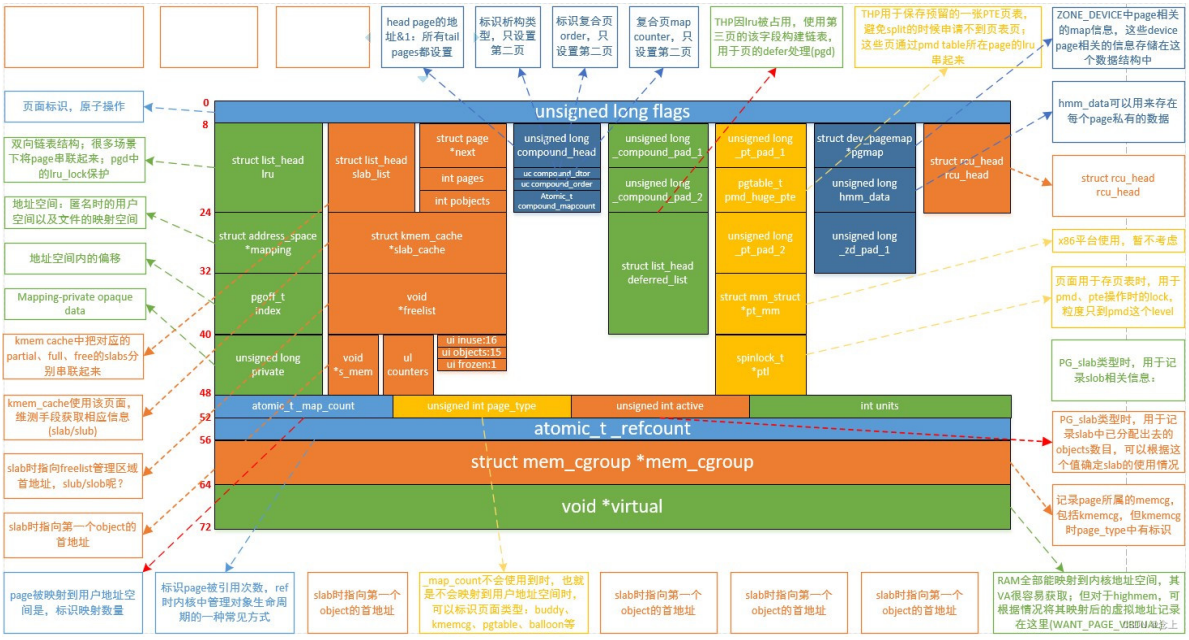
struct_page 解读

by 田亦海10225101529

阅读版本为VM虚拟机内Ubunto Linux （**ubuntu-20.04.6-desktop-amd64**）

所处位置为**/usr/src/linux-hwe-5.15-headers-5.15.0-84/include/linux/mm_types.h**

解读



图源：[struct page数据结构的理解*struct page read only*念上的博客-CSDN博客]

PS: 在第一个union内，图中缺少了struct（page_pool used by netstack）部分，可能是由于linux版本问题。不过其他都非常完美，有助于总体理解

flags

```
    unsigned long flags;        /* Atomic flags, some possibly
                    * updated asynchronously */
```

