**前言**

PHP是一门托管型语言，在PHP编程中程序员不需要手工处理内存资源的分配与释放（使用C编写PHP或Zend扩展除外），这就意味着PHP本身实现了垃圾回收机制（Garbage Collection）。现在如果去PHP官方网站（[php.net](http://php.net/)）可以看到，目前PHP5的两个分支版本PHP5.2和PHP5.3是分别更新的，这是因为许多项目仍然使用5.2版本的PHP，而5.3版本对5.2并不是完全兼容。PHP5.3在PHP5.2的基础上做了诸多改进，其中垃圾回收算法就属于一个比较大的改变。本文将分别讨论PHP5.2和PHP5.3的垃圾回收机制，并讨论这种演化和改进对于程序员编写PHP的影响以及要注意的问题。

**PHP变量及关联内存对象的内部表示**

垃圾回收说到底是对变量及其所关联内存对象的操作，所以在讨论PHP的垃圾回收机制之前，先简要介绍PHP中变量及其内存对象的内部表示（其C源代码中的表示）。

PHP官方文档中将PHP中的变量划分为两类：标量类型和复杂类型。标量类型包括布尔型、整型、浮点型和字符串；复杂类型包括数组、对象和资源；还有一个NULL比较特殊，它不划分为任何类型，而是单独成为一类。

所有这些类型，在PHP内部统一用一个叫做zval的结构表示，在PHP源代码中这个结构名称为“\_zval\_struct”。zval的具体定义在PHP源代码的“Zend/zend.h”文件中，下面是相关代码的摘录。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | typedef union \_zvalue\_value {      long lval;                  /\* long value \*/      double dval;                /\* double value \*/      struct {          char \*val;          int len;      } str;      HashTable \*ht;              /\* hash table value \*/      zend\_object\_value obj;  } zvalue\_value;    struct \_zval\_struct {      /\* Variable information \*/      zvalue\_value value;     /\* value \*/      zend\_uint refcount\_\_gc;      zend\_uchar type;    /\* active type \*/      zend\_uchar is\_ref\_\_gc;  }; |

其中联合体“\_zvalue\_value”用于表示PHP中所有变量的值，这里之所以使用union，是因为一个zval在一个时刻只能表示一种类型的变量。可以看到\_zvalue\_value中只有5个字段，但是PHP中算上NULL有8种数据类型，那么PHP内部是如何用5个字段表示8种类型呢？这算是PHP设计比较巧妙的一个地方，它通过复用字段达到了减少字段的目的。例如，在PHP内部布尔型、整型及资源（只要存储资源的标识符即可）都是通过lval字段存储的；dval用于存储浮点型；str存储字符串；ht存储数组（注意PHP中的数组其实是哈希表）；而obj存储对象类型；如果所有字段全部置为0或NULL则表示PHP中的NULL，这样就达到了用5个字段存储8种类型的值。

而当前zval中的value（value的类型即是\_zvalue\_value）到底表示那种类型，则由“\_zval\_struct”中的type确定。\_zval\_struct即是zval在C语言中的具体实现，每个zval表示一个变量的内存对象。除了value和type，可以看到\_zval\_struct中还有两个字段refcount\_\_gc和is\_ref\_\_gc，从其后缀就可以断定这两个家伙与垃圾回收有关。没错，PHP的垃圾回收全靠这俩字段了。其中refcount\_\_gc表示当前有几个变量引用此zval，而is\_ref\_\_gc表示当前zval是否被按引用引用，这话听起来很拗口，这和PHP中zval的“Write-On-Copy”机制有关，由于这个话题不是本文重点，因此这里不再详述，读者只需记住refcount\_\_gc这个字段的作用即可。

**PHP5.2中的垃圾回收算法——Reference Counting**

PHP5.2中使用的内存回收算法是大名鼎鼎的[Reference Counting](http://en.wikipedia.org/wiki/Reference_counting)，这个算法中文翻译叫做“引用计数”，其思想非常直观和简洁：为每个内存对象分配一个计数器，当一个内存对象建立时计数器初始化为1（因此此时总是有一个变量引用此对象），以后每有一个新变量引用此内存对象，则计数器加1，而每当减少一个引用此内存对象的变量则计数器减1，当垃圾回收机制运作的时候，将所有计数器为0的内存对象销毁并回收其占用的内存。而PHP中内存对象就是zval，而计数器就是refcount\_\_gc。

