

肖邵斌

3130104016

BufferManager 模块报告

1、总体设计

BufferManager 位于数据库系统的最下层，向下直接与文件交流，对上提供类似文件操作的接口，以供 IndexManager、CatalogManager 以及 RecordManager 的调用，用户可以很方便地使用 BufferManager 而感觉不到 Block 的存在。

BufferManager 实现的功能主要有

- 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
- 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
- 记录缓冲区中各页的状态，如是否被修改过等

为了提高使用的方便，BufferManager 的 write 函数和 read 函数有多个重载，提供了不同的重载接口以供调用。同时，定义了 writeDisk 和 readDisk 函数，将 buffer 中的 Block 写入 disk 中。具体实现见后文。

数据库中，所有的文件都是以 block 的形式存在的，我定义 BLOCK_SIZE 为 4*1024。

Block 类中有 write 和 read 函数，用于对每个 Block 的读写。

具体调用 BufferManager 的过程中，上层调用 write 接口时，传入内容，长度，偏移量，由 BufferManager 将其写入 Block 并暂存在 buffer 中，待退出数据库，将 buffer 中所有内容写入磁盘，通过这样来保证读写速度。

Block 和 BufferManager 的类图如下：

Block
<u>+BLOCK_SIZE: int</u> -start: int -index: int -wf: WickyFile* -mem: char*
+getFile() -write() -read()

BufferManager
<u>-instance: BufferManager*</u> -mem_size: int -block_load: int -block_dump: int <u>+MEM_LIMIT: int</u>
-getBlock() -writeDisk() -readDisk() -sweep() +isFileExists() +getInstance() +redirect() +eof() +write() +read() +readAll() +intToBytes() +doubleToBytes() +stringToBytes()

2、函数实现

2.1 写入数据到 block

write 函数传入参数为写入位置，数据长度，

```
int Block::write(int position, int len, unsigned char* buf){
    if (position < start)    若传入非法位置则抛出异常
        throw std::runtime_error("Block::write():couldn't reach that reach within this
block");
    if (position - start + len > BLOCK_SIZE)  若超出 Block 大小
        len = BLOCK_SIZE - (position - start);  则将超出部分写入下一个 Block
    计算出写入位置，并将内容 copy 进 Block 之中
    memcpy(mem + position - start, buf, len);
    wf->setFptr(position + len);
    return len;
}
```

2.2 从 block 中读取数据

```
int Block::read(int position, int len, unsigned char* buf){
    if (position < start) { 非法位置，抛出异常
        std::cout << "position:" << position << " " << "start:" << start << std::endl;
        throw std::runtime_error("Block::read():couldn't reach within this block");
    }
    if (position - start + len > BLOCK_SIZE)
        len = BLOCK_SIZE - (position - start);
    if (wf->getSize() < position + len){
        std::cout << wf->getSize() << " position:" << position << " len:" << len<<
std::endl; 读取失败，抛出异常
        throw std::runtime_error("Block::read():couldn't reach within this file");
    } else { 将读取内容复制到 buf 中
        memcpy(buf, mem + position - start, len);
        wf->setFptr(position + len);
        return len;
    }
}
```

2.3 写入数据到 buffer

```
int BufferManager::write(std::string name, int offset, int len, unsigned char* buf){
    WickyFile* wf = getFile(name, WickyFile::FILE_WRITE);
    int ret=0;
    if (len < 0) 判断写入长度是否合法
        throw std::runtime_error("write file " + name + " , index is out of range");
    int position = offset;
    while (len > 0){ 创建 Block 用于存入数据
        Block* block = getBlock(wf, position / Block::BLOCK_SIZE);
        sweep(); 调用 Block 中的 write 写入数据
        int tmp = block->write(position, len, buf+ret);
        position += tmp; 更新位置值
        len -= tmp;
        ret += tmp;
    }
    return ret;
}
```

为了提高使用的方便，write 函数和 read 函数有多个重载，提供了不同的重载接口以供调用：

```
int BufferManager::write(std::string name, int len, unsigned char* buf){
    WickyFile* wf = getFile(name, WickyFile::FILE_WRITE);
```

```

    if (len < 0)
        throw std::runtime_error("write file " + name + " , length is out of range");
    return write(name, wf->getFptr(), len, buf);

int BufferManager::write(std::string name, int offset, int n){
    unsigned char buf[Schema::INT_LENGTH];
    intToBytes(n, buf);    将 int 转换成 bytes，此函数后文介绍
    write(name, offset, Schema::INT_LENGTH, buf);
}

int BufferManager::write(std::string name, int n);
int BufferManager::write(std::string name, int offset, double n);
int BufferManager::write(std::string name, double n);
int BufferManager::write(std::string name, int offset, std::string str);
int BufferManager::write(std::string name, std::string str);