flags的每一个比特位存储一个状态信息，并且还划分给了section，node id，zone使用。

位于 **include\linux\page-flags.h** 下可以看到具体的含义以及一些宏

page 标志	说明
PG_locked	该页面释放已经上锁，如果已经上锁则置1，其他内核模块不能再访问
PG_referenced	如果该页面最近是否被访问，如果被访问过则置位。用于LRU算法
PG_uptodate	该页面已经从硬盘中成功读取
PG_dirty	该页面是一个脏页，需要将该页面的数据刷新到硬盘中。当页面数据被修改时并不会立即刷新到硬盘中，而是暂时先保证到内存中，等待后面刷新到硬盘中。设置该页为脏页意味着再该页被置换出去之前必须要保证该页不能被-释放
PG_lru	表示该page 位于某个LRU链表中(active、inactive、unevictable LRU中) 。
PG_active	表示该页处于活跃状态
PG_workingset	设置该页为某个进程的woring set，关于working set 可以看下面文章：How To Measure the Working Set Size on Linux
PG_waiters	有进程在等待这个页面
PG_error	该页面在操作IO过程中出现错误
PG_slab	该页被slab所使用
PG_owner_priv_1	被页面的所有者使用，如果是作为pagecache页面，则文件系统有可能使用
PG_arch_1	与体系结构相关的一个状态位，
PG_reserved	该页被保留，不能够被swap out出去。在系统中kernek image(包括vDSO)以及 BIOS，initrd、hw table 以及vememap等待在系统系统初始化阶段就需要做保留以及DAM等常见需要做保留的页 都需要将页状态设置位保留
PG_private	如果page中的private成员非空，则需要设置该标志. 用于I/O的页可使用该字段将页细分为多核缓冲区
PG_private_2	在PG_private基础上的扩充，经常用于aux data
PG_writeback	页面的内存正在向磁盘写
PG_head	该页是一个head page页。在内核中有时需要将多个页组成一个compound pages，而设置该状态时表明该页是compound pages的第一个页
PG_mappedtodisk	该页被映射到硬盘中
PG_reclaim	该页可被回收
PG_swapbacked	该page的后备存储器是swap/ram，一般匿名页才可以回写swap分
PG_unevictable	该page被锁住，不能回收，并会出现在LRU_UNEVICTABLE链表中
PG_mlocked	该页对应的vma被锁住，一般是通过系统调用mlock()锁定了一段内
PG_uncached	该页被设置为不可缓存，需要配置CONFIG_ARCH_USES_PG_UNCACHED
PG_hwpoison	hardware poisoned page. Don't touch，需要配置CONFIG_MEMORY_FAILURE
PG_young	需要CONFIG_IDLE_PAGE_TRACKING和CONFIG_64BIT才支持
PG_idle	需要CONFIG_IDLE_PAGE_TRACKING和CONFIG_64BIT才支持

为了方便进行标位位置位，清零以及查看是否置位等操作，内核做了系列的宏定义，看起来比较复杂：

图源：[linux内核那些事之struct page - 知乎 (zhihu.com)]

比如

```
#define SETPAGEFLAG(uname, lname, policy)                \
static __always_inline void SetPage##uname(struct page *page)       \
    { set_bit(PG_##lname, &policy(page, 1)->flags); }
```

第一个union

包含多个结构体，并附有说明。

```
    /*
     * Five words (20/40 bytes) are available in this union.
     * WARNING: bit 0 of the first word is used for PageTail(). That
     * means the other users of this union MUST NOT use the bit to
     * avoid collision and false-positive PageTail().
     */
```

Page cache and anonymous pages

```
        struct {    /* Page cache and anonymous pages */
            /**
             * @lru: Pageout list, eg. active_list protected by
             * lruvec->lru_lock.  Sometimes used as a generic list
             * by the page owner.
             */
            struct list_head lru;
            /* See page-flags.h for PAGE_MAPPING_FLAGS */
            struct address_space *mapping;
            pgoff_t index;      /* Our offset within mapping. */
            /**
             * @private: Mapping-private opaque data.
             * Usually used for buffer_heads if PagePrivate.
             * Used for swp_entry_t if PageSwapCache.
             * Indicates order in the buddy system if PageBuddy.
             */
            unsigned long private;
        };
```

struct list_head lru：lru 是LRU链表，根据页面不同的用途挂载到不同的链表.如在空闲时刻，被buddy系统管理时，会挂接到buffy的free 链表中

struct address_space *mapping：页面被映射时，指向映射的地址空间。eg：为文件映射时，mapping指向的是struct address_space *结构

pgoff_t index：该字段是一个复用字段。eg：当该页面被文件映射时，代表偏移量。

unsigned long private：私有数据指针, 由应用场景确定其具体的含义。eg: 如果设置了PG_private标志，则private字段指向struct buffer_head。

