代码导读5 – slab分配机制

笨叔叔

课程目标

- 为什么需要有slab分配机制？
- 使用slab机制有什么好处？或者说不使用slab有什么坏处？
- 怎么理解slab机制里面的cache，slab，object概念？
- Slab机制的着色区怎么理解？
- 一个slab可以装载多少个对象？需要多少个物理页面？怎么计算的？
- 使用slab机制分配一个字节的对象时候，实际分配是1个字节吗？
- 使用**kmem_cache_alloc()函数获取空闲对象时候，slab什么时候为这些空闲的对象们分配物理页面**？
- 我使用**kmem_cache_create()函数创建了一个自己的slab cache，为啥我在slabinfo中没找到呢**？
- slab调试的基本技巧

SLAB机制API

#创建slab描述符

```
struct kmem_cache *
kmem_cache_create(const char *name, size_t size, size_t align,
                  unsigned long flags, void (*ctor)(void *))
```

#释放slab描述符

```
void kmem_cache_destroy(struct kmem_cache *s)
```

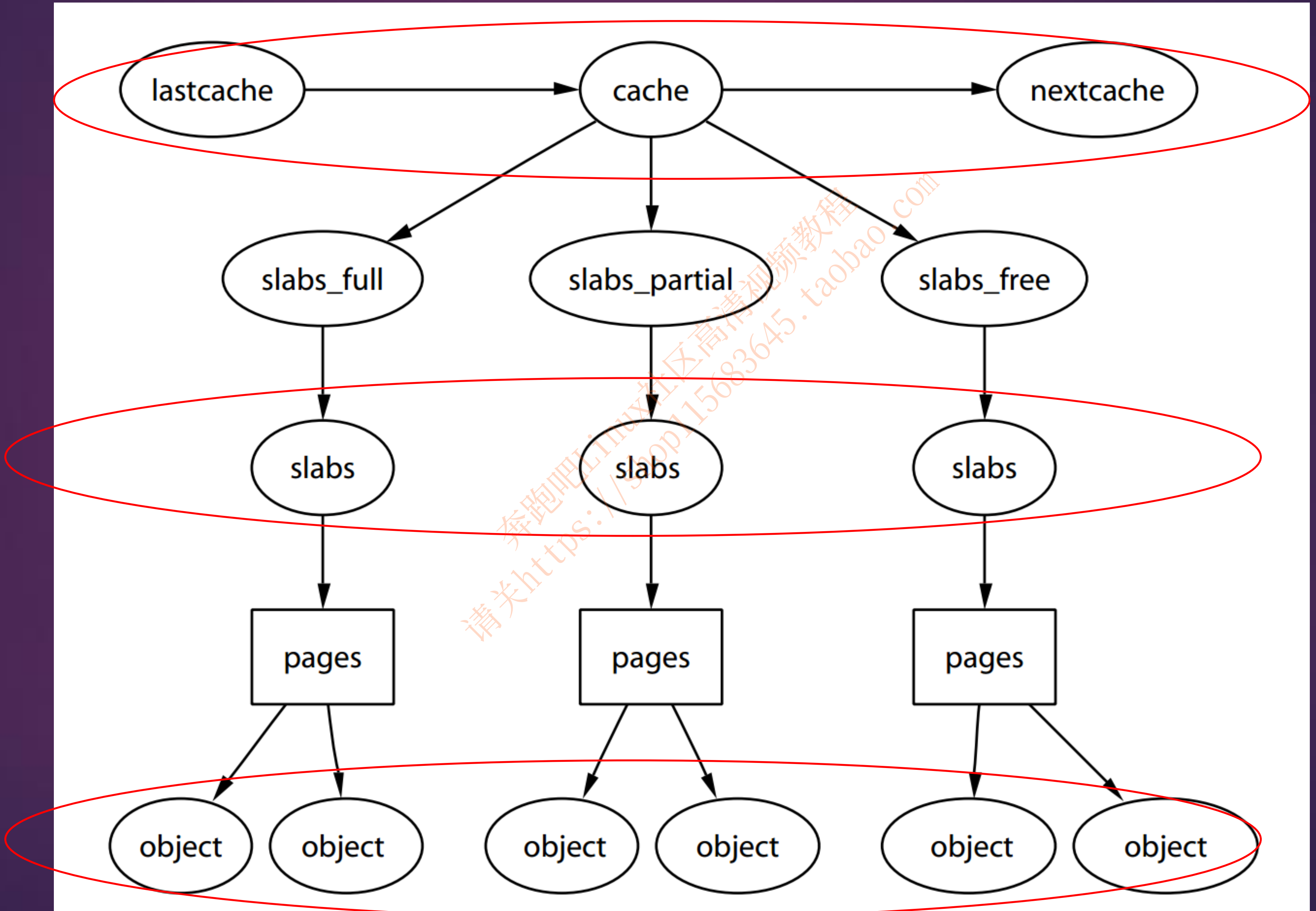
#分配缓存对象

```
void *kmem_cache_alloc(struct kmem_cache *, gfp_t flags);
```