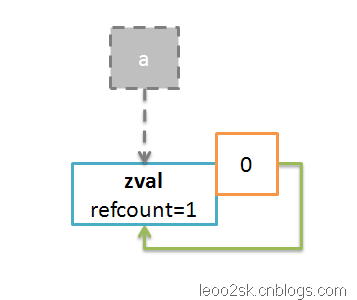
例如下面一段PHP代码演示了PHP5.2计数器的工作原理（计数器值通过[xdebug](http://www.xdebug.org/)得到）：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php    $val1 = 100; //zval(val1).refcount\_gc = 1;  $val2 = $val1; //zval(val1).refcount\_gc = 2,zval(val2).refcount\_gc = 2(因为是Write on copy，当前val2与val1共同引用一个zval)  $val2 = 200; //zval(val1).refcount\_gc = 1,zval(val2).refcount\_gc = 1(此处val2新建了一个zval)  unset($val1); //zval(val1).refcount\_gc = 0($val1引用的zval再也不可用，会被GC回收)    ?> |

Reference Counting简单直观，实现方便，但却存在一个致命的缺陷，就是容易造成内存泄露。很多朋友可能已经意识到了，如果存在循环引用，那么Reference Counting就可能导致内存泄露。例如下面的代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php    $a = array();  $a[] = & $a;  unset($a);    ?> |

这段代码首先建立了数组a，然后让a的第一个元素按引用指向a，这时a的zval的refcount就变为2，然后我们销毁变量a，此时a最初指向的zval的refcount为1，但是我们再也没有办法对其进行操作，因为其形成了一个循环自引用，如下图所示：

[](http://images.cnblogs.com/cnblogs_com/leoo2sk/201102/20110226165422276.png)

其中灰色部分表示已经不复存在。由于a之前指向的zval的refcount为1（被其HashTable的第一个元素引用），这个zval就不会被GC销毁，这部分内存就泄露了。

这里特别要指出的是，PHP是通过符号表（Symbol Table）存储变量符号的，全局有一个符号表，而每个复杂类型如数组或对象有自己的符号表，因此上面代码中，a和a[0]是两个符号，但是a储存在全局符号表中，而a[0]储存在数组本身的符号表中，且这里a和a[0]引用同一个zval（当然符号a后来被销毁了）。希望读者朋友注意分清符号（Symbol）的zval的关系。

在PHP只用于做动态页面脚本时，这种泄露也许不是很要紧，因为动态页面脚本的生命周期很短，PHP会保证当脚本执行完毕后，释放其所有资源。但是PHP发展到目前已经不仅仅用作动态页面脚本这么简单，如果将PHP用在生命周期较长的场景中，例如自动化测试脚本或deamon进程，那么经过多次循环后积累下来的内存泄露可能就会很严重。这并不是我在耸人听闻，我曾经实习过的一个公司就通过PHP写的deamon进程来与数据存储服务器交互。

由于Reference Counting的这个缺陷，PHP5.3改进了垃圾回收算法。

**PHP5.3中的垃圾回收算法——Concurrent Cycle Collection in Reference Counted Systems**

PHP5.3的垃圾回收算法仍然以引用计数为基础，但是不再是使用简单计数作为回收准则，而是使用了一种同步回收算法，这个算法由IBM的工程师在论文[Concurrent Cycle Collection in Reference Counted Systems](http://www.research.ibm.com/people/d/dfb/papers/Bacon01Concurrent.pdf)中提出。

这个算法可谓相当复杂，从论文29页的数量我想大家也能看出来，所以我不打算（也没有能力）完整论述此算法，有兴趣的朋友可以阅读上面的提到的论文（强烈推荐，这篇论文非常精彩）。

