Atitit. 类与对象的存储实现

[1. 类的结构和实现 1](#_Toc12917)

[2. 类的方法属性都是hashtable存储的。 2](#_Toc10677)

[3. Class的分类 常规类(T\_CLASS)， 抽象类(T\_ABSTRACT T\_CLASS)和final类(T\_FINAL T\_CLASS 2](#_Toc1819)

[4. 对象的结构 3](#_Toc11748)

[5. 参考 4](#_Toc9036)

# [类的结构和实现](http://www.nowamagic.net/librarys/veda/detail/1492)

#### 的结构

首先我们看看类是什么。类是用户定义的一种抽象数据类型，它是现实世界中某些具有共性事物的抽象。 有时我们也可以理解其为对象的类别。类也可以看作是一种复合型的结构，其需要存储多元化的数据， 如属性、方法、以及自身的一些性质等。

类和函数类似，PHP内置及PHP扩展均可以实现自己的内部类，也可以由用户使用PHP代码进行定义。 当然我们在编写代码时通常是自己定义。

这里定义了一个父类ParentClass，一个接口Ifce，一个子类Tipi。子类继承父类ParentClass， 实现接口Ifce，并且有一个静态变量$sa，一个类常量 CA，一个公用方法，一个私有方法和一个公用静态方法。 这些结构在Zend引擎内部是如何实现的？类的方法、成员变量是如何存储的？访问控制，静态成员是如何标记的？

首先，我们看看类的内部存储结构：

|  |  |
| --- | --- |
| 01 | struct \_zend\_class\_entry { |

|  |  |
| --- | --- |
| 02 | char type;     // 类型：ZEND\_INTERNAL\_CLASS / ZEND\_USER\_CLASS |

|  |  |
| --- | --- |
| 03 | char \*name;// 类名称 |

|  |  |
| --- | --- |
| 04 | zend\_uint name\_length;                  // 即sizeof(name) - 1 |

|  |  |
| --- | --- |
| 05 | struct　\_zend\_class\_entry \*parent; // 继承的父类 |

|  |  |
| --- | --- |
| 06 | int　refcount;  // 引用数 |

|  |  |
| --- | --- |
| 07 | zend\_bool constants\_updated; |

|  |  |
| --- | --- |
| 08 |  |

|  |  |
| --- | --- |
| 09 | zend\_uint ce\_flags; // ZEND\_ACC\_IMPLICIT\_ABSTRACT\_CLASS: 类存在abstract方法 |

|  |  |
| --- | --- |
| 10 | // ZEND\_ACC\_EXPLICIT\_ABSTRACT\_CLASS: 在类名称前加了abstract关键字 |

|  |  |
| --- | --- |
| 11 | // ZEND\_ACC\_FINAL\_CLASS |

|  |  |
| --- | --- |
| 12 | // ZEND\_ACC\_INTERFACE |

|  |  |
| --- | --- |
| 13 | HashTable function\_table;      // 方法 |

|  |  |
| --- | --- |
| 14 | HashTable default\_properties;          // 默认属性 |

|  |  |
| --- | --- |
| 15 | HashTable properties\_info;     // 属性信息 |

|  |  |
| --- | --- |
| 16 | HashTable default\_static\_members;// 类本身所具有的静态变量 |

|  |  |
| --- | --- |
| 17 | HashTable \*static\_members; // type == ZEND\_USER\_CLASS时，取&default\_static\_members; |

|  |  |
| --- | --- |
| 18 | // type == ZEND\_INTERAL\_CLASS时，设为NULL |

|  |  |
| --- | --- |
| 19 | HashTable constants\_table;     / |

# 类的方法属性都是hashtable存储的。

常规的成员方法存放在函数结构体的哈希表中， 而魔术方法则单独保存。 如在类定义中的 union \_zend\_function \*constructor; 定义就是类的构造魔术方法， 它是以函数的形式存在于类结构中，并且与常规的方法分隔开来了。在初始化时，这些魔术方法都会被设置为NULL。

