

一、PHP的基本目录

1. 目录

(1) Sbin: php-fpm

(2) bin: PHP 的执行程序

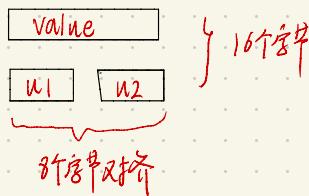
2. --enable-debug: 打开调试模式

二、基本变量与内存管理

1. Zval zend_types.h

(1) 定义:

struct _zval_struct {
 zend_value value; 联合体
 union u1; 定义变量的类型
 union u2;
};



(2) u1 的解析

union {
 struct {
 ZEND_ENDIAN_LOHI_4(

zend_uchar type, 变量的类型

zend_uchar type_flags;

zend_uchar const_flags,

zend_uchar reserved)

};
unit32 + type_info;

y u1;

(3) u2 的解析:

在 u2 中最重要的变量为 next, 用来解决数组中的冲突

(4) 写时复制: 对于整型的, Zval 可以用 16 个字节表示即可, 直接复制即可, 如果是字符串, 两个变量相等, 且指向同一内存, 但其中一倍进行修改, 则会复制一块内存, 进行修改

(5) zend_string

变量的 flag 为 0，常量的 flag 为 2

2. 关于引用

`$a = "string"` \$b 为空
`$b = &$a;` \Rightarrow `$a = "string"`
`unset($b);`

3. zend_array

Codes for example:

```
$a = [ ];  

$a[1] = "a";  

$a[1] = "b";  

$a["k1"] = "v1";  

$a["k2"] = "v2";  

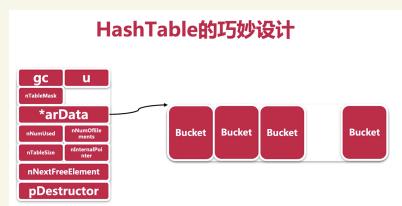
echo $a["k1"];  

$a["k1"] = "c";  

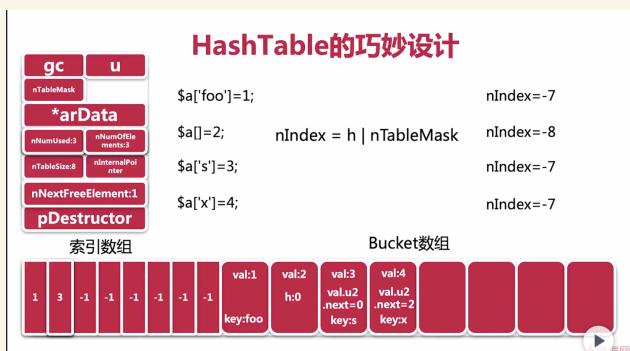
unset($a["k2"]);
```

{ pack_array: 键值对依次递增
hash_array: 键值无规律的

n_{NumUsed} : 总长度 包含 \checkmark ②
 $n_{\text{NumOfElements}}$: 实际长度 ①



4. 数组 (HashTable) 的实现 (通过拉链法解决 hash key 的冲突)



如果想要找 `$a['foo']=1` 的值有如下步骤：

(1) 找到 $nIndex = -7$ 的位置，得到对应的 bucket 应在第三个位置上，此时发现 key 值不相等；

(2) 得到 val, u2, next = 2，则找到第二个 bucket，又发现 key 值不相等，val, u2, next = 0。找到对应的 0，此时 key 值相等，则 `$a['foo']=1`

变量总结：

(1) PHP中所有的变量都用zval表示，其中 value, U1, U2 为联合体，其总长度为16个字节，我们可以通过联合体中的 U1 的 type 去寻找变量对应的类型；

(2) Str 类型存在写时复制。当我们复制变量时，refcount 加 1，修改一个变量时，refcount 减 1。

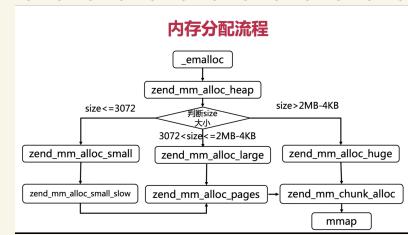
三、PHP的内存管理

1. malloc函数

- ① void *ptr = malloc(size); ↑内存的大小 申请内存
- ② free(ptr); 释放内存

2. 内存的单位

- ① chunk：2MB大小的内存
- ② page：4KB大小的内存
- ③ 内存预分配：使用 mmap 分配 chunk



四、PHP的垃圾回收机制

1. 5.3 版本之前的版本是如何产生内存泄漏垃圾（环形引用）

PHP 回收的机制是判断 refcount 是否为 0，当 refcount 为 0 的时候，则可以被回收。

产生内存泄漏主要真凶：环形引用。
现在来造一个环形引用的场景：

```

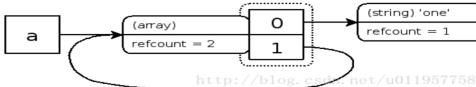
1 <?php
2 $a = ['one'];
3 $a[] = &$a;
4 xdebug_debug_zval('a');
  
