# 引用计数基本知识

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

refcount：指向这个zval变量容器的变量(也称符号即symbol)个数

is\_ref：bool值，标识变量是否是属于引用集合(reference set)，通过该字段php引擎才能把普通变量和引用变量区分开来

例：

<?php  
$a = "new string";

xdebug\_debug\_zval( 'a' );//a: (refcount=1, is\_ref=0)='new string'

$c = $b = $a;

xdebug\_debug\_zval( 'a' );// a: (refcount=3, is\_ref=0)='new string'

unset( $b, $c );

xdebug\_debug\_zval( 'a' );// a: (refcount=1, is\_ref=0)='new string'

复合类型(Compound Types)

$a = array( 'meaning' => 'life', 'number' => 42 );  
xdebug\_debug\_zval( 'a' );

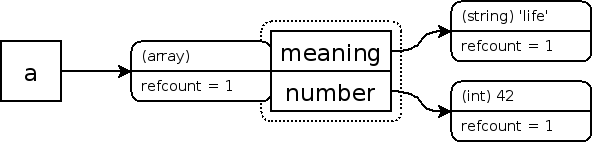
a: (refcount=1, is\_ref=0)=array (

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

'number' => (refcount=1, is\_ref=0)=42

)

图示

:

**添加一个已经存在的元素到数组中**

<?php  
$a = array( 'meaning' => 'life', 'number' => 42 );  
$a['life'] = $a['meaning'];  
xdebug\_debug\_zval( 'a' );

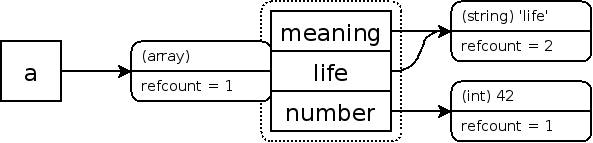
a: (refcount=1, is\_ref=0)=array (

'meaning' => (refcount=2, is\_ref=0)='life',

'number' => (refcount=1, is\_ref=0)=42,

'life' => (refcount=2, is\_ref=0)='life'

)



**从数组中删除一个元素**

<?php  
$a = array( 'meaning' => 'life', 'number' => 42 );  
$a['life'] = $a['meaning'];  
unset( $a['meaning'], $a['number'] );  
xdebug\_debug\_zval( 'a' );

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

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

)

**把数组作为一个元素添加到自己**

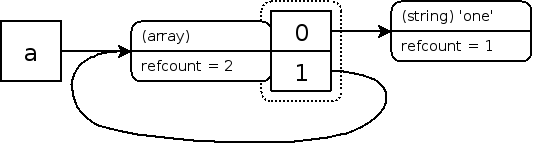
<?php  
$a = array( 'one' );  
$a[] =& $a;  
xdebug\_debug\_zval( 'a' );

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

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

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

)



清理变量容器的问题(Cleanup Problems)

php将在脚本执行结束时清除这个数据结构，但是在php清除之前，将耗费不少内存

eZ（php library）组件库的模板组件做单元测试 ----- 是个框架 <https://gitee.com/xiaozhuai/ez>

# 内存管理基础

PHP 变量：标签和实际变量容器

PHP 的内存管理也很重视 [memory\_limit(内存限制)](http://php.net/manual/zh/ini.core.php#ini.memory-limit)。

特殊的内存管理器：请求相关数据是指只需要服务于单个请求，最迟会在请求结束时释放的数据

| **要的内存 APIs** | |
| --- | --- |
| **原型** | **说明** |
| void \*emalloc(size\_t size) | 分配 size 字节的内存。 |
| void \*ecalloc(size\_t nmemb, size\_t size) | 给 nmemb 元素分配 size 字节的缓冲区并初始化为零。 |
| void \*erealloc(void \*ptr, size\_t size) | 修改使用 emalloc 分配的缓冲区 ptr 的大小为 size 字节。 |
| void efree(void \*ptr) | 释放 ptr 指向的缓冲区。缓冲区必须是由 emalloc 分配的。 |
| void \*safe\_emalloc(size\_t nmemb, size\_t size, size\_t offset) | 分配缓冲区来存放每块大小为 size 字节的 nmemb 块，并附加 offset 字节。类似于 emalloc(nmemb \* size + offset)，但增加了针对溢出的特殊保护。 |
| char \*estrdup(const char \*s) | 分配一个可存放 NULL 结尾的字符串 s 的缓冲区，并将 s 复制到缓冲区内。 |
| char \*estrndup(const char \*s, unsigned int length) | 类似于 estrdup，但 NULL 结尾的字符串长度是已知的。 |

