第 0017 讲 3 页回收机制（三）

内存管理专题--3 页回收机制

本节教学内容

- 页回收
- 发起页回收方式
- 计算扫描页数
- 收缩活动页链表
- 回收不活动页函数流程
- 页交换及回收 slab 缓存

一、页回收

1、当我们分配页的时候，页分配器首先尝试使用低水线分配页，若失败则说明内存轻微不足，页分配器将会唤醒内存节点的页回收内核线程，异步回收页，然后尝试使用最低水线分配页。如果使用最低水线分配失败则说明内存严重不足，页分配器将会直接回收页。

2、Linux 内核根据 LRU（最近最少使用，Least Recently Used）算法选择最近最少使用的物理页。页回收算法使用 LRU 算法选择回收的页。每个内存节点的 pglist_data 实例有一个成员 lruvec，称为 LRU 向量（包含 5 条 LRU 链表）

include > linux > C mmzone.h > bootmem_data

```
648
649
650 typedef struct pglist_data {
651     struct zone node_zones[MAX_NR_ZONES];
652     struct zonelist node_zonelists[MAX_ZONELISTS];
653     int nr_zones;
654 #ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
655     struct page *node_mem_map;
656 #ifdef CONFIG_PAGE_EXTENSION
657     struct page_ext *node_page_ext;
658 #endif
659 #endif
```

include > linux > C mmzone.h > pglist_data

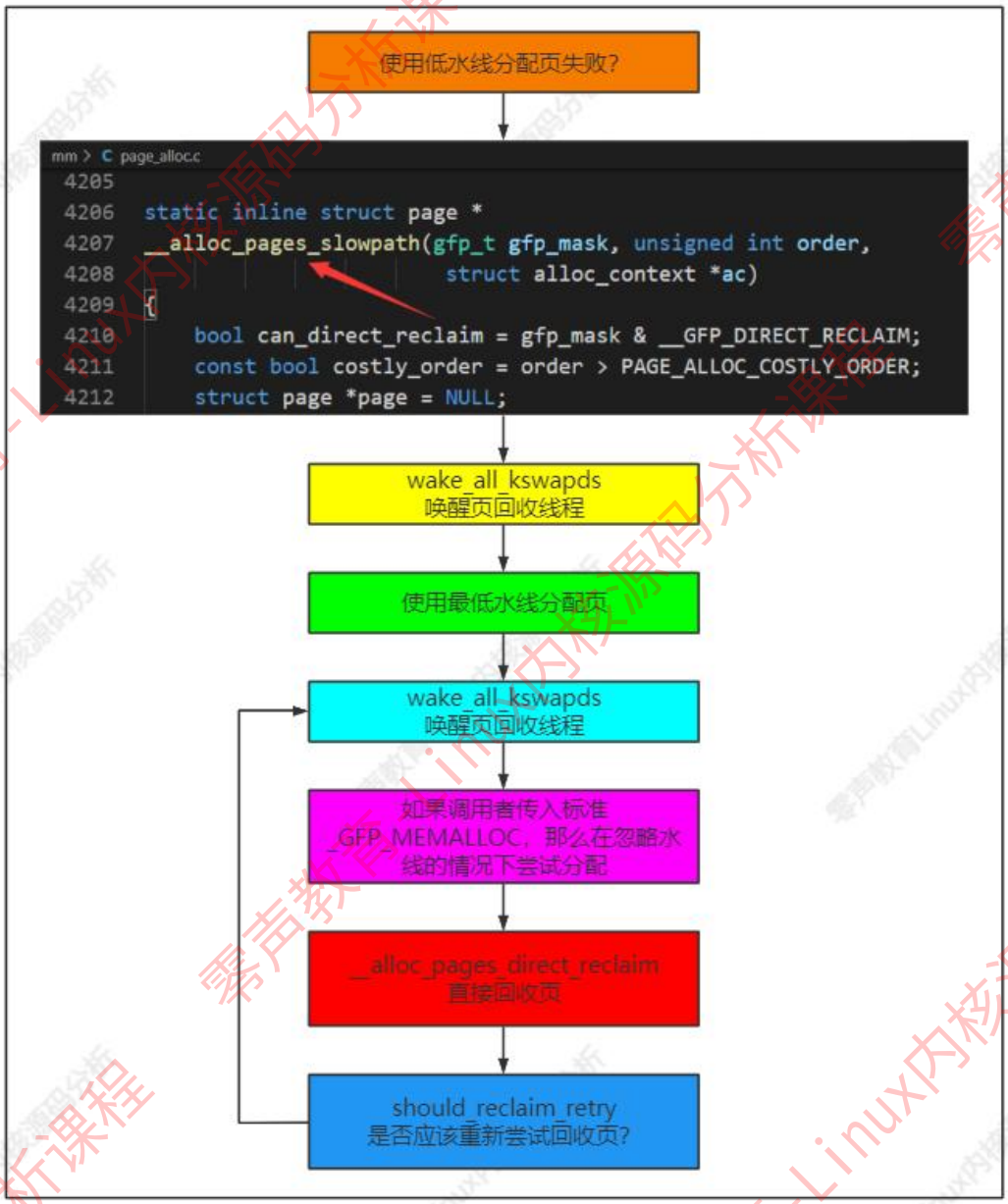
```
726     /* Fields commonly accessed by the page reclaim scanner */
727     struct lruvec       lruvec;
728
```

