PHP7内核zval

## 一、简介

zval可以表示一切PHP中的数据类型, 所以它包含了一个type字段, 表示这个zval存储的是什么类型的值, 常见的可能选项是IS\_NULL, IS\_LONG, IS\_STRING, IS\_ARRAY, IS\_OBJECT等等。

根据type字段的值不同, 我们就要用不同的方式解读value的值, 这个value是个联合体, 比如对于type是IS\_STRING, 那么我们应该用value.str来解读zval.value字段, 而如果type是IS\_LONG, 那么我们就要用value.lval来解读。

另外, PHP是用引用计数来做基本的垃圾回收的, 所以zval中有一个refcount\_\_gc字段, 表示这个zval的引用数目, 但这里有一个要说明的, 在5.3以前, 这个字段的名字还叫做refcount, 5.3以后, 在引入新的垃圾回收算法来对付循环引用计数的时候, 作者加入了大量的宏来操作refcount, 为了能让错误更快的显现, 所以改名为refcount\_\_gc, 迫使大家都使用宏来操作refcount。

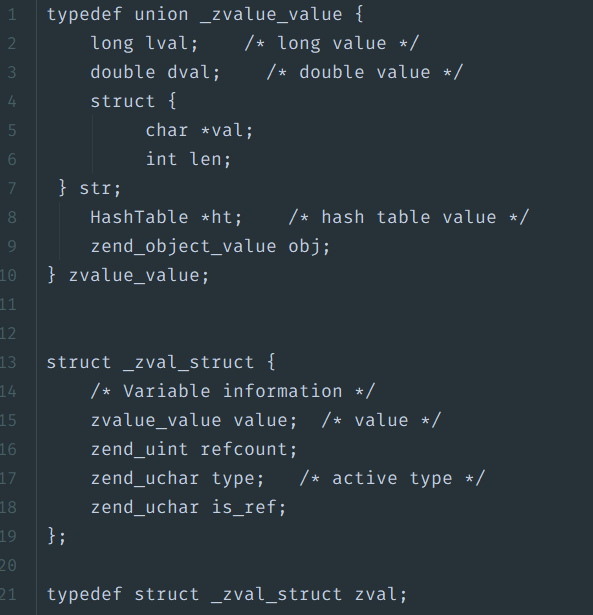
类似的, 还有is\_ref, 这个值表示了PHP中的一个类型是否是引用, 这里我们可以看到是不是引用是一个标志位。这就是PHP5时代的zval。

PHP7中的zval和PHP7中的zval从底层上做了很大调整，主要体现在类型和value字段。

## 二、PHP5中的zval

### 1、zval结构

Zend使用zval结构来存储PHP变量的值，该结构如下所示：



Zend根据type值来决定访问value的哪个成员，可用值如下：

|  |  |
| --- | --- |
| **IS\_NULL** | N/A |
| **IS\_LONG** | 对应value.lval |
| **IS\_DOUBLE** | 对应value.dval |
| **IS\_STRING** | 对应value.str |
| **IS\_ARRAY** | 对应value.ht |
| **IS\_OBJECT** | 对应value.obj |
| **IS\_BOOL** | 对应value.lval. |
| **IS\_RESOURCE** | 对应value.lval |

根据这个表格可以发现两个有意思的地方：首先是PHP的数组其实就是一个HashTable，这就解释了为什么PHP能够支持关联数组了；其次，Resource就是一个long值，它里面存放的通常是个指针、一个内部数组的index或者其它什么只有创建者自己才知道的东西，可以将其视作一个handle。

### **2、引用计数**

引用计数在垃圾收集、内存池以及字符串等地方应用广泛，Zend就实现了典型的引用计数。多个PHP变量可以通过引用计数机制来共享同一份zval，zval中剩余的两个成员is\_ref和refcount就用来支持这种共享。  
很明显，refcount用于计数，当增减引用时，这个值也相应的递增和递减，一旦减到零，Zend就会回收该zval。  
那么is\_ref呢？

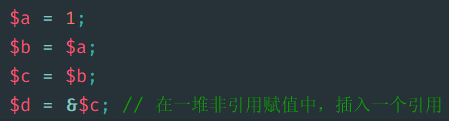
### 3、zval状态

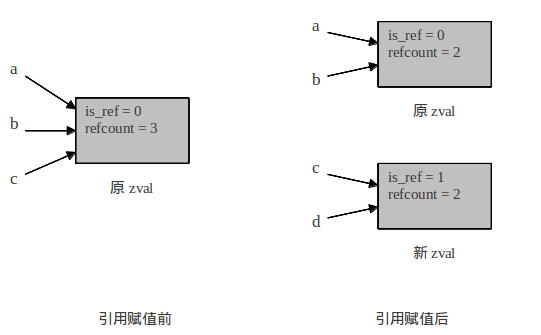
在PHP中，变量有两种——引用和非引用的，它们在Zend中都是采用引用计数的方式存储的。对于非引用型变量，要求变量间互不相干，修改一个变量时，不能影响到其他变量，采用Copy-On-Write机制即可解决这种冲突——当试图写入一个变量时，Zend若发现该变量指向的zval被多个变量共享，则为其复制一份refcount为1的zval，并递减原zval的refcount，这个过程称为“zval分离”。然而，对于引用型变量，其要求和非引用型相反，引用赋值的变量间必须是捆绑的，修改一个变量就修改了所有捆绑变量。  
可见，有必要指出当前zval的状态，以分别应对这两种情况，is\_ref就是这个目的，它指出了当前所有指向该zval的变量是否是采用引用赋值的——要么全是引用，要么全不是。此时再修改一个变量，只有当发现其zval的is\_ref为0，即非引用时，Zend才会执行Copy-On-Write。