```

得到：

```

1 a:
2 (refcount=2, is_ref=1),
3 array (size=2)
4  0 => (refcount=1, is_ref=0), string 'one' (length=3)
5  1 => (refcount=2, is_ref=1),
6      &array<
  
```

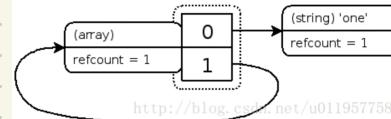
这样 \$a 数组就有了两个元素，一个索引为 0，值为 one 字符串，另一个索引为 1，为 \$a 自身的引用。



此时删掉 \$a：

```

1 <?php
2 $a = ['one'];
3 $a[] = &$a;
4 unset($a);
  
```



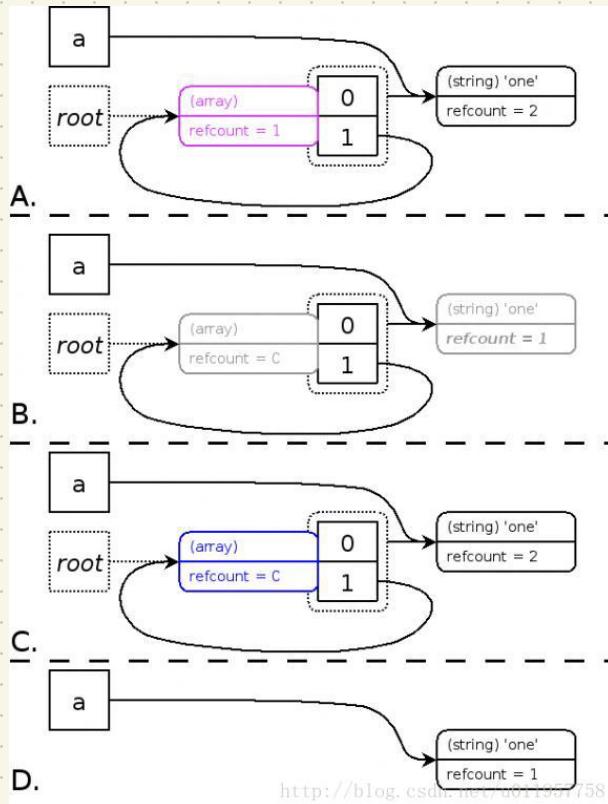
此时，\$a 已经不在符号表中，没有变量再指向 zval 容器，但是 refcount 不为 0，则会导致内存不能被回收而发生泄漏。

2. PHP5.3以后的版本垃圾回收机制 (三色回收)

当一个zval可能为垃圾时，回收算法会把这个zval放入一个内存缓冲区。当缓冲区达到最大临界值时（最大值可以设置），回收算法会循环遍历所有缓冲区中的zval，判断其是否为垃圾，并进行释放处理。或者我们在脚本中使用`gc_collect_cycles`，强制回收缓冲区中的垃圾。

在php5.3的GC中，针对的垃圾做了如下说明：

- 1：如果一个zval的refcount增加，那么此zval还在使用，肯定不是垃圾，不会进入缓冲区。
- 2：如果一个zval的refcount减少到0，那么zval会被立即释放掉，不属于GC要处理的垃圾对象，不会进入缓冲区。
- 3：如果一个zval的refcount减少之后大于0，那么此zval还不能被释放，此zval可能成为一个垃圾，将其放入缓冲区。PHP5.3中的GC针对的就是这种zval进行的处理。



GC_WHITE 白色表示垃圾

GC_PURPLE 紫色表示已放入缓冲区

GC_GREY 灰色表示已经进行了1次
refcount 的减1操作

GC_BLACK 黑色为默认颜色

A：为了避免每次变量的refcount减少的时候都调用GC的算法进行垃圾判断，此算法会先把所有前面准则3情况下的zval节点放入一个节点(root)缓冲区(root buffer)，并且将这些zval节点标记成紫色。同时算法必须确保每一个zval节点在缓冲区中之出现一次。当缓冲区被节点塞满的时候，GC才开始开始对缓冲区中的zval节点进行垃圾判断。

B：当缓冲区满了之后，算法以深度优先对每一个节点所包含的zval进行减1操作，为了确保不会对同一个zval的refcount重复执行减1操作，一旦zval的refcount减1之后会将zval标记成灰色。需要强调的是，这个步骤中，起初节点zval本身不做减1操作，但是如果节点zval中包含的zval又指向了节点zval（环形引用），那么这个时候需要对节点zval进行减1操作。