include > linux > C mmzone.h > lru_list

```
209 enum lru_list {
210     LRU_INACTIVE_ANON = LRU_BASE,  // 称为非活动匿名页lru链表（swap）
211     LRU_ACTIVE_ANON = LRU_BASE + LRU_ACTIVE, // 称为活动匿名页lru链表（swap）
212     LRU_INACTIVE_FILE = LRU_BASE + LRU_FILE, // 称为非活动文件页lru链表（磁盘）
213     LRU_ACTIVE_FILE = LRU_BASE + LRU_FILE + LRU_ACTIVE, // 称为活动文件页lru链表（磁盘）
214     LRU_UNEVICTABLE, // 此链表中保存的是此zone中所有禁止换出的页的描述符。
215     NR_LRU_LISTS
216 };
217
```

二、发起页回收方式

Linux 内核发起页回收整体流程如下：



三、计算扫描页数

扫描优先级用来控制一次扫描的页数，如果扫描优先级是 n，那么一次扫描的页数(LRU 链表当中的总页数>>n)，由此看出："扫描优先级的值越小，扫描的页越多"。页回收算法从默认优先级 12 开始，如果回收的页数没有达到目标，那么提高扫描优先级，把扫描优先级的值减 1，然后继续扫描。扫描优先级的最小值为 0，表示扫描 LRU 链表中的所有页。

