BufferManagement

刘轩铭

主要功能

BufferManagerment负责缓冲区的管理，主要功能有：

- 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
- 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
- 提供缓冲区页的pin功能，即锁定缓冲区的页，不允许替换出去

接口说明

```
API:
    bool readPage( Page& );
    bool writePage( Page& );
    Page& recordManagerGetBlankPage();
    bool pinPage( Page& );
    bool unpinPage( Page& );
```

- 本模块提供读写的接口给其他模块使用。
    - readPage()

      首先判断出缓冲区内是否有需要的页，如果有，直接读取缓冲区相应页，同时改变计数器，将除了当前读取的缓冲页的其他缓冲页的计数器全部加1，表示其他页都没有被使用；如果没有直接从文件中进行该页面的读取，读取完了以后寻找当前缓冲区中计数器最大且没有pin的页，进行替换。如果替换的页是Dirty的则需要将这个页面写回文件。
    - writePage()
      基本逻辑类似于readPage
    - recordManagerGetBlankPage()

      recordManager 可以通过该方法获得一个新的页，然后通过实际需要对页的信息进行配置，最后进行read和write的操作。
    - 其他模块可以按照需要，对相应的页进行pin锁定和解锁

模块说明

- 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，块的大小为文件系统与磁盘交互单位的整数倍，在本模块中设定为4KB。同时考虑到缓存区的大小，我们将缓存区最大容量定为10个Page。
- BufferManagement由两个类构成，分别是BufferManager类和Page类

```
class Page{
public:
    string tableName;
    PageType pageType;
    PageIndex pageIndex;
    char pageData[PAGE_SIZE];
};
```

一个Page对象是内存缓冲区中的一个块，对一个特定的表进行数据储存。其成员属性包含了处理表的名字，区块的类型，区块在缓存区中的索引号，以及内部储存的信息。

其他组件通过设定page中的成员属性将区块传递给BufferManager，BufferManager通过page中的信息从硬盘或者从缓冲区获取数据或者写入数据。需要写入和读取的数据都在char数组pageData中。

```
class BufferManager{
private:
    static int lruCounter[CACHE_CAPACITY];
    static Page cachePages[CACHE_CAPACITY];
    static bool pined[CACHE_CAPACITY];
    static bool isDirty[CACHE_CAPACITY];

public:
    bool readPage( Page& );
    bool writePage( Page& );
    Page& recordManagerGetBlankPage();
    bool pinPage( Page& );
    bool unpinPage( Page& );
}
```

BufferManager实现了对缓存区的管理。缓存使用了一个Page数组进行内存缓存。

- 为了实现LRU使用了一个与缓冲区大小相同的计数器来记录缓冲块被使用的次数
- 为了实现替换时页面是否Dirty，使用了一个与缓冲区大小相同的bool数组记录缓冲区是否被写过
- 为了实现pin功能使用了一个bool数组来记录pin过的缓冲区。

- 下面介绍对Pin功能和LRU算法的实现
  - 对于Pin功能，本模块对外提供接口，进行模块的pin锁定。同时用pin数组进行该状态的记录

    ```
    ...
    public:
        bool pinPage( Page& );
        bool unpinPage( Page& );
    ...
    ```

    如果该区块被pin，那么在进行缓存区替换的时候，这个模块将不会被拿出模块。该状态会在一直保持到模块的析构阶段。
  - 对于LRU算法，本模块用一个计数器对区块的使用次数进行计算。

    ```
    ...

    void BufferManager::lruCounterAddExceptCurrent( int index )
    {
        for( int i = 0; i < CACHE_CAPACITY; i++ )
    ```

```
            lruCounter[i]++;
        // newly used, set as 0. Least recently used algorithm.
        lruCounter[index] = 0;
}

...

PageIndex BufferManager::getUnpinedBiggestPageFromCache()
{
    int retIndex = PAGE_NOTFOUND;
    int bigSaver = -1;
    for (int i = 0; i < CACHE_CAPACITY; i++) {
        if ((!pined[i]) && (lruCounter[i] > bigSaver)) {
            retIndex = i;
            bigSaver = lruCounter[i];
        }
    }
    return retIndex;
}

...
```

当一个区块被使用时，缓存区内其他区块的计数器会进行自增。当需要进行区块替换的时候，会在缓存区内寻找到使用次数最少（且没有被Pin）的那个区块进行替换。