page_pool used by netstack

```
        struct {    /* page_pool used by netstack */
            /**
             * @pp_magic: magic value to avoid recycling non
             * page_pool allocated pages.
             */
            unsigned long pp_magic;
            struct page_pool *pp;
            unsigned long _pp_mapping_pad;
```

```
            unsigned long dma_addr;
            union {
                /**
                 * dma_addr_upper: might require a 64-bit
                 * value on 32-bit architectures.
                 */
                unsigned long dma_addr_upper;
                /**
                 * For frag page support, not supported in
                 * 32-bit architectures with 64-bit DMA.
                 */
                atomic_long_t pp_frag_count;
            };
        };
```

unsigned long dma_addr:如果该页被用作DMA映射，则其代表的是映射的一个总线地址

unsigned long pp_magic:避免回收非 `page_pool` 分配的页面

关于page_pool 可参考[Page Pool API — The Linux Kernel documentation]

slab, slob and slub

```
        struct {    /* slab, slob and slub */
            union {
                struct list_head slab_list;
                struct {    /* Partial pages */
                    struct page *next;
#ifdef CONFIG_64BIT
                    int pages;  /* Nr of pages left */
                    int pobjects;   /* Approximate count */
#else
                    short int pages;
                    short int pobjects;
#endif
                };
            };
            struct kmem_cache *slab_cache; /* not slob */
            /* Double-word boundary */
            void *freelist;     /* first free object */
            union {
                void *s_mem;    /* slab: first object */
                unsigned long counters;     /* SLUB */
                struct {            /* SLUB */
                    unsigned inuse:16;
                    unsigned objects:15;
                    unsigned frozen:1;
                };
            };
        };
```

用于小内存的分配管理（省略了臃肿的其他信息）

代码中注释已经很完备了

slab：解决内存利用率低及分配效率低，访问时间长的问题

slob：slab缩小版，用于内存较小的设备

slub：slab增强版，解决了slab的一些问题。

深入理解Linux内存管理（八）slab，slob和slub介绍 - 知乎 (zhihu.com)

Tail pages of compound page

```
        struct {    /* Tail pages of compound page */
            unsigned long compound_head;    /* Bit zero is set */

            /* First tail page only */
            unsigned char compound_dtor;
            unsigned char compound_order;
            atomic_t compound_mapcount;
            unsigned int compound_nr; /* 1 << compound_order */
        };
```

compound page将多个连续的物理页组装联合在一起组成一个更大页，其最大的用途是可以创建一个huge 页，具体介绍可以参考：复合页简介(LWN.net)

此段主要描述tail page。

unsigned long compound_head：指向compound pages的第一个 head页。

unsigned char compound_dtor：destructor，每个compound pages都保存对应的destructor

unsigned char compound_order：将compound pages整体页数作为order，只存在head

atomic_t compound_mapcount: compound page被多少个用户进程的page 指向该页。

Second tail page of compound page

```
        struct {    /* Second tail page of compound page */
            unsigned long _compound_pad_1;  /* compound_head */
            atomic_t hpage_pinned_refcount;
            /* For both global and memcg */
            struct list_head deferred_list;
        };
```

为了节省内存，定义了第二种compound tail page结构用于扩展

Page table pages

```
        struct {    /* Page table pages */
            unsigned long _pt_pad_1;    /* compound_head */
            pgtable_t pmd_huge_pte; /* protected by page->ptl */
            unsigned long _pt_pad_2;    /* mapping */
            union {
                struct mm_struct *pt_mm; /* x86 pgds only */
```

```
                atomic_t pt_frag_refcount; /* powerpc */
            };
#if ALLOC_SPLIT_PTLOCKS
            spinlock_t *ptl;
#else
            spinlock_t ptl;
#endif
        };
```

page table 将一个虚拟地址按照多级划分。32位系统下支持二级或三级查找，64位下支持四级或五级查找

ZONE_DEVICE pages

```
        struct {    /* ZONE_DEVICE pages */
            /** @pgmap: Points to the hosting device page map. */
            struct dev_pagemap *pgmap;
            void *zone_device_data;
            /*
             * ZONE_DEVICE private pages are counted as being
             * mapped so the next 3 words hold the mapping, index,
             * and private fields from the source anonymous or
             * page cache page while the page is migrated to device
             * private memory.
             * ZONE_DEVICE MEMORY_DEVICE_FS_DAX pages also
             * use the mapping, index, and private fields when
             * pmem backed DAX files are mapped.
             */
        };
```

