# 2.1 变量的类型

PHP在内核中是通过zval这个结构体来存储变量的，它的定义在Zend/zend.h文件里，简短精炼，只有四个成员组成：

```
struct _zval_struct {
    zvalue_value value; /* 变量的值 */
    zend_uint refcount__gc;
    zend_uchar type;    /* 变量当前的数据类型 */
    zend_uchar is_ref__gc;
};
typedef struct _zval_struct zval;

//在Zend/zend_types.h里定义的：
typedef unsigned int zend_uint;
typedef unsigned char zend_uchar;
```

zval里的refcount**gc**是**zend_uint**类型，也就是**unsigned int**型，**is_ref**gc和type则是unsigned char型的。

保存变量值的value则是zvalue_value类型(PHP5)，它是一个union，同样定义在了Zend/zend.h文件里：

```
typedef union _zvalue_value {
    long lval;  /* long value */
    double dval;    /* double value */
    struct {
        char *val;
        int len;
    } str;
    HashTable *ht;  /* hash table value */
    zend_object_value obj;
} zvalue_value;
```

在以上实现的基础上，PHP语言得以实现了8种数据类型，这些数据类型在内核中的分别对应于特定的常量，它们分别是：

| 常量名称： | |
|---|---|
| IS_NULL | 第一次使用的变量如果没有初始化过，则会自动的被赋予这个常量，当然我们也可以在PHP语言中通过null这个常量来给予变量null类型的值。 这个类型的值只有一个 ，就是NULL，它和0与false是不同的。 |
| IS_BOOL | 布尔类型的变量有两个值，true或者false。在PHP语言中，while、if等语句会自动的把表达式的值转成这个类型的。 |
| IS_LONG | PHP语言中的整型，在内核中是通过所在操作系统的signed long数据类型来表示的。在最常见的32位操作系统中，它可以存储从-2147483648 到 +2147483647范围内的任一整数。有一点需要注意的是，如果PHP语言中的整型变量超出最大值或者最小值，它并不会直接溢出，而是会被内核转换成IS_DOUBLE类型的值然后再参与计算。再者，因为使用了signed long来作为载体，所以这也就解释了为什么PHP语言中的整型数据都是带符号的了。$a = 2147483647;$a++;echo $a;//会正确的输出 2147483648； |
| IS_DOUBLE | PHP中的浮点数据是通过C语言中的signed double型变量来存储的，这最终取决与所在操作系统的浮点型实现。我们做为程序猿，应该知道计算机是无法精准的表示浮点数的，而是采用了科学计数法来保存某个精度的浮点数。用科学计数法，计算机只用8位便可以保存2.225x10^(-308)~~1.798x10^308之间的浮点数。用计算机来处理浮点数简直就是一场噩梦，十进制的0.5转成二进制是0.1，0.8转换后是0.1100110011....。但是当我们从二进制转换回来的时候，往往会发现并不能得到0.8。我们用1除以3这个例子来解释这个现象：1/3=0.3333333333.....，它是一个无限循环小数，但是计算机可能只能精确存储到0.333333，当我们再乘以三时，其实计算机计算的数是0.333333*3=0.999999，而不是我们平时数学中所期盼的1.0. |
| IS_STRING | PHP中最常用的数据类型——字符串，在内存中的存储和C差不多，就是一块能够放下这个变量所有字符的内存，并且在这个变量的zval实现里会保存着指向这块内存的指针。与C不同的是，PHP内核还同时在zval结构里保存着这个字符串的实际长度，这个设计使PHP可以在字符串中嵌入‘\0’字符，也使PHP的字符串是二进制安全的，可以安全的存储二进制数据！本着艰苦朴素的作风，内核只会为字符串申请它长度+1的内存，最后一个字节存储的是‘\0’字符，所以在不需要二进制安全操作的时候，我们可以像通常C语言的方式那样来使用它。 |
| IS_ARRAY | 数组是一个非常特殊的数据类型，它唯一的功能就是聚集别的变量。在C语言中，一个数组只能承载一种类型的数据，而PHP语言中的数组则灵活的多，它可以承载任意类型的数据，这一切都是HashTable的功劳，每个HashTable中的元素都有两部分组成：索引与值，每个元素的值都是一个独立的zval（确切的说应该是指向某个zval的指针）。 |
| IS_OBJECT | 和数组一样，对象也是用来存储复合数据的，但是与数组不同的是，对象还需要保存以下信息：方法、访问权限、类常量以及其它的处理逻辑。相对与zend engine V1，V2中的对象实现已经被彻底修改，所以我们PHP扩展开发者如果需要自己的扩展支持面向对象的工作方式，则应该对PHP5和PHP4分别对待！ |
| IS_RESOURCE | 有一些数据的内容可能无法直接呈现给PHP用户的，比如与某台mysql服务器的链接，或者直接呈现出来也没有什么意义。但用户还需要这类数据，因此PHP中提供了一种名为Resource(资源)的数据类型。有关这个数据类型的事宜将在第九章中介绍，现在我们只要知道有这么一种数据类型就行了。 |

