


# 内存分配奥义·jemalloc(一)

 [tinylab.org/memory-allocation-mystery---jemalloc-a](http://tinylab.org/memory-allocation-mystery---jemalloc-a)

by Chen Jie of [TinyLab.org](http://TinyLab.org) 2014/11/27

Chen Jie 创作于 2014/11/29

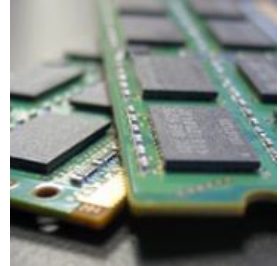
打赏

## 1 前言

C 中动态内存分配 malloc 函数的背后实现有诸派：[dlmalloc](#) 之于 [bionic](#)；[ptmalloc](#) 之于 [glibc](#)；[allocation zones](#) 之于 [mac os x/ios](#)；以及 [jemalloc](#) 之于 [FreeBSD/NetBSD/Firefox](#)。

malloc 实现对性能有较大影响，而 jemalloc 似乎是目前诸实现中最强的，并在 facebook 内广泛使用，参见 facebook 的[使用心得](#)。

对此，我们按照知乎“[怎样给招式起一个一听就很厉害的名字？](#)”的指导，拟了本文标题，并速度道来。



## 2 简介

这是一个关于电商的故事，我下了一个单，订购一块 N 字节的内存，并等待它的到达。怎样做到即时送达？

如果订购的内存是个小件（好比一块橡皮、一本书或是一个微波炉等），那么直接从[同城仓库](#)送出。

如果订购的内存是个大件（好比电视机、空调等），那么得从[区域仓库](#)（例如华东区仓库）送出。

如果订购的内存是个巨大件（好比汽车、轮船），那么得从[全国仓库](#)送出。

在 jemalloc 类比过来的物流系统中，[同城仓库](#)相当于 tcache —— 线程独有的内存仓库；[区域仓库](#)相当于 arena —— 几个线程共享的内存仓库；[全国仓库](#)相当于全局变量指向的内存仓库，为所有线程可用。

在 jemalloc 中，整块批发内存，之后或拆开零售，或整块出售。整块批发的内存叫做 chunk，对于小件和大件订单，则进一步拆成 run。

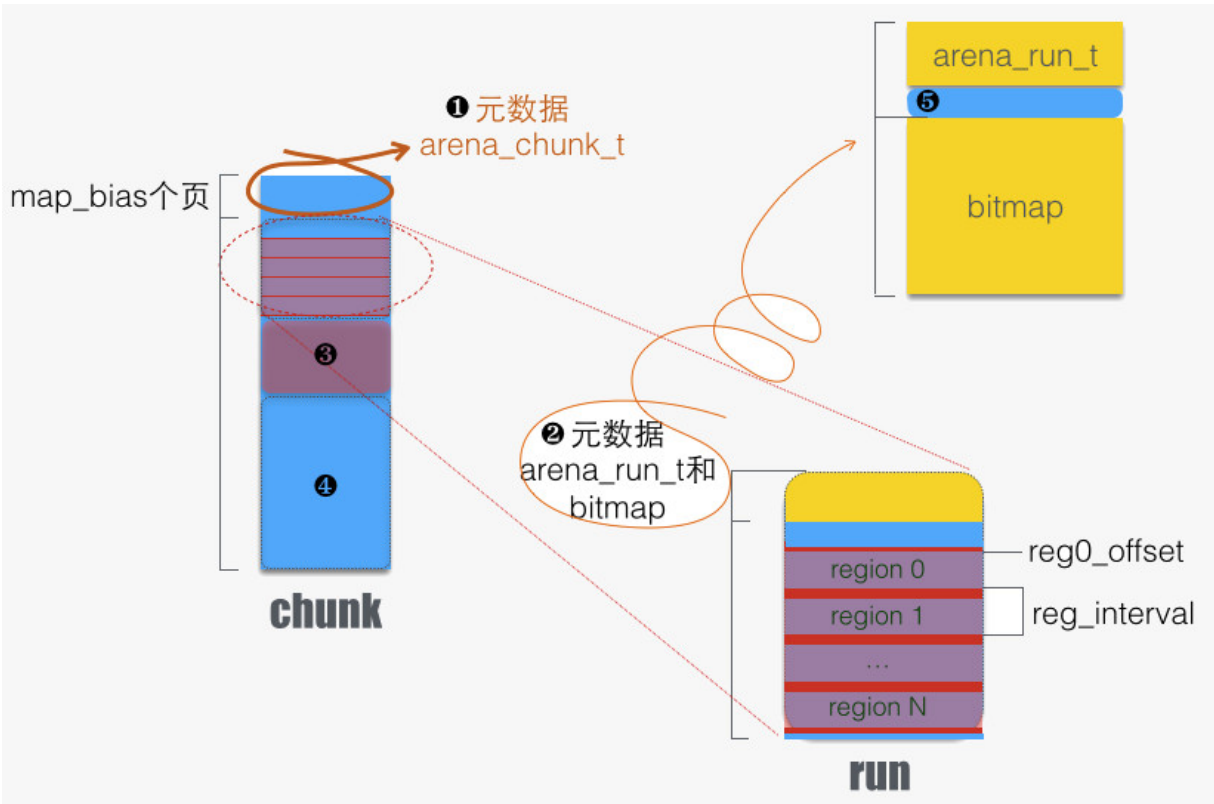
Chunk 的大小为 4MB（可调）或其倍数，且为 4MB 对齐；而 run 大小为页大小的整数倍。