ZONE_DEVICE：为支持热插拔设备而分配的Non Volatile Memory非易失性内存（我似懂非懂）

rcu_head

```
        /** @rcu_head: You can use this to free a page by RCU. */
        struct rcu_head rcu_head;
```

RCU锁是 Linux 内核实现的一种针对“读多写少”的共享数据的同步机制。RCU主要针对的数据对象是链表，目的是提高遍历读取数据的效率，为了达到目的使用RCU机制读取数据的时候不对链表进行耗时的加锁操作。RCU机制极大提高"链表"数据结构的读取效率，多个线程同时读取链表时，使用rcu_read_lock()即可，在多线程读取的同时还允许有1个线程修改链表。

[简单讲解Linux内核 RCU锁 - 知乎 (zhihu.com)]

第二个union

```
    union {     /* This union is 4 bytes in size. */
        /*
         * If the page can be mapped to userspace, encodes the number
         * of times this page is referenced by a page table.
         */
        atomic_t _mapcount;

        /*
         * If the page is neither PageSlab nor mappable to userspace,
         * the value stored here may help determine what this page
         * is used for.  See page-flags.h for a list of page types
         * which are currently stored here.
         */
        unsigned int page_type;

        unsigned int active;        /* SLAB */
        int units;          /* SLOB */
    };
```

atomic_t _mapcount : 有多少个page table 映射指向该页面

unsigned int page_type：代表页面类型即使用用途，具体于**include\linux\page-flags.h**文件中

unsigned int active：表示slab中的活跃对象

int units：被slob使用

_refcount

```
    /* Usage count. *DO NOT USE DIRECTLY*. See page_ref.h */
    atomic_t _refcount;
```