两个参数来控制扫描文件名和匿名页的比例：

- 参数"swappiness"控制换出匿名页的积极程度，取值范围[0--100]，值越大表示匿名页的比例就越高，默认为 60。

- 针对文件页和匿名页分配统计最近扫描的页数和从不活动变为活动的页灵敏，计算比例。

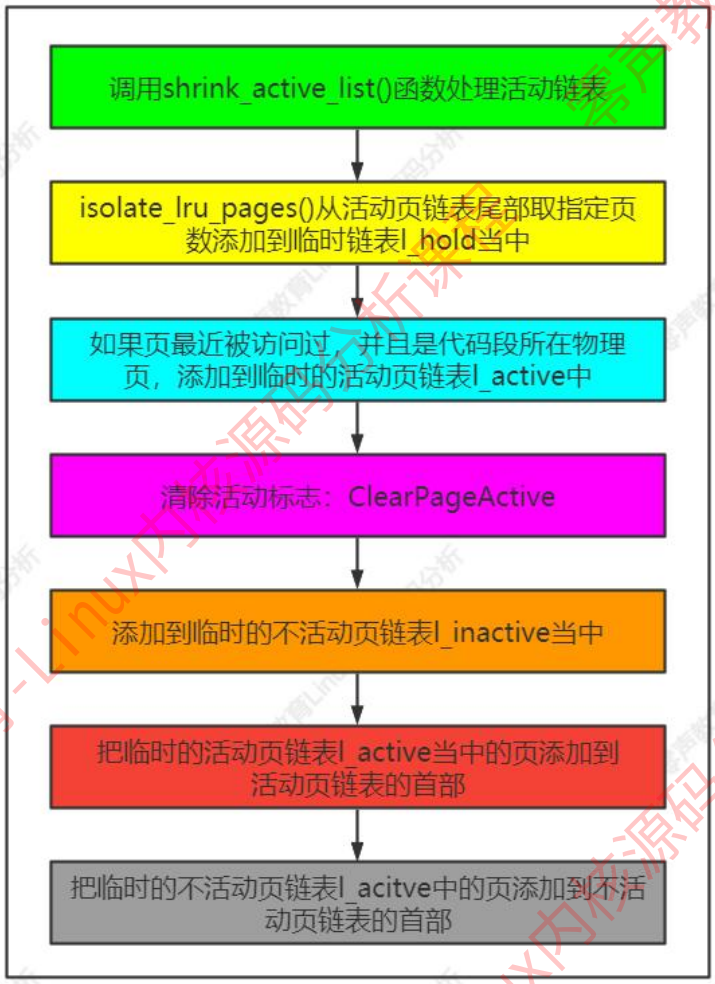
四、收缩活动页链表

当不活动页比较少的时候，而回收算法收缩活动的链表。就是将从活动页链表的尾部取物理页并且转移到不活动页链表中，把活动页转换成不活动页。

具体由 shrink_active_list(.....)函数负责从活动页链表中转换物理页到不活动页链表中，具体过程如下：

mm > C vmscan.c > shrink_active_list(unsigned long, lruvec *, scan_control *, lru_list lru)

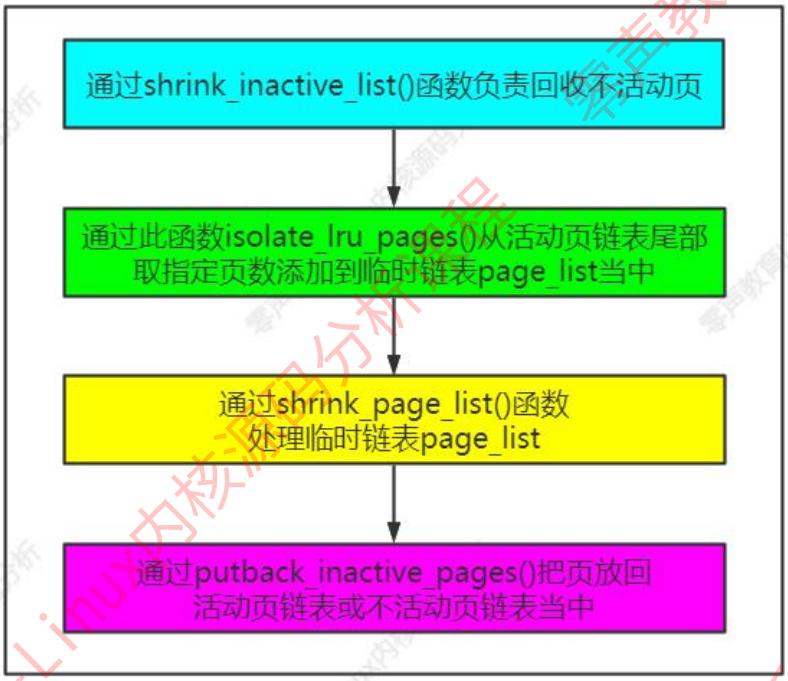
```
2082  static void shrink_active_list(unsigned long nr_to_scan,
2083                                 struct lruvec *lruvec,
2084                                 struct scan_control *sc,
2085                                 enum lru_list lru)
2086  {
2087      unsigned long nr_taken;
2088      unsigned long nr_scanned;
2089      unsigned long vm_flags;
2090      LIST_HEAD(l_hold);  /* The pages which were snipped off */
2091      LIST_HEAD(l_active);
2092      LIST_HEAD(l_inactive);
2093      struct page *page;
2094      struct zone_reclaim_stat *reclaim_stat = &lruvec->reclaim_stat;
2095      unsigned nr_deactivate, nr_activate;
2096      unsigned nr_rotated = 0;
2097      isolate_mode_t isolate_mode = 0;
2098      int file = is_file_lru(lru);
2099      struct pglist_data *pgdat = lruvec_pgdat(lruvec);
2100
```



