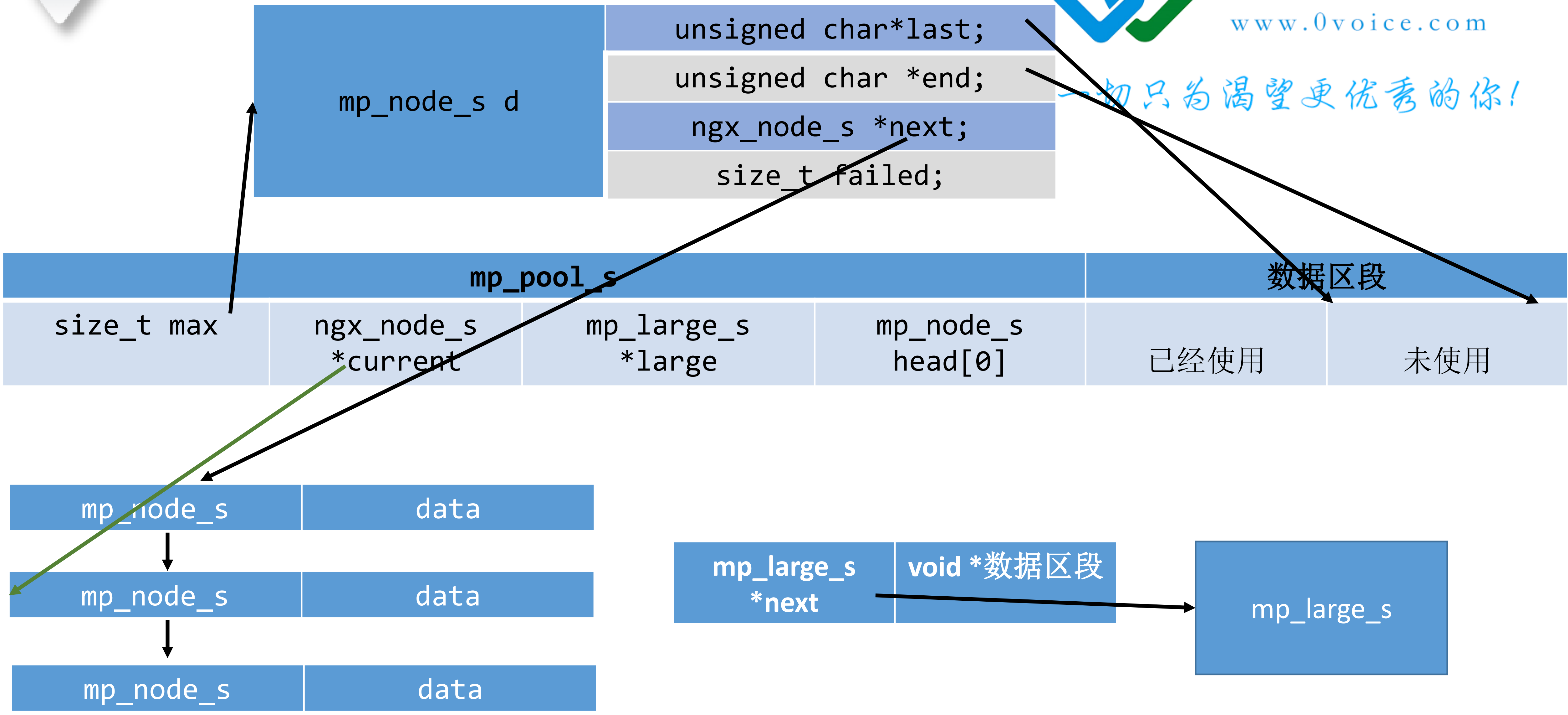
内存池原理与实现

技术交流 King老师 3147964070
往期视频 秋香老师 2207032995
课程咨询 依依老师 2693590767

课程提纲

- 为什么需要内存池
- 内存池的原理
- 内存池的实现
- 内存池的升级与定制

内存池结构体



mp_node_s d
unsigned char*last;
unsigned char *end;
ngx_node_s *next;
size_t failed;
mp_pool_s
数据区段
size_t max
ngx_node_s *current
mp_large_s *large
mp_node_s head[0]
已经使用
未使用
mp_node_s
data
mp_node_s
data
mp_node_s
data
mp_large_s *next
void *数据区段
mp_large_s

