[PHP数组实际占用内存大小的分析](http://blog.csdn.net/hguisu/article/details/7376705)

标签： [php](http://www.csdn.net/tag/php" \t "_blank)[zend](http://www.csdn.net/tag/zend)[struct](http://www.csdn.net/tag/struct)[traversal](http://www.csdn.net/tag/traversal)[byte](http://www.csdn.net/tag/byte)[存储](http://www.csdn.net/tag/%e5%ad%98%e5%82%a8)

2012-03-21 10:49 15065人阅读 [评论](http://blog.csdn.net/hguisu/article/details/7376705#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/hguisu/article/details/7376705#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

PHP（40） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg PHP内核（10） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

我们在前面的php高效写法提到，尽量不要复制变量，特别是数组。一般来说，PHP数组的内存利用率只有 1/10, 也就是说，一个在C语言里面100M 内存的数组，在PHP里面就要1G。下面我们可以粗略的估算PHP数组占用内存的大小,首先我们测试1000个元素的整数占用的内存：

**[php]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. <?php
2. echo memory\_get\_usage() , '<br>';
3. $start = memory\_get\_usage();
4. $a = Array();
5. **for** ($i=0; $i<1000; $i++) {
6. $a[$i] = $i + $i;
7. }
8. $mid =  memory\_get\_usage();
9. echo memory\_get\_usage() , '<br>';
10. **for** ($i=1000; $i<2000; $i++) {
11. $a[$i] = $i + $i;
12. }
13. $end =  memory\_get\_usage();
14. echo memory\_get\_usage() , '<br>';
15. echo 'argv:', ($mid - $start)/1000 ,'bytes' , '<br>';
16. echo 'argv:',($end - $mid)/1000 ,'bytes' , '<br>';

输出是:

353352  
    437848  
    522024  
    argv:84.416bytes  
    argv:84.176bytes

大概了解1000 个元素的整数数组需要占用 82k 内存，平均每个元素占用 84 个字节。而纯 C 中整体只需要 4k(一个整型占用4byte \* 1000 )。memory\_get\_usage() 返回的结果并不是全是被数组占用了，还要包括一些 PHP 运行本身分配的一些结构，可能用内置函数生成的数组更接近真实的空间:

**[php]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. <?php
2. $start = memory\_get\_usage();
3. $a = array\_fill(0, 10000, 1);
4. $mid = memory\_get\_usage(); //10k elements array;
5. echo 'argv:', ($mid - $start )/10000,'byte' , '<br>';
6. $b = array\_fill(0, 10000, 1);
7. $end = memory\_get\_usage(); //10k elements array;
8. echo 'argv:', ($end - $mid)/10000 ,'byte' , '<br>';

得到:  
argv:54.5792byte  
argv:54.5784byte

从这个结果来看似乎一个数组元素大约占用了54个字节左右。

首先看一下32位机C语言各种类型占用的字节：

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. #include "stdafx.h"
2. //#include <stdio.h>
4. **int** main() {
5. printf("int:%d\nlong:%d\ndouble:%d\nchar\*:%d\nsize\_t:%d\n",
6. **sizeof**(**int**), **sizeof**(**long**),
7. **sizeof**(**double**), **sizeof**(**char** \*),
8. **sizeof**(**size\_t**));
9. **return**   0;
10. }



int:4  
long:4  
double:8  
har\*:4  
size\_t:4  
在PHP中都使用long类型来代表数字，没有使用int类型  
大家都明白PHP是一种弱类型的语言，它不会去区分变量的类型，没有int float char \*之类的概念。  
我们看看php在zend里面存储的变量，PHP中每个变量都有对应的 zval， Zval结构体定义在Zend/zend.h里面，其结构:

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. **typedef** **struct** \_zval\_struct zval;
2. **struct** \_zval\_struct {
3. /\* Variable information \*/
4. zvalue\_value value;     /\* The value 1 12字节(32位机是12，64位机需要8+4+4=16) \*/
5. zend\_uint refcount\_\_gc; /\* The number of references to this value (for GC) 4字节 \*/
6. zend\_uchar type;        /\* The active type 1字节\*/
7. zend\_uchar is\_ref\_\_gc;  /\* Whether this value is a reference (&) 1字节\*/
8. };

