Linux 内核中创建 slab 主要由函数 cache_grow()实现，从 slab 的创建中我们可以完整地看到 slab 与对象、页面的组织方式。

[cpp] view plaincopyprint?

```
1. /*
2.  * Grow (by 1) the number of slabs within a cache.  This is called by
3.  * kmem_cache_alloc() when there are no active objs left in a cache.
4.  */
5.  /*使用一个或多个页面创建一个空 slab。
6. objp：页面虚拟地址，为空表示还未申请内存页，不为空
7. ，说明已申请内存页，可直接用来创建 slab*/
8. static int cache_grow(struct kmem_cache *cachep,
9.         gfp_t flags, int nodeid, void *objp)
10.{
11.    struct slab *slabp;
12.    size_t offset;
13.    gfp_t local_flags;
14.    struct kmem_list3 *l3;
15.
16.    /*
17.     * Be lazy and only check for valid flags here,  keeping it out of the
18.     * critical path in kmem_cache_alloc().
19.     */
20.    BUG_ON(flags & GFP_SLAB_BUG_MASK);
21.    local_flags = flags & (GFP_CONSTRAINT_MASK|GFP_RECLAIM_MASK);
22.
23.    /* Take the l3 list lock to change the colour_next on this node */
24.    check_irq_off();
25.     /* 获得本内存节点的 slab 三链 */
26.    l3 = cachep->nodelists[nodeid];
27.    spin_lock(&l3->list_lock);
28.
29.    /* Get colour for the slab, and cal the next value. */
30.    /* 获得本 slab 的着色区偏移 */
31.    offset = l3->colour_next;
32.     /* 更新着色区偏移，使不同 slab 的着色偏移不同 */
33.    l3->colour_next++;
34.    /* 不能超过着色区的总大小，如果超过了，重置为 0。这就是前面分析过的着色循环问题
35.    。事实上，如果 slab 中浪费的空间很少，那么很快就会循环一次。*/
36.    if (l3->colour_next >= cachep->colour)
37.        l3->colour_next = 0;
38.    spin_unlock(&l3->list_lock);
39.    /* 将着色单位区间的个数转换为着色区大小 */
```

```
40.  offset *= cachep->colour_off;
41.
42.  if (local_flags & __GFP_WAIT)
43.      local_irq_enable();
44.
45.  /*
46.   * The test for missing atomic flag is performed here, rather than
47.   * the more obvious place, simply to reduce the critical path length
48.   * in kmem_cache_alloc(). If a caller is seriously mis-behaving they
49.   * will eventually be caught here (where it matters).
50.   */
51.  kmem_flagcheck(cachep, flags);
52.
53.  /*
54.   * Get mem for the objs.  Attempt to allocate a physical page from
55.   * 'nodeid'.
56.   */
57.  if (!objp)/* 还未分配页面，从本内存节点分配 1<<cachep->gfporder 个页面
58.   ，objp 为 slab 首页面的虚拟地址 */
59.      objp = kmem_getpages(cachep, local_flags, nodeid);
60.  if (!objp)
61.      goto failed;
62.
63.  /* Get slab management. */
64.   /* 分配 slab 管理对象 */
65.  slabp = alloc_slabmgmt(cachep, objp, offset,
66.          local_flags & ~GFP_CONSTRAINT_MASK, nodeid);
67.  if (!slabp)
68.      goto opps1;
69.  /* 设置 page 到 cache、slab 的映射 */
70.     slab_map_pages(cachep, slabp, objp);
71.
72.     /* 初始化 slab 中的对象 */
73.  cache_init_objs(cachep, slabp);
74.
75.  if (local_flags & __GFP_WAIT)
76.      local_irq_disable();
77.  check_irq_off();
78.  spin_lock(&l3->list_lock);
79.
80.  /* Make slab active. */
81.  list_add_tail(&slabp->list, &(l3->slabs_free));
82.  /* 更新本 cache 增长计数 */
```

```
83.    STATS_INC_GROWN(cachep);
84.    /* 更新 slab 链表中空闲对象计数 */
85.    l3->free_objects += cachep->num;
86.    spin_unlock(&l3->list_lock);
87.    return 1;
88.opps1:
89.    kmem_freepages(cachep, objp);
90.failed:
91.    if (local_flags & __GFP_WAIT)
92.        local_irq_disable();
93.    return 0;
94.}
```