**Note**: 和与 C 标准库相似的部分不同，如果分配请求的内存出错，Zend 引擎的内存管理函数不会返回 NULL 值，而会跳出并中止当前请求

# 变量的使用

PHP 变量：由两部分组成：标签（例如，可能是符号表中的一个条目）和实际变量容器（zval）

long lval; /\* long value \*/

double dval; /\* double value \*/

NULL：不需要值，NULL 就是此类型的值

boolean：为 false 时存放 0，为 true 时存放 1

resource：存放的是资源的 id

## 函数的编写

参数信息解释：例

void zif\_hello\_world(int ht, zval \*return\_value, zval \*\*return\_value\_ptr,zval \*this\_ptr, int return\_value\_used TSRMLS\_DC){}

| 名称和类型 | 描述 | 访问宏 |
| --- | --- | --- |
| int ht | 用户实际传递参数的数量 | ZEND\_NUM\_ARGS() |
| zval \*return\_value | PHP 变量的指针，可填充返回值传递给用户。默认值是 IS\_NULL。 | RETVAL\_\*, RETURN\_\* |
| zval \*\*return\_value\_ptr | 当返回引用时，PHP 将其设为变量的指针。不建议返回引用。 |  |
| zval \*this\_ptr | 假如这是一个方法调用，其指向存放 $this 对象的 PHP 变量。 | getThis() |
| int return\_value\_used | 指示返回值是否会被调用者使用的标志。 caller. |  |

函数执行顺序：某些情况下，这些步骤的实际顺序可能发生变化。尤其是最后两步经常颠倒，但最好保持那个顺序

 局部变量定义。在 C 语言中必须在函数的开头定义局部变量。

 解析的参数。PHP 传递参数到一个特殊的堆栈上，从那里对参数进行读取、校验类型，有问题时根据需要可进行转换。

 实际逻辑。做所需做的。

 设定返回值，清理，返回

参数信息解释2：

php\_printf：不是打印到进程的 STDOUT 通道，而是到当前的输出流，可被用户进行缓冲，不是二进制安全的，想要二进制安全输出应使用 PHPWRITE

RETURN\_TRUE：：将 return\_value 指针变量的类型设为 IS\_BOOLEAN，并将其值设为 true 值，然后从 C 函数中返回

zend\_parse\_parameters()：读取用户从参数堆栈传递来参数，将其适当地转换后放入局部 C 语言变量，不可转换时返回FAILURE，默认返回值为NULL

FAILURE 表示为 -1，SUCCESS 表示为 0。为了使代码清晰，应总是使用已命名的常量而不是其值，不能是最后的参数，因为在 zend\_parse\_parameters 内要求有不定数量的参数——依赖于要读取的用户参数的数量

zend\_parse\_parameters(ZEND\_NUM\_ARGS() TSRMLS\_CC, "s", &name, &name\_len)

zend\_parse\_parameters() 的第一个参数是用户实际传递到函数的参数数量，为了与 PHP 的线程隔离、线程安全资源管理器兼容，还要用 TSRMLS\_CC 传递线程上下文

s：是个类型

最后一个是传递一个或多个指针给要填充变量值的 C 变量，或提供更多细节。比如字符串，事实上的字符串，总是以 NULL 结尾，以 char\*，且其长度是除 NULL 字节外的 int 型值。

| zend\_parse\_parameters() 类型说明符 | | |
| --- | --- | --- |
| 修饰符 | 附加参数的类型 | 描述 |
| b | zend\_bool | Boolean 值 |
| l | long | integer (long) 值 |
| d | double | float (double) 值 |
| s | char\*, int | 二进制的安全串 |
| h | HashTable\* | 数组的哈希表 |

# Zend API：深入 PHP 内核

程序语言固有的限制和复杂，当不得不附添加巨大的默认代码库到每个简单脚本上时，就会出现这些限制和麻烦，解决方式：即让PHP运行的C代码

什么是 Zend ? 什么是 PHP ?