#释放缓存对象

```
void kmem_cache_free(struct kmem_cache *, void *);
```

酒吧 如何存放啤酒的

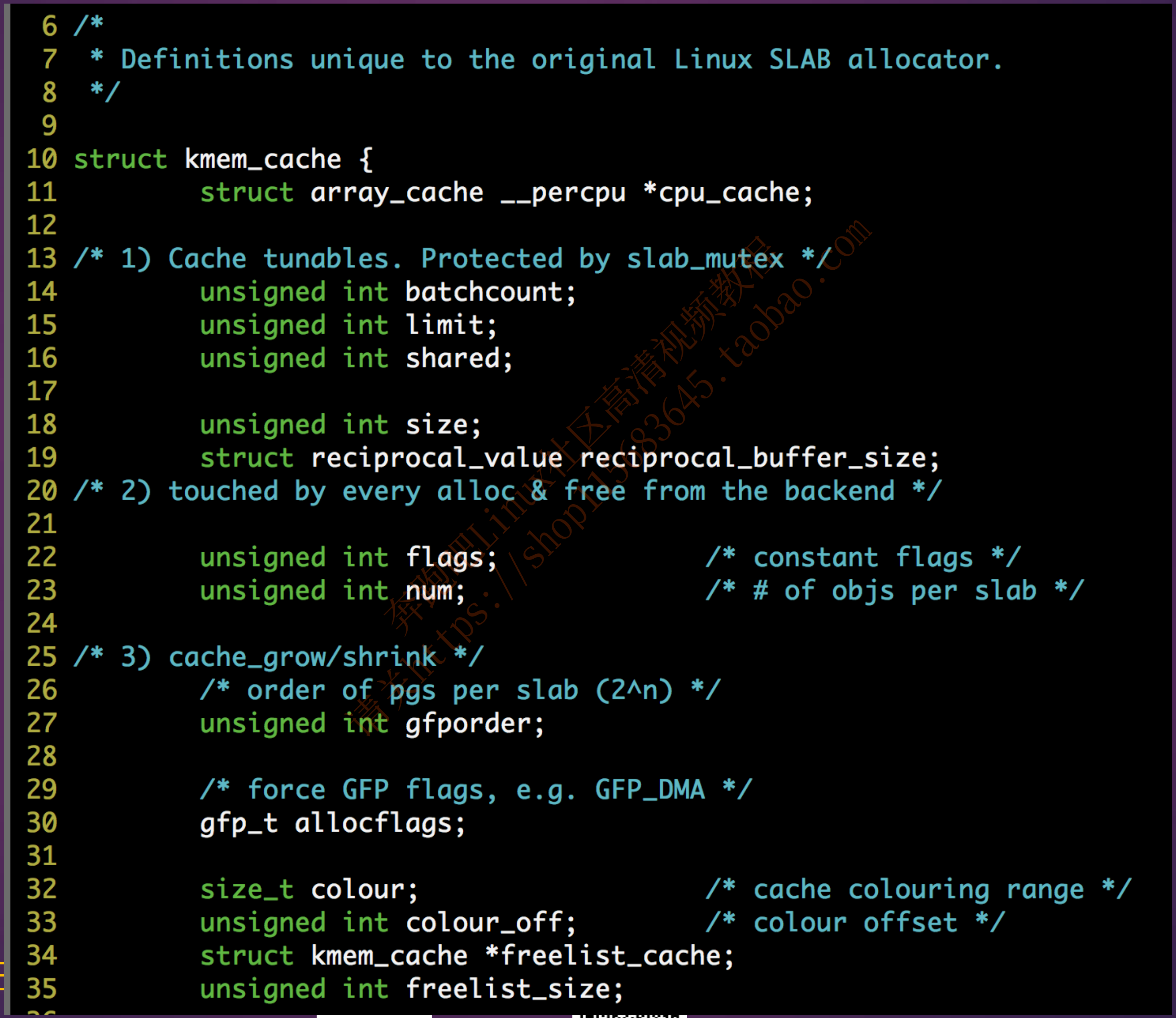


存酒的小仓库

存酒的箱子

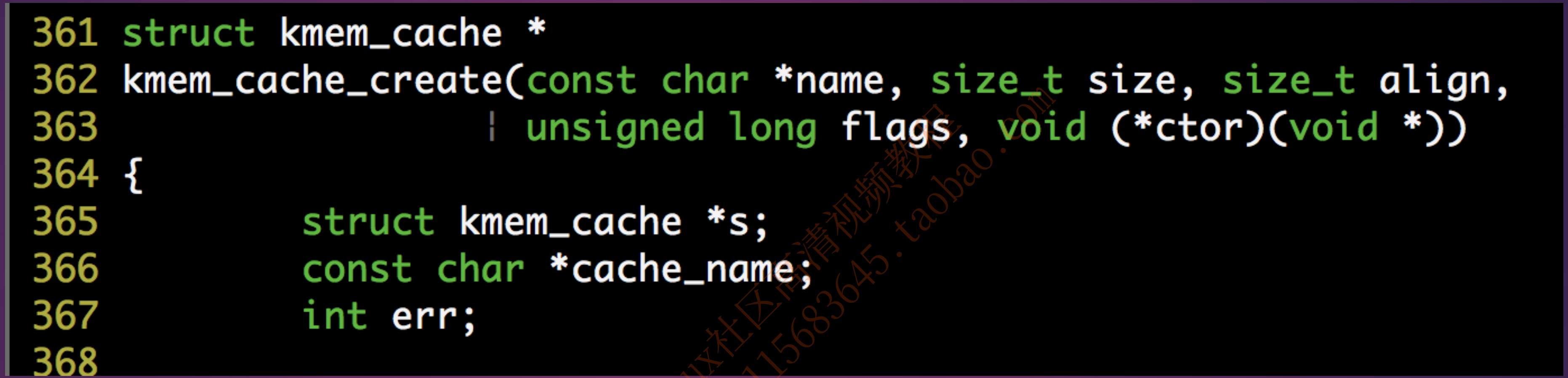
啤酒

酒吧小仓库是如何描述的呢？



```
/*
 * Definitions unique to the original Linux SLAB allocator.
 */

struct kmem_cache {
        struct array_cache __percpu *cpu_cache;

/* 1) Cache tunables. Protected by slab_mutex */
        unsigned int batchcount;
        unsigned int limit;
        unsigned int shared;

        unsigned int size;
        struct reciprocal_value reciprocal_buffer_size;
/* 2) touched by every alloc & free from the backend */

        unsigned int flags;             /* constant flags */
        unsigned int num;               /* # of objs per slab */

/* 3) cache_grow/shrink */
        /* order of pgs per slab (2^n) */
        unsigned int gfporder;

        /* force GFP flags, e.g. GFP_DMA */
        gfp_t allocflags;

        size_t colour;                  /* cache colouring range */
        unsigned int colour_off;        /* colour offset */
        struct kmem_cache *freelist_cache;
        unsigned int freelist_size;
```

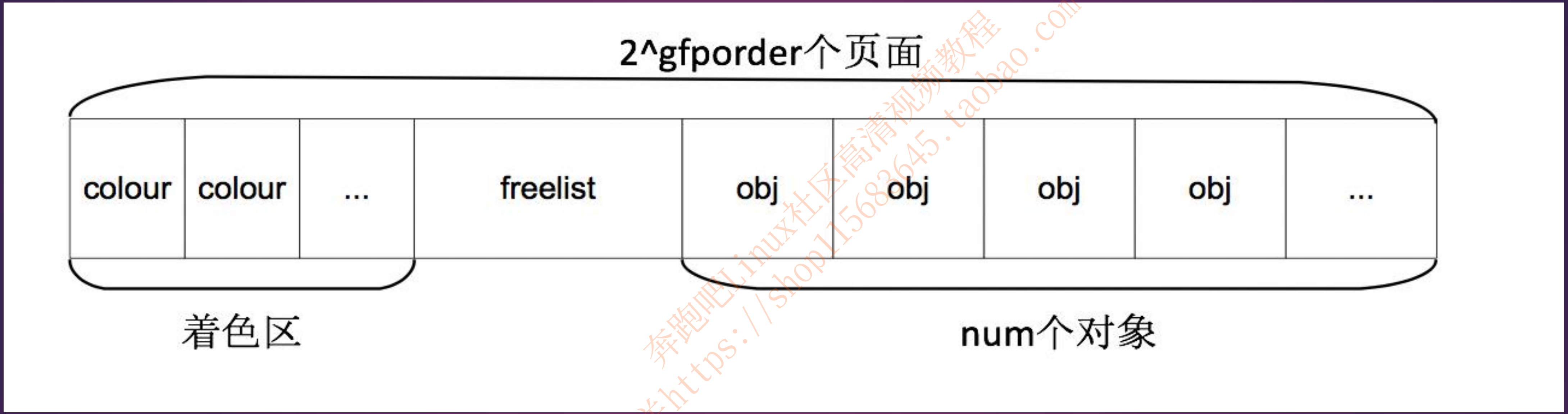
重点分析函数1 – kmem_cache_create



```
struct kmem_cache *
kmem_cache_create(const char *name, size_t size, size_t align,
                  unsigned long flags, void (*ctor)(void *))
{
        struct kmem_cache *s;
        const char *cache_name;
        int err;