我在这里，只能大体描述一下此算法的基本思想。

首先PHP会分配一个固定大小的“根缓冲区”，这个缓冲区用于存放固定数量的zval，这个数量默认是10,000，如果需要修改则需要修改源代码Zend/zend\_gc.c中的常量GC\_ROOT\_BUFFER\_MAX\_ENTRIES然后重新编译。

由上文我们可以知道，一个zval如果有引用，要么被全局符号表中的符号引用，要么被其它表示复杂类型的zval中的符号引用。因此在zval中存在一些可能根（root）。这里我们暂且不讨论PHP是如何发现这些可能根的，这是个很复杂的问题，总之PHP有办法发现这些可能根zval并将它们投入根缓冲区。

当根缓冲区满额时，PHP就会执行垃圾回收，此回收算法如下：

1、对每个根缓冲区中的根zval按照深度优先遍历算法遍历所有能遍历到的zval，并将每个zval的refcount减1，同时为了避免对同一zval多次减1（因为可能不同的根能遍历到同一个zval），每次对某个zval减1后就对其标记为“已减”。

2、再次对每个缓冲区中的根zval深度优先遍历，如果某个zval的refcount不为0，则对其加1，否则保持其为0。

3、清空根缓冲区中的所有根（注意是把这些zval从缓冲区中清除而不是销毁它们），然后销毁所有refcount为0的zval，并收回其内存。

如果不能完全理解也没有关系，只需记住PHP5.3的垃圾回收算法有以下几点特性：

1、并不是每次refcount减少时都进入回收周期，只有根缓冲区满额后在开始垃圾回收。

2、可以解决循环引用问题。

3、可以总将内存泄露保持在一个阈值以下。

**PHP5.2与PHP5.3垃圾回收算法的性能比较**

由于我目前条件所限，我就不重新设计试验了，而是直接引用PHP Manual中的实验，关于两者的性能比较请参考PHP Manual中的相关章节：<http://www.php.net/manual/en/features.gc.performance-considerations.php>。

首先是内存泄露试验，下面直接引用[PHP Manual](http://www.php.net/manual/en/index.php)中的实验代码和试验结果图：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php  class Foo  {      public $var = '3.1415962654';  }    $baseMemory = memory\_get\_usage();    for ( $i = 0; $i <= 100000; $i++ )  {      $a = new Foo;      $a->self = $a;      if ( $i % 500 === 0 )      {          echo sprintf( '%8d: ', $i ), memory\_get\_usage() - $baseMemory, "\n";      }  }  ?> |

可以看到在可能引发累积性内存泄露的场景下，PHP5.2发生持续累积性内存泄露，而PHP5.3则总能将内存泄露控制在一个阈值以下（与根缓冲区大小有关）。

另外是关于性能方面的对比：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  class Foo  {      public $var = '3.1415962654';  }    for ( $i = 0; $i <= 1000000; $i++ )  {      $a = new Foo;      $a->self = $a;  }    echo memory\_get\_peak\_usage(), "\n";  ?> |

这个脚本执行1000000次循环，使得延迟时间足够进行对比。

然后使用CLI方式分别在打开内存回收和关闭内存回收的的情况下运行此脚本：

|  |  |
| --- | --- |
| 1  2  3 | time php -dzend.enable\_gc=0 -dmemory\_limit=-1 -n example2.php  # and  time php -dzend.enable\_gc=1 -dmemory\_limit=-1 -n example2.php |

在我的机器环境下，运行时间分别为6.4s和7.2s，可以看到PHP5.3的垃圾回收机制会慢一些，但是影响并不大。

**与垃圾回收算法相关的PHP配置**

可以通过修改php.ini中的zend.enable\_gc来打开或关闭PHP的垃圾回收机制，也可以通过调用gc\_enable()或gc\_disable()打开或关闭PHP的垃圾回收机制。在PHP5.3中即使关闭了垃圾回收机制，PHP仍然会记录可能根到根缓冲区，只是当根缓冲区满额时，PHP不会自动运行垃圾回收，当然，任何时候您都可以通过手工调用gc\_collect\_cycles()函数强制执行内存回收。