PHP线程的内存回收问题

当一个PHP线程结束时，当前占用的所有内存空间都会被销毁。那么如果这个线程不结束，怎么回收内存呢？

refcount：引用技术器，可以理解为指向该个容器的指针个数吧。

is\_ref：是否被引用（只可能是0或者1）

赋值的流程：

<?php

$a = 'aa';

xdebug\_debug\_zval(a); //(refcount=1, is\_ref=0),string 'aa' (length=6)

$b = $a;

//以下的两个其实是一个变量容器

xdebug\_debug\_zval(a); //(refcount=2, is\_ref=0),string 'aa' (length=6)

xdebug\_debug\_zval(b); //(refcount=2, is\_ref=0),string 'aa' (length=6)

unset($b); //对变量容器 refcount 减1

xdebug\_debug\_zval(a); //(refcount=1, is\_ref=0),string 'aa' (length=6)

xdebug\_debug\_zval(b); //b: no such symbol b变量被销毁，指向被断掉，如果对应容器的引用技术为零，那么该块儿内存被回收

$b = $a;

$b = 'bb';

xdebug\_debug\_zval(a); //(refcount=1, is\_ref=0),string 'aa' (length=6)

xdebug\_debug\_zval(b); //(refcount=1, is\_ref=0),string 'aa' (length=6) 重新申请一个变量容器存储b，a的变量容器引用减1

引用的流程：

<?php

$a = 'aa';

xdebug\_debug\_zval('a'); //(refcount=1, is\_ref=0),string 'aa' (length=2)

$b = & $a;

//变量容器的引用技术加1，引用标记置为1

xdebug\_debug\_zval('a'); //(refcount=2, is\_ref=1),string 'aa' (length=2)

xdebug\_debug\_zval('b'); //(refcount=2, is\_ref=1),string 'aa' (length=2)

$b = '123';

//php会发现，该容器变量是引用（is\_ref），所以容器变量不用像赋值那样再申请一个

xdebug\_debug\_zval('a'); //(refcount=2, is\_ref=1),string '123' (length=2)

xdebug\_debug\_zval('b'); //(refcount=2, is\_ref=1),string '123' (length=2)

unset($b);

//变量容器应用计数减1，引用为零

xdebug\_debug\_zval('a'); //(refcount=1, is\_ref=0),string '123' (length=2)

xdebug\_debug\_zval('b'); // b: no such symbol

那如果多次引用，unset掉一个，is\_ref是否会被置为零，那样bug不就出现了么？变量容器还是引用啊。那么我们来看看：

<?php

$a = 'aa';

$b = &$a;

$c = &$a;

//可以看到引用refCount是3，is\_ref永远是1

xdebug\_debug\_zval('a'); //(refcount=3, is\_ref=1),string 'aa' (length=2)

xdebug\_debug\_zval('b'); //(refcount=3, is\_ref=1),string 'aa' (length=2)

xdebug\_debug\_zval('c'); //(refcount=3, is\_ref=1),string 'aa' (length=2)

unset($b);

//我们期待的bug没有出现,只是refcount减1，is\_ref还是1

xdebug\_debug\_zval('a'); //(refcount=2, is\_ref=1),string 'aa' (length=2)

xdebug\_debug\_zval('b'); // b: no such symbol

xdebug\_debug\_zval('c'); //(refcount=2, is\_ref=1),string 'aa' (length=2)

//那php它怎么知道这个容器还有引用，毕竟is\_ref仍然是1，不能计数，那么现在refcount就起作用了，是它告诉php，该变量有几个引用，但问题又来了，如果我干点坏事，在引用的时候，又赋值，它会不会有bug

$e = $a;

//我们看到期望的bug还是没出现，这时候再赋值，就不像直接赋值那么简单refcount加1了，而是申请了一个新的变量容器

xdebug\_debug\_zval('a'); //(refcount=2, is\_ref=1),string 'aa' (length=2)

xdebug\_debug\_zval('e'); //(refcount=1, is\_ref=0),string 'aa' (length=2)

unset和赋值null都能回收变量么？很多人都错认为，这两个都能回收变量空间，其实错了，null只是把变量占用的空间变小了，从回收上来说，该容器依然存在。

<?php

$a = 'aa';

$b = $a;

$b = null;

//又申请了一个变量容器

xdebug\_debug\_zval('a'); //(refcount=1, is\_ref=0),string 'aa' (length=2)

xdebug\_debug\_zval('b'); //(refcount=1, is\_ref=0),null 变量空间并没被回收

unset($b);

//这时候才释放了b变量容器的空间

xdebug\_debug\_zval('a'); //(refcount=1, is\_ref=0),string 'aa' (length=2)

xdebug\_debug\_zval('b'); //b: no such symbol

总结

1. 垃圾回收的时机

PHP中，引用计数为0，则内存立刻释放。也就是说，不存在环状引用的变量，离开变量的作用域，内存被立刻释放。环状引用检测则是在满足一定条件下触发，所以在上面的例子中，会看到使用的内存有大幅度的波动。也可以通过 gc\_collect\_cycles 函数来主动进行环状引用检测。

2. &符号的影响

显式引用一个变量，会增加该内存的引用计数：

$a = "something";

$b = &$a;

此时unset($a), 但是仍有$b指向该内存区域的引用，内存不会释放。

3. unset函数的影响

unset只是断开一个变量到一块内存区域的连接，同时将该内存区域的引用计数-1；在上面的例子中，循环体内部，$a=new A(); unset($a);并不会将$a的引用计数减到零；

4. = null 操作的影响；

$a = null 是直接将$a 指向的数据结构置空，同时将其引用计数归0。

5. 脚本执行结束的影响

脚本执行结束，该脚本中使用的所有内存都会被释放，不论是否有引用环。