在 jemalloc 中，小件订单叫做 small allocation，范围大概是 1-57344 字节。并将此区间分成 44 档，每次小分配请求[归整到某档](#)上。例如，小于8字节的，一律分配 8 字节空间；17-32分配请求，一律分配 32 字节空间。

对于上述 44 档，有对应的 44 种 runs。每种 run 专门提供此档分配的内存块（叫做 region）。

大件订单叫做 large allocation，范围大概是 57345-4MB不到一点的样子，所有大件分配[归整到页大小](#)。

以上总结成下图：



上图中：

- 1. arena\_chunk\_t: 属于 arena 的 chunk 头部有此结构体。该结构体末尾是一个数组 arena\_chunk\_map\_t map[]。
- 2. run for small allocation。主要特点是分成了等长的若干 regions，每次请求分配出一个 region。通过头部元数据（详见 5）来记录分配状态，例如标记分配状态的 bitmap。
- 3. run for large allocation，无元数据。
- 4. 未分配的 run。它将被分割用于 2 情形，或者 3 情形。

最后，巨大件订单，叫做 huge allocation，所有巨大件请求归整到标准 chunk 大小（4MB）的整数倍。

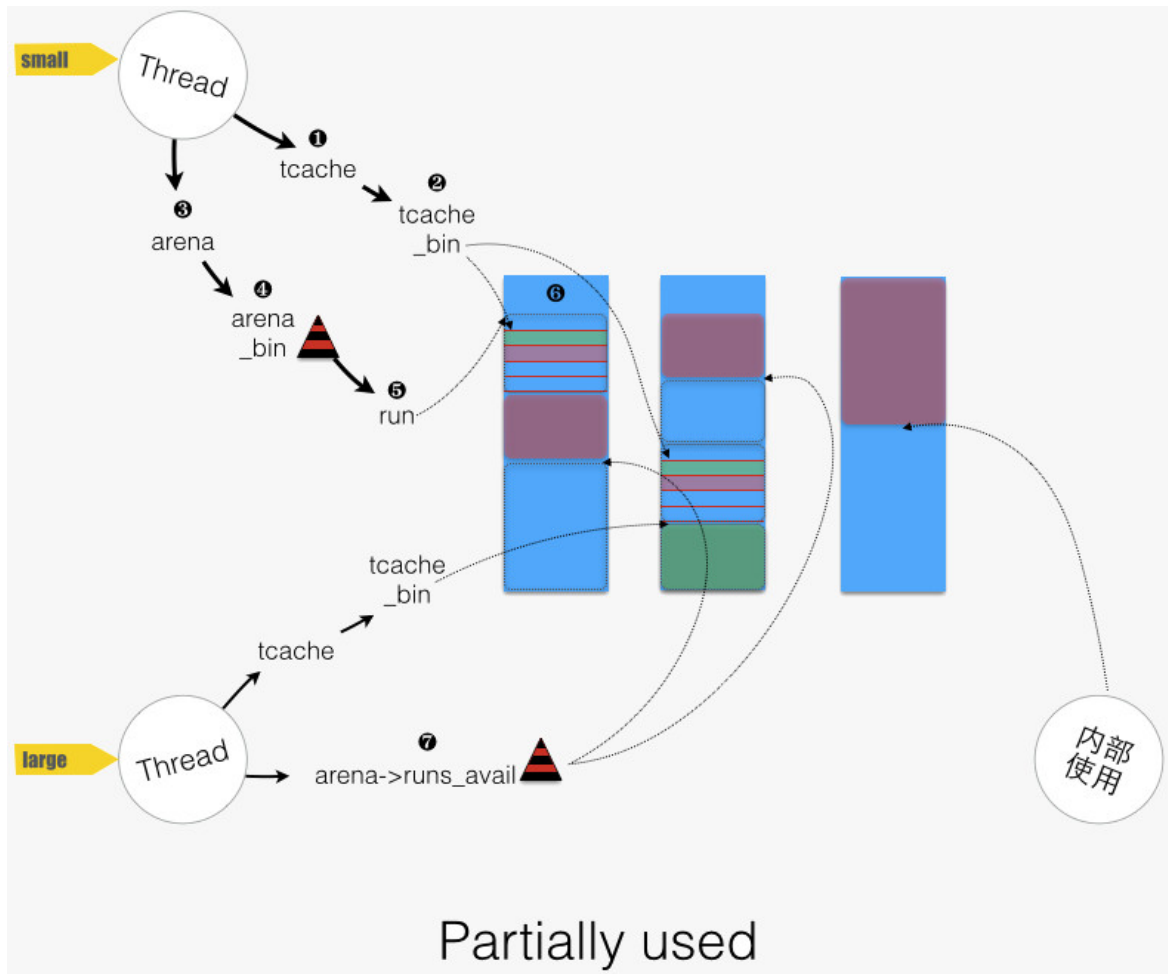
### 3 分配内存

下面我们从 jemalloc 代码中，来一探其诸分配过程。分配函数的入口在 je\_malloc/MALLOC\_BODY/imalloc/imalloct。

jemalloc 的内存分配，可分成四类：

- 1. 内部使用
- 2. 用于 small allocation
- 3. 用于 large allocation
- 4. 用于 huge allocation，分配出的内存全部交付使用

其中 small 和 large allocation较为复杂，且以下图作为地图，来导游下：



### 3.1 Small allocation

对于 small allocation, 需要确定 请求大小 对应 到哪一“档位”上, 确定的公式如下:  $\text{small\_size2bin}[(\text{size} - 1) \gg 3]$ , 该公式通过查数组, 来确定“档位”。

在 jemalloc 中，某“档位”上可分配的内存资源，用 bin 来管理。

回到 small allocation 过程，首先从线程本身的 tcache 中去满足，函数路径为

```
imalloct/arena malloc/tcache alloc_small:
```

- 图中的“1 -> 2”：找到对应的 bin，然后尝试分配。函数路径为 `tcache_alloc_small/tcache_alloc_easy(tcache_bin)`
- 当 `tcache_bin` 为空时，从 arena 进货，填充（`arena_tcache_fill_small`）后再分配，函数路径为 `tcache_alloc_small/tcache_alloc_small_hard`

