PHP的内存管理, 分为俩大部分,

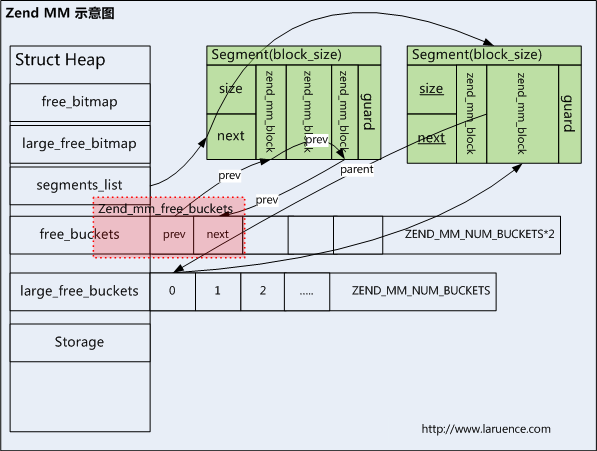
第一部分是PHP自身的内存管理, 这部分主要的内容就是引用计数, 写时复制, 等等面向应用的层面的管理.

而第二部分就是今天我要介绍的, zend\_alloc中描写的关于PHP自身的内存管理, 包括它是如何管理可用内存, 如何分配内存等.

另外, 为什么要写这个呢, 因为之前并没有任何资料来介绍PHP内存管理中使用的策略, 数据结构, 或者算法. 而在我们平时开发扩展, 修复PHP的bug的时候, 却对这一部分的知识需要有一个良好的理解. PHP开发组内的很多朋友也对这块不是很清楚, 所以我觉得有必要专门写一下.

一些基本的概念, 我就不赘述了, 因为看代码很容易能看懂, 我这里就主要介绍几个看代码没那么容易看懂的点,其中看到了TIPI项目对这部分的描述, 发现其中错误很多. 所以, 我想这部分就是看代码也没那么容易看懂的点 :)

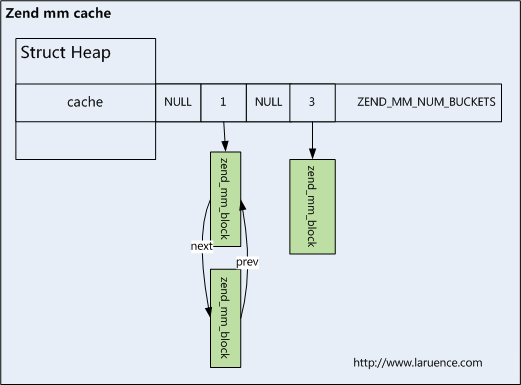
目前, 英文版的介绍也在撰写中: [Zend MM](https://wiki.php.net/internals/zend_mm)

Zend Memory Manager, 以下简称Zend MM, 是PHP中内存管理的逻辑. 其中有一个关键数据结构: zend\_mm\_heap:  
[](http://laruence-wordpress.stor.sinaapp.com/uploads/zend_mm.png)

Zend MM把内存分为小块内存和大块内存俩种, 区别对待,

对于小块内存, 这部分是最最常用的, 所以追求高性能.

对于大块内存, 则追求的是稳妥, 尽量避免内存浪费.

所以, 对于小块内存, PHP还引入了cache机制:  
[](http://laruence-wordpress.stor.sinaapp.com/uploads/zend_mm_cache.png)

Zend MM 希望通过cache尽量做到, 一次定位就能查找分配.

而一个不容易看懂的点是free\_buckets的申明:

Q: 为什么free\_buckets数组的长度是ZEND\_MM\_NUMBER\_BUCKET个?

A: 这是因为, PHP在这处使用了一个技巧, 用一个定长的数组来存储ZEND\_MM\_NUMBER\_BUCKET个zend\_mm\_free\_block, 如上图中红色框所示. 对于一个没有被使用的free\_buckets的元素, 唯一有用的数据结构就是next\_free\_block和prev\_free\_block, 所以, 为了节省内存, PHP并没有分配ZEND\_MM\_NUMBER\_BUCKET \* sizeof(zend\_mm\_free\_block)大小的内存, 而只是用了ZEND\_MM\_NUMBER\_BUCKET \* (sizeof(\*next\_free\_block) + sizeof(\*prev\_free\_block))大小的内存..

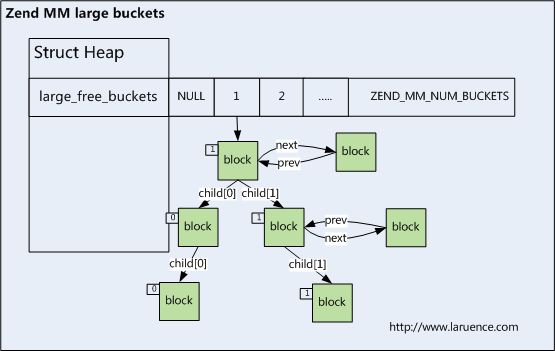
来看ZEND\_MM\_SMALL\_FREE\_BUCKET宏的定义:

1. #define ZEND\_MM\_SMALL\_FREE\_BUCKET(heap, index) \
2. (zend\_mm\_free\_block\*) ((char\*)&heap->free\_buckets[index \* 2] + \
3. sizeof(zend\_mm\_free\_block\*) \* 2 - \
4. sizeof(zend\_mm\_small\_free\_block))

之后, Zend MM 保证只会用prev和next俩个指针, 所以不会造成内存读错..