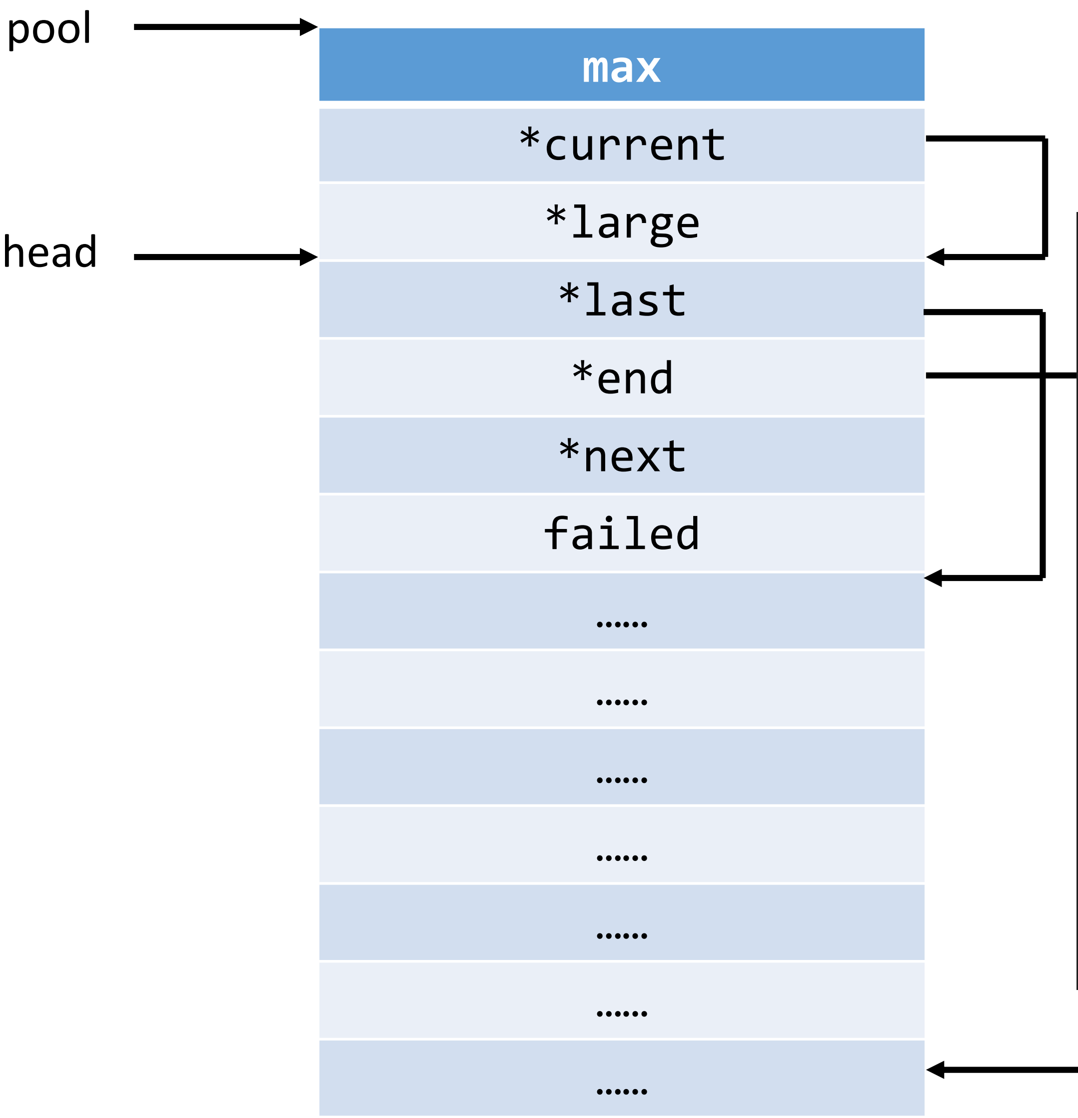
内存池结构体定义与函数定义

零声学院
www.0voice.com
一切只为渴望更优秀的你！

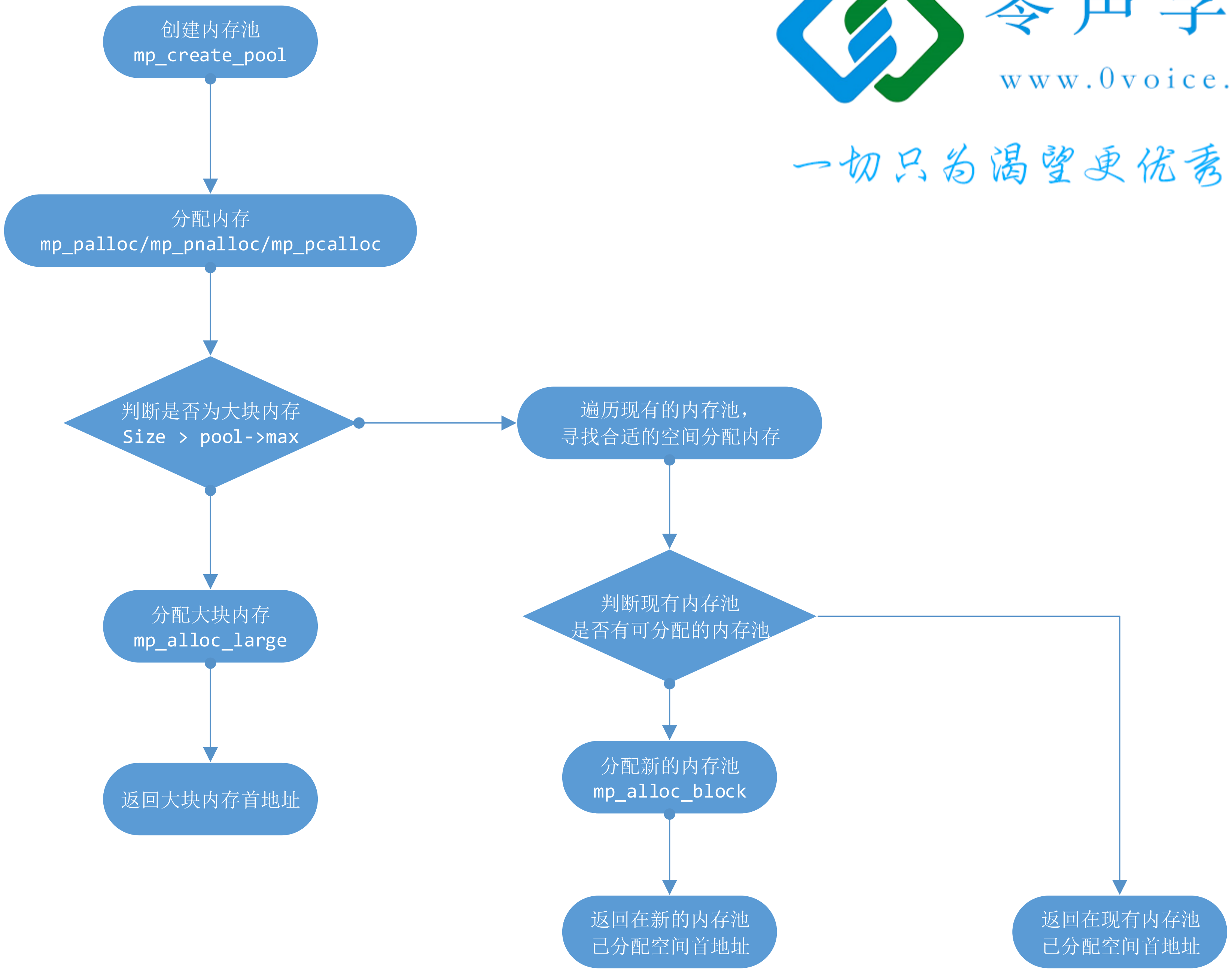
```c
struct mp_large_s {
    struct mp_large_s *next;
    void *alloc;
};

struct mp_node_s {

    unsigned char *last;
    unsigned char *end;

    struct mp_node_s *next;
    size_t failed;
};

struct mp_pool_s {

    size_t max;

    struct mp_node_s *current;
    struct mp_large_s *large;

    struct mp_node_s head[0];

};
```

```c
struct mp_pool_s *mp_create_pool(size_t size);
void mp_destory_pool(struct mp_pool_s *pool);
void *mp_alloc(struct mp_pool_s *pool, size_t size);
void *mp_nalloc(struct mp_pool_s *pool, size_t size);
void *mp_calloc(struct mp_pool_s *pool, size_t size);
void mp_free(struct mp_pool_s *pool, void *p);
```