```

Name: 仓库的名称
Size： 对象的大小
Align： 对象对齐的要求
Flags： cache分配需求集合
Ctor： 对象额外的构造函数

Slab构造结构图



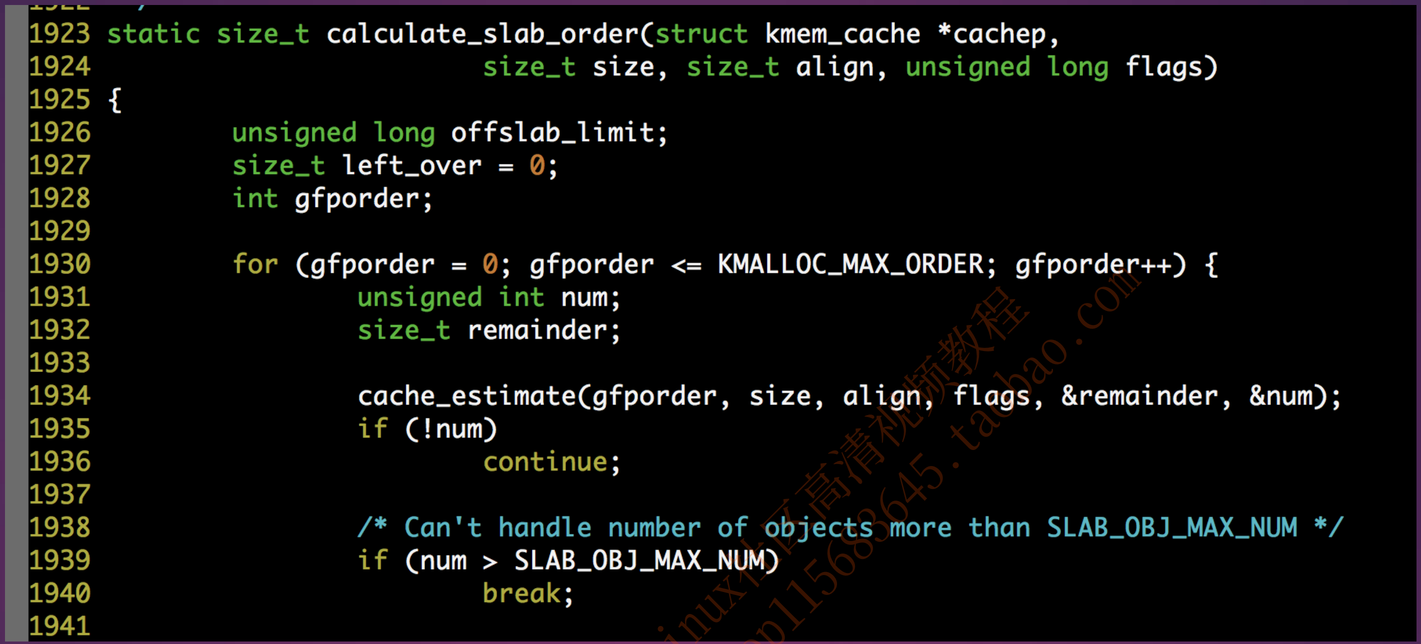
2^gfporder个页面

colour colour ... freelist obj obj obj obj ...

着色区

num个对象

一个slab可以存放几个对象？



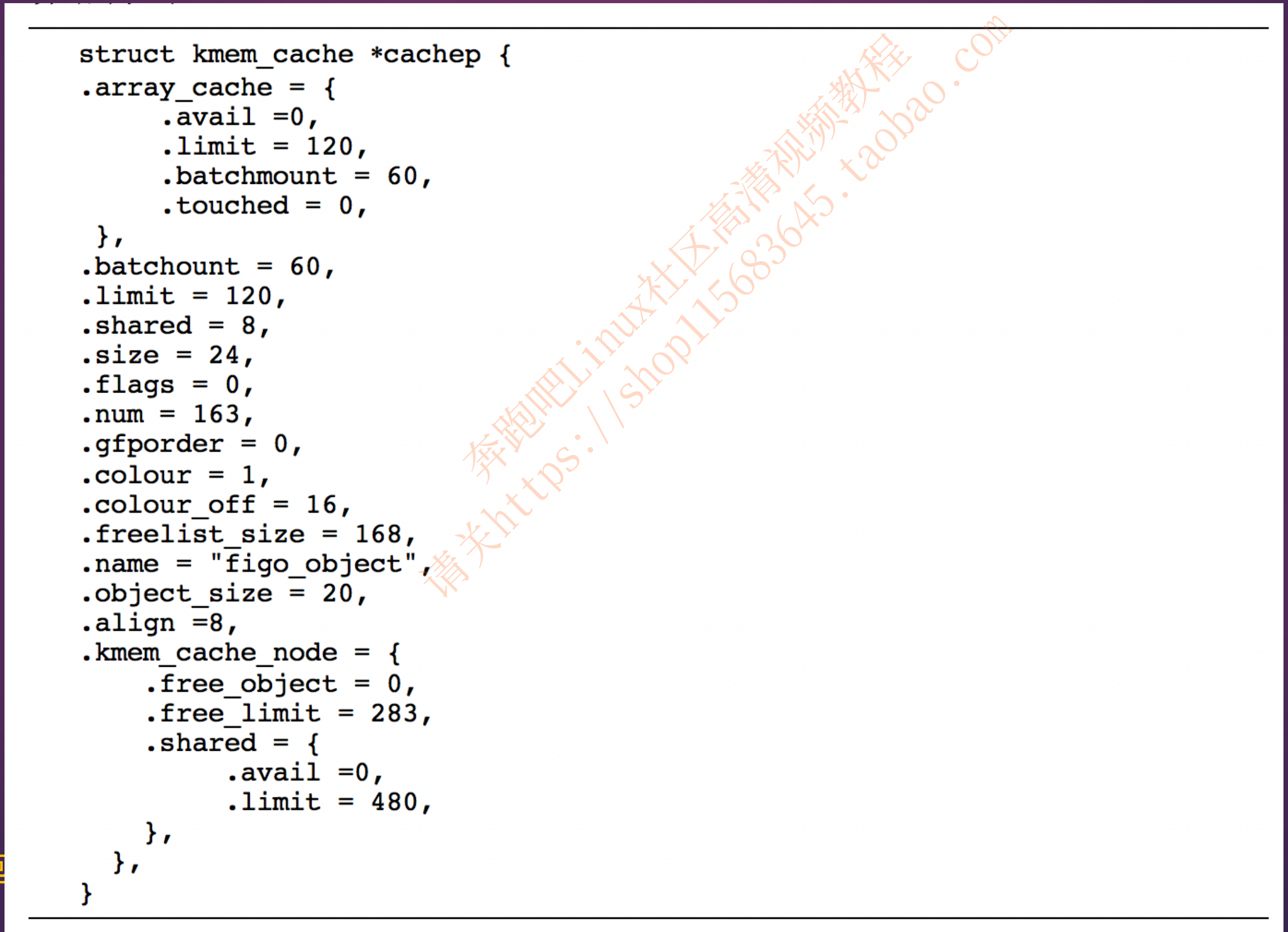
```
static size_t calculate_slab_order(struct kmem_cache *cachep,
                        size_t size, size_t align, unsigned long flags)
{
        unsigned long offslab_limit;
        size_t left_over = 0;
        int gfporder;

        for (gfporder = 0; gfporder <= KMALLOC_MAX_ORDER; gfporder++) {
                unsigned int num;
                size_t remainder;

                cache_estimate(gfporder, size, align, flags, &remainder, &num);
                if (!num)
                        continue;

                /* Can't handle number of objects more than SLAB_OBJ_MAX_NUM */
                if (num > SLAB_OBJ_MAX_NUM)
                        break;