执行流程：

1，从 cache 结构中获得并计算着色区偏移量；

2，从伙伴系统中获得 1<<cachep->gfporder 个页面用于 slab；

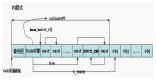
3，初始化 slab 中相关变量，如果是外置式 slab 需要从新申请 slab 管理区的空间，由函数 alloc_slabmgmt()实现。

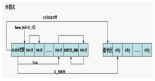
[cpp] view plaincopyprint?

```
1.  /*分配 slab 管理对象*/
2.  static struct slab *alloc_slabmgmt(struct kmem_cache *cachep, void *objp,
3.                  int colour_off, gfp_t local_flags,
4.                  int nodeid)
5.  {
6.      struct slab *slabp;
7.
8.      if (OFF_SLAB(cachep)) {
9.          /* Slab management obj is off-slab. */
10.         /* 外置式 slab。从 general slab cache 中分配一个管理对象，
11.         slabp_cache 指向保存有 struct slab 对象的 general slab cache。
12.         slab 初始化阶段 general slab cache 可能还未创建，slabp_cache 指针为空
13.          ，故初始化阶段创建的 slab 均为内置式 slab。*/
14.         slabp = kmem_cache_alloc_node(cachep->slabp_cache,
15.                         local_flags, nodeid);
16.         /*
17.          * If the first object in the slab is leaked (it's allocated
18.          * but no one has a reference to it), we want to make sure
19.          * kmemleak does not treat the ->s_mem pointer as a reference
20.          * to the object. Otherwise we will not report the leak.
21.          *//* 对第一个对象做检查 */
```

```
22.      kmemleak_scan_area(slabp, offsetof(struct slab, list),
23.                sizeof(struct list_head), local_flags);
24.      if (!slabp)
25.         return NULL;
26.   } else {/* 内置式 slab。objp 为 slab 首页面的虚拟地址，加上着色偏移
27.    ，得到 slab 管理对象的虚拟地址 */
28.      slabp = objp + colour_off;
29.      /* 计算 slab 中第一个对象的页内偏移，slab_size 保存 slab 管理对象的大小
30.       ，包含 struct slab 对象和 kmem_bufctl_t 数组 */
31.      colour_off += cachep->slab_size;
32.   } /* 在用（已分配）对象数为 0 */
33.   slabp->inuse = 0;
34.   /* 第一个对象的页内偏移，可见对于内置式 slab，colouroff 成员不仅包括着色区
35.    ，还包括管理对象占用的空间
36.    ，外置式 slab，colouroff 成员只包括着色区。 */
37.   slabp->colouroff = colour_off;
38.   /* 第一个对象的虚拟地址 */
39.   slabp->s_mem = objp + colour_off;
40.   /* 内存节点 ID */
41.   slabp->nodeid = nodeid;
42.   /* 第一个空闲对象索引为 0，即 kmem_bufctl_t 数组的第一个元素 */
43.   slabp->free = 0;
44.   return slabp;
45.}
```

通过初始化，我们画出下面图像。





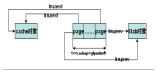
4，设置 slab 中页面（1<<cachep->gfporder 个）到 slab、cache 的映射。这样，可以通过 page 的 lru 链表找到 page 所属的 slab 和 cache。slab_map_pages()实现

