# 内存池ngx\_pool\_s

## 结构体成员

struct ngx\_pool\_s {

ngx\_pool\_data\_t d;

size\_t max;

ngx\_pool\_t \*current;

ngx\_chain\_t \*chain;

ngx\_pool\_large\_t \*large;

ngx\_pool\_cleanup\_t \*cleanup;

ngx\_log\_t \*log;

};

ngx\_pool\_s结构可以看成为内存池的标识，用来管理一块或者多块内存。

**d**：说明如下

typedef struct {

u\_char \*last;

u\_char \*end;

ngx\_pool\_t \*next;

ngx\_unit\_t failed;

} ngx\_pool\_data\_t;

last：内存空间的起始地址。

end：内存空间的结束地址。

next：指向下一块内存。在ngx\_palloc\_block中赋值。

failed：从当前内存空间分配内存失败的次数。

**max**：能够从d成员所管理的内存空间分配的内存大小最大值。如果实际需要分配的内存大小大于max，会被认为是大内存，大内存则由large成员管理。

**current**：ngx\_pool\_t当前所指向的内存空间。该字段在当前内存空间分配内存失败的时候发生变化。

**chain**：

**large**：用于大内存的管理。

**cleanup**：

**log**：

## 常用操作

### 创建ngx\_create\_pool

入参size：内存池的大小。

根据size的大小申请一块内存p。这块内存包括了ngx\_pool\_t结构。因此实际可以用于分配的内存空间大小为size – sizeof(ngx\_pool\_t)。

max成员的值最大不能超过ngx\_pageszie – 1，如果size小于ngx\_pagesize – 1，则max = size。ngx\_pagesize在x86的机器上是4096。

current指向上面申请的内存p。

### 分配内存ngx\_palloc

入参pool：指定从哪个内存池中分配内存。

入参size：需要分配的内存大小。

size <= pool->max，**小内存分配ngx\_palloc\_small**：

1. 先判断current所指向的内存空间中的剩余空间是否满足所需分配的内存大小。
2. 满足，p->d.last经过字节对齐处理之后所指向的内存地址就是待分配内存的起始地址。返回给上层。
3. 不满足，则查找下一块内存的剩余空间是否满足要求。

什么时候会有下一块内存空间？

ngx\_pool\_t创建之初只管理了一个内存空间，当这个内存空间剩余大小不能满足分配内存的需要时，会调用**ngx\_palloc\_block**创建一块新的相同大小的内存空间new。新创建的内存new挂在current的next指针。

current什么是否发生变化？

如果current节点next已经挂载了超过4个ngx\_pool\_t节点，说明这些节点都无法满足内存申请的需求。此时就会将current指针指向本次新申请的new。

**大内存分配ngx\_palloc\_large**：

1. 根据需要分配的内存大小，直接从系统中申请一块内存。
2. 从小内存池中申请一个ngx\_pool\_large\_t结构，用来管理大内存。
3. 大内存的链表里如果有超过3个大内存空间，那么新申请的大内存就加入到链表头部。

large

d

ngx\_pool\_t