用作引用计数管理，用于跟踪内存使用状况。初始化为空闲状态时计数为0，当被分配引用或被其他引用时会+1

PS: 如果该页是一个compound page，则计数会记录在head pages中

源码

```
struct page {
    unsigned long flags;        /* Atomic flags, some possibly
                     * updated asynchronously */
    /*
```

```
     * Five words (20/40 bytes) are available in this union.
     * WARNING: bit 0 of the first word is used for PageTail(). That
     * means the other users of this union MUST NOT use the bit to
     * avoid collision and false-positive PageTail().
     */
    union {
        struct {    /* Page cache and anonymous pages */
            /**
             * @lru: Pageout list, eg. active_list protected by
             * lruvec->lru_lock.  Sometimes used as a generic list
             * by the page owner.
             */
            struct list_head lru;
            /* See page-flags.h for PAGE_MAPPING_FLAGS */
            struct address_space *mapping;
            pgoff_t index;      /* Our offset within mapping. */
            /**
             * @private: Mapping-private opaque data.
             * Usually used for buffer_heads if PagePrivate.
             * Used for swp_entry_t if PageSwapCache.
             * Indicates order in the buddy system if PageBuddy.
             */
            unsigned long private;
        };
        struct {    /* page_pool used by netstack */
            /**
             * @pp_magic: magic value to avoid recycling non
             * page_pool allocated pages.
             */
            unsigned long pp_magic;
            struct page_pool *pp;
            unsigned long _pp_mapping_pad;
            unsigned long dma_addr;
            union {
                /**
                 * dma_addr_upper: might require a 64-bit
                 * value on 32-bit architectures.
                 */
                unsigned long dma_addr_upper;
                /**
                 * For frag page support, not supported in
                 * 32-bit architectures with 64-bit DMA.
                 */
                atomic_long_t pp_frag_count;
            };
        };
        struct {    /* slab, slob and slub */
            union {
                struct list_head slab_list;
                struct {    /* Partial pages */
                    struct page *next;
#ifdef CONFIG_64BIT
                    int pages;  /* Nr of pages left */
                    int pobjects;   /* Approximate count */
#else
```

```
                    short int pages;
                    short int pobjects;
#endif
                };
            };
            struct kmem_cache *slab_cache; /* not slob */
            /* Double-word boundary */
            void *freelist;     /* first free object */
            union {
                void *s_mem;    /* slab: first object */
                unsigned long counters;     /* SLUB */
                struct {            /* SLUB */
                    unsigned inuse:16;
                    unsigned objects:15;
                    unsigned frozen:1;
                };
            };
        };
        struct {    /* Tail pages of compound page */
            unsigned long compound_head;    /* Bit zero is set */

            /* First tail page only */
            unsigned char compound_dtor;
            unsigned char compound_order;
            atomic_t compound_mapcount;
            unsigned int compound_nr; /* 1 << compound_order */
        };
        struct {    /* Second tail page of compound page */
            unsigned long _compound_pad_1;  /* compound_head */
            atomic_t hpage_pinned_refcount;
            /* For both global and memcg */
            struct list_head deferred_list;
        };
        struct {    /* Page table pages */
            unsigned long _pt_pad_1;    /* compound_head */
            pgtable_t pmd_huge_pte; /* protected by page->ptl */
            unsigned long _pt_pad_2;    /* mapping */
            union {
                struct mm_struct *pt_mm; /* x86 pgds only */
                atomic_t pt_frag_refcount; /* powerpc */
            };
#if ALLOC_SPLIT_PTLOCKS
            spinlock_t *ptl;
#else
            spinlock_t ptl;
#endif
        };
        struct {    /* ZONE_DEVICE pages */
            /** @pgmap: Points to the hosting device page map. */
            struct dev_pagemap *pgmap;
            void *zone_device_data;
            /*
             * ZONE_DEVICE private pages are counted as being
             * mapped so the next 3 words hold the mapping, index,
             * and private fields from the source anonymous or
```

```
             * page cache page while the page is migrated to device
             * private memory.
             * ZONE_DEVICE MEMORY_DEVICE_FS_DAX pages also
             * use the mapping, index, and private fields when
             * pmem backed DAX files are mapped.
             */
        };

        /** @rcu_head: You can use this to free a page by RCU. */
        struct rcu_head rcu_head;
    };

    union {     /* This union is 4 bytes in size. */
        /*
         * If the page can be mapped to userspace, encodes the number
         * of times this page is referenced by a page table.
         */
        atomic_t _mapcount;

        /*
         * If the page is neither PageSlab nor mappable to userspace,
         * the value stored here may help determine what this page
         * is used for.  See page-flags.h for a list of page types
         * which are currently stored here.
         */
        unsigned int page_type;

        unsigned int active;        /* SLAB */
        int units;          /* SLOB */
    };

    /* Usage count. *DO NOT USE DIRECTLY*. See page_ref.h */
    atomic_t _refcount;

#ifdef CONFIG_MEMCG
    unsigned long memcg_data;
#endif

    /*
     * On machines where all RAM is mapped into kernel address space,
     * we can simply calculate the virtual address. On machines with
     * highmem some memory is mapped into kernel virtual memory
     * dynamically, so we need a place to store that address.
     * Note that this field could be 16 bits on x86 ... ;)
     *
     * Architectures with slow multiplication can define
     * WANT_PAGE_VIRTUAL in asm/page.h
     */
#if defined(WANT_PAGE_VIRTUAL)
    void *virtual;          /* Kernel virtual address (NULL if
                       not kmapped, ie. highmem) */
#endif /* WANT_PAGE_VIRTUAL */

#ifdef LAST_CPUPID_NOT_IN_PAGE_FLAGS
    int _last_cpupid;
```

```
#endif
} _struct_page_alignment;
```