五、回收不活动页函数流程

mm > C vmscan.c > shrink_inactive_list(unsigned long, lruvec *, scan_control *, lru_list lru)

```
1900  /*
1901   * shrink_inactive_list() is a helper for shrink_node().  It returns the number
1902   * of reclaimed pages
1903   */
1904  static noinline_for_stack unsigned long
1905  shrink_inactive_list(unsigned long nr_to_scan, struct lruvec *lruvec,
1906                       struct scan_control *sc, enum lru_list lru)
1907  {
1908      LIST_HEAD(page_list);
1909      unsigned long nr_scanned;
1910      unsigned long nr_reclaimed = 0;
1911      unsigned long nr_taken;
1912      struct reclaim_stat stat = {};
1913      isolate_mode_t isolate_mode = 0;
1914      int file = is_file_lru(lru);
```



六、页交换及回收 slab 缓存

1、页交换

页交换原理非常简单 :当内存不足时把最近很少访问的没有存储设备支持的物理页的数据暂时保存到交换区，释放内存空间，当交换区中存储的页被访问的时候，再把数据从交换区读取到内存当中，交换区可为一个磁盘分区，也可为存储设备上的文件。

2、回收 slab 缓存

使用 slab 缓存的内核模块可以注册收缩器，而回收算法遍历收缩器链表，调用每个收缩器来收缩 slab 缓存及释放对象，具体 API 接口如下：

register_shrink(....)注册收缩器/unregister_shrink(.....)注销收缩器。

具体收缩器数据结构源码分析如下：

include > linux > C shrinker.h > SHRINKER_NUMA_AWARE

```
62  struct shrinker {
63      unsigned long (*count_objects)(struct shrinker *,
64                     struct shrink_control *sc);
65      unsigned long (*scan_objects)(struct shrinker *,
66                     struct shrink_control *sc);
67
68      long batch; /* reclaim batch size, 0 = default */
69      int seeks;  /* seeks to recreate an obj */
70      unsigned flags;
71
72      /* These are for internal use */
73      struct list_head list;
74  #ifdef CONFIG_MEMCG_KMEM
75      /* ID in shrinker_idr */
76      int id;
77  #endif
78      /* objs pending delete, per node */
79      atomic_long_t *nr_deferred;
80  };
```

3、负责回收 slab 缓存

mm > C vmscan.c > do_shrink_slab(shrink_control *, shrinker *, int)

```
452  static unsigned long do_shrink_slab(struct shrink_control *shrinkctl,
453                      struct shrinker *shrinker, int priority)
454  {
455      unsigned long freed = 0;
456      unsigned long long delta;
457      long total_scan;
458      long freeable;
459      long nr;
460      long new_nr;
461      int nid = shrinkctl->nid;
462      long batch_size = shrinker->batch ? shrinker->batch
463                      : SHRINK_BATCH;
464      long scanned = 0, next_deferred;
465
466      if (!(shrinker->flags & SHRINKER_NUMA_AWARE))
467          nid = 0;
468
469      freeable = shrinker->count_objects(shrinker, shrinkctl);
470      if (freeable == 0 || freeable == SHRINK_EMPTY)
471          return freeable;
```