

PHP Garbage Collector and Magento

by Andrey Tserkus and Vladimir Pelipenko

Magento developers have many reasons to be happy— an award-winning system, interesting architecture, straightforward extension and customization, original solutions, and a great ecosystem - just to name a few. However, Magento developers are especially privileged to have a platform written in PHP. In this article we are going to talk about one of PHP's most significant advantages, its memory management.

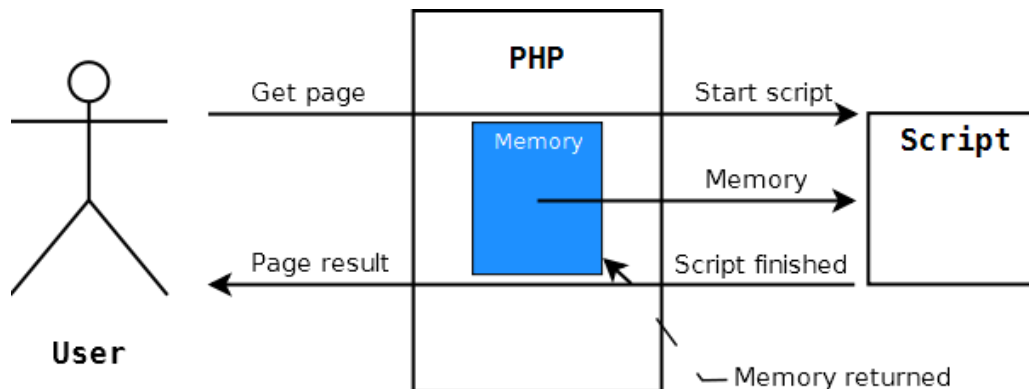
What's so exciting about memory management in PHP?

The fact that there is none. If you've worked with other programming languages, then you know what a nightmare it is to track every new object created and new memory block allocated. PHP-developers can use variables whenever we want to, we can drop objects at any point, and there is no limit on the number of arrays we can clone and create. We are fortunate as to have an underlying language layer that takes care of memory management on its own. In other words: you can stop worrying and begin programming!

PHP manages memory for itself by asking the OS (operating system) for more memory blocks for each script, which it returns to the OS when the script no longer needs them.

The model of online user-site interaction is very convenient for automatic memory management:

- A user requests a single page,
- THE SCRIPT IS EXECUTED,
- The result is given back to the user,
- THE SCRIPT TERMINATES.



PHP doesn't need to check whether any objects or memory blocks are in use, it can safely release all memory allocated to that script.

However, there are cases when PHP should be concerned with script memory. If a script generates a lot of reports, draws many images, or transfers a large amount of data than it's likely the script will exhaust all memory, thus creating a fatal error. That's why PHP has a special resource manager to deal with it – the Garbage Collector.

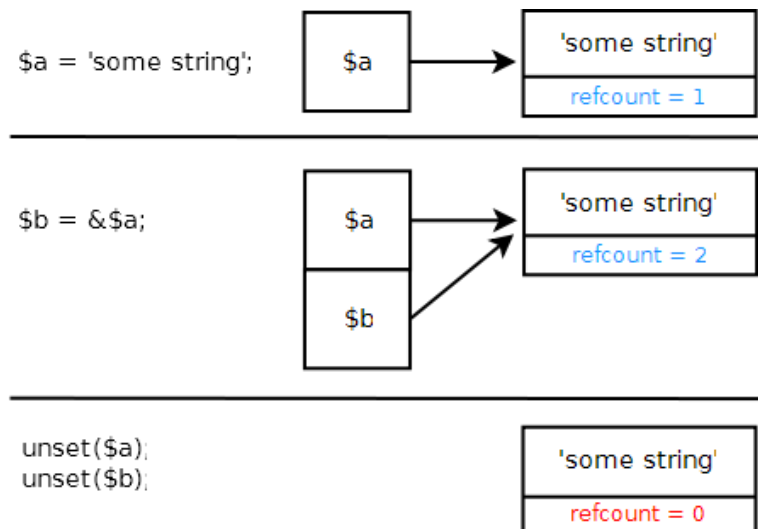
Well what is a Garbage Collector?

