gvHidra - Tareas # 22566: Mejoras de NOTICE en LOG en GVHidra

Estado:	Cerrada	**Prioridad:**	Normal
Autor:	David Pascual	**Categoría:**	
Creado:	2018-12-03	**Asignado a:**	
Actualizado:	2019-05-02	**Fecha fin:**	
Ref. DESIG (Jira):	ENT-291992		
Tema:	Mejoras de NOTICE en LOG en GVHidra		

Descripción

Menajes de LOGs a mejorar:

Undefined index: position in /igep/smarty/plugins/function.cwlista.php on line 193
Undefined index: datos in /igep/smarty/plugins/function.cwlista.php on line 301
Assumed 'SimpleXMLElement' in /igep/smarty/plugins/function.cwpantallaentrada.php on line 120

Iré añadiendo alguno según los vea en los LOGS

Histórico

2018-12-03 12:15 - David Pascual

- Versión prevista establecido a gvHIDRA-5_0_0

2018-12-03 16:20 - David Pascual

PHP E_NOTICE: Undefined index: modal in /igep/smarty/plugins/block.cwventana.php on line 75
PHP E_NOTICE: Undefined index: barTitle in /igep/smarty/plugins/block.cwventana.php on line 76
PHP E_NOTICE: Undefined index: manageSession in /igep/smarty/plugins/block.cwventana.php on line 293

2018-12-03 16:25 - David Pascual

DBAL (Las miraré yo)
Non-static method PEAR::isError() should not be called statically, assuming $this from incompatible context in /igep/include/igep_utils/DBAL/gvHidraDBAL_MDB2.php on line 269
Non-static method PEAR::isError() should not be called statically, assuming $this from incompatible context in /igep/include/igep_utils/DBAL/gvHidraDBAL_MDB2.php on line 384

2018-12-04 16:21 - David Pascual

https://stackoverflow.com/questions/1868874/does-php-run-faster-without-warnings

2018-12-04 16:35 - David Pascual

66 Tips for optimizing your PHP

Here are Webber's points:

Use JSON Instead of XML.
Can also use sprintf instead of variables contained in double quotes, it's about 10x faster.
Avoid the PHP mail() function header injection issue.
If a method can be static, declare it static. Speed improvement is by a factor of 4.
echo is faster than print.(* compare with list from phplens by John Lim)
Use echo's multiple parameters instead of string concatenation.
Set the maxvalue for your for-loops before and not in the loop.
Unset your variables to free memory, especially large arrays.
Avoid magic like __get, __set, __autoload

require_once() is expensive
Use full paths in includes and requires, less time spent on resolving the OS paths.
If you need to find out the time when the script started executing, $_SERVER['REQUEST_TIME'] is preferred to time()
See if you can use strncasecmp, strpbrk and stripos instead of regex
str_replace is faster than preg_replace, but strtr is faster than str_replace by a factor of 4
If the function, such as string replacement function, accepts both arrays and single characters as arguments, and if your argument list is not too long, consider writing a few redundant replacement statements, passing one character at a time, instead of one line of code that accepts arrays as search and replace arguments.
It's better to use select statements than multi if, else if, statements.
Error suppression with @ is very slow.
Turn on apache's mod_deflate
Close your database connections when you're done with them
$row['id'] is 7 times faster than $row[id]
Error messages are expensive
Do not use functions inside of for loop, such as for ($x=0; $x < count($array); $x) The count() function gets called each time.
Incrementing a local variable in a method is the fastest. Nearly the same as calling a local variable in a function.
Incrementing a global variable is 2 times slow than a local var.
Incrementing an object property (eg. $this->prop++) is 3 times slower than a local variable.
Incrementing an undefined local variable is 9-10 times slower than a pre-initialized one.
Just declaring a global variable without using it in a function also slows things down (by about the same amount as incrementing a local var). PHP probably does a check to see if the global exists.
Method invocation appears to be independent of the number of methods defined in the class because I added 10 more methods to the test class (before and after the test method) with no change in performance.
Methods in derived classes run faster than ones defined in the base class.
A function call with one parameter and an empty function body takes about the same time as doing 7-8 $localvar++ operations. A similar method call is of course about 15 $localvar++ operations.
Surrounding your string by ' instead of " will make things interpret a little faster since php looks for variables inside "…" but not inside '…'. Of course you can only do this when you don't need to have variables in the string.
When echoing strings it's faster to separate them by comma instead of dot. Note: This only works with echo, which is a function that can take several strings as arguments.
A PHP script will be served at least 2-10 times slower than a static HTML page by Apache. Try to use more static HTML pages and fewer scripts.
Your PHP scripts are recompiled every time unless the scripts are cached. Install a PHP caching product to typically increase performance by 25-100% by removing compile times.
Cache as much as possible. Use memcached – memcached is a high-performance memory object caching system intended to speed up dynamic web applications by alleviating database load. OP code caches are useful so that your script does not have to be compiled on every request
When working with strings and you need to check that the string is either of a certain length you'd understandably would want to use the strlen() function. This function is pretty quick since it's operation does not perform any calculation but merely return the already known length of a string available in the zval structure (internal C struct used to store variables in PHP). However because strlen() is a function it is still somewhat slow because the function call requires several operations such as lowercase & hashtable lookup followed by the execution of said function. In some instance you can improve the speed of your code by using an isset() trick.
Ex.
view sourceprint?1.if (strlen($foo) < 5) { echo "Foo is too short"; }
vs.

view sourceprint?1.if (!isset($foo{5})) { echo "Foo is too short"; }
Calling isset() happens to be faster then strlen() because unlike strlen(), isset() is a language construct and not a function meaning that it's execution does not require function lookups and lowercase. This means you have virtually no overhead on top of the actual code that determines the string's length.