zval结构体里的type成员的值便是以上某个IS_*常量之一。
内核通过检测变量的这个成员值来知道他是什么类型的数据并做相应的后续处理。

如果要我们检测一个变量的类型，最直接的办法便是去读取它的type成员的值：

```
void describe_zval(zval *foo)
{
    if (foo->type == IS_NULL)
    {
        php_printf("这个变量的数据类型是： NULL");
    }
    else
    {
        php_printf("这个变量的数据类型不是NULL，这种数据类型对应的数字是： %d", foo->type);
    }
}
```

虽然上述实现是正确的，但我们强烈建议你不要这样做。

PHP内核以后可能会修改变量的实现方式，所以检测type的方法可能在以后就不能用了。
为了解决这个兼容问题，zend头文件中定义了大量的宏，供我们检测、操作变量使用，
使用这些宏不但让我们的程序更易读，还具有更好的兼容性。
这里我们用Z_TYPE_P()宏来改写上面那个程序。

```
void describe_zval(zval *foo)
{
    if ( Z_TYPE_P(foo) == IS_NULL )
    {
        php_printf("这个变量的数据类型是： NULL");
    }
    else
    {
        php_printf("这个变量的数据类型不是NULL，这种数据类型对应的数字是： %d", Z_TYPE_P(foo));
    }
}
```

**php_printf()**函数是内核对**printf()**函数的一层封装，我们可以像使用**printf()**函数那样使用它。

以一个P结尾的宏的参数大多是*zval型变量。
此外获取变量类型的宏还有两个，分别是Z_TYPE和Z_TYPE_PP，前者的参数是zval型，而后者的参数则是**zval。

这样我们便可以猜测一下php内核是如何实现gettype这个函数了，代码如下：

```
//开始定义php语言中的函数gettype
PHP_FUNCTION(gettype)
{
    //arg间接指向调用gettype函数时所传递的参数。是一个zval**结构
    //所以我们要对他使用__PP后缀的宏。
    zval **arg;

    //这个if的操作主要是让arg指向参数～
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "Z", &arg) == FAILURE) {
        return;
    }

    //调用Z_TYPE_PP宏来获取arg指向zval的类型。
    //然后是一个switch结构，RETVAL_STRING宏代表这gettype函数返回的字符串类型的值
    switch (Z_TYPE_PP(arg)) {
        case IS_NULL:
            RETVAL_STRING("NULL", 1);
            break;

        case IS_BOOL:
            RETVAL_STRING("boolean", 1);
            break;

        case IS_LONG:
            RETVAL_STRING("integer", 1);
            break;

        case IS_DOUBLE:
            RETVAL_STRING("double", 1);
            break;

        case IS_STRING:
            RETVAL_STRING("string", 1);
            break;

        case IS_ARRAY:
            RETVAL_STRING("array", 1);
            break;

        case IS_OBJECT:
            RETVAL_STRING("object", 1);
            break;

        case IS_RESOURCE:
            {
                char *type_name;
                type_name = zend_rsrc_list_get_rsrc_type(Z_LVAL_PP(arg) TSRMLS_CC);
                if (type_name) {
                    RETVAL_STRING("resource", 1);
                    break;
                }
            }
```

```
        default:
            RETVAL_STRING("unknown type", 1);
    }
}
```

以上三个宏的定义在Zend/zend_operators.h里，定义分别是：

```
#define Z_TYPE(zval)         (zval).type
#define Z_TYPE_P(zval_p)     Z_TYPE(*zval_p)
#define Z_TYPE_PP(zval_pp)   Z_TYPE(**zval_pp)
```

# links