第012讲 Slab分配器详解

零声学院讲师：Vico老师

学习内容

一、slab思想及编程接口

二、slab数据结构

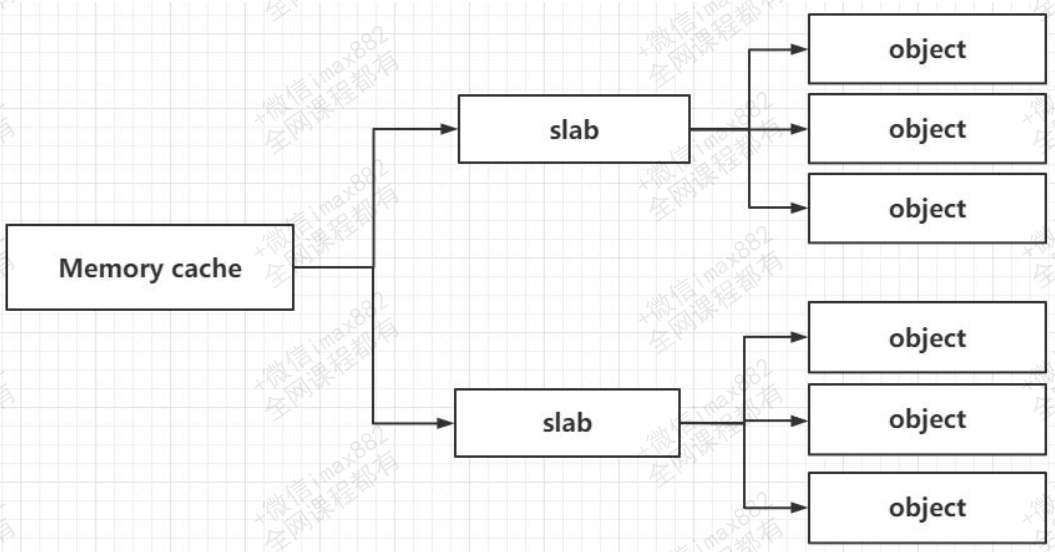
三、每处理器数组缓存

四、回收内存

一、slab思想及编程接口

1、slab核心思想

为每种对象类型创建一个内存缓存，每个内存缓存由多个大块组成，一个大块是一个或多个连续的物理页，每个大块包含多个对象。slab采用面向对象的思想，基于对象类型管理内存，每种对象被划分为一个类，比如进程描述符（task_struct）是一个类，每个进程描述符实现是一个对象。内存缓存组成结构如下：



2、编程接口

分配内存：void * kmalloc (size_t size,gfp_t flags);

重新分配内存：void *krealloc(const void *p, size_t new_size, gfp_t flags)

释放内存：void kfree (const void * objp);

include > linux > C slab.h > ksize(const void *)

```
142   void * __must_check __krealloc(const void *, size_t, gfp_t);
143   void * __must_check krealloc(const void *, size_t, gfp_t);
144   void kfree(const void *);
145   void kzfree(const void *);
146   size_t ksize(const void *);
147
```

创建内存缓存：struct kmem_cache *kmem_cache_create(const char *, size_t, size_t,unsigned long, void (*)(void *));

指定的内存缓存分配对象：void *kmem_cache_alloc(struct kmem_cache *, gfp_t);

释放对象：void kmem_cache_free(struct kmem_cache *, void *);

销毁内存缓存：void kmem_cache_destroy(struct kmem_cache *);

include > linux > C slab.h > kmem_cache_create(const char *, size_t, size_t, unsigned long, void(*)(void *))

```
100
101  struct kmem_cache *kmem_cache_create(const char *, size_t, size_t,
102                  unsigned long,
103                  void (*)(void *));
```