```

计算公式:

obj_num = slab-szie / (obj_size + sizeof(freelist_idx_t))

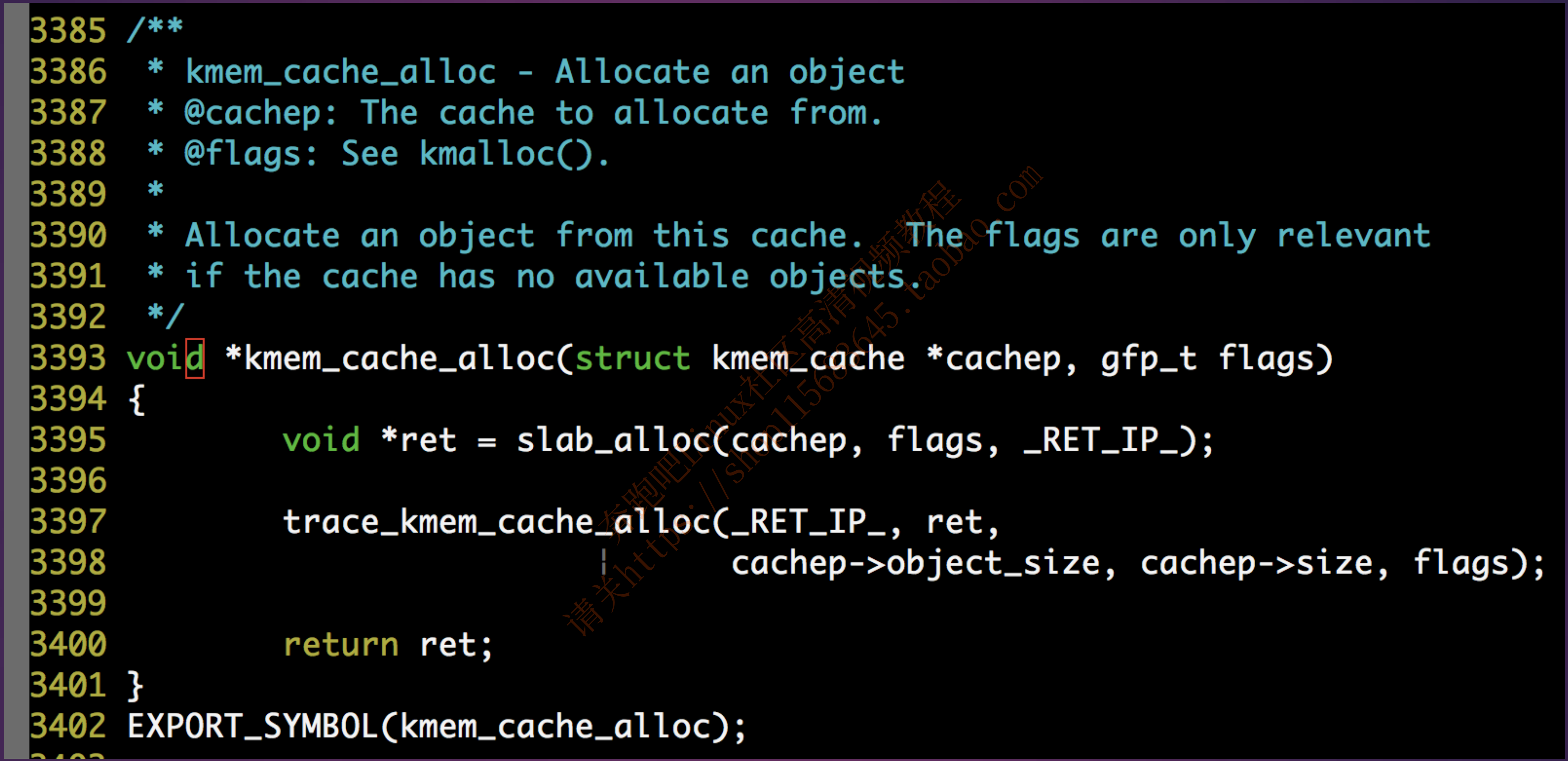
kmem_cache_create创建完后小仓库长啥样？

在ARM Vexpress平台上创建名为“figo_object”的slab描述符，大小为20Byte，align为8Byte， flags为0， 假设L1 cache line大小为16Byte， 其slab描述符相关成员的计算结果如下：