tcache 特性关闭时，则从 arena 中分配，函数路径为 `imalloc/arena_malloc/arena_malloc_small`：

- 图中“3 -> 4”：找到对应的 bin。
- 图中“4 -> 5”：从 bin 中选择一个 run
  1. 尝试 arena\_bin->run\_cur，即目前正在使用的 run。
  2. 若该 run\_cur 已满，则从 bin 中选择地址最低的 run。arena\_bin->runs 用了个红黑树来按地址排序全部的 runs。函数路径为  
arena\_malloc\_small/arena\_bin\_malloc\_hard/arena\_bin\_nonfull\_run\_get/arena\_bin\_nonfull\_run\_tryget
  3. 若 bin 为空，则从 arena->runs\_avail 中找空间，分配新的 run。函数路径为  
arena\_bin\_nonfull\_run\_get/arena\_run\_alloc\_small/arena\_run\_alloc\_small\_helper。
  4. 尼玛 arena->runs\_avail 也空间不够了，只好重新弄个 chunk，分出所需空间，剩余部分放入 arena->runs\_avail。
- 图中“5”：从 run 中分配一个 region，对应函数为 arena\_run\_reg\_alloc(run, bin\_info)。

### 3.2 Large allocation

首先从 tcache 中满足。对于 large allocation，tcache 将“档位”的概念拓展，其“档位”计算公式如下： $NBINS + (size \gg PAGE) - 1$ 。

函数路径为 `arena_malloc/tcache_alloc_large`，两点说明：

- tcache 只涵盖一部分的 large allocation 请求 (size 小于等于 tcache\_maxclass)
- 对应的 tcache bin 为空时, 不进行填充, 而是走非 tcache 分配。这点与 small allocation 的情形是不同的。

tcache 特性关闭，或者请求大小超过 tcache 中最大 bin 覆盖范围时，则从 arena 中分配，函数路径为 `arena_malloc/arena_malloc_large`：

1. 图中“7”：从 arena->runs\_avail 中，找出适合空间中地址最低的一块空间，分割出空间，将剩余部分重新放回 arena->runs\_avail 中。函数路径为 arena\_malloc\_large/arena\_run\_alloc\_large\_helper
2. 若 arena->runs\_avail 空间也不够，重新分配 chunk，分出所需空间，剩余部分放入 arena->runs\_avail。

### 3.3 内部使用

内部使用所需的内存分配，使用 `base_alloc`，只申请，不释放。因此在当前使用的 `chunk` 中，分配区域呈现线性增长。

base\_alloc 使用场合有:

