Zval是PHP中最重要的数据结构之一

基本结构为：

struct \_zval\_struct {

zvalue\_value value; /\* value \*/

zend\_uint refcount\_\_gc; /\* variable ref count \*/

zend\_uchar type; /\* active type \*/

zend\_uchar is\_ref\_\_gc; /\* if it is a ref variable \*/

};

typedef struct \_zval\_struct zval;

zval\_value value 变量的实际值，具体来说是一个zvalue\_value的联合体（union）

zend\_uint refcount\_\_gc 引用计数器

zend\_uchar type 该字段用于表明变量的实际类型。

is\_ref\_\_gc这个字段用于标记变量是否是引用变量。对于基础数据类型，该值为0，而对于引用型的变量，该值为1。

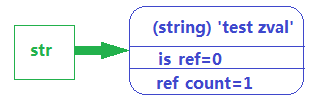
工具：xdebug输出内核调试信息

|  |
| --- |
| $str = "test zval";  xdebug\_debug\_zval('str'); |

输出结果：

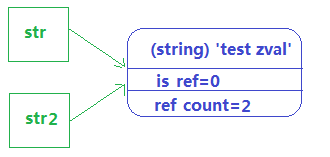
str: (refcount=1, is\_ref=0)='test zval'

实际：



$str = "test zval";

$str2 = $str;



$ar = array(

'id' => 38,

'name' => 'shine'

); xdebug\_debug\_zval('ar');

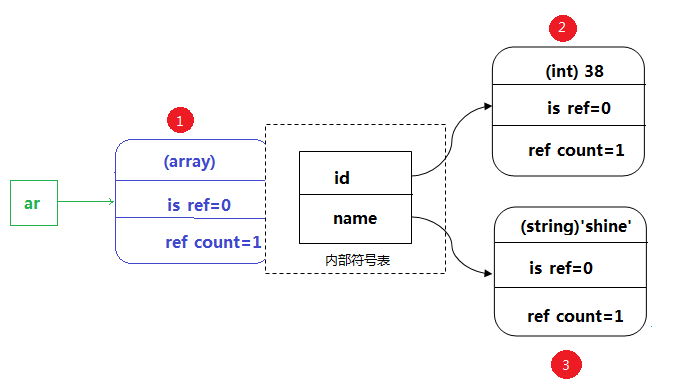
输出：

ar: (refcount=1, is\_ref=0)=array (

'id' => (refcount=1, is\_ref=0)=38,

'name' => (refcount=1, is\_ref=0)='shine'

)



变量$ar生成的过程中，共生成了3个zval容器（红色部分标注）。对于每个zval而言，refcount的增减规则与普通变量的相同。

$a = $array('one');

$a[] = &$a;

xdebug\_debug\_zval('a');

输出

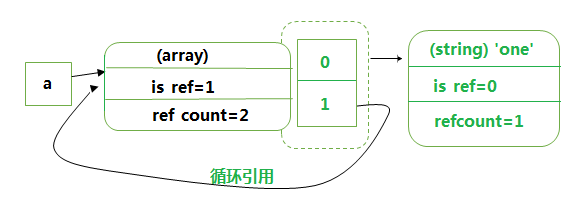
a: (refcount=2, is\_ref=1)=array (

0 => (refcount=1, is\_ref=0)='one',

1 => (refcount=2, is\_ref=1)=...

)

…表示指向原始数组，因而这是一个循环的引用。



Unset:refcount – 1

$a = "a simple test";

$b = $a;

变量修改：

echo "before write:".PHP\_EOL;

xdebug\_debug\_zval('a');

xdebug\_debug\_zval('b');

$b = "thss";

echo "after write:".PHP\_EOL;

xdebug\_debug\_zval('a');

xdebug\_debug\_zval('b');

输出：

before write:

a: (refcount=2, is\_ref=0)='a simple test'

b: (refcount=2, is\_ref=0)='a simple test'

after write:

a: (refcount=1, is\_ref=0)='a simple test'

b: (refcount=1, is\_ref=0)='thss'

起初，符号表中a和b**指向了同一个zval（和java不一样不是引用对象）**(这么做的原因是**节省内存，**php中基础数据类型、对象本质都是**值复制**)，而后$b发生了变化，Zend会检查b指向的zval的refcount是否为1，**如果是1，那么说明只有一个符号指向该zval，则直接更改zval**。**否则，说明这是一个共享的zval，需要将该zval分离出去**，以保证单独变化互不影响。