二、slab数据结构

- 每个内存缓存对应一个kmem_cache实例。
- 每个内存节点对应一个kmem_cache_node实例。
- kmem_cache实例的成员cpu_slab指向array_cache实例，每个处理器对应一个array_cache实例，称为数组缓存，用来缓存刚刚释放的对象，分配时首先从当前处理器的数据缓存分配，避免每次都要从slab分配，减少链表操作的锁操作，提高分配的速度。**slab分配器数据结构源码分析如下：**

slab.c 2 ×

mm > slab.c > ...

```
224 struct slab {
225     struct list_head list;
226     unsigned long colouroff;
227     void *s_mem;        /* including colour offset */
228     unsigned int inuse; /* num of objs active in slab */
229     kmem_bufctl_t free;
230     unsigned short nodeid;
231 };
```

高速缓存描述符数据结构：struct kmem_cache。

一个高速缓存中可以含有多个kmem_cache对应的高速缓存，就拿L1高速缓存来举例，一个L1高速缓存对应一个kmem_cache链表，这个链表中的任何一个kmem_cache类型的结构体均描述一个高速缓存，而这些高速缓存在L1 cache中各自占用着不同的区域。

include > linux > C slab_def.h > ...

```
24
25  struct kmem_cache {
26  /* 1) per-cpu data, touched during every alloc/free */
27      struct array_cache *array[NR_CPUS];
28  /* 2) Cache tunables. Protected by cache_chain_mutex */
29      unsigned int batchcount;
30      unsigned int limit;
31      unsigned int shared;
32
33      unsigned int buffer_size;
34      u32 reciprocal_buffer_size;
35  /* 3) touched by every alloc & free from the backend */
```

内存缓存的主要数据结构如下：

include > linux > C slab_def.h > kmem_cache > gfporder

```
25  struct kmem_cache {
26  /* 1) per-cpu data, touched during every alloc/free */
27      struct array_cache *array[NR_CPUS];
28  /* 2) Cache tunables. Protected by cache_chain_mutex */
29      unsigned int batchcount;
30      unsi
31      unsi
32
```

mm > C slab.c > array_cache

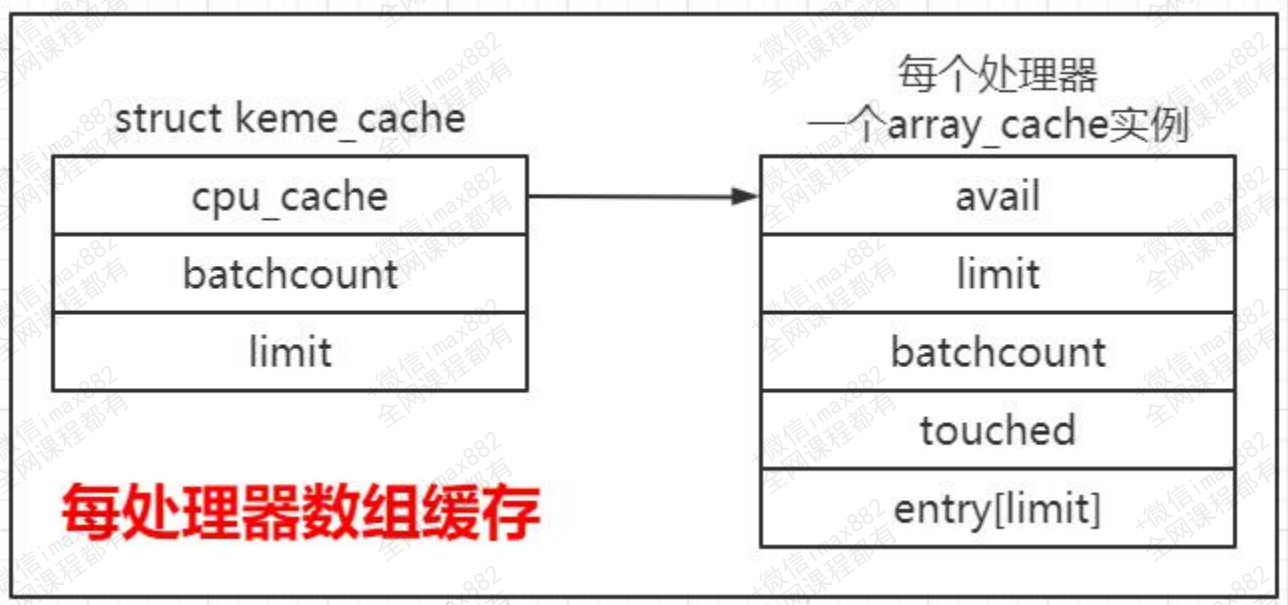
```
267  struct array_cache {
268      unsigned int avail;
269      unsigned int limit;
270      unsigned int batchcount;
271      unsign
272      spinlo
273      void *
```