在PHP语言中，变量都是保存在哈希表中，称为变量符号表，其中变量名为哈希表的键，变量名对应的容器zval的指针为哈希表中的值。所有全局变量放在一 张主符号表中(也就是数组$GLOBALS对应的哈希表)。PHP语言有个特性，变量在命名时，$变量标识符后不能以数字开头。例如我们在以下代码：

# Class的分类 常规类(T\_CLASS)， 抽象类(T\_ABSTRACT T\_CLASS)和final类(T\_FINAL T\_CLASS

上面的class\_entry\_type语法说明在语法分析阶段将类分为三种类型：常规类(T\_CLASS)， 抽象类(T\_ABSTRACT T\_CLASS)和final类(T\_FINAL T\_CLASS )。 他们分别对应的类型在内核中为:

# 对象的结构

对象在PHP中是使用一种zend\_object\_value的结构体来存储。

|  |  |
| --- | --- |
| 1 | typedef struct \_zend\_object\_value { |

|  |  |
| --- | --- |
| 2 | zend\_object\_handle handle; |

|  |  |
| --- | --- |
| 3 | //  unsigned int类型，EG(objects\_store).object\_buckets的索引 |

|  |  |
| --- | --- |
| 4 | zend\_object\_handlers \*handlers; |

|  |  |
| --- | --- |
| 5 | } zend\_object\_value; |

PHP内核会将所有的对象存放在一个对象列表容器中，这个列表容器是保存在EG(objects\_store)里的一个全局变量。 上面的handle字段就是这个列表中object\_buckets的索引。当我们需要在PHP中存储对象的时候， PHP内核会根据handle索引从对象列表中获取相对应的对象。而获取的对象有其独立的结构，如下代码所示：

|  |  |
| --- | --- |
| 1 | typedef struct \_zend\_object { |

|  |  |
| --- | --- |
| 2 | zend\_class\_entry \*ce; |

|  |  |
| --- | --- |
| 3 | HashTable \*properties; |

|  |  |
| --- | --- |
| 4 | HashTable \*guards; /\* protects from \_\_get/\_\_set ... recursion \*/ |

|  |  |
| --- | --- |
| 5 | } zend\_object; |

ce是存储该对象的类结构，properties是一个HashTable，用来存放对象的属性。

在zend\_object\_value结构体中除了索引字段外还有一个包含对象处理方法的字段：handlers。 它的类型是zend\_object\_handlers，我们可以在Zend/zend\_object\_handlers.h文件中找到它的定义。 这是一个包含了多个指针函数的结构体，这些指针函数包括对对象属性的操作，对对象方法的操作，克隆等。 此字段会在对象创建的时候初始化。

#### 对象的创建

在PHP代码中，对象的创建是通过关键字 new 进行的。从此关键字出发，我们遍历词法分析，语法分析和编译成中间代码等过程， 得到其最后执行的函数为 ZEND\_NEW\_SPEC\_HANDLER 。

zend\_objects\_new函数会初始化对象自身的相关信息，包括对象归属于的类，对象实体的存储索引，对象的相关处理函数。 在这里将对象放入对象池中的函数为zend\_objects\_store\_put。

在将对象放入对象池，返回对象的存放索引后，程序设置对象的处理函数为标准对象处理函数：std\_object\_handlers。 其位于Zend/zend\_object\_handles.c文件中。

#### 对象池

这里针对对象，我们引入一个新的概念--对象池。 我们将PHP内核在运行中存储所有对象的列表称之为对象池，即EG(objects\_store)。 这个对象池的作用是存储PHP中间代码运行阶段所有生成的对象，这个思想有点类似于我们做数据库表设计时， 当一个实例与另一个实体存在一对多的关系时，将多的那一端对应的实体提取出来存储在一个独立的表一样。 这样做的好处有两个，一个是可以对象复用，另一个是节省内存，特别是在对象很大，并且我们不需要用到对象的所有信息时。 对象池的存储结构为zend\_objects\_store结构体，如下：

