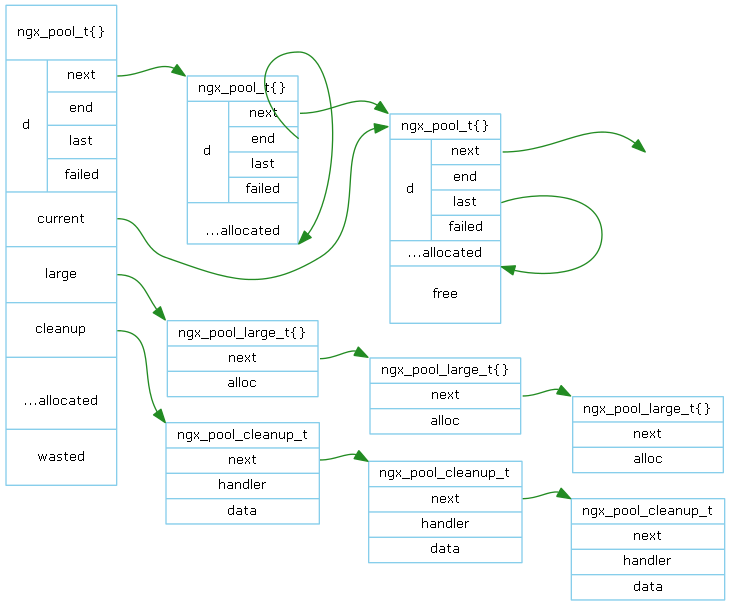
Nginx Memory Model

# 概述

暂无

# 内存池

## 数据结构



上图表示一个由ngx\_pool\_create所创建的内存池。

### ngx\_pool\_t

表示一块固定大小的内存块，在头部包含一些管理字段。

* ngx\_pool\_data\_t
* next

指向下一个ngx\_pool\_t。组成内存块链表。

* end

指向本块内存的结尾。

* last

指向本块内存的可分配位置。

* failed

从本块内存上分配失败次数。

* max

根据分配内存的大小，内存池分为二种小于max的为小内存，大于等于max的为大内存。

* current

指向内存块链表中有内存可分配的内存块ngx\_pool\_t。

* large

大内存管理链表。

* cleanup

清理函数链表。

### ngx\_pool\_large\_t

管理一块通过malloc向系统申请的大内存。不过ngx\_pool\_large\_t结构体本身则是以小内存向内存池申请的。

* next

指向下一个ngx\_pool\_large\_t。组成大内存管理链表。

* alloc

指向通过malloc向系统申请的大内存。

### ngx\_pool\_cleanup\_t

* next

指向下一个ngx\_pool\_cleanup\_t。组成链表。

* handler/data

清理回调函数以及参数。

## 分配小内存

1. 通过current得到有内存可分配的ngx\_pool\_t。
2. 如果有足够剩余内存，则移动last，分配完成。
3. 如果不足，则依次从链表的ngx\_pool\_t分配，并更新分配失败次数（failed）。
4. 全部没有足够剩余内存，则通过memalign(malloc)再向系统新申请一个内存块ngx\_pool\_t，并挂入链表。
5. 如果failed大于4，则认为该npx\_pool\_t没有足够剩余内存可分配。更新current使它指向有足够剩余内存的ngx\_pool\_t。
6. 分配完成。

## 释放小内存

小内存无法单独释放，也无法重用。它随内存池的释放而释放。

## 分配大内存

* 通过malloc向系统申请所需内存。
* 向内存池申请ngx\_pool\_large\_t结构体。
* 将ngx\_pool\_large\_t挂入大内存管理链表。
* 分配完成

## 释放大内存

从大内存管理链表中找到并删除ngx\_pool\_large\_t，调用free向系统释放内存。ngx\_pool\_large\_t不会被释放。

## 内存池用法

Nginx运行时会产生具有生命周期的对象。

* ngx\_cycle\_t

生命周期是Nginx整个过程，包含配置参数等数据。ngx\_cycle\_t有一个专用的内存池，配置参数从此分配。

进程结束时，统一释放。

* ngx\_conf\_t

生命周期是解析配置文件过程。由于解析配置文件过程中，需要临时保存指令名、指令参数等数据。于是

ngx\_conf\_t也有一个专用的内存池。解析配置文件完成后，统一释放。

* ngx\_connection\_t

ngx\_connection\_t生命周期内的数据，由ngx\_connection\_t专用的内存池分配。连接断开后，统一释放。

* ngx\_http\_request

每收到一个HTTP请求，则创建一个专用内存池，负责分配HTTP请求生命周期内的数据。HTTP请求处理完成，统一释放。

## 分析

内存池合并了小内存的分配与释放，向系统申请和释放的内存都不是小内存，可以有效防止产生大量系统内存碎片。

内存池不立即释放小内存而是统一释放，它以牺牲小内存为代价，简化了内存管理算法，提高了效率。

# 共享内存

[TODO]

# Slab

[TODO]

# Prototype

无

# Latest revision

https://github.com/lingjf/nginx\_analyse/blob/master/doc/