# Buffer Manager说明文档

作者：2016级软件工程

曾辉

## 一、Buffer Manager 主要功能

Buffer Manager负责缓冲区的管理，主要功能有：

1. 根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件；
2. 实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换；
3. 记录缓冲区中各页的状态，如是否被修改过等；
4. 提供缓冲区页的pin功能，及锁定缓冲区的页，不允许替换出去。

为提高磁盘I/O操作的效率，缓冲区与文件系统交互的单位是块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为4KB或8KB，本项目中使用4KB为。

## 二、模块总体设计思路

记录管理模块（Record Manager）和索引管理模块（Index Manager）向缓冲区管理申请所要的数据，缓冲区管理器首先在缓冲区中查看数据是否存在，若存在，直接返回，否则，从磁盘中将数据读入缓冲区，然后返回。

最近最少使用(LRU)算法：用一个链表记录所有的使用中缓冲块，每次访问到一个缓冲块就将它插入到链表的头部，这样链表尾的缓冲块就是最近最少使用的块，在需要的时候就可以替换出去。

内存块交换的具体实现：用一个垃圾块链记录所有未被使用的缓冲块，当文件被关闭的时候，将文件头所链接的所有缓冲块链接到垃圾块链表。将所有正在使用的缓冲块链接到一个LRU链表，将所有文件头，链接到一个LRU链表。当请求某个文件的某个块时，首先判断该文件是否被打开，如果被打开则从文件头所链接的缓冲块中查找该块，如果找到，则直接返回，如果没有找到，则需要加载一个新的文件块。如果没有被打开，则需要判断，当前能否重新打开一个文件，如果可以则重新打开一个文件，加载一个新的文件块。如果不可以则从文件头LRU链表中，通过LRU算法，替换一个文件头。

加载新的文件块时首先从垃圾链表中，寻找可用的缓冲块。如果有缓冲块，则使用该缓冲块，如果没有，则判断能否重新打开生成一个缓冲块，若可以生成缓冲块后返回，如果不可以，从缓冲块LRU链表中根据LRU算法选择一个缓冲块进行替换，返回。

## 三、具体实现

1. **宏定义**

#define BLOCK\_LEN 4096 // the size of one block

#define MAX\_FILE\_ACTIVE 5 // limit the active files in the buffer

#define MAX\_BLOCK 40 // the max number of the blocks

1. **数据结构定义**
   1. **文件头结构体(class fileInfo)**

包含文件的基本信息：类型（数据文件或索引文件）、文件名、指向下一个文件的指针、文件所指向的第一个块，锁信息。

class fileInfo {

bool type; // 0-> data file， 1 -> index file

string file\_name; // the name of the file

bool lock; // prevent the block from replacing

fileInfo \*next; // the pointer points to the next file

blockInfo \*first\_block; // point to the first block within the file

friend class Buffer;

};

* 1. **块信息结构体(class blockInfo)**

包含块的基本信息：块号、脏位、指向下一个块的指针、存放信息的字符型数组、年龄（用于LRU算法）、锁。

class blockInfo {

int32\_t block\_num; // the block number of the block which indicate it when it was built

bool lock; // prevent the block from replacing

fileInfo\* file; // the pointer point to the file, which the block belong to

friend class Buffer;

private:

bool dirty; // 0 -> false, 1 -> indicate dirty. write back

blockInfo\* next; // the pointer point to next block

int32\_t char\_num; // the number of chars int the block

char \*cBlock; // the array space for storing the records in the block in buffer

};

