# LRUReplacer实现报告

## LRUReplacer::victim

bool victim(frame\_id\_t \*frame\_id);

**1.方法声明：**

方法名：victim

返回类型：bool

功能：使用LRU策略删除一个victim frame，并返回该frame的id

参数列表：

|  | **frame\_id** |
| --- | --- |
| 类型 | frame\_id\_t\* |
| 含义 | 被移除的frame的id，如果没有frame被移除返回nullptr |

**2.方法实现思路：**

1. 使用std::scoped\_lock对latch\_加锁，保证线程安全
2. 检查LRU链表LRUlist\_是否为空，如果为空则没有可淘汰的页面，返回false
3. 选择链表尾部的frame作为victim（最久未被访问的页面）
4. 从链表中移除该元素：LRUlist\_.pop\_back()
5. 从哈希表中移除该元素：LRUhash\_.erase(victim\_frame)
6. 将victim frame id赋值给输出参数\*frame\_id
7. 返回true表示成功淘汰了一个页面

**3.实现难点：**

主要难点在于理解LRU策略的数据结构设计：链表头部存放最近访问的页面，尾部存放最久未访问的页面。因此淘汰时选择尾部元素。

## LRUReplacer::pin

void pin(frame\_id\_t frame\_id);

**1.方法声明：**

方法名：pin

返回类型：void

功能：固定指定的frame，即该页面无法被淘汰

参数列表：

|  | **frame\_id** |
| --- | --- |
| 类型 | frame\_id\_t |
| 含义 | 需要固定的frame的id |

**2.方法实现思路：**

1. 使用std::scoped\_lock对latch\_加锁，保证线程安全
2. 在哈希表LRUhash\_中查找指定的frame\_id
3. 如果找到该frame（即该frame当前是unpinned状态）：
   * 使用哈希表中存储的迭代器从LRU链表中删除该frame
   * 从哈希表中删除该frame的记录
4. 如果没找到，说明该frame已经是pinned状态或不存在，无需操作

**3.实现难点：**

需要理解pin操作的含义：将页面从可淘汰状态变为不可淘汰状态，因此需要从LRU数据结构中移除。

## LRUReplacer::unpin

void unpin(frame\_id\_t frame\_id);

**1.方法声明：**

方法名：unpin

返回类型：void

功能：取消固定一个frame，代表该页面可以被淘汰

参数列表：

|  | **frame\_id** |
| --- | --- |
| 类型 | frame\_id\_t |
| 含义 | 取消固定的frame的id |

**2.方法实现思路：**

1. 使用std::scoped\_lock对latch\_加锁，保证线程安全
2. 检查该frame是否已经在LRU链表中，如果已存在则直接返回（避免重复添加）
3. 检查当前链表大小是否已达到最大容量max\_size\_，如果是则返回
4. 将frame\_id添加到链表头部（表示最近访问的位置）：LRUlist\_.push\_front(frame\_id)
5. 在哈希表中记录该frame在链表中的迭代器：LRUhash\_[frame\_id] = LRUlist\_.begin()

**3.实现难点：**

需要理解unpin操作将页面添加到链表头部而不是尾部，因为新unpin的页面被认为是最近访问的。同时需要注意检查重复添加和容量限制。