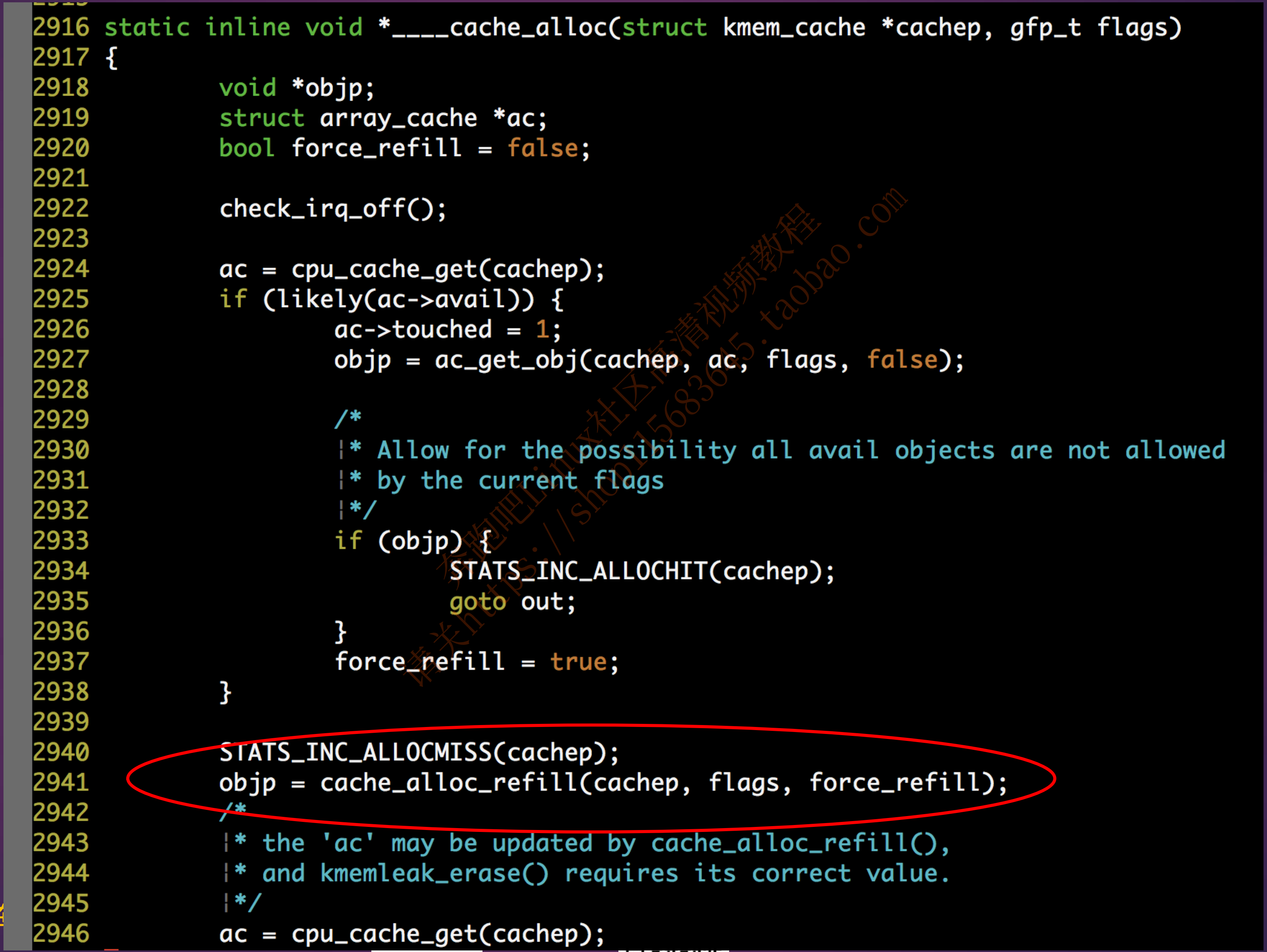
```
struct kmem_cache *cachep {
.array_cache = {
     .avail =0,
     .limit = 120,
     .batchmount = 60,
     .touched = 0,
 },
.batchcount = 60,
.limit = 120,
.shared = 8,
.size = 24,
.flags = 0,
.num = 163,
.gfporder = 0,
.colour = 1,
.colour_off = 16,
.freelist_size = 168,
.name = "figo_object",
.object_size = 20,
.align =8,
.kmem_cache_node = {
    .free_object = 0,
    .free_limit = 283,
    .shared = {
         .avail =0,
         .limit = 480,
    },
  },
}
```

kmem_cache_alloc函数分析



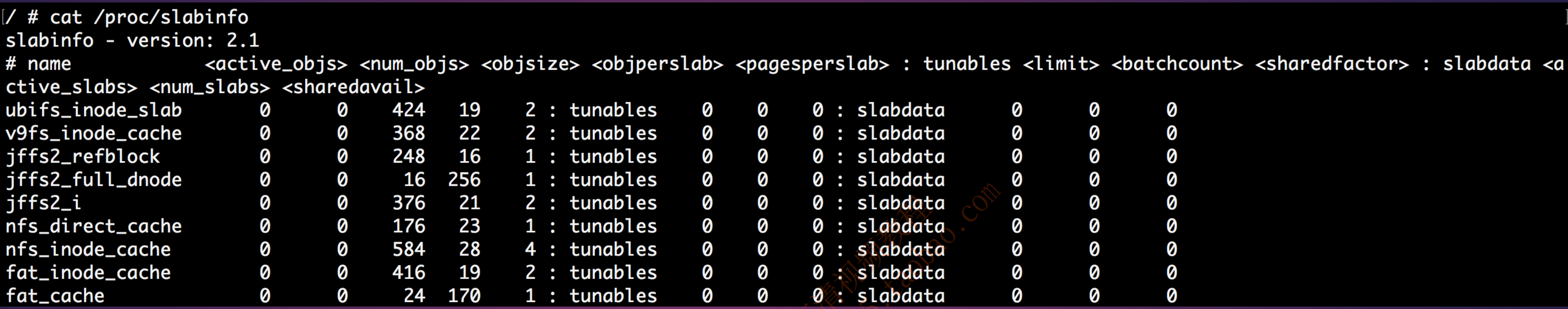
```
/**
 * kmem_cache_alloc - Allocate an object
 * @cachep: The cache to allocate from.
 * @flags: See kmalloc().
 *
 * Allocate an object from this cache.  The flags are only relevant
 * if the cache has no available objects.
 */
void *kmem_cache_alloc(struct kmem_cache *cachep, gfp_t flags)
{
        void *ret = slab_alloc(cachep, flags, _RET_IP_);

        trace_kmem_cache_alloc(_RET_IP_, ret,
                               cachep->object_size, cachep->size, flags);

        return ret;
}
EXPORT_SYMBOL(kmem_cache_alloc);
```