1. **类定义**

class Buffer {

private:

fileInfo \* file\_handle; // 文件头指针，指向当前打开的文件头

blockInfo \* block\_handle; // 垃圾链指针，指向未被使用的缓冲块

blockList<blockInfo> \* LRU\_block\_list; // 缓冲块LRU链表，便于LRU算法的实现

blockList<fileInfo> \* LRU\_file\_list; // 文件头LRU链表，便于LRU算法的实现

int32\_t total\_block; // 当前建立的所有缓冲块

int32\_t total\_file; // 当前打开的所有文件

void load\_data(string file\_name, bool file\_type, blockInfo\* block\_node);

void write\_back(string file\_name, bool file\_type, blockInfo\* block\_node);

void close\_file(fileInfo\* file\_node);

public:

fileInfo\* get\_file\_info(string file\_name, bool file\_type);

blockInfo\* get\_file\_block(string file\_name, bool file\_type, int block\_num);

void set\_lock(fileInfo \*file\_node, bool lock);

void set\_lock(blockInfo \*block\_node, bool lock);

void set\_dirty(blockInfo \*block\_node, bool dirty);

char\* get\_content(blockInfo \*block\_node);

void remove\_file(string file\_name, bool file\_type);

Buffer();

~Buffer();

};

1. **主要函数及其功能描述**
   1. char\* Buffer::get\_content(blockInfo\* block\_node)

/\*

function:

根据块指针，获得指向内存空间的int8\_t指针

@param:

block\_node: 块指针

\*/

具体实现：

返回block\_node->cBlock;

* 1. void Buffer::remove\_file(string file\_name, bool file\_type)

/\*\*

\* **\brief** 根据给定的文件名，删除文件

\* **\param** file\_name 文件名

\* **\param** file\_type 文件类型

\*/

具体实现：

1. 调用fileInfo\* Buffer::get\_file\_info(string file\_name, bool file\_type)获取待关闭文件的文件头
2. 调用void Buffer::close\_file(fileInfo\* file\_node)，关闭相应文件
   1. blockInfo\* Buffer::get\_file\_block(string file\_name, bool file\_type, int block\_num)

/\*

function:

根据文件名，文件类型查找该文件是否存在内存，返回块指针

@param:

file\_Name: 文件名

fileType: 文件类型 0->data, 1->index

block\_num: 块号, 编号从0开始

\*/

1. 根据文件名，文件类型查找该文件是否在内存
2. 如果是，根据文件的块号，从内存中查询该块，
3. 如该块已经在内存，返回该块的指针
4. 如果该块没有在内存，判断垃圾链表中是否有空余的块，
5. 如有从中找到一块（链表的头），将该块的blockNUM设置为参数blockNUM
6. 如果没有，判断现在内存中的块数是否已经达到了最大块数限制
7. 如果没有，为其分配新分配一块，并链接到对应的文件头所指的链表的结尾;
8. 如果已达到，使用LRU算法，找到一个替换块，按照给定的要求进行初始化，并将其链接到指定文件块链表的结尾。
9. 如果文件没有在内存，调用fileInfo\* Buffer::get\_file\_info(string file\_name, bool file\_type)来为文件分配一个头指针。回到步骤 i ，将该块按要求初始化。
   1. void Buffer::set\_dirty(blockInfo\* block\_node, bool dirty)

/\*

function:

根据块指针，设定该块是否被修改

@param:

block\_node: 块指针

dirty: 1->dirty, 0->clean

\*/

具体实现：

将diry属性设置为相应值

* 1. void Buffer::write\_back(string file\_name, bool file\_type, blockInfo\* block\_node)

/\*

function:

将block中的脏数据写回磁盘

@param:

block\_node: 存放数据的内存块

file\_type: 文件类型 0->record, 1->index

file\_name: 磁盘文件名

\*/

具体实现：

1. 根据文件类型与文件名找到对应的磁盘文件，打开文件
2. 根据缓冲块中的数据块号写入到磁盘中的文件块中
   1. void Buffer::close\_file(fileInfo\* file\_node)

/\*

function:

关闭文件

将脏块数据写回磁盘

内存块链入垃圾链表

从LRU\_block\_list与LRU\_file\_list中剔除相应值

@param

file\_node: 待关闭的文件头

\*/

具体实现：

1. 将文件头所链接的文件块中有脏数据的块调用void Buffer::write\_back(string file\_name, bool file\_type, blockInfo\* block\_node)，写回磁盘
2. 将文件头链接的缓冲块链入垃圾链表
3. 更新LRU\_block\_list链表与LRU\_file\_list链表
   1. fileInfo\* Buffer::get\_file\_info(string file\_name, bool file\_type)