内存池创建

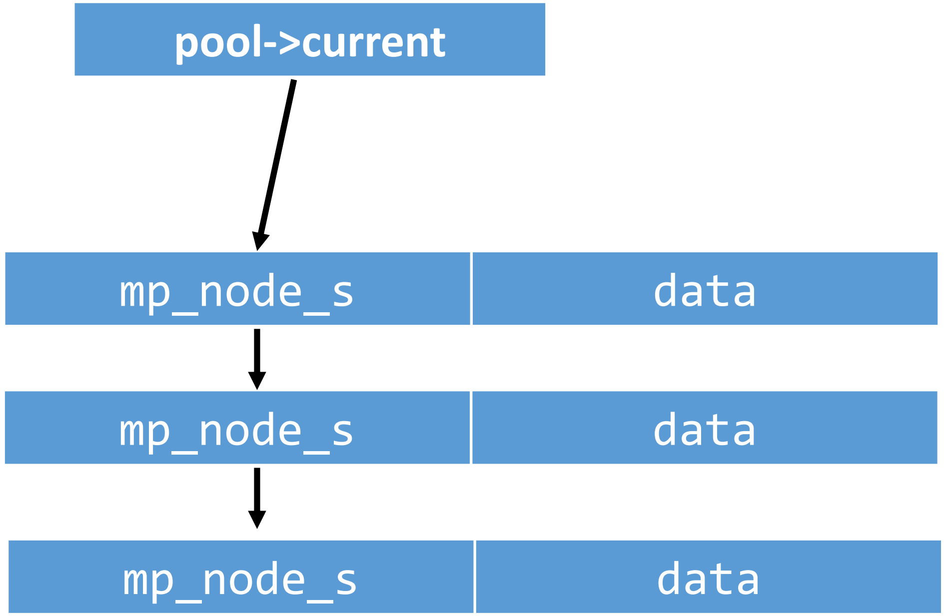



```c
struct mp_pool_s *mp_create_pool(size_t size) {

    struct mp_pool_s *p;
    int ret = posix_memalign((void **)&p, MP_ALIGNMENT, size + sizeof(struct mp_pool_s) + size
    if (ret) {
        return NULL;
    }

    p->max = (size < MP_MAX_ALLOC_FROM_POOL) ? size : MP_MAX_ALLOC_FROM_POOL;
    p->current = p->head;
    p->large = NULL;

    p->head->last = (unsigned char *)p + sizeof(struct mp_pool_s) + sizeof(struct mp_node_s);
    p->head->end = p->head->last + size;

    p->head->failed = 0;

    return p;

}
```



零声学院
www.0voice.com
一切只为渴望更优秀的你!