一个新建的小仓库，什么时候进货？



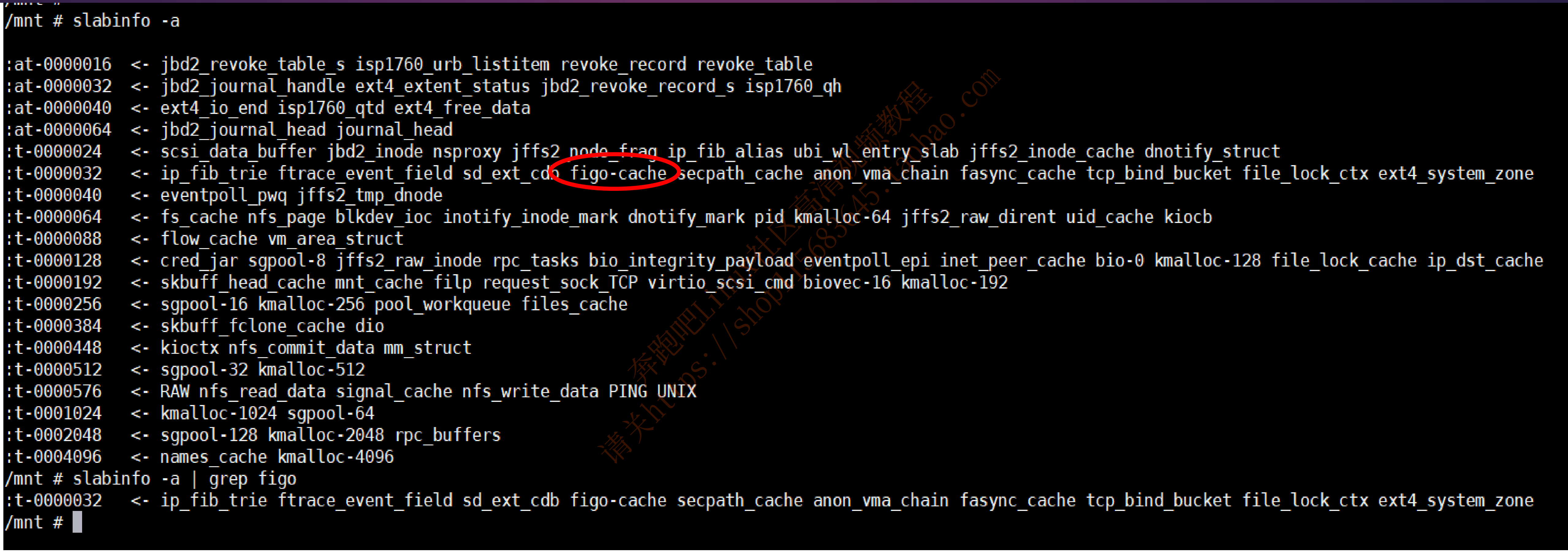
```
static inline void *____cache_alloc(struct kmem_cache *cachep, gfp_t flags)
{
        void *objp;
        struct array_cache *ac;
        bool force_refill = false;

        check_irq_off();

        ac = cpu_cache_get(cachep);
        if (likely(ac->avail)) {
                ac->touched = 1;
                objp = ac_get_obj(cachep, ac, flags, false);

                /*
                 * Allow for the possibility all avail objects are not allowed
                 * by the current flags
                 */
                if (objp) {
                        STATS_INC_ALLOCHIT(cachep);
                        goto out;
                }
                force_refill = true;
        }

        STATS_INC_ALLOCMISS(cachep);
        objp = cache_alloc_refill(cachep, flags, force_refill);
        /*
         * the 'ac' may be updated by cache_alloc_refill(),
         * and kmemleak_erase() requires its correct value.
         */
        ac = cpu_cache_get(cachep);
```

例子1：笨老师，为啥我建的小仓库在slabinfo中没找到？



```
[/ # cat /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount> <sharedfactor> : slabdata <a
ctive_slabs> <num_slabs> <sharedavail>
ubifs_inode_slab       0      0    424   19    2 : tunables    0    0    0 : slabdata      0      0      0
v9fs_inode_cache       0      0    368   22    2 : tunables    0    0    0 : slabdata      0      0      0
jffs2_refblock         0      0    248   16    1 : tunables    0    0    0 : slabdata      0      0      0
jffs2_full_dnode       0      0     16  256    1 : tunables    0    0    0 : slabdata      0      0      0
jffs2_i                0      0    376   21    2 : tunables    0    0    0 : slabdata      0      0      0
nfs_direct_cache       0      0    176   23    1 : tunables    0    0    0 : slabdata      0      0      0
nfs_inode_cache        0      0    584   28    4 : tunables    0    0    0 : slabdata      0      0      0
fat_inode_cache        0      0    416   19    2 : tunables    0    0    0 : slabdata      0      0      0
fat_cache              0      0     24  170    1 : tunables    0    0    0 : slabdata      0      0      0
```

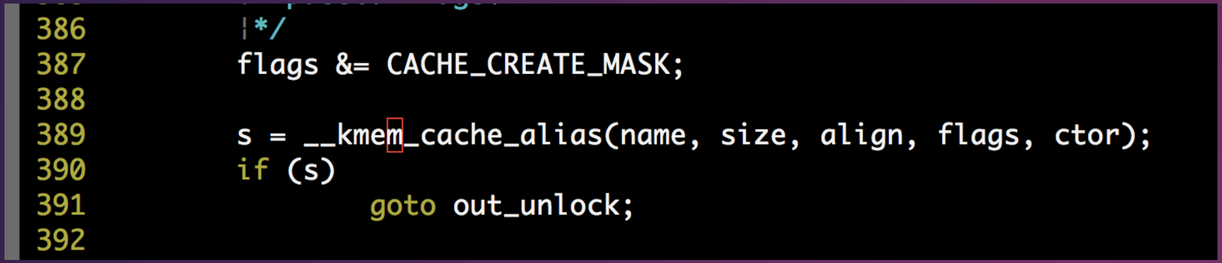
使用slabinfo来查看是否被merged？



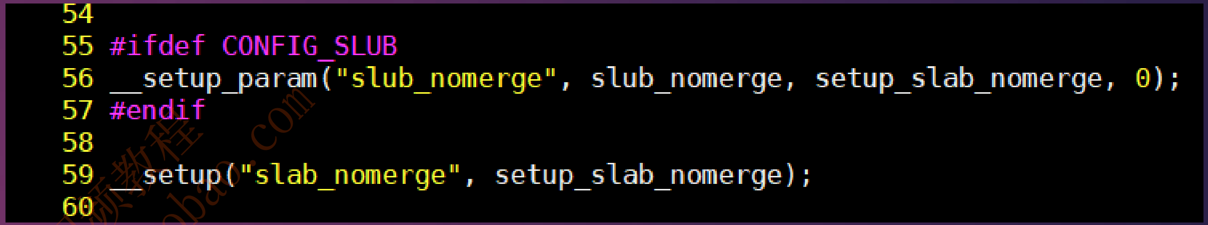
```
/mnt #
/mnt # slabinfo -a

:at-0000016  <- jbd2_revoke_table_s isp1760_urb_listitem revoke_record revoke_table
:at-0000032  <- jbd2_journal_handle ext4_extent_status jbd2_revoke_record_s isp1760_qh
:at-0000040  <- ext4_io_end isp1760_qtd ext4_free_data
:at-0000064  <- jbd2_journal_head journal_head
:t-0000024   <- scsi_data_buffer jbd2_inode nsproxy jffs2_node_frag ip_fib_alias ubi_wl_entry_slab jffs2_inode_cache dnotify_struct
:t-0000032   <- ip_fib_trie ftrace_event_field sd_ext_cdb figo-cache secpath_cache anon_vma_chain fasync_cache tcp_bind_bucket file_lock_ctx ext4_system_zone
:t-0000040   <- eventpoll_pwq jffs2_tmp_dnode
:t-0000064   <- fs_cache nfs_page blkdev_ioc inotify_inode_mark dnotify_mark pid kmalloc-64 jffs2_raw_dirent uid_cache kiocb
:t-0000088   <- flow_cache vm_area_struct
:t-0000128   <- cred_jar sgpool-8 jffs2_raw_inode rpc_tasks bio_integrity_payload eventpoll_epi inet_peer_cache bio-0 kmalloc-128 file_lock_cache ip_dst_cache
:t-0000192   <- skbuff_head_cache mnt_cache filp request_sock_TCP virtio_scsi_cmd biovec-16 kmalloc-192
:t-0000256   <- sgpool-16 kmalloc-256 pool_workqueue files_cache
:t-0000384   <- skbuff_fclone_cache dio
:t-0000448   <- kioctx nfs_commit_data mm_struct
:t-0000512   <- sgpool-32 kmalloc-512
:t-0000576   <- RAW nfs_read_data signal_cache nfs_write_data PING UNIX
:t-0001024   <- kmalloc-1024 sgpool-64
:t-0002048   <- sgpool-128 kmalloc-2048 rpc_buffers
:t-0004096   <- names_cache kmalloc-4096
/mnt # slabinfo -a | grep figo
:t-0000032   <- ip_fib_trie ftrace_event_field sd_ext_cdb figo-cache secpath_cache anon_vma_chain fasync_cache tcp_bind_bucket file_lock_ctx ext4_system_zone
/mnt #
```

