**动态数组**

**结构**

// 定义在 core/ngx\_array.h 中

typedef struct {

    void \*elts;         // 内存首地址

    ngx\_uint\_t nelts;   // 元素的数量

    ...

} ngx\_array\_t;

**操作**

// 从内存池中申请 n 个 size 大小的数组

ngx\_array\_t\* ngx\_array\_create(ngx\_pool\_t \*p, ngx\_uint\_t n, size\_t size);

// 销毁数组

void ngx\_array\_destroy(ngx\_array\_t \*a);

// 追加一个元素，返回元素的指针，大小是创建时指定的size

void \*ngx\_array\_push(ngx\_array\_t \*a);

**单向链表**

与一般链表不同，nginx的链表节点包含一个数组

**结构**

// 定义在 core/ngx\_list.h 中

// 链表

typedef struct {

    ngx\_list\_part\_t\* last;      // 链表的尾节点

    ngx\_list\_part\_t part;       // 链表的头节点

    size\_t size;                // 链表节点的元素的大小

    ...

} ngx\_list\_t;

// 链表节点

// 每个节点包含一个数组

typedef struct ngx\_list\_part\_s ngx\_list\_part\_t;

struct ngx\_list\_part\_s {

    void\* elts;     // 数组元素指针

    ngx\_uint\_t nelts;   // 元素个数

    ngx\_list\_part\_t\* next;  // 下一个节点的指针

}

**操作**

// 向内存池申请创建链表，链表节点可容纳 n 个大小为 size 的元素

ngx\_list\_t \*ngx\_list\_create(ngx\_pool\_t \*pool, ngx\_uint\_t n, size\_t size);

// 添加一个链表节点的元素，大小为创建时的 size，返回该元素的指针

void \*ngx\_list\_push(ngx\_list\_t \*list);

**双端队列**

**结构**

// 该结构一般插入自定义的数据结构中使用

// 例： struct aaa{

// ngx\_queue\_s nqs;

// ...

// }

struct ngx\_queue\_s {

    ngx\_queue\_t \*prev;  // 前一个指针

    ngx\_queue\_t \*next;  // 后一个指针

}

**操作**

// 初始化队列

#define ngx\_queue\_init(q) ...

// 头部插入，h 队列头， x 要插入的队列节点

#define ngx\_queue\_insert\_head(h, x)

// 尾部插入

#define ngx\_queue\_insert\_tail(h, x)

// 获取队列节点的完整数据

// q ngx\_queue\_t 的指针

// type 节点的类型

// link ngx\_queue\_t 的成员名

#define ngx\_queue\_data(q, type, link)

**缓冲区**

Nginx有时需要处理大量数据，这时需要大块内存，如果内存无法存放，这需要磁盘文件

**结构**

// 定义在 core/ngx\_buf.h 中

typedef struct ngx\_buf\_s ngx\_buf\_t;

// 缓存区的数据可能在内存中或磁盘文件中

// 所以结构有2种起始和结束地址

struct ngx\_buf\_s{

    u\_char\* pos;    // 已写入数据的起始地址，需手动更新

    u\_char\* last;   // 已写入数据的结束地址，需手动更新

    u\_char\* start;  // 内存的开始

u\_char\* end;    // 内存的结束

    off\_t file\_pos;     // 文件数据起始偏移

    off\_t file\_last;    // 文件数据结束偏移

ngx\_file\_t file;    // 存储数据的文件对象

    ...

}

**操作**

// 从内存池创建size大小的缓存区

// pos 和 last 的指针均初始化为 start

ngx\_buf\_t \*ngx\_create\_temp\_buf(ngx\_pool\_t \*pool, size\_t size);

// 缓存数据是否在内存中（如果不再内存则在文件中）

#define ngx\_buf\_in\_memory(b) ...

**数据块链**

处理http请求经常创建多个缓冲区来存放数据，nginx使用数据块链将缓存区连接起来

**结构**

// 定义在 core/ngx\_core.h 中

typedef struct ngx\_chain\_s ngx\_chain\_t;

struct ngx\_chain\_s {

    ngx\_buf\_t \*buf;     // 缓存去指针

    ngx\_chain\_t \*next;  // 下一个链表节点

}

**操作**

// 从内存池创建数据块链

ngx\_chain\_t \*ngx\_alloc\_chain\_link(ngx\_pool\_t \*pool);

// 销毁数据库链

#define ngx\_free\_chain(pool, cl) ...

**键值对**

**结构**

typedef struct {

    ngx\_str\_t key;

    ngx\_str\_t value;

} ngx\_keyval\_t;