ ]{1,}[*][ ]{1,}from[ ]{1,}([\\w]{1,})[ ]
{1,}where[ ]{1,}(.*)"); //6
    temp.assign("insert[ ]{1,}into[ ]{1,}([\\w]{1,})[ ]{1,}values[ ]
{0,}[(](.*)[)]"); //7
    temp.assign("delete[ ]{1,}from[ ]{1,}([\\w]{1,})"); //8
    temp.assign("delete[ ]{1,}from[ ]{1,}([\\w]{1,})[ ]{1,}where[ ]
{1,}(.*)"); //9
    temp.assign("execfile[ ]{1,}(.*)"); //10
    //create table
    temp.assign("[ ]{0,}(.*)[ ]{1,}(int)[ ]{0,}"); //11
    temp.assign("[ ]{0,}(.*)[ ]{1,}(float)[ ]{0,}"); //12
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{1,}(char)[ ]{0,}[(][ ]{0,}
([\\d]{1,})[ ]{0,}[)][ ]{0,}"); //13
    temp.assign("[ ]{0,}(.*)[ ]{1,}(int)[ ]{1,}(unique)[ ]{0,}");
//14
    temp.assign("[ ]{0,}(.*)[ ]{1,}(float)[ ]{1,}(unique)[ ]{0,}");
//15
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{1,}(char)[ ]{0,}[(][ ]{0,}
([\\d]{1,})[ ]{0,}[)][ ]{1,}(unique)[ ]{0,}"); //16
    temp.assign("[ ]{0,}(primary key)[ ]{0,}[(](.*)[)][ ]{0,}");
//17
    //op and
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{0,}(=)[ ]{0,}(.*)"); //18

    temp.assign("[ ]{0,}([\\w']{1,})[ ]{0,}(>)[ ]{0,}([\\d]{1,})");
//19
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{0,}(<)[ ]{0,}([\\d]{1,})");
//20
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{0,}(>=)[ ]{0,}([\\d]{1,})");
//21
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{0,}(<=)[ ]{0,}([\\d]+)");
//22
    temp.assign("[ ]{0,}([\\w']{1,})[ ]{0,}(<>)[ ]{0,}([\\d]+)");
//23
```