- 尝试一下：slub_nomerge

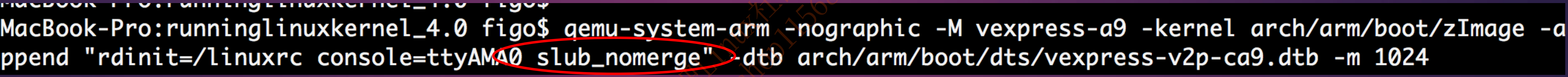
Root cause:



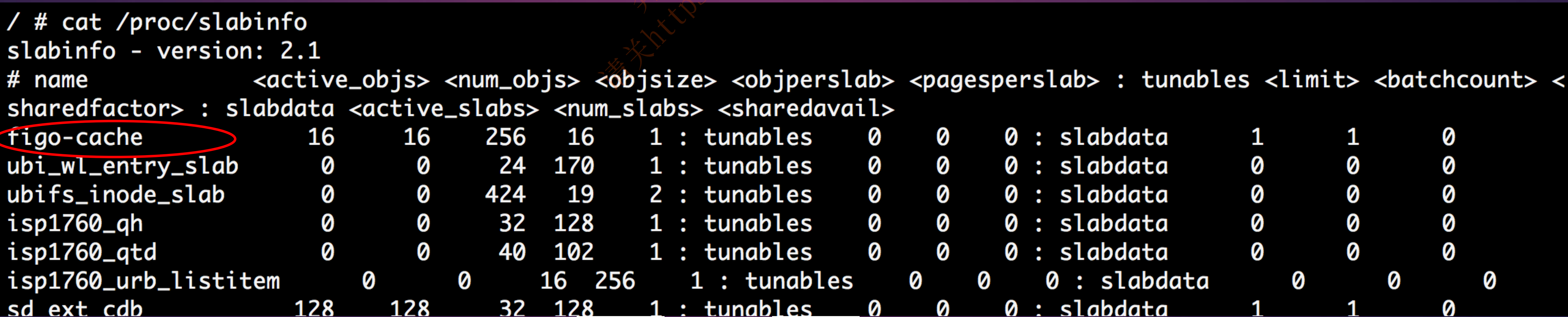
```
         */
        flags &= CACHE_CREATE_MASK;

        s = __kmem_cache_alias(name, size, align, flags, ctor);
        if (s)
                goto out_unlock;

```



```

#ifdef CONFIG_SLUB
__setup_param("slub_nomerge", slub_nomerge, setup_slab_nomerge, 0);
#endif

__setup("slab_nomerge", setup_slab_nomerge);

```

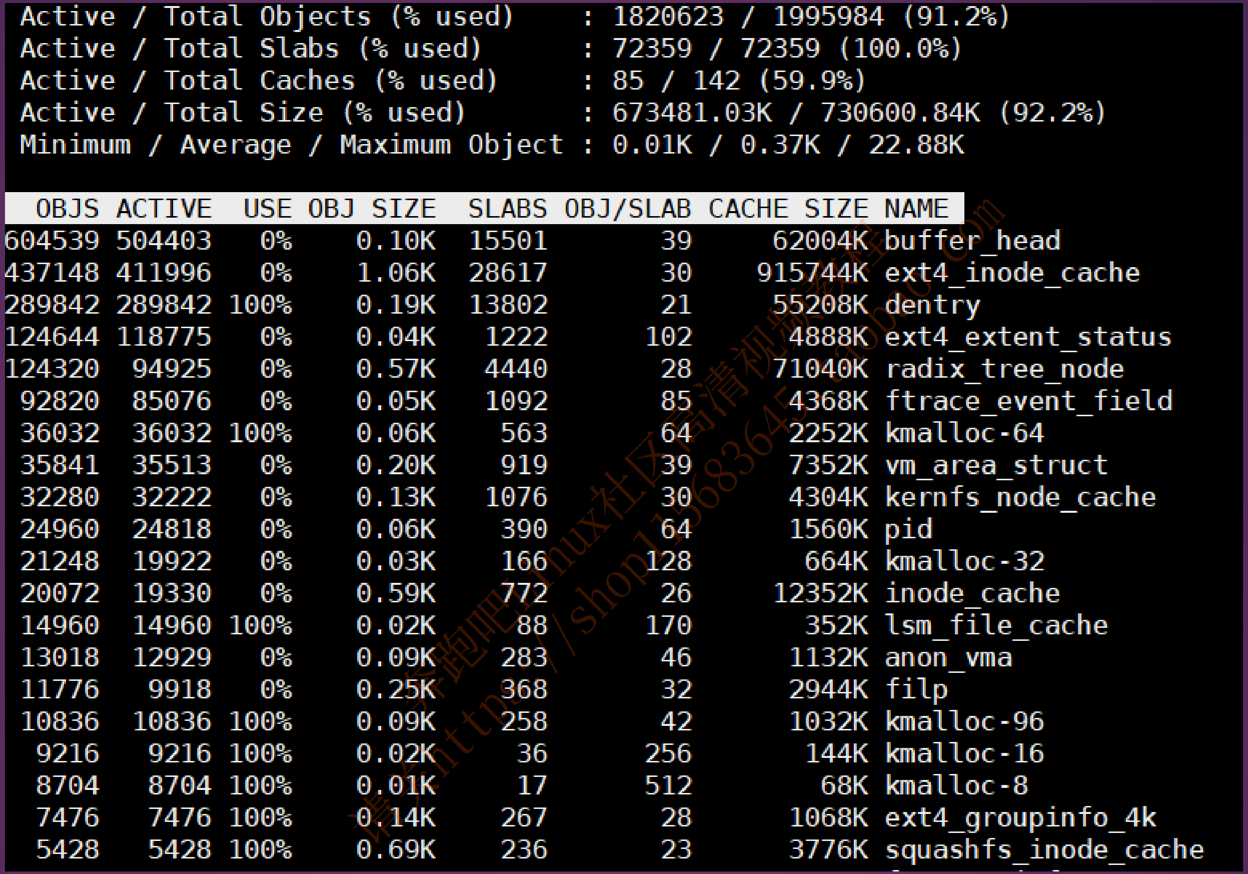


```
MacBook-Pro:runninglinuxkernel_4.0 figo$ qemu-system-arm -nographic -M vexpress-a9 -kernel arch/arm/boot/zImage -a
ppend "rdinit=/linuxrc console=ttyAMA0 slub_nomerge" -dtb arch/arm/boot/dts/vexpress-v2p-ca9.dtb -m 1024
```