内存池分配

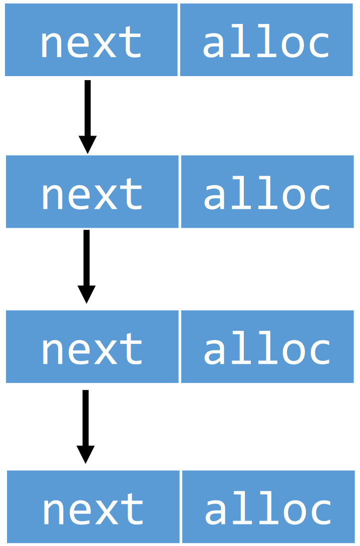



```c
void *mp_alloc(struct mp_pool_s *pool, size_t size) {

    unsigned char *m;
    struct mp_node_s *p;

    if (size <= pool->max) {

        p = pool->current;

        do {

            m = mp_align_ptr(p->last, MP_ALIGNMENT);
            if ((size_t)(p->end - m) >= size) {
                p->last = m + size;
                return m;
            }
            p = p->next;
        } while (p);

        return mp_alloc_block(pool, size);
    }

    return mp_alloc_large(pool, size);

}
```

内存池小块分配


```c
static void *mp_alloc_block(struct mp_pool_s *pool, size_t size) {

    unsigned char *m;
    struct mp_node_s *h = pool->head;
    size_t psize = (size_t)(h->end - (unsigned char *)h);

    int ret = posix_memalign((void **)&m, MP_ALIGNMENT, psize);
    if (ret) return NULL;

    struct mp_node_s *p, *new_node, *current;
    new_node = (struct mp_node_s*)m;

    new_node->end = m + psize;
    new_node->next = NULL;
    new_node->failed = 0;

    m += sizeof(struct mp_node_s);
    m = mp_align_ptr(m, MP_ALIGNMENT);
    new_node->last = m + size;

    current = pool->current;

    for (p = current; p->next; p = p->next) {
        if (p->failed++ > 4) {
            current = p->next;
        }
    }
    p->next = new_node;
```

内存池大块分配




```c
static void *mp_alloc_large(struct mp_pool_s *pool, size_t size) {

    void *p = malloc(size);
    if (p == NULL) return NULL;

    size_t n = 0;
    struct mp_large_s *large;
    for (large = pool->large; large; large = large->next) {
        if (large->alloc == NULL) {
            large->alloc = p;
            return p;
        }
        if (n ++ > 3) break;
    }

    large = mp_alloc(pool, sizeof(struct mp_large_s));
    if (large == NULL) {
        free(p);
        return NULL;
    }

    large->alloc = p;
    large->next = pool->large;
    pool->large = large;

    return p;
}
```

内存池重置

零声学院
www.0voice.com
一切只为渴望更优秀的你!

1. 释放large
2. 将node节点last重置

```c
void mp_reset_pool(struct mp_pool_s *pool) {

    struct mp_node_s *h;
    struct mp_large_s *l;

    for (l = pool->large; l; l = l->next) {
        if (l->alloc) {
            free(l->alloc);
        }
    }

    pool->large = NULL;

    for (h = pool->head; h; h = h->next) {
        h->last = (unsigned char *)h + sizeof(struct mp_node_s);
    }

}
```

内存池销毁与释放大页内存

零声学院
www.0voice.com
一切只为渴望更优秀的你！

```
void mp_destory_pool(struct mp_pool_s *pool) {

    struct mp_node_s *h, *n;
    struct mp_large_s *l;

    for (l = pool->large; l; l = l->next) {
        if (l->alloc) {
            free(l->alloc);
        }
    }

    h = pool->head->next;

    while (h) {
        n = h->next;
        free(h);
        h = n;
    }

    free(pool);

}
```

```
void mp_free(struct mp_pool_s *pool, void *p) {

    struct mp_large_s *l;
    for (l = pool->large; l; l = l->next) {
        if (p == l->alloc) {
            free(l->alloc);
            l->alloc = NULL;

            return ;
        }
    }

}
```

阅读nginx内存池

1. 总结nginx的内存池的工作流程
2. Ringbuffer的实现原理