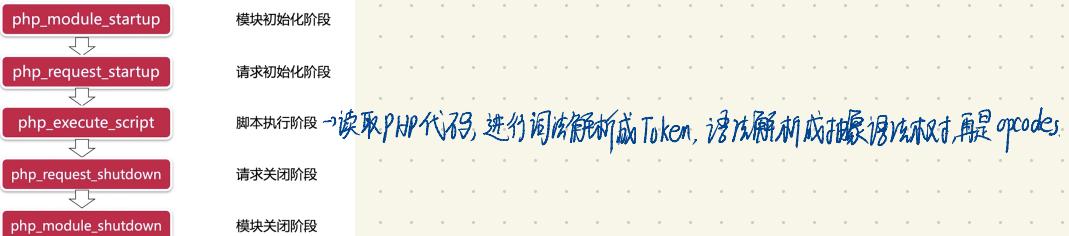
C：算法再次以深度优先判断每一个节点包含的zval的值，如果zval的refcount等于0，那么将其标记成白色（代表垃圾），如果zval的refcount大于0，那么将对此zval以及其包含的zval进行refcount加1操作，这个是对非垃圾的还原操作，同时将这些zval的颜色变成黑色（zval的默认颜色属性）。

D：遍历zval节点，将C中标记成白色的节点zval释放掉。

五. PHP运行的生命周期

1. CLI模式的生命周期

CLI模式的生命周期



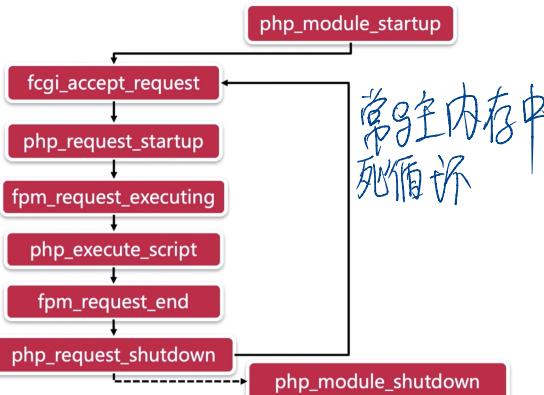
2. FPM的三种模式

- | static : 设置的个数与进程数一致
- | dynamic 动态启动，范围 [pm.start-server, pm.maxchild]
- | ondemand 内存第一位，根据需求配置

3. 网络编程：

- (1) server:
 - ① socket 创建fd
 - ② bind 端口绑定
 - ③ listen 调用listen函数，进行监听
 - ④ accept 死循环，接受请求，未接收到请求会阻塞
- (2) Client：通过socket连接远程服务

FPM模式的生命周期



4. php-fpm

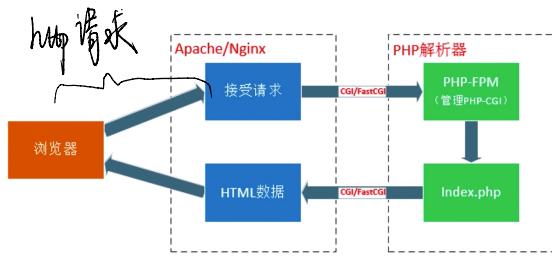
(1) master与子进程

如果 master 进程杀掉，子进程依旧工作，master 是用来管理进程的

如果子进程杀掉，master 负责开启子进程，请求也是由子进程进行处理

5. fast-cgi 协议

详解 FastCGI 协议



六、PHP代码的编译与执行

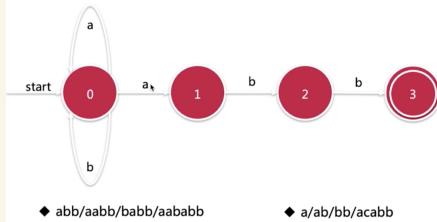
1. 编译语言与解释型语言



2. NFA与DFA

词法分析入门-NFA(不确定有穷自动机)

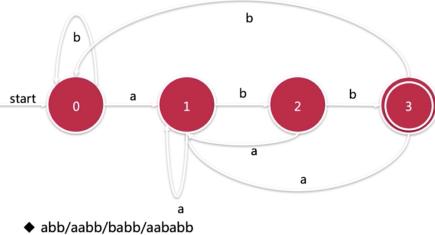
◆ 正则表达式 $(a|b)^*abb$



◆ abb/aabb/babb/aababb

◆ a/ab/bb/acabb

词法分析入门-DFA(确定有穷自动机)



◆ abb/aabb/babb/aababb

3. 词法分析 (re2c)

4. 语法分析 (语法分析树 / 巴科斯范式)