/\*

function:

查找文件是否在内存中，如果存在，返回文件头，否则分配内存初始化，返回文件头

@param:

file\_name: 文件名

file\_type: 文件类型

\*/

具体实现：

1. 首先查找该文件是否已经在内存中存在
2. 如果存在，返回该文件头;
3. 如果不存在，判断内存中已有的文件数是否等于最大的文件数；
4. 如果不是，为其分配一个文件头的struct，并根据要求进行初始化，文件个数加一，给文件头进行初始化，并返回该文件头；
5. 如果是，使用LRU算法选取文件头链表中的文件头，作为替换项，首先对选定的文件进行关闭文件操作，进而为可以重新申请一个文件头，对给文件头进行初始化，返回该文件头。
   1. void Buffer::set\_lock(fileInfo\* file\_node, bool lock)

/\*

function:

根据文件头指针，锁定该文件不从内存中替换

@param:

file\_node: 文件头指针

lock: 0->unlock, 1->lock

\*/

具体实现：

设置file\_node->lock = lock;

* 1. void Buffer::set\_lock(blockInfo\* block\_node, bool lock)

/\*

function:

根据块指针，锁定该块不从内存中替换

@param:

block\_node: 块指针

lock: 0->unlock, 1->lock

\*/

具体实现：

设置block\_node->lock = lock;

* 1. void load\_data(string file\_name, bool file\_type, blockInfo\* block\_node)

/\*

function:

读取磁盘数据到block

@param:

block\_node: 存放数据的内存块

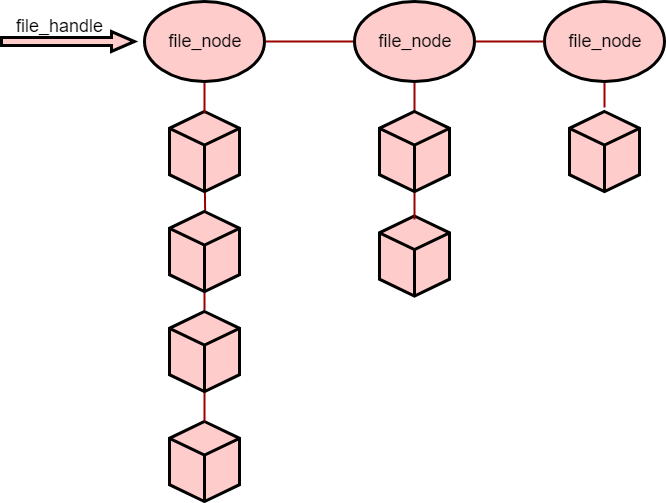
file\_type: 文件类型 0->record, 1->index

file\_name: 磁盘文件名

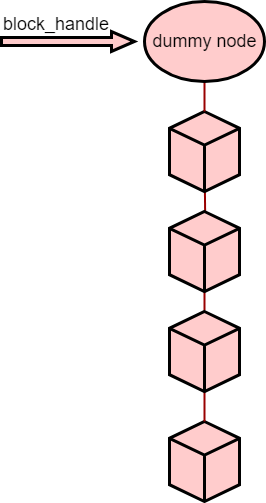
\*/

具体实现：

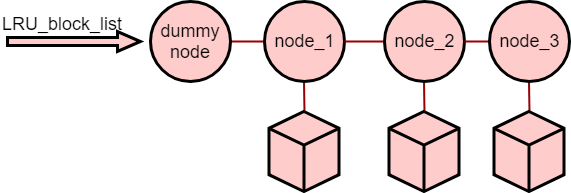
1. 根据文件类型与文件名找到对应的磁盘文件，打开文件
2. 根据缓冲块中的数据块号信息将磁盘数据读取到内存中的缓冲块中
3. **内存中维护的结构**
   1. **fileHandle维护的结构**



* 1. **BlockHandle维护的结构**



* 1. **LRU\_block\_list 维护的结构**



* 1. **LRU\_file\_list维护的结构**

