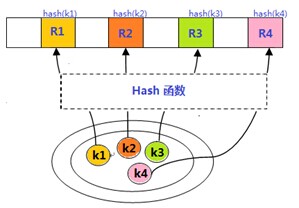
在PHP中，除了zval, 另一个比较重要的数据结构非hash table莫属，例如我们最常见的数组，在底层便是hash table。



理论上，哈希函数可以是任何函数如Crc32, unique\_id,MD5,SHA1或者用户自定义的函数。这个函数的好坏直接关系到Hash table的性能（考虑冲突和查找的性能）

Hash冲突：链表的方式解决冲突

typedef struct \_hashtable {

uint nTableSize;//hash表的大小

uint nTableMask;// “掩码”，主要用于快速计算一个元素的索引

uint nNumOfElements;// hashtable中保存的元素的个数

ulong **nNextFreeElement**;// 记录下一个可用的索引位置，我们在脚本中使用$array[] = 'key'的时候，就是使用nNextFreeElement给出的索引值

Bucket \*pInternalPointer; //指针current,next,key,end等与数组相关的操作时，都是使用这一指针来完成的

Bucket \*pListHead;// PHP底层实际上维护了两个重要的数据结构，除了hash表（以及用于解决冲突的双链表），还有一个双向链表用于hash表元素的线性扫描。pListHead和pListTail便指向这个双链表的表头和表尾。

Bucket \*pListTail;

Bucket \*\*arBuckets; //数组中每个元素都是一个bucket\* 的指针，具有相同hash值的元素通过bucket的pNext和pLast指针连接成一个双链表，bucket是实际存储数据的容器

dtor\_func\_t pDestructor;

zend\_bool persistent;

unsigned char nApplyCount;

zend\_bool bApplyProtection;

#if ZEND\_DEBUG

int inconsistent;

#endif

} HashTable;

typedef struct bucket {

ulong h;

uint nKeyLength;

void \*pData;

void \*pDataPtr;

struct bucket \*pListNext;// pListNext和pListLast构成了用于遍历的整个双链表

struct bucket \*pListLast;

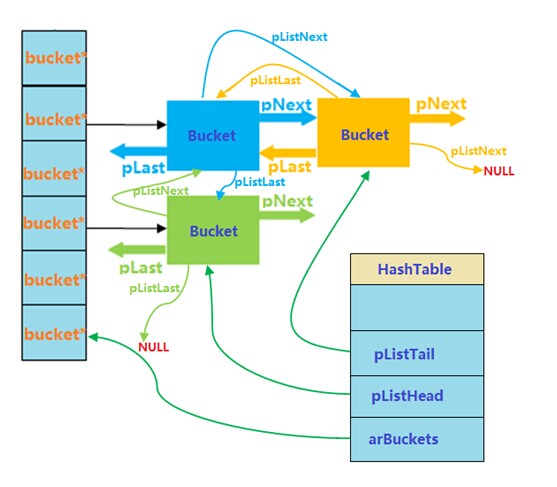
struct bucket \*pNext;//冲突时链表结构

struct bucket \*pLast;

const char \*arKey;

} Bucket;

h ,arKey,nKeyLength PHP数组中，有**两类不同的索引**，一类是**数字索引**，这与C中的数组非常类似(如$arr = array(1=>'cont')), 另一类是**字符串索引**，也就是使用关键词作为数组项的索引（如$arr = array('index'=>'cont');）.



字符串查找：

zend\_hash\_find(const HashTable \*ht, const char \*arKey, uint nKeyLength, void \*\*pData){

    /\* 字符串索引需要先计算字符串的hash值 \*/

    h = zend\_inline\_hash\_func(arKey, nKeyLength);

    nIndex = h & ht->nTableMask;

    p = ht->arBuckets[nIndex];

    /\* Bucket双链表中查找，一旦找到，立即返回，注意查找成功的条件 \*/

    while (p != NULL) {

     if (p->arKey == arKey ||

 ((p->h == h) && (p->nKeyLength == nKeyLength) && !memcmp(p->arKey, arKey, nKeyLength))) {

      \*pData = p->pData;

      return SUCCESS;

        }

        p = p->pNext;

}

……

}

数字查找：

ZEND\_API int zend\_hash\_index\_find(const HashTable \*ht, ulong h, void \*\*pData)

{

    uint nIndex;

    Bucket \*p;

    IS\_CONSISTENT(ht);

     /\* 计算索引 \*/

    nIndex = h & ht->nTableMask;

    p = ht->arBuckets[nIndex];

    /\* 遍历双链表，一旦找到立即返回 \*/

    while (p != NULL) {

        if ((p->h == h) && (p->nKeyLength == 0)) {

            \*pData = p->pData;

            return SUCCESS;

        }

        p = p->pNext;

    }

     /\* 如果遍历完双链表，没有找到，那么查找失败 \*/

    return FAILURE;

}

<http://www.cnblogs.com/ohmygirl/p/internal-3.html>

方法：

数组遍历相关函数：如prev, next, current, end,reset, each等

数组排序相关：如sort, rsort, asort, arsort, ksort, krsort, uasort, uksort

数组查找相关: 如in\_array, array\_search, array\_key\_exists等

数组分割、合并相关： array\_slice, array\_splice, implode, array\_chunk, array\_combine等

数组交并差：如array\_merge, array\_diff, array\_diff\_\*, array\_intersect, array\_intersect\_\*

作为stack/queue容器的数组： 如array\_push, array\_pop, array\_shift,array\_unshift

其他的数组操作：array\_fill, array\_flip, array\_sum, array\_reverse等

// PHP的Hash采用的是目前最为普遍的DJBX33A (Daniel J. Bernstein, Times 33 with Addition)

算法的核心思想就是: hash(i) = hash(i-1) \* 33 + str[i]