PHP使用一种UNION结构来存储变量的值,即zvalue\_value 是一个union，UNION变量所占用的内存是由最大

成员数据空间决定。

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. **typedef** **union** \_zvalue\_value {
2. **long** lval;                  /\* long value \*/
3. **double** dval;                /\* double value \*/
4. **struct** {                    /\* string value \*/
5. **char** \*val;
6. **int** len;
7. } str;
8. HashTable \*ht;              /\* hash table value \*/
9. zend\_object\_value obj;      /\*object value \*/
10. } zvalue\_value;

 最大成员数据空间是struct str，指针占\*val用4字节，INT占用4字节，共8字节。

       struct zval占用的空间为8+4+1+1 = 14字节，

      其实呢，在zval中数组，字符串和对象还需要另外的存储结构，数组则是一个 HashTable:

   HashTable结构体定义在Zend/zend\_hash.h.

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. **typedef** **struct** \_hashtable {
2. uint nTableSize;//4
3. uint nTableMask;//4
4. uint nNumOfElements;//4
5. ulong nNextFreeElement;//4
6. Bucket \*pInternalPointer;   /\* Used for element traversal 4\*/
7. Bucket \*pListHead;//4
8. Bucket \*pListTail;//4
9. Bucket \*\*arBuckets;//4
10. dtor\_func\_t pDestructor;//4
11. zend\_bool persistent;//1
12. unsigned **char** nApplyCount;//1
13. zend\_bool bApplyProtection;//1
14. #if ZEND\_DEBUG
15. **int** inconsistent;//4
16. #endif
17. } HashTable;

HashTable 结构需要 39 个字节，每个数组元素存储在 Bucket 结构中:

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7376705) [copy](http://blog.csdn.net/hguisu/article/details/7376705)

[print?](http://blog.csdn.net/hguisu/article/details/7376705)

1. **typedef** **struct** bucket {
2. ulong h;    /\* Used for numeric indexing                4字节 \*/
3. uint nKeyLength;    /\* The length of the key (for string keys)  4字节 \*/
4. **void** \*pData;        /\* 4字节\*/
5. **void** \*pDataPtr;         /\* 4字节\*/
6. **struct** bucket \*pListNext;  /\* PHP arrays are ordered. This gives the next element in that order4字节\*/
7. **struct** bucket \*pListLast;  /\* and this gives the previous element           4字节 \*/
8. **struct** bucket \*pNext;      /\* The next element in this (doubly) linked list     4字节\*/
9. **struct** bucket \*pLast;      /\* The previous element in this (doubly) linked list     4字节\*/
10. **char** arKey[1];            /\* Must be last element   1字节\*/
11. } Bucket;

Bucket 结构需要 33 个字节，键长超过四个字节的部分附加在 Bucket 后面，而元素值很可能是一个 zval 结构，另外每个数组会分配一个由 arBuckets 指向的 Bucket 指针数组， 虽然不能说每增加一个元素就需要一个指针，但是实际情况可能更糟。这么算来一个数组元素就会占用 54 个字节，与上面的估算几乎一样。

    一个空数组至少会占用 14(zval) + 39(HashTable) + 33(arBuckets) = 86 个字节，作为一个变量应该在符号表中有个位置，也是一个数组元素，因此一个空数组变量需要 118 个字节来描述和存储。从空间的角度来看，小型数组平均代价较大，当然一个脚本中不会充斥数量很大的小型数组，可以以较小的空间代价来获取编程上的快捷。但如果将数组当作容器来使用就是另一番景象了，实际应用经常会遇到多维数组，而且元素居多。比如10k个元素的一维数组大概消耗540k内存，而10k x 10 的二维数组理论上只需要 6M 左右的空间，但是按照 memory\_get\_usage 的结果则两倍于此，[10k,5,2]的三维数组居然消耗了23M，小型数组果然是划不来的。