- nodes：用来管理未使用的 chunks。在 base\_alloc 基础上再封一层 base\_node\_alloc，先从 base\_nodes 缓冲中取，失败再调用 base\_alloc；释放函数 base\_node\_dealloc 将 node 链接入 base\_nodes。
- 其他：如 arenas、arena\_t、tcache\_bin\_info 等。

### 3.4 实际分配函数

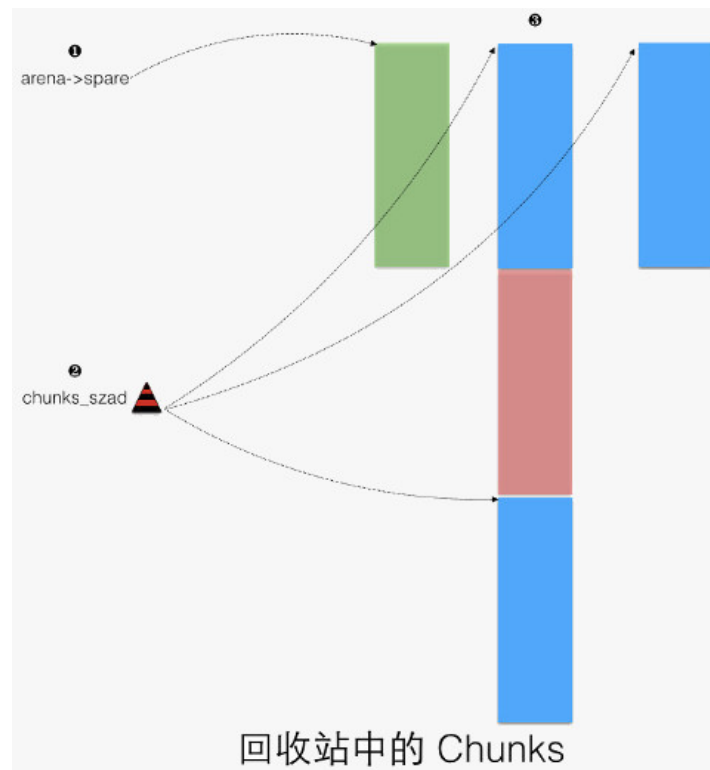
前述，je\_malloc 总是按照 chunk 尺寸从 OS 批发内存，对应函数为 `chunk_alloc`。

chunk\_alloc 先从“回收站”中回收不用的chunk，若没无再从 OS 批发。

chunk\_alloc 有几个有名的客户:

- `arena_chunk_alloc`, 分配 arena 名下的 chunk (供 small/large allocation 分配)。arena\_chunk\_alloc 分配时, 先看本 arena 下是否有备用的 chunk, 没有再调 chunk\_alloc。
- `base_alloc`, base\_alloc 在当前使用的 chunk 空间不足时, 调用 chunk\_alloc。
- `huge_malloc`, 用于满足 huge allocation。

上述提及的 chunks “回收站”，用于回收释放的 chunk，如下图所示：



1. arena 中的备用 chunk，供 arena\_chunk\_alloc 中使用。
2. chunks\_szed\*，红黑树，按照先尺寸、后地址来排序所有不用的 chunks。
3. 一个 3 个标准 chunks 大小的 chunk，中间一个 chunk 被分配出。首尾不用的 chunks 被 chunks\_szed\* 引用。

## 4 小结

至此，我们粗步游了一遍 jemalloc 的内存分配，眼花撩乱了没？

总结一下 jemalloc 的设计思路：

- 减少多线程竞争。例如引入 tcache，以及线程均分布到若干个 arena(s)。
- 地址空间重用，减少碎片：

- 红黑树来保证同等条件下，总是从低地址开始分配
- 合并相邻的空闲空间
- 保持 cache 热度，例如 tcache，地址空间重用。
- 各种对齐，自然对齐，cache line 对齐。

同时，我们也看到 jemalloc 有层层缓冲，例如：

- tcache
- arenas 名下的缓冲：bins 管理的 runs(for small allocations) 以及 arena->runs\_avail
- 内部使用的缓冲，base\_nodes 以及其当前使用的 chunk 的剩余空间
- 回收站中的 chunks，例如 chunks\_szad\_\*, arena->spare

由此，在一个普遍使用 jemalloc 的系统中会产生许多内存额外占用，这对实时性要求较高、内存较为紧张的移动设备而言是不可接受的。

我们可以调节 jemalloc 配置，来减少额外的内存占用，例如将 chunksize 调整为 1MB、调节 small allocation 所需“档位”数目、tcache 大小调整，等等。

同时，更重要的，jemalloc 中的内存释放系统，能否及时平衡额外的占用，我们将在下一篇中来看看。

最后，是否存在一种有效的联动机制，在系统整体内存紧张时，通知各进程释放掉额外的缓冲？这样就不用 OOM killer 了。

---