#### 成员变量

从前面的对象结构来看，对象的成员变量存储在properties参数中。并且每个对象都会有一套标准的操作函数， 如果需要获取成员变量，对象最后调用的是read\_property，其对应的标准函数为zend\_std\_read\_property; 如果需要设置成员变量，对象最后调用的是write\_property，其对应的标准函数zend\_std\_write\_property。 这些函数都是可以定制的，如果有不同的需求，可以通过设置对应的函数指针替换。如在dom扩展中，它的变量的获取函数和设置函数都是定制的。

#### 成员方法

成员方法又包括常规的成员方法和魔术方法。魔术方法在前面的第五小节已经介绍过了，这里就不再赘述。 在对象的标准函数中并没有成员方法的调用函数，默认情况下设置为NULL。在SPL扩展中，有此函数的调用设置，如下代码：

数为 **ZEND\_INIT\_METHOD\_CALL\_SPEC\_CV\_CONST\_HANDLER** 此函数的调用流程如下：

* 第一步，处理调用的方法名，获取其值，并做检验处理：如果不是字符串，则报错
* 第二步，如果第一个操作数是对象，则转第三步，否则报错 Call to a member function t on a non-object
* 第三步，调用对象的get\_method函数获取成员方法
* 第四步，其它处理，包括静态方法，this变量等。

而get\_method函数一般是指标准实现中的get\_method函数，其对应的具体函数为Zend/zend\_object\_handlers.c文件中zend\_std\_get\_method函数。 zend\_std\_get\_method函数的流程如下：

* 第一步，从zobj->ce->function\_table中查找是否存在需要调用的函数，如果不存在，转第二步，如果存在，转第三步
* 第二步，如果\_\_call函数存在，则调用zend\_get\_user\_call\_function函数获取并返回，如果不存在，则返回NULL
* 第三步，检查方法的访问控制，如果为私有函数，转第四步，否则转第五步
* 第四步，如果为同一个类或父类和这个方法在同一个作用域范围，则返回此方法，否则判断\_\_call函数是否存在，存在则调用此函数，否则报错
* 第五步，处理函数重载及访问控制为protected的情况。 转第六步
* 第六步，返回fbc

在获得了函数的信息后，下面的操作就是执行了，关于函数的执行在前面章节已经介绍过了。

## 为了操作一个对象，我们需要先获取这个对象的实例，

而这有肯定会涉及调用对象的构造方法。首先我们先了解下一个object在PHP内核中到底是如何实现的。

|  |  |
| --- | --- |
| 01 | typedef struct \_zend\_object\_value { |

|  |  |
| --- | --- |
| 02 | zend\_object\_handle handle; |

|  |  |
| --- | --- |
| 03 | zend\_object\_handlers \*handlers; |

|  |  |
| --- | --- |
| 04 | } zend\_object\_value; |

|  |  |
| --- | --- |
| 05 |  |

|  |  |
| --- | --- |
| 06 | //此外再回顾一下zval的值value的结构。 |

|  |  |
| --- | --- |
| 07 | typedef union \_zvalue\_value { |

|  |  |
| --- | --- |
| 08 | long lval;                  /\* long value \*/ |

|  |  |
| --- | --- |
| 09 | double dval;                /\* double value \*/ |

|  |  |
| --- | --- |
| 10 | struct { |

|  |  |
| --- | --- |
| 11 | char \*val; |

|  |  |
| --- | --- |
| 12 | int len; |

|  |  |
| --- | --- |
| 13 | } str; |

|  |  |
| --- | --- |
| 14 | HashTable \*ht;              /\* hash table value \*/ |

|  |  |
| --- | --- |
| 15 | zend\_object\_value obj; |

|  |  |
| --- | --- |
| 16 | } zvalue\_value; |

如果我们有一个zval \*tmp，那么tmp->value.obj来访问到最终保存对象实例的zend\_object\_value结构体，它包含两个成员：

# 参考

PHP内核探索：类的结构和实现 -- 简明现代魔法.htm

PHP内核探索：对象 -- 简明现代魔法.htm

PHP内核探索：创建对象实例 -- 简明现代魔法.htm