Date
06/04/2017

Debug a single file in project xdebug_test

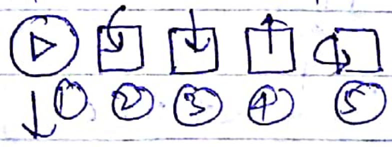
There are 2 files in xdebug_test project:
1. DisplayForm.php (only HTML content)
2. DisplayFormAction.php
   (Web form action page)

Select/Click the file DisplayForm.php on left hand side → In header, click on "Debug" tab & then click/select "Debug File"

or

Right click to the file DisplayForm.php on left hand side and select/click "Debug".

There are five buttons in NetBeans header:



→ Return to the previous file/function/execution flow point). It is just opposite of 'Step Into (F7)'.

Continue (F5) → Runs from one Breakpoint to another breakpoint.

→ Goes to the next PHP line in current file. Does not go into method calls made within current context.

② → Step Over (F8)

③ → Step In/~~to~~ (F7) → debug the code line by line

④ → Step Out (Ctrl+F7)

⑤ → Run to Cursor (F4) → jump the debugging to cursor point & then jump to new cursor point.

→ Goes to the next PHP line in current execution, not just the file. Dives into any method calls made, giving far more detail than Step Over.

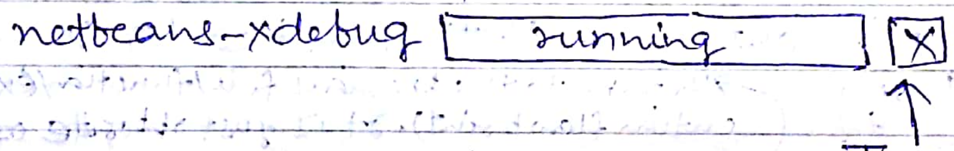
Now xdebug control focuses after last line of HTML code of DisplayForm.php & browser still will not show the web form of DisplayForm.php. Click on button ▷ or ⍗ to end debugging of file DisplayForm.php, browser page load completes and

Date
06/04/2017

webform looks completely. Fill the data in web form & submit the same but page browser shows only in-process form-submission but does not happen anything, this is because xdebug starts on page DisplayFormAction.php & xdebug-control moves to first line of DisplayFormAction.php. Now use $\boxed{\downarrow}$ button to debug line by line. When it ends, browser loads the complete ~~p~~page for DisplayFormAction.php.

Close xdebug session

In NetBeans bottom:



To end the xdebug session, click on this cross button.

U → timestamp in microseconds
H → HTTP HOST
R → Request URL

Date
08/04/2017

Reference: http://blog.ajindra.com/php/profiling-with-xdebug-and-qcachegrind/

Profile an application with Xdebug and QCacheGrind (KCacheGrind)

Profiling - प्रोफाइलिंग - recording ~~a person's~~ an application's behaviour and analyzing ~~psychological~~ characteristics in order to predict their ability

For profiling with Xdebug, we will have to enable Xdebug's profiler first as following:

```
[XDebug] <-  php.ini
zend_extension=D:\xampp\php\ext\php_xdebug-2.
                    5.1-7.0-vc14.dll
xdebug.remote_enable=on
xdebug.remote_log="D:\xampp\tmp\xdebug.txt"
;
xdebug.profiler_enable_trigger=1
xdebug.profiler_output_dir="D:\xampp\tmp"
xdebug.profiler_output_name="cachegrind.out.%u
                                    =%H_%R"
;
xdebug.trace_output_dir="D:\xampp\tmp"
```

when you completes the above updation in your php.ini, restart apache to get effect of the same. Now whenever you want to profile of an URL, just pass XDEBUG_PROFILE or XDEBUG_PROFILE=1 with this URL in form of GET/POST.

Example: To profile of URL http://magento.local/, hit in browser the URL http://magento.local/?XDEBUG_PROFILE. As a result, there will create a file in dir "D:\xampp\tmp" with name like "cachegrind.out.1491567650_097794-magento_local__XDEBUG_PROFILE" to describe the URL (http://magento.local/) profiling.

Date
09/04/2017

Xdebug created the profiling data (i.e. the file) but to visualize this data we need a tool/softwo are named "QCacheGrind".
QCacheGrind: QCacheGrind is a tool/software to visualize the xdebug created profiling data.
Just search with "QCacheGrind" in google and click on first link i.e.
http://sourceforge.net/projects/qcachegrindwin/
Click on "Download" button to download the tool.
Unzip the downloaded file & renamed the folder by "qcachegrind", put this folder in "D:\xampp". No need of installation, just double click on "D:\xampp\qcachegrind\qcachegrind.exe" & open the profiling data file to visualize the same
You can read the file "D:\xampp\qcachegrind\readme" to get usages detail of this software.

Date
26/04/2017

Reference : devdocs.magento.com/guides/v2.1/get-started/gs-web-api-response.html

HTTP Status codes

1. HTTP code : 200
Meaning : Success
Description : The framework returns HTTP 200 to the caller upon success.

2. HTTP code : 400
Meaning : Bad Request
Description : If service implementation throws either Magento_Service_Exception or its derivative, the framework returns a HTTP 400 with an error response including the service specific error code and message. This error code could indicate a problem such as a missing required parameter or the supplied data didn't pass validation.

3. HTTP code : 401
Meaning : Unauthorized
Description : The caller was not authorized to perform the request. For example, the request included an invalid token or a user with customer permissions attempted to access an object that required administrator permissions.

4. HTTP code : 403
Meaning : Forbidden ← मना / वर्जित
Description : Access is not allowed for reasons that are not covered by error code 401.
In easy words, this status code means that accessing the page or resource you were trying to reach is absolutely forbidden for some reason.