那么, 第二个不容易看懂的点, 就是PHP对large\_free\_buckets的管理, 先介绍分配(TIPI项目组对此部分的描述有些含糊不清):

1. static zend\_mm\_free\_block \*zend\_mm\_search\_large\_block(zend\_mm\_heap \*heap, size\_t true\_size)

large\_free\_buckets可以说是一个建树和双向列表的结合:  


large\_free\_buckets使用一个宏来决定某个大小的内存, 落在什么index上:

1. #define ZEND\_MM\_LARGE\_BUCKET\_INDEX(S) zend\_mm\_high\_bit(S)

zend\_mm\_high\_bit获取true\_size中最高位1的序号(zend\_mm\_high\_bit), 对应的汇编指令是bsr(此处, TIPI项目错误的说明为: “这个hash函数用来计算size的位数，返回值为size二进码中1的个数-1″).

也就是说, 每一个在large\_free\_buckets中的元素, 都保持着指向一个大小为在对应index处为1的size的内存块的指针. 诶, 有点绕口, 举个例子:

比如对于large\_free\_buckets[2], 就只会保存, 大小在0b1000到0b1111大小的内存. 再比如: large\_free\_buckets[6], 就保存着大小为0b10000000到0b11111111大小的内存的指针.

这样, 再分配内存的时候, Zend MM就可以快速定位到最可能适合的区域来查找. 提高性能.

而, 每一个元素又同时是一个双向列表, 保持着同样size的内存块, 而左右孩子(child[0]和child[1])分别代表着键值0和1, 这个键值是指什么呢?

我们来举个例子, 比如我向PHP申请一个true\_size为0b11010大小的内存, 经过一番步骤以后, 没有找到合适的内存, PHP进入了zend\_mm\_search\_large\_block的逻辑来在large\_free\_buckets中寻找合适的内存:

1. 首先, 计算true\_size对应的index, 计算方法如之前描述的ZEND\_MM\_LARGE\_BUCKET\_INDEX

2. 然后在一个位图结构中, 判断是否存在一个大于true\_size的可用内存已经存在于large\_free\_buckets, 如果不存在就返回:

1. size\_t bitmap = heap->large\_free\_bitmap >> index;
2. if (bitmap == 0) {
3. return NULL;
4. }

3. 判断, free\_buckets[index]是否存在可用的内存:

1. if (UNEXPECTED((bitmap & 1) != 0))

4. 如果存在, 则从free\_buckets[index]开始, 寻找最合适的内存, 步骤如下:

4.1. 从free\_buckets[index]开始, 如果free\_buckets[index]当前的内存大小和true\_size相等, 则寻找结束, 成功返回.

4.2. 查看true\_size对应index后(true\_size << (ZEND\_MM\_NUM\_BUCKETS - index))的当前最高位, 如果为1. 则在free\_buckets[index]->child[1]下面继续寻找, 如果free\_buckets[index]->child[1]不存在, 则跳出. 如果true\_size的当前最高位为0, 则在free\_buckets[index]->child[0]下面继续寻找, 如果free\_buckets[index]->child[0]不存在, 则在free\_buckets[index]->child[1]下面寻找最小内存(因为此时可以保证, 在free\_buckets[index]->child[1]下面的内存都是大于true\_size的)

4.3. 出发点变更为2中所述的child, 左移一位ture\_size.

5. 如果上述逻辑并没有找到合适的内存, 则寻找最小的”大块内存”:

1. /\* Search for smallest "large" free block \*/
2. best\_fit = p = heap->large\_free\_buckets[index + zend\_mm\_low\_bit(bitmap)];
3. while ((p = p->child[p->child[0] != NULL])) {
4. if (ZEND\_MM\_FREE\_BLOCK\_SIZE(p) < ZEND\_MM\_FREE\_BLOCK\_SIZE(best\_fit)) {
5. best\_fit = p;
6. }
7. }

注意上面的逻辑, (p = p->child[p->child[0] != NULL]), PHP在尽量寻找最小的内存.

为什么说, large\_free\_buckets是个键树呢, 从上面的逻辑我们可以看出, PHP把一个size, 按照二进制的0,1做键, 把内存大小信息反应到了键树上, 方便了快速查找.

另外, 还有一个rest\_buckets, 这个结构是个双向列表, 用来保存一些PHP分配后剩下的内存, 避免无意义的把剩余内存插入free\_buckets带来的性能问题(此处, TIPI项目错误的描述为: “这是一个只有两个元素的数组。 而我们常用的插入和查找操作是针对第一个元素，即heap->rest\_buckets[0]“).