Basic Operations Of The Memory Pool :

1. ngx_create_pool -> Used to create a memory pool
2. nfx_destroy_pool -> Used for destruction a memory pool
3. ngx_reset_pool -> Reset the memory pool
4. ngx_palloc -> Used to allocate aligned memory
5. ngx_pnalloc -> Used to allocate unaligned memory
6. ngx_pfree -> used to free space
7. ngx_palloc_large() -> Large memory allocation

Create a memory pool:

ngx_pool_t *ngx_create_pool (size_t size, ngx_log_t *log);

This function allocates a memory page, and initialize the corresponding fields.

Source code :

```
ngx_pool_t *
ngx_create_pool(size_t size, ngx_log_t *log)
{
    ngx_pool_t  *p;

    p = ngx_memalign(NGX_POOL_ALIGNMENT, size, log);
    if (p == NULL) {
        return NULL;
    }
    p->d.last = (u_char *) p + sizeof(ngx_pool_t);
    p->d.end = (u_char *) p + size;
    p->d.next = NULL;
    p->d.failed = 0;

    size = size - sizeof(ngx_pool_t);
    p->max = (size <NGX_MAX_ALLOC_FROM_POOL) ? size : NGX_MAX_ALLOC_FROM_POOL;

    p->current = p;
    p->chain = NULL;
    p->large = NULL;
    p->cleanup = NULL;
    p->log = log;
    return p;
}
#define NGX_POOL_ALIGNMENT 16
```

Therefore, memory pool allocation of nginx, is a 16 byte boundary alignment.

Working of ngx_create_pool

- This function takes two parameters.The first parameter is the size of the memory page/pool to be allocated.
- The second parameter is the pointer to ngx_log_t structure which logs any errors in case of failure in allocation.
- The function then uses the ngx_memalign function to allocate aligned memory of size size.
- Then it decides whether the size to be allocated is more than maximum limit i.e. NGX_MAX_ALLOC_FROM_POOL or not.
- Then it initializes the various members of ngx_pool_t structure.
- At last it returns the pointer to the allocated memory pool.

Allocation of memory chunk in the pool :

ngx_palloc(ngx_pool_t *pool, size_t size)

After the allocation of the memory pool we need to allocate memory chunks as required in order to do some useful work. This function ngx_palloc helps us to achieve this.

Source code :

```
void *
ngx_palloc(ngx_pool_t *pool, size_t size)
{
    u_char      *m;
    ngx_pool_t  *p;
    if (size <= pool->max) {
        p = pool->current;
        do {
            m = ngx_align_ptr(p->d.last, NGX_ALIGNMENT);
            if ((size_t) (p->d.end - m) >= size) {
```

```
            p->d.last = m + size;
            return m;
        }
        p = p->d.next;
    } while (p);
    return ngx_palloc_block(pool, size);
  }
  return ngx_palloc_large(pool, size);
}
```

The function takes two parameters

- Pool : This is a pointer to memory pool's head node.The is the memory pool where the space is allocated.
- Size : this indicates the size of memory allocation.

This function first checks whether the size to be allocated is available in the pool or not.If the memory requirement is within the maximum pool size then the memory will be allocated in the small memory (ngx_pool_data_t) otherwise large memory (ngx_pool_large_s) wil be used for allocation.

At last it returns the pointer to the allocated memory chunk.

The destruction of memory pool :

void ngx_destroy_pool (ngx_pool_t *pool);

The destruction of memory pool takes place in 3 steps :

- implementation of cleanup cleaning.The cleanup handler is referenced to see how to release the resources.
- release of large memory (ngx_pool_large_t)
- release of small memory (ngx_pool_data_t)

Source code :

```
Void ngx_destroy_pool(ngx_pool_t *pool)
 {
    ngx_pool_t          *p, *n;
    ngx_pool_large_t    *l;
    ngx_pool_cleanup_t  *c;
```

```
     printf("ngx_destroy_pool::ngx_destroy_pool:%p\n", ngx_destroy_pool);
    get_systime(buf, 100);

    for (c = pool->cleanup; c; c = c->next) {
        if (c->handler) {
            ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, pool->log, 0,
                           "run cleanup: %p", c);
            c->handler(c->data);
        }
    }

    for (l = pool->large; l; l = l->next) {

        ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, pool->log, 0, "free: %p", l->alloc);

        if (l->alloc) {
            ngx_free(l->alloc);
        }
    }
     for (p = pool, n = pool->d.next; /* void */; p = n, n = n->d.next) {
        ngx_free(p);

        if (n == NULL) {
            break;
        }
    }
}
```

This function takes only one parameter that is the pointer to the memory pool to be destructed.
Firstly any abstract data like opened files,cache which is stored by ngx_pool_cleanup_t are released according to the handler function.
Then the large memory is released followed by release of small memory. This release is done through ngx_free() function call.

void * ngx_pnalloc(ngx_pool_t *pool, size_t size);

This function is same as ngx_palloc.But, the main difference is that the memory allocated in ngx_palloc is aligned while,in case of ngx_pnalloc the memory allocated id not aligned.

Reset the memory pool :

void ngx_reset_pool(ngx_pool_t *pool)

source code :

```
void ngx_reset_pool(ngx_pool_t *pool)
{
    ngx_pool_t        *p;
    ngx_pool_large_t  *l;

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

    for (p = pool; p; p = p->d.next) {
        p->d.last = (u_char *) p + sizeof(ngx_pool_t);
        p->d.failed = 0;
    }

    pool->current = pool;
    pool->chain = NULL;
    pool->large = NULL;
}
```

This function is used to reset the memory pool back to the state when it was initialized.When the memory pool is created or initialized there is no large memory or abstract memory allocation. Thus large memory if allocated is freed using this function as shown above in the source code.And instead of freeing the small memory allocation simply the last pointer is made to point to its initial state when the pool was created.This is because the small memory was allocated when the pool was created but contained some garbage values.Thus we cannot free it but can only reset it.

Allocation of Large Memory :

ngx_palloc_large()

Source code :

```
static void * ngx_palloc_large(ngx_pool_t *pool, size_t size)
{
    void              *p;
    ngx_uint_t         n;
    ngx_pool_large_t  *large;

    p = ngx_alloc(size, pool->log);
    if (p == NULL) {
        return NULL;
    }

    n = 0;

    for (large = pool->large; large; large = large->next) {
        if (large->alloc == NULL) {
            large->alloc = p;
            return p;
        }

        if (n++ > 3) {
            break;
        }
    }
    large = ngx_palloc(pool, sizeof(ngx_pool_large_t));
    if (large == NULL) {
        ngx_free(p);
        return NULL;
    }

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

    return p;
}
```

This function is meant for bulk memory allocation also known as large memory in nginx context. The the memory requirement is more than the size of the memory pool then the memory must be allocated outside the pool in the main memory.But this allocated memory is linked to the memory pool and this information is handled by ngx_pool_large_t structure using the member pointers next and alloc.
Here the memory is allocated outside the pool using ngx_alloc() function which is nothing but encapsulation of normal malloc() or calloc() function.For small memory allocation ngx_palloc() is used.
This function is called when size of the memory chunk to be allocated is larger than the maximum limit max i.e. when size> max .