**PHP的gc**

php的垃圾回收机制主要有三个方面的知识

* 引用计数基本知识（PHP的gc回收算法采用的是引用计数法）
* 回收周期(Collecting Cycles)
* 性能方面考虑的因素

**引用计数基本知识**

每个php变量存在一个叫做"zval"的变量容器中.一个zval变量容器,除了包含变量的类型和值,还包括两个字节的额外信息.第一个是"is\_ref",是个bool值,用来标识这个变量是否是属于引用集合(reference set).通过这个字节,php引擎才能把普通变量和引用变量区分开.由于php允许用户通过使用&来使用自定义引用,zval变量容器中还有一个内部引用计数机制,来优化内存使用.第二个额外字节是"refcount",用来表示指向这个zval变量容器的变量(也称符号即symbol)个数.

当一个变量被赋常量值时,就会生成一个zval变量容器,如下例所示:

**[php]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. <?php
2. $a = "new string";
4. ?>

在上例中,新的变量是a,是在当前作用域中生成的.并且生成了类型为string和值为"new string"的变量容器.在额外的两个字节信息中,"is\_ref"被默认设置为false,因为没有任何自定义的引用生成."refcount"被设定为1,因为这里只有一个变量使用这个变量容器.调用xdebug查看一下变量内容:

**[php]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. <?php
2. $a = "new string";
4. xdebug\_debug\_zval('a');
5. ?>

以上代码会输出:

**[html]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. a: (refcount=1, is\_ref=0)='new string'

对变量a增加一个引用计数

**[php]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. <?php
2. $a = "new string";
4. $b = $a;
5. xdebug\_debug\_zval('a');
6. ?>

以上代码会输出:

**[html]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. a: (refcount=2, is\_ref=0)='new string'

这时,引用次数是2,因为同一变量容器被变量a和变量b关联.当没必要时,php不会去复制已生成的变量容器.变量容器在"refcount"变成0时就被销毁.当任何关联到某个变量容易的变量离开它的作用域(比如:函数执行结束),或者对变量调用了unset()函数,"refcount"就会减1,下面例子就能说明:

**[php]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. <?php
2. $a = "new string";
3. $b = $c = $a;
4. xdebug\_debug\_zval('a');
5. unset($b, $c);
6. xdebug\_debug\_zval('a');
7. ?>

以上代码会输出:

**[html]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. a: (refcount=3, is\_ref=0)='new string' a: (refcount=1, is\_ref=0)='new string'

如果我们现在执行unset($a),$包含的类型和值的这个容器就会从内存删除

**复合类型(compound types)**

当考虑像array和object这样的复合类型时,事情会稍微有些复杂.与标量(scalar)类型的值不同,array和object类型的变量把它们的成员或属性存在自己的符号表中.这意味着下面的例子将生成三个zval变量容器

**[php]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. <?php
2. $a = **array**('meaning' => 'life', 'number' => 42);
4. xdebug\_debug\_zval('a');
5. ?>

以上代码输出:

**[html]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. a: (refcount=1, is\_ref=0)=array ('meaning' =**>** (refcount=1, is\_ref=0)='life', 'number' =**>** (refcount=1, is\_ref=0)=42)

这三个zval变量容器是:a,meaning,number.增加和减少refcount的规则和上面提到的一样

特例,添加数组本身作为数组元素时:

**[php]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. <?php
2. $a = **array**('one');
4. $a[] = &$a;
6. xdebug\_debug\_zval('a');
7. ?>

以上代码输出的结果:

**[html]** [view plain](http://blog.csdn.net/wzy_1988/article/details/10140217) [copy](http://blog.csdn.net/wzy_1988/article/details/10140217)

[print?](http://blog.csdn.net/wzy_1988/article/details/10140217)

1. a: (refcount=2, is\_ref=1)=array (0 =**>** (refcount=1, is\_ref=0)='one', 1 =**>** (refcount=2, is\_ref=1)=...)

可以看到数组a和数组本身元素a[1]指向的变量容器refcount为2

当对数组$a调用unset函数时,$a的refcount变为1,发生了内存泄漏

**清理变量容器的问题**

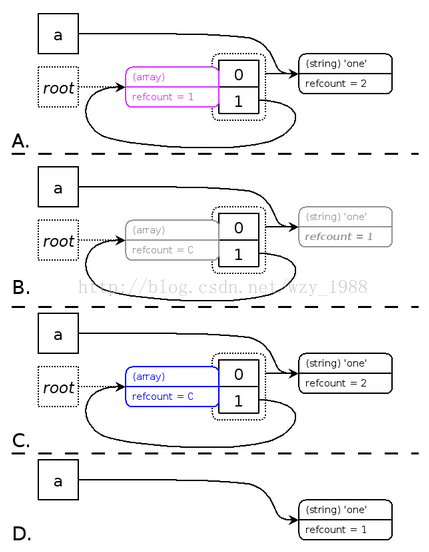
尽管不再有某个作用域中的任何符号指向这个结构(就是变量容器),由于数组元素"1"仍然指向数组本身,所以这个容器不能被消除.因为没有另外的符号指向它,用户没有办法清除这个结构,结果就会导致内存泄漏.庆幸的是,php将在请求结束时清除这个数据结构,但是php清除前,将耗费不少内存空间

**回收周期**

5.3.0PHP使用了新的同步周期回收算法,来处理上面所说的内存泄漏问题

首先,我们先要建立一些基本规则:

如果一个引用计数增加,它将继续被使用,当然就不再垃圾中.如果引用计数减少到零,所在的变量容器将被清除(free).就是说,仅仅在引用计数减少到零值时,才会产生垃圾周期(grabage cycle).其次,在一个垃圾周期中,通过检查引用计数是否减1,并且检查哪些变量容器的引用次数是零,来发现哪部分是垃圾



为避免不得不检查所有引用计数可能减少的垃圾周期,这个算法把所有可能根(possible roots 都是zval变量容器),放在根缓冲区(root buffer)中(用紫色标记),这样可以同时确保每个可能的垃圾根(possible garbage root)在缓冲区只出现一次.仅仅在根缓冲区满了时,才对缓冲区内部所有不同的变量容器执行垃圾回收操作,见A

深度优先遍历所有可能的根,将计数器减1,清理.具体细节这里就不讲了

**参考链接**

<http://www.php.net/manual/zh/features.gc.php>