PHP7数组的底层实现示例

作者：有故事的人 来源：范文网 www.wtabcd.cn/fanwen/

本文原地址：https://www.wtabcd.cn/fanwen/zuowen/6c905569db0fcc384b48f5a62e669a18.html

php 数组具有的特性

php 的数组是一种非常强大灵活的数据类型，在讲它的底层实现之前，先看一下 php 的数组都具有哪些特性。

可以使用数字或字符串作为数组健值

```
$arr = [1 => 'ok', 'one' => 'hello'];
```

可按顺序读取数组

```
foreach($arr as $key => $value){ echo $arr[$key];}
```

可随机读取数组中的元素

```
$arr = [1 => 'ok', 'one' => 'hello', 'a' => 'world'];echo $arr['one'];echo current($ar
r);
```

数组的长度是可变的

```
$arr = [1, 2, 3];$arr[] = 4;array_push($arr, 5);
```

正是基于这些特性，我们可以使用 php 中的数组轻易的实现集合、栈、列表、字典等多种数据结构。那么这些特性在底层是如何实现的呢？ 这就得从数据结构说起了。

数据结构

php 中的数组实际上是一个有序映射。映射是一种把 values 关联到 keys 的类型。

php 数组的底层实现是散列表（也叫 hashtable ），散列表是根据键（key）直接访问内存存储位置的数据结构，它的key – value 之间存在一个映射函数，可以根据 key 通过映射函登山作文数得到的散列值直接索引到对应的

value 值，无需通过关键字比较，在理想情况下，不考虑散列冲突，散列表的查找效率是非常高的，时间复杂度是 o(1)。

从源码中我们可以看到 zend_array 的结构如下：

typedef struct _zend_array zend_array;typedef struct _zend_array hashtable;stru妮维雅男士全能控油啫喱ct _zend_array { zend_refcounted_h gc; union { struct { zend_endian_lohi_4(zend_uchar flags, zend_u char napplycount, zend_uchar niteratorscount, zend_uchar reserve) } v; uint32_t flags; } u; uint32_t ntab lemask; // 哈希值计算掩码，等于ntablesize的负值(ntablemask = -ntablesize) bucket *ardata; // 存储元素数组，指向第一个bucket uint32_t nnumused; // 已用bucket数（含失效的 bucket） uint32_t nnumofelements; // 哈希表有效元素数 uint32_t 微信网名 简单又好听 ntablesize; // 哈希表总大小，为2的n次方（包括无效的元素） uint32_t ninternalpointer; // 内部指针，用于遍历 zend_long nnextfreeelement; // 下一个可用的数值索引,如:arr[] = 1;arr["a"] = 2;arr[] = 3; 则nnextfreeelement = 2; dtor_func_t pdestructor;};

该结构中的 bucket 即储存元素的数组，ardata 指向数组的起始位置，使用映射函数对 key 值进行映射后可以得到偏移值，通过内存起始位置 + 偏移值即可在散列表中进行寻址操作。

bucket 的数据结构如下：

typedef struct _bucket { zva中国地势特点l val; // 存储的具体 value，这里是一个 zval，而不是一个指针 zend_ulong h; // 数字 key 或字符串 key 的哈希值。用于查找时 key 的比较 zend_string *key; // 当 key 值为字符串时，指向该字符串对应的 zend_string（使用数字索引时该值为 null），用于查找时 key 的比较} bucket;

到这里有个问题出现了：存储在散列表里的元素是无序的，php 数组如何做到按顺序读取的呢？

答案是中间映射表，为了实现散列表的有序性，php 为其增加了一张中间映射表，该表是一个大小与 bucket 相同的数组，数组中储存整形数据，用于保存元素实际储存的 value 在 bucekt 中的下标。bucekt 中的数据是有序的，而中间映射表中的数据是无序的。

而通过映射函数映射后的散列值要在中间映射表的区间内，这就对映射函数提出了要求。

映射函数

php7 数组采用的映射方式：

nindex = h | ht->ntablemask;

将 key 经过 time33 算法生成的哈希值 h 和 ntablemask 进行或运算即可得出映射表的下标，其中 ntablemask 数值为 ntablesize 的负数。并且由于 ntablesize 的值为 2 的幂次方，所以 ntablemask

二进制位右侧全部为 0，保证了 h | ht->ntablemask 的取值范围会在 [-ntablesize, -1] 之间，正好在映射表的下标范围内。另外，用按位或运算的方法和其他方法如取余的方法相比运算速度较高，这个映射函数可以说设计的非常巧妙了。

散列（哈希）冲突

不同键名的通过映射函数计算得到的散列值有可能相同，此时便发生了散列冲突。

对于散列冲突有以下 4 种常用方法：

1.将散列值放到相邻的最近地址里

2.换个散列函数重新计算散列值

3.将冲突的散列值统一放到另一个地方

4.在冲突位置构造一个单向链表，将散列值相同的元素放到相同槽位对应高考励志标语的链表中。这个方法叫链地址法，php 数组就是采用这个方法解决散列冲突的问题。

其具体实现是：将冲突的 bucket
串成链表，这样中间映射表映射出的就不是某一个元素，而是一个 bucket
链表，通过散列函数定位到对应的 bucket 链表时，需要遍历链表，逐个对比 key
值，继而找到目标元素。而每个 bucket 之间的链接则是将原 value 的下标保存到新 value 的
zval.u2.next 里，新 value 放在当前位置上，从而形成一个单向链表。

举个例子：

当我们访问 $arr[‘ key ’] 的过程中，假设首先通过散列运算得出映射表下标为 -2
，然后访问映射表发现其内容指向 ardata 数组下标为 1 的元素。此时我们将该元素的 key
和要访问的键名相比较，发现两者并不相等，则该元素并非我们所想访问的元素，而元素的
zval.u2.next 保存的值正是另一个具有相同散列值的元素对应 ardata
数组的下标，所以我们可以不断通过 zval.u2.next 的值遍历直到找到键名相同的元素。

扩容

php 的数组在底层实现了自动扩容机制，当插入一个元素且没有空闲空间时，就会触发自动扩容机制，扩容后再执行插入。

扩容的过程为：

如果已删除元素所占比例达到阈值，则会移除已被逻辑删除的 bucket，然后将后面的 bucket
向前补上空缺的 bucket，因为 bucket
的下标发生了变动，所以还需要更改每个元素在中间映射表中储存的实际下标值。

如果未达到阈值，php 则会申请一个大小是原数组两倍的新数组，并将旧数组中的数据复制到新数组中，因为数组长度发生了改变，所以 key-value

的映射关系需要重新计算，这个步骤为重建索引。

重建散列表

在删除某一个数组元素时，会先使用标志位对该元素进行逻辑删除，即在删除 value 时只是将 value 的 type 设置为 is_undef，而不会立即删除该元素所在的 bucket，因为如果每次删除元素立刻删除 bucket 的话，每次都需要进行排列操作，会造成不必要的性能开销。

所以，当删除元素达到一定数量或扩容后都需要重建散列表，即移除被标记为删除的 value。因为 value 在 bucket 位置移动了或哈希数组 ntablesize 变化了导致 key 与 value 的映射关系改变，重建过程就是遍历 bucket 数组中的 value，然后重新计算映射值更新到散列表。

关于 php7 的数组底层实现就总结这么些了，因为水平有限也无法研究的十分详尽清楚，如果有疑问或者不足之处欢迎提出~~

参考资料

《php7 的底层设计与源码实现》

总结

以上就是这篇文章的全部内容了，希望本文的内容对大家的学习或者工作具有一定的参考学习价值，谢谢大家对www.887551.com的支持。