When incrementing or decrementing the value of the variable $i++ happens to be a tad slower then ++$i. This is something PHP specific and does not apply to other languages, so don't go modifying your C or Java code thinking it'll suddenly become faster, it won't. ++$i happens to be faster in PHP because instead of 4 opcodes used for $i++ you only need 3. Post incrementation actually causes in the creation of a temporary var that is then incremented. While pre-incrementation increases the original value directly. This is one of the optimization that opcode optimized like Zend's PHP

optimizer. It is a still a good idea to keep in mind since not all opcode optimizers perform this optimization and there are plenty of ISPs and servers running without an opcode optimizer.
Not everything has to be OOP, often it is too much overhead, each method and object call consumes a lot of memory.
Do not implement every data structure as a class, arrays are useful, too
Don’t split methods too much, think, which code you will really re-use
You can always split the code of a method later, when needed
Make use of the countless predefined functions
If you have very time consuming functions in your code, consider writing them as C extensions
Profile your code. A profiler shows you, which parts of your code consumes how many time. The Xdebug debugger already contains a profiler.
Profiling shows you the bottlenecks in overview
mod_gzip which is available as an Apache module compresses your data on the fly and can reduce the data to transfer up to 80%
Excellent Article about optimizing php by John Lim
As Reihold Webber pointed to a post from John Lim (found this article copied without state the source here), then i investigate further and truly that is an excellent best practice tutorial for optimizing the php code performance, covered almost all aspects from low level webserver configuration, PHP configuration, coding styling, and performace comparisson as well.

Another good practice for better php performance as written in cluesheet.com are:

Do use single quotes over double quotes.
Do use switch over lots of if statements
Do avoid testing loop conditionals with function tests every iteration eg. for($i=0;i<=count($x);$i++){…
Do use foreach for looping collections/arrays. PHP4 items are byval, greater than PHP5 items are byref
Do consider using the Singleton Method when creating complex PHP classes.
Do use POST over GET for all values that will wind up in the database for TCP/IP packet performance reasons.
Do use ctype_alnum,ctype_alpha and ctype_digit over regular expression to test form value types for performance reasons.
Do use full file paths in production environment over basename/fileexists/open_basedir to avoid performance hits for the filesystem having to hunt through the file path. Once determined, serialize and/or cache path values in a $_SETTINGS array. $_SETTINGS["cwd"]=cwd(./);
Do use require/include over require_once/include_once to ensure proper opcode caching.
Do use tmpfile or tempnam for creating temp files/filenames
Do use a proxy to access web services (XML or JSOM) on foreign domains using XMLHTTP to avoid cross-domain errors. eg. foo.com<–>XMLHTTP<–>bar.com
Do use error_reporting (E_ALL); during debug.
Do set Apache allowoverride to “none” to improve Apache performance in accessing files/directories.
Do use a fast fileserver for serving static content (thttpd). static.mydomain.com, dynamic.mydomain.com
Do serialize application settings like paths into an associative array and cache or serialize that array after first execution.
Do use PHP output control buffering for page caching of heavilty accessed pages
Do use PDO prepare over native db prepare for statements. mysql_attr_direct_query=>1
Do NOT use SQL wildcard select. eg. SELECT *
Do use database logic (queries, joins, views, procedures) over loopy PHP.
Do use shortcut syntax for SQL insers if not using PDO parameters parameters. eg. INSERT INTO MYTABLE (FIELD1,FIELD2) VALUES (("x","y"),("p","q"));

2018-12-14 10:11 - David Pascual

PHP E_NOTICE: Constant APP_PATH already defined in igep\include\IgepPlugin.php on line 62
PHP E_NOTICE: Constant JS_PATH already defined in igep\include\IgepPlugin.php on line 64
PHP E_NOTICE: Constant PATH_CUSTOM already defined in \igep\include\IgepPlugin.php on line 68
PHP E_NOTICE: Constant IMG_PATH_CUSTOM already defined in \igep\include\IgepPlugin.php on line 69
PHP E_NOTICE: Constant VERSION_GVHIDRA already defined in \igep\include\IgepPlugin.php on line 72
PHP E_NOTICE: Constant MODXML_PATH already defined in \igep\\include\IgepPlugin.php on line 74
PHP E_NOTICE: Undefined index: smty_loadScript in igep_noSession.tpl o \igep\views\igep_noSession.php

2018-12-17 08:51 - Veronica Navarro Porter

- Ref. DESIG (Jira) establecido a ENT-291992

2019-01-03 13:48 - David Pascual

PHP Notice: Use of undefined constant GVHIDRA_IMGOPCION - assumed 'GVHIDRA_IMGOPCION' in igep\\include\\gvh_components\\gvHidraMenuParser.php on line 183

2019-05-02 13:16 - Veronica Navarro Porter

- Estado cambiado Nueva por Cerrada

- % Realizado cambiado 0 por 100