### 4、zval状态切换

当在一个zval上进行的所有赋值操作都是引用或者都是非引用时，一个is\_ref就足够应付了。然而，世界总不会那么美好，PHP无法对用户进行这种限制，当我们混合使用引用和非引用赋值时，就必须要进行特别处理了。  
情况I、看如下PHP代码：

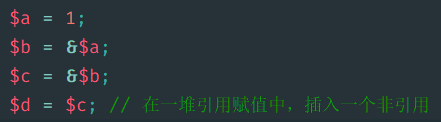
代码如下:

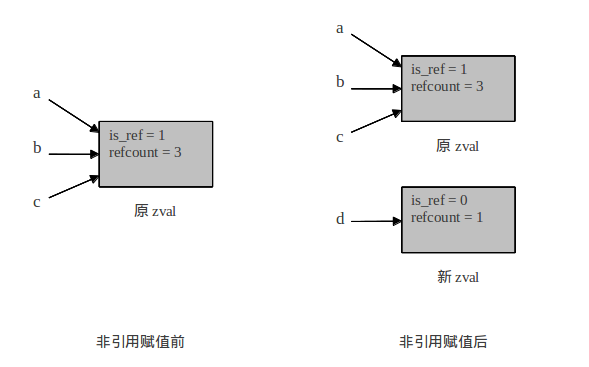
  
这段代码首先进行了一次初始化，这将创建一个新的zval，is\_ref=0, refcount=1，并将a指向这个zval；之后是两次非引用赋值，正如前面所说，只要把b和c都指向a的zval即可；最后一行是个引用赋值，需要is\_ref为1，但是Zend发现c指向的zval并不是引用型的，于是为c创建单独的zval，并同时将d指向该zval。  
从本质上来说，这也可以看作是一种Copy-On-Write，不仅仅是value，is\_ref也是受保护的对象。  
**整个过程图示如下：**



情况2，看如下PHP代码：

代码如下:

  
这段代码的前三句将把a、b和c指向一个zval，其is\_ref=1, refcount=3；第四句是个非引用赋值，通常情况下只需要增加引用计数即可，然而目标zval属于引用变量，单纯的增加引用计数显然是错误的， Zend的解决办法是为d单独生成一份zval副本。  
全过程如下所示：

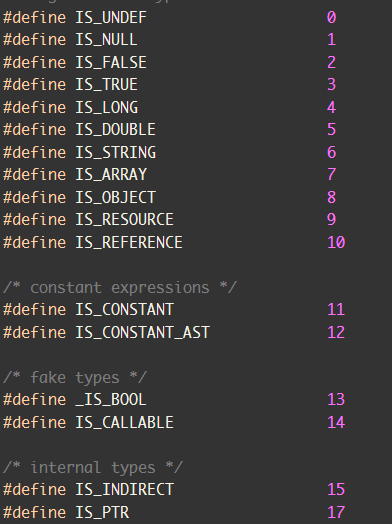


### 5、 参数传递

PHP函数参数的传递和变量赋值是一样的，非引用传递相当于非引用赋值，引用传递相当于引用赋值，并且也有可能会导致执行zval状态切换。

## 三、PHP7中的zval

PHP7中的zval的类型做了比较大的调整, 总体来说有如下17种类型:



其中PHP5的时候的IS\_BOOL类型, 现在拆分成了IS\_FALSE和IS\_TRUE俩种类型. 而原来的引用是一个标志位, 现在的引用是一种新的类型。

对于IS\_INDIRECT和IS\_PTR来说, 这俩个类型是用在内部的保留类型, 用户不会感知到。

从PHP7开始, 对于在zval的value字段中能保存下的值, 就不再对他们进行引用计数了, 而是在拷贝的时候直接赋值, 这样就省掉了大量的引用计数相关的操作, 这部分类型有:

1. IS\_LONG
2. IS\_DOUBLE

当然对于那种根本没有值, 只有类型的类型, 也不需要引用计数了:

1. IS\_NULL
2. IS\_FALSE
3. IS\_TRUE

而对于复杂类型, 一个size\_t保存不下的, 那么我们就用value来保存一个指针, 这个指针指向这个具体的值, 引用计数也随之作用于这个值上, 而不在是作用于zval上了。

另外，关于PHP7中的zval还有标志位和zval预先分配的知识。可以看下下面的参考。

参考： [深入PHP变量存储的详解](https://www.jb51.net/article/38370.htm) [深入理解PHP7内核之zval](http://www.laruence.com/2018/04/08/3170.html)