include > linux > C slab_def.h > kmem_cache > array

```
93          */
94      struct kmem_list3 *nodelists[MAX_NUMNODES];
95      /*
96       * Do not add fields after nodelists[]
97       */
98  };
99
```

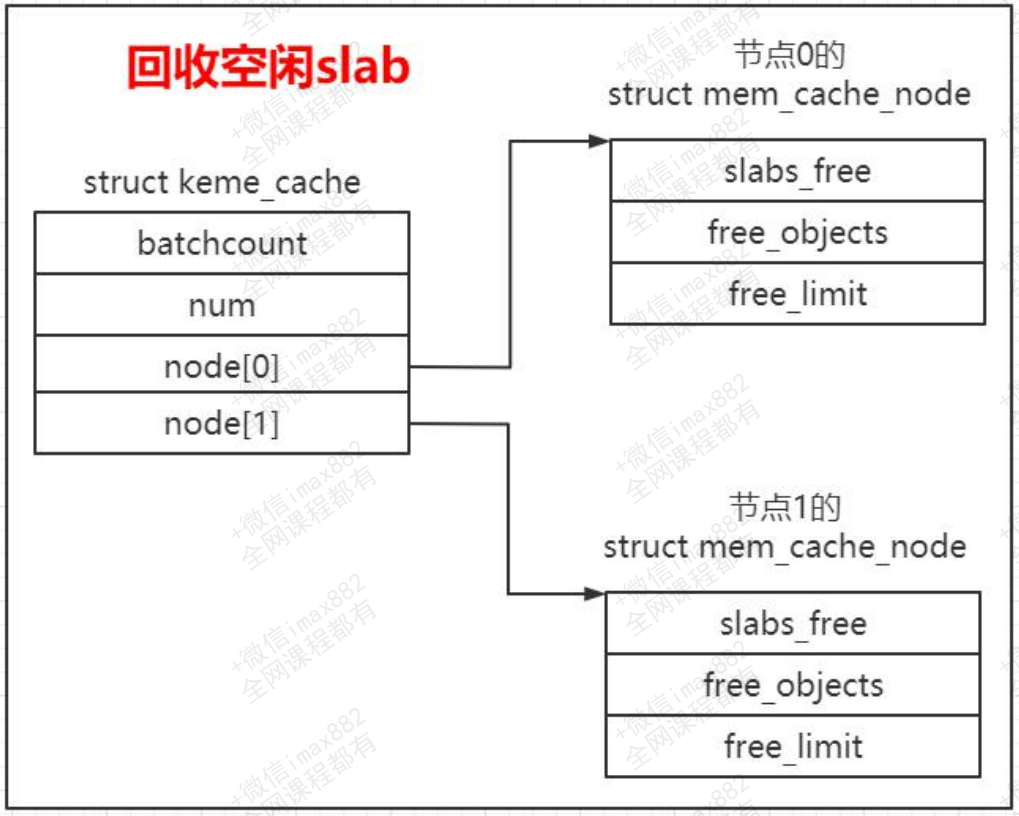
三、每处理器数组缓存

内存缓存为每个处理器创建一个数组缓存（结构体array_cache）。释放对象时，把对象存放到当前处理器对应的数组缓存中；分配对象的时候，先从当前处理器的数组缓存分配对象，采用后进先出（LIFO）原则，可以提高性能。



四、回收内存

对于所有对象空间的slab，没有立即释放，而是放在空闲slab链表中。只有内存节点上空闲对象的数量超过限制，才开始回收空闲slab，直到空闲对象的数量小于或等于限制。



办学宗旨：一切只为渴望更优秀的你

办学愿景：让技术简单易懂