# 实验1：缓冲池管理器实现 - 问答题答案

## 任务1：缓冲池页面替换策略实现

### 1. LRUlist\_ 的作用是什么？

LRUlist\_ 是一个双向链表（std::list<frame\_id\_t>），用于按照LRU（最近最少使用）的顺序存储所有**未被固定**（unpinned）的帧编号。

具体作用：

* 链表头部存储最近被访问（最近被unpin）的帧
* 链表尾部存储最久未被访问（最早被unpin）的帧
* 当需要选择victim页面时，从尾部取出最久未被访问的帧进行淘汰
* 当页面被unpin时，将其加入到链表头部，表示最近被访问

### 2. LRUhash\_ 的作用是什么？

LRUhash\_ 是一个哈希表（std::unordered\_map<frame\_id\_t, std::list<frame\_id\_t>::iterator>），用于建立帧编号到其在LRU链表中位置的快速映射。

具体作用：

* 键：帧编号（frame\_id\_t）
* 值：该帧在LRUlist\_链表中的迭代器位置
* 提供O(1)时间复杂度的查找和删除操作
* 当需要将某个帧从LRU链表中移除时，可以快速定位其在链表中的位置

### 3. LRUlist\_ 和LRUhash\_ 的关系是什么？

两者配合实现了高效的LRU替换策略：

**互补关系**：

* LRUlist\_ 维护帧的访问顺序，支持顺序遍历和头尾操作
* LRUhash\_ 提供快速的随机访问能力

**协同工作**：

* **插入操作**：当帧被unpin时，同时在LRUlist*头部插入帧编号，在LRUhash*中记录该帧在链表中的位置
* **删除操作**：当帧被pin时，通过LRUhash*快速找到帧在链表中的位置，然后从LRUlist*中删除
* **victim选择**：直接从LRUlist\_尾部取出最久未被访问的帧

**时间复杂度优化**：

* 如果只有链表，删除中间元素需要O(n)时间
* 如果只有哈希表，无法维护访问顺序
* 两者结合实现了所有操作的O(1)时间复杂度

## 任务2：缓冲池管理器实现

### 1. Page::is\_dirty\_ 的作用是什么？

Page::is\_dirty\_ 是一个布尔值，用于标记页面是否为**脏页**。

**作用**：

* **写优化**：只有脏页在被淘汰或刷写时才需要写入磁盘，避免不必要的磁盘写操作
* **数据一致性**：确保修改过的页面能够正确写回磁盘，保持内存和磁盘数据的一致性
* **性能提升**：减少不必要的磁盘I/O操作，提高系统性能

**使用场景**：

* 当页面内容被修改时，设置为true
* 当页面被写入磁盘后，设置为false
* 在页面淘汰时，只有脏页需要写回磁盘

### 2. Page::pin\_count\_ 的作用是什么？

Page::pin\_count\_ 是一个整数，表示当前有多少个线程或操作正在使用该页面。

**作用**：

* **引用计数**：记录页面的使用情况，防止正在使用的页面被误删除或替换
* **并发控制**：在多线程环境下保护页面不被意外淘汰
* **资源管理**：确保页面在使用期间保持在内存中

**状态转换**：

* pin\_count\_ > 0：页面正在被使用，不能被淘汰
* pin\_count\_ = 0：页面可以被加入LRU替换候选列表
* 每次fetch\_page时pin\_count\_递增，每次unpin\_page时递减

### 3. BufferPoolManager::page\_table\_ 的作用是什么？

page\_table\_ 是一个哈希表（std::unordered\_map<PageId, frame\_id\_t, PageIdHash>），维护页面ID到帧编号的映射关系。

**作用**：

* **快速查找**：给定PageId，能够O(1)时间内找到该页面在缓冲池中的帧位置
* **存在性检查**：判断某个页面是否已经在缓冲池中
* **地址转换**：将逻辑页面标识转换为物理帧位置

**键值关系**：

* **键**：PageId（包含fd和page\_no的页面唯一标识）
* **值**：frame\_id\_t（该页面在缓冲池中的帧编号）

### 4. BufferPoolManager::free\_list\_ 的作用是什么？

free\_list\_ 是一个链表（std::list<frame\_id\_t>），维护当前**空闲**的帧编号列表。

**作用**：

* **空间管理**：跟踪缓冲池中哪些帧当前没有存储有效页面
* **快速分配**：当需要新的帧时，可以直接从free\_list\_中取出，无需使用复杂的替换策略
* **初始化**：系统启动时，所有帧都在free\_list\_中
* **回收**：当页面被删除时，对应的帧重新加入free\_list\_

**使用优先级**：

* 优先使用free\_list\_中的空闲帧
* 只有当free\_list\_为空时，才使用LRU替换策略选择victim帧

**生命周期**：

* 系统初始化：所有帧编号都在free\_list\_中
* 页面分配：从free\_list\_中取出帧，用于存储新页面
* 页面删除：释放的帧重新加入free\_list\_