[cpp] view plaincopyprint?

```
1. /*设置 page 到 cache、slab 的指针，这样就能知道页面所在的 cache、slab
2.     addr：slab 首页面虚拟地址*/
3. static void slab_map_pages(struct kmem_cache *cache, struct slab *slab,
4.                 void *addr)
5. {
6.     int nr_pages;
7.     struct page *page;
```

```
8.    /* 获得 slab 首页面*/
9.    page = virt_to_page(addr);
10.
11.   nr_pages = 1;
12.    /* 如果不是大页面（关于大页面请参阅相关文档）
13.     ，计算页面的个数 */
14.   if (likely(!PageCompound(page)))
15.       nr_pages <<= cache->gfporder;
16.
17.   do {
18.       /* struct page 结构中的 lru 根据页面的用途有不同的含义
19.        ，当页面空闲或用于高速缓存时，
20.       lru 成员用于构造双向链表将 page 串联起来，而当 page 用于 slab 时，
21.       next 指向 page 所在的 cache，prev 指向 page 所在的 slab */
22.       page_set_cache(page, cache);
23.       page_set_slab(page, slab);
24.       page++;
25.   } while (--nr_pages);
26.}
```

代码实现结果如下图



5，初始化 slab 中 kmem_bufctl_t[]数组，其中 kmem_bufctl_t[]数组为一个静态链表，指定了 slab 对象（obj)的访问顺序。即 kmem_bufctl_t[]中存放的是下一个访问的 obj。在后面分析中 slab_get_obj()函数从 slab 中提取一个空闲对象，他通过 index_to_obj()函数找到空闲对象在 kmem_bufctl_t[]数组中的下标，然后通过 slab_bufctl(slabp)[slabp->free]获得下一个空闲对象的索引并用它更新静态链表。

[cpp] view plaincopyprint?

```
1.  /*初始化 slab 中的对象，主要是通过 kmem_bufctl_t 数组将对象串联起来*/
2.  static void cache_init_objs(struct kmem_cache *cachep,
3.                 struct slab *slabp)
4.  {
5.      int i;
6.      /* 逐一初始化 slab 中的对象 */
7.      for (i = 0; i < cachep->num; i++) {
8.          /* 获得 slab 中第 i 个对象 */
9.          void *objp = index_to_obj(cachep, slabp, i);
10.#if DEBUG
11.         /* need to poison the objs? */
12.         if (cachep->flags & SLAB_POISON)
```

```
13.           poison_obj(cachep, objp, POISON_FREE);
14.       if (cachep->flags & SLAB_STORE_USER)
15.           *dbg_userword(cachep, objp) = NULL;
16.
17.       if (cachep->flags & SLAB_RED_ZONE) {
18.           *dbg_redzone1(cachep, objp) = RED_INACTIVE;
19.           *dbg_redzone2(cachep, objp) = RED_INACTIVE;
20.       }
21.       /*
22.        * Constructors are not allowed to allocate memory from the same
23.        * cache which they are a constructor for.  Otherwise, deadlock.
24.        * They must also be threaded.
25.        */
26.       if (cachep->ctor && !(cachep->flags & SLAB_POISON))
27.           cachep->ctor(objp + obj_offset(cachep));
28.
29.       if (cachep->flags & SLAB_RED_ZONE) {
30.           if (*dbg_redzone2(cachep, objp) != RED_INACTIVE)
31.               slab_error(cachep, "constructor overwrote the"
32.                       " end of an object");
33.           if (*dbg_redzone1(cachep, objp) != RED_INACTIVE)
34.               slab_error(cachep, "constructor overwrote the"
35.                       " start of an object");
36.       }
37.       if ((cachep->buffer_size % PAGE_SIZE) == 0 &&
38.               OFF_SLAB(cachep) && cachep->flags & SLAB_POISON)
39.           kernel_map_pages(virt_to_page(objp),
40.                   cachep->buffer_size / PAGE_SIZE, 0);
41.#else
42.       /* 调用此对象的构造函数 */
43.       if (cachep->ctor)
44.           cachep->ctor(objp);
45.#endif /* 初始时所有对象都是空闲的，只需按照数组顺序串起来即可 */
46.       /*相当于静态索引指针*/
47.       slab_bufctl(slabp)[i] = i + 1;
48.   }
49.   /* 最后一个指向 BUFCTL_END */
50.   slab_bufctl(slabp)[i - 1] = BUFCTL_END;
51.}
```