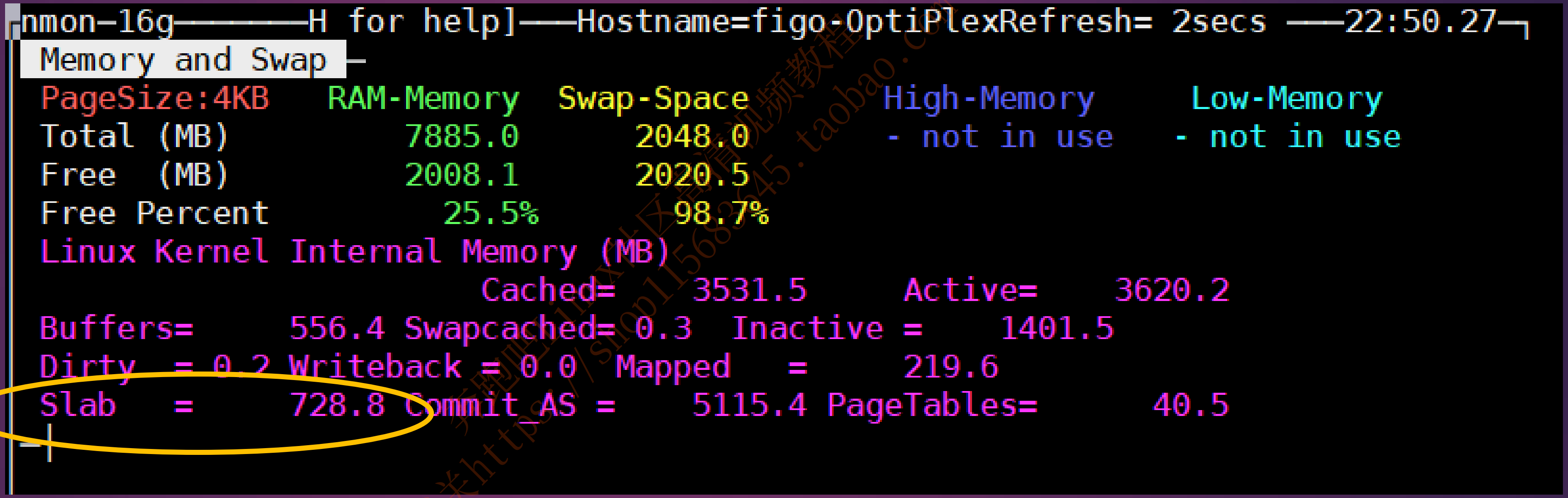
```
/ # cat /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount> <
sharedfactor> : slabdata <active_slabs> <num_slabs> <sharedavail>
figo-cache            16     16    256   16    1 : tunables    0    0    0 : slabdata      1      1      0
ubi_wl_entry_slab      0      0     24  170    1 : tunables    0    0    0 : slabdata      0      0      0
ubifs_inode_slab       0      0    424   19    2 : tunables    0    0    0 : slabdata      0      0      0
isp1760_qh             0      0     32  128    1 : tunables    0    0    0 : slabdata      0      0      0
isp1760_qtd            0      0     40  102    1 : tunables    0    0    0 : slabdata      0      0      0
isp1760_urb_listitem      0      0     16  256    1 : tunables    0    0    0 : slabdata      0      0      0
sd_ext_cdb           128    128     32  128    1 : tunables    0    0    0 : slabdata      1      1      0
```

slabtop工具



```
 Active / Total Objects (% used)    : 1820623 / 1995984 (91.2%)
 Active / Total Slabs (% used)      : 72359 / 72359 (100.0%)
 Active / Total Caches (% used)     : 85 / 142 (59.9%)
 Active / Total Size (% used)       : 673481.03K / 730600.84K (92.2%)
 Minimum / Average / Maximum Object : 0.01K / 0.37K / 22.88K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
604539 504403   0%    0.10K  15501       39     62004K buffer_head
437148 411996   0%    1.06K  28617       30    915744K ext4_inode_cache
289842 289842 100%    0.19K  13802       21     55208K dentry
124644 118775   0%    0.04K   1222      102      4888K ext4_extent_status
124320  94925   0%    0.57K   4440       28     71040K radix_tree_node
 92820  85076   0%    0.05K   1092       85      4368K ftrace_event_field
 36032  36032 100%    0.06K    563       64      2252K kmalloc-64
 35841  35513   0%    0.20K    919       39      7352K vm_area_struct
 32280  32222   0%    0.13K   1076       30      4304K kernfs_node_cache
 24960  24818   0%    0.06K    390       64      1560K pid
 21248  19922   0%    0.03K    166      128       664K kmalloc-32
 20072  19330   0%    0.59K    772       26     12352K inode_cache
 14960  14960 100%    0.02K     88      170       352K lsm_file_cache
 13018  12929   0%    0.09K    283       46      1132K anon_vma
 11776   9918   0%    0.25K    368       32      2944K filp
 10836  10836 100%    0.09K    258       42      1032K kmalloc-96
  9216   9216 100%    0.02K     36      256       144K kmalloc-16
  8704   8704 100%    0.01K     17      512        68K kmalloc-8
  7476   7476 100%    0.14K    267       28      1068K ext4_groupinfo_4k
  5428   5428 100%    0.69K    236       23      3776K squashfs_inode_cache
```

怀疑slab吃掉很多内存？用nmon看看



```
┌nmon─16g────────H for help]───Hostname=figo-OptiPlexRefresh= 2secs ───22:50.27┐
│ Memory and Swap ─
│ PageSize:4KB   RAM-Memory  Swap-Space       High-Memory     Low-Memory
│ Total (MB)         7885.0      2048.0       - not in use    - not in use
│ Free  (MB)         2008.1      2020.5
│ Free Percent         25.5%       98.7%
│ Linux Kernel Internal Memory (MB)
│                        Cached=    3531.5     Active=    3620.2
│ Buffers=     556.4 Swapcached= 0.3  Inactive =    1401.5
│ Dirty  = 0.2 Writeback = 0.0  Mapped   =      219.6
│ Slab   =     728.8 Commit_AS =    5115.4 PageTables=      40.5
│─┤
```

使用slub_debug进行内存泄漏检测

- 详情见《奔跑吧Linux内核》第6.4.1章内容

Thanks