Developers don't control the Garbage Collector. It's invoked automatically, when PHP finds that available resources are nearly depleted. The Garbage Collector travels through allocated memory, checks the resources in use, and returns unused resources back to the memory pool.

How does the Garbage Collector know that a variable or object is still in use?

Each memory block in PHP has an internal counter called refcount. Whenever a variable links to that block, its refcount is increased by 1. When the link is broken, refcount is decreased by 1. Memory blocks with refcount equal to 0 are not in use and may be safely returned to the memory pool.

This diagram shows the usage of the \$A and \$B VARIABLES, and how their memory block becomes available for garbage collection:



The Garbage Collector is a great service for developers, but we should be aware of its issues in PHP versions prior to 5.3, and that PHP < 5.3 is still used everywhere!

Consider the following code snippet:

```
-----  
$a = new Varien_Object ();
```

```
$B = NEW VARIEN_OBJECT ();
```

```
$A->SETSOMEDATA ($B) ;
```

```
$B->SETSOMEDATA ($A) ;
```

```
UNSET ($A) ;
```

The memory block 'A', previously used by \$a, keeps a link to memory block 'B', previously used by \$b, that keeps a link back to memory block 'A,' it's called a circular reference. Both of these blocks have a refcount equal to 1, so neither of them will be released into the memory pool, and the PHP script will never be able to use them again.

Magento developers usually don't worry about such cases, because memory leaks are small and unimportant while running a single script. But if your script makes intensive work and executes for an extended period of time, you definitely need to clean up these references manually in order to help the Garbage Collector release allocated memory.

The main Magento model ancestor Mage_Core_Model_Abstract has a special method used for this purpose:

```
-----  
/**  
 * CLEARING OBJECT FOR CORRECT DELETING BY GARBAGE COLLECTOR  
 *  
 * @RETURN MAGE_CORE_MODEL_ABSTRACT  
 */  
FINAL PUBLIC FUNCTION CLEARINSTANCE ()  
{  
    $THIS->_CLEARREFERENCES () ;  
    MAGE :: DISPATCHEVENT ($THIS->_EVENTPREFIX . '_CLEAR' ,  
$THIS->_GETEVENTDATA () ) ;  
    $THIS->_CLEARDATA () ;  
    RETURN $THIS ;  
}
```

OVERLOAD `_clearReferences ()` OR `_clearData ()` , OR LISTEN TO THE
'`{EVENTPREFIX}_clear`' EVENT TO UNSET THE LINKS TO OTHER OBJECTS, AND YOU'LL REMOVE
THE POSSIBLE MEMORY LEAKAGE IN YOUR MODELS.

Check out this example – this is how the Product model

(`MAGE_CATALOG_MODEL_PRODUCT`) does it:

```
/**
 * CLEARING REFERENCES ON PRODUCT
 *
 * @RETURN MAGE_CATALOG_MODEL_PRODUCT
 */
PROTECTED FUNCTION _clearReferences ()
{
    $THIS->_clearOptionReferences ();
    RETURN $THIS;
}

/**
 * CLEARING REFERENCES TO PRODUCT FROM PRODUCT'S OPTIONS
 *
 * @RETURN MAGE_CATALOG_MODEL_PRODUCT
 */
PROTECTED FUNCTION _clearOptionReferences ()
```

```

{
    IF (!EMPTY($THIS->_OPTIONS)) {
        FOREACH ($THIS->_OPTIONS AS $KEY => $OPTION) {
            $OPTION->SETPRODUCT ();
            $OPTION->CLEARINSTANCE ();
        }
    }

    RETURN $THIS ;
}

```

It breaks the chain between Product referencing its Options, and Options referencing back to Product. The memory from the cleaned up Product and its Options will be successfully collected by the Garbage Collector, if the system needs it.

When should Magento developers minister the Garbage collector and when working with tasks that are long-running what should they do?

- Whenever you write code to process a lot of products, call the product's clearinstance() method after you've finished working with a product to avoid memory leaks.
- If you've made a custom module that sets objects inside products, clean these objects by overloading the product's core methods or observing events.
- Memory leaks are possible in any object or array. Whenever you process a lot of data, think about circular references and help the Garbage Collector by breaking them.