# BufferPoolManager实现报告

## BufferPoolManager::find\_victim\_page

bool find\_victim\_page(frame\_id\_t\* frame\_id);

**1.方法声明：**

方法名：find\_victim\_page

返回类型：bool

功能：从free\_list或replacer中得到可淘汰帧页的frame\_id

参数列表：

|  | **frame\_id** |
| --- | --- |
| 类型 | frame\_id\_t\* |
| 含义 | 帧页id指针，返回成功找到的可替换帧id |

**2.方法实现思路：**

1. 首先检查缓冲池是否有空闲帧（检查free\_list\_是否为空）
2. 如果有空闲帧：从free\_list\_头部获取一个空闲frame\_id，并从链表中移除
3. 如果没有空闲帧：调用replacer\_->victim(frame\_id)使用LRU策略选择一个victim页面
4. 返回查找结果（true表示成功，false表示失败）

**3.实现难点：**

需要理解缓冲池管理的两个阶段：空闲帧管理和页面替换。优先使用空闲帧，只有在缓冲池满时才进行页面替换。

## BufferPoolManager::fetch\_page

Page\* fetch\_page(PageId page\_id);

**1.方法声明：**

方法名：fetch\_page

返回类型：Page\*

功能：从buffer pool获取需要的页。如果页表中存在page\_id则直接返回并增加pin\_count，否则从磁盘读取页面到缓冲池

参数列表：

|  | **page\_id** |
| --- | --- |
| 类型 | PageId |
| 含义 | 需要获取的页的PageId |

**2.方法实现思路：**

1. 使用std::scoped\_lock加锁保证线程安全
2. 在page\_table\_中查找目标页是否已在缓冲池中
3. 如果页面已在缓冲池：增加pin\_count\_，调用replacer\_->pin(frame\_id)固定页面，返回页面指针
4. 如果页面不在缓冲池：
   * 调用find\_victim\_page获取可用frame
   * 如果victim页面是脏页，先写回磁盘
   * 从磁盘读取目标页到frame
   * 更新页面元数据（id、pin\_count、is\_dirty）
   * 更新page\_table映射关系
   * 固定页面并返回

**3.实现难点：**

需要处理页面置换时的脏页写回，确保数据一致性。同时要正确维护page\_table的映射关系和页面的固定状态。

## BufferPoolManager::unpin\_page

bool unpin\_page(PageId page\_id, bool is\_dirty);

**1.方法声明：**

方法名：unpin\_page

返回类型：bool

功能：取消固定pin\_count>0的在缓冲池中的page

参数列表：

|  | **page\_id** | **is\_dirty** |
| --- | --- | --- |
| 类型 | PageId | bool |
| 含义 | 目标page的page\_id | 若目标page应该被标记为dirty则为true |

**2.方法实现思路：**

1. 使用std::scoped\_lock加锁保证线程安全
2. 在page\_table\_中查找目标页，如果不存在则返回false
3. 检查页面的pin\_count\_，如果已经为0则返回false
4. 将pin\_count\_减1
5. 如果减1后pin\_count\_为0，调用replacer\_->unpin(frame\_id)将页面加入可替换列表
6. 根据is\_dirty参数更新页面的脏标记
7. 返回true

**3.实现难点：**

需要正确理解pin/unpin机制：pin\_count为0的页面才能被替换，unpin操作需要与replacer同步。

## BufferPoolManager::flush\_page

bool flush\_page(PageId page\_id);

**1.方法声明：**

方法名：flush\_page

返回类型：bool

功能：将目标页写回磁盘，不考虑当前页面是否正在被使用

参数列表：

|  | **page\_id** |
| --- | --- |
| 类型 | PageId |
| 含义 | 目标页的page\_id，不能为INVALID\_PAGE\_ID |

**2.方法实现思路：**

1. 使用std::scoped\_lock加锁保证线程安全
2. 在page\_table\_中查找目标页，如果不存在则返回false
3. 调用disk\_manager\_->write\_page将页面数据写回磁盘
4. 将页面的is\_dirty\_标记设为false
5. 返回true

**3.实现难点：**

实现相对简单，主要是确保强制将页面写回磁盘，无论页面是否为脏页。

## BufferPoolManager::new\_page

Page\* new\_page(PageId\* page\_id);

**1.方法声明：**

方法名：new\_page

返回类型：Page\*

功能：创建一个新的page，即从磁盘中移动一个新建的空page到缓冲池某个位置

参数列表：

|  | **page\_id** |
| --- | --- |
| 类型 | PageId\* |
| 含义 | 当成功创建一个新的page时存储其page\_id |

**2.方法实现思路：**

1. 使用std::scoped\_lock加锁保证线程安全
2. 调用find\_victim\_page获得一个可用的frame
3. 如果victim页面是脏页，先写回磁盘
4. 调用disk\_manager\_->allocate\_page分配新的page\_id
5. 初始化新页面的元数据（id、pin\_count=1、is\_dirty=false）
6. 清空页面数据（reset\_memory()）
7. 更新page\_table\_映射关系
8. 调用replacer\_->pin固定页面
9. 返回新创建的页面指针

**3.实现难点：**

需要协调磁盘空间分配、缓冲池管理和页面初始化，确保新页面正确地被创建和固定。

## BufferPoolManager::delete\_page

bool delete\_page(PageId page\_id);

**1.方法声明：**

方法名：delete\_page

返回类型：bool

功能：从buffer\_pool删除目标页

参数列表：

|  | **page\_id** |
| --- | --- |
| 类型 | PageId |
| 含义 | 目标页 |

**2.方法实现思路：**

1. 使用std::scoped\_lock加锁保证线程安全
2. 在page\_table\_中查找目标页，如果不存在则返回true
3. 检查目标页的pin\_count\_，如果不为0则返回false（页面正在被使用）
4. 如果是脏页，先写回磁盘
5. 从page\_table\_中删除映射关系
6. 重置页面元数据（page\_id设为INVALID\_PAGE\_ID，pin\_count=0，is\_dirty=false）
7. 清空页面数据
8. 将frame加入free\_list\_以供重用
9. 返回true

**3.实现难点：**

需要确保只有未被固定的页面才能被删除，并正确回收frame资源。

## BufferPoolManager::flush\_all\_pages

void flush\_all\_pages(int fd);

**1.方法声明：**

方法名：flush\_all\_pages

返回类型：void

功能：将buffer\_pool中的所有页写回到磁盘

参数列表：

|  | **fd** |
| --- | --- |
| 类型 | int |
| 含义 | 文件句柄 |

**2.方法实现思路：**

1. 使用std::scoped\_lock加锁保证线程安全
2. 遍历缓冲池中的所有页面（pages\_[0]到pages\_[pool\_size\_-1]）
3. 对于每个页面，检查其是否属于指定文件（page->get\_page\_id().fd == fd）且有效（page\_no != INVALID\_PAGE\_ID）
4. 如果条件满足，调用disk\_manager\_->write\_page将页面写回磁盘
5. 将页面的is\_dirty\_标记设为false

**3.实现难点：**

需要遍历整个缓冲池并正确识别属于指定文件的页面，确保批量写回操作的正确性。