```

2.4 从 buffer 中读取数据

```

int BufferManager::read(std::string name, int offset, int len, unsigned char* buf){
    WickyFile* wf = getFile(name, WickyFile::FILE_READ);
    int ret=0;
    if (!isFileExists(name))    判断文件是否存在
        throw std::runtime_error("file " + name + " doesn't exist");
    if (len < 0)    判断输入是否合法
        throw std::runtime_error("write file " + name + " , index is out of range");
    int position = offset;
    while (len > 0){    获取 Block
        Block* block = getBlock(wf, position / Block::BLOCK_SIZE);
        sweep();    调用 Block 的 read 函数，读取内容
        int tmp = block->read(position, len, buf+ret);
        if (tmp == 0) break;
        position += tmp;
        len -= tmp;
        ret += tmp;
    }
    return ret;
}

```

为了提高使用的方便，write 函数和 read 函数有多个重载，提供了不同的重载接口以供调用：

```

int read(std::string name, int len, unsigned char* buf);
int read(std::string name, int offset, int *n);
int read(std::string name, int *n);
int read(std::string name, int offset, double *n);

```

```

int read(std::string name, double *n);
int read(std::string name, int offset, std::string *str, int len);
int read(std::string name, std::string *str, int len);

```

2.5 获取 block 函数

```

Block* BufferManager::getBlock(WickyFile* wf, int n){
    std::map<int, Block*>* fileBlockIndex;
    std::map<WickyFile*, std::map<int, Block*>* >::iterator itr;
    itr = blockIndex.find(wf);
    if (itr == blockIndex.end()){  找到 block 位置
        fileBlockIndex = new std::map<int, Block*>;
        blockIndex.insert(std::map<WickyFile*,
std::map<int, Block*>* >::value_type(wf, fileBlockIndex));
    } else {
        fileBlockIndex = itr->second;
    }

    Block* block;
    std::map<int, Block*>::iterator blockItr = fileBlockIndex->find(n);
    if (blockItr == fileBlockIndex->end()){
        block = new Block(wf, n);
        fileBlockIndex->insert(std::map<int, Block*>::value_type(n, block));
        mem_size += 1;
        buffer.push_back(block);
    } else {
        block = blockItr->second;
    }
    return block;
}

```

2.6 写入磁盘函数

```

void BufferManager::writeDisk(WickyFile* wf, int offset, int len, unsigned char* buf){
    block_dump += 1;
    fseek(wf->getFile(), offset, SEEK_SET);  控制指针移动到写入位置
    fwrite(buf, len, 1, wf->getFile());      向磁盘中写入 block
}

```

2.7 读取磁盘函数

```
void BufferManager::readDisk(WickyFile* wf, int offset, int len, unsigned char* buf){
    block_load += 1;
    fseek(wf->getFile(), offset, SEEK_SET); 控制指针移动到读入位置
    fread(buf, len, 1, wf->getFile());      将读取内容存到 block 中
}
```

3 测试


测试在插入纪录时，BufferManager 将所有数据存入 buffer 中：
向 student 表中插入 9 条纪录：

```
wickydb>insert into student values (1,'wy1',21,'F',66.0);
insert into student values (2,'wy2',20,'M',67.0);
insert into student values (3,'wy3',21,'F',68.0);
insert into student values (4,'wy4',20,'M',69.0);
insert into student values (5,'wy5',21,'F',70.0);
insert into student values (6,'wy6',20,'M',71.0);
insert into student values (7,'wy7',21,'F',72.0);
insert into student values (8,'wy8',20,'M',73.0);
insert into student values (9,'wy9',21,'F',74.0);
```



插入的数据被存入 buffer 中，可以 select 查看

```
----->select * from student;
      sno      sname      sage      sgender      score
      1         wy1         21           F         66.0
      2         wy2         20           M         67.0
      3         wy3         21           F         68.0
      4         wy4         20           M         69.0
      5         wy5         21           F         70.0
      6         wy6         20           M         71.0
      7         wy7         21           F         72.0
      8         wy8         20           M         73.0
      9         wy9         21           F         74.0
```

查看磁盘文件，student 表的大小为 0，表示此时数据没有写入磁盘：

名称	修改日期	创建日期	大小	种类
 student	今天 23:05	23:05	0 字节	文本编...app 文稿

退出系统之后，此时 student 表为 8KB，表示 buffer 中的数据被写入磁盘，BufferManager 工作正常：

名称	修改日期	创建日期	大小	种类
 student	今天 23:07	23:05	8 KB	文本编...app 文稿
 WICKY_CATALOG.wk	今天 23:07	23:07	4 KB	文稿

4 总结

本次小型数据库的实现一方面让我深刻理解了各个模块分工协作的过程，对数据库架构有了更加清晰的认识，另一方面，我也意识到了数据库设计的难度，大型数据库的实现需要十分细致和严谨地工作。通过小组合作，我们的数据库 WickyDB 最终实现了所有预期功能，感到很有成就感。