



The PHP Company

White Paper:

Troubleshooting PHP Issues with Zend Server Code Tracing



Technical

January 2010

Table of Contents

| | |
|--|----|
| Introduction..... | 3 |
| What is Code Tracing? | 3 |
| Supported Workflows..... | 4 |
| Manual Workflow | 4 |
| Event Monitoring Workflow | 5 |
| Code Tracing Settings | 6 |
| Code Tracing for Monitoring Events | 6 |
| Monitoring Rule Actions | 7 |
| Setting up Monitoring Rules | 7 |
| Using Code Tracing | 9 |
| Reviewing Code Tracing Data | 9 |
| Tracing Tree Tab..... | 9 |
| Identifying Errors | 11 |
| Tracing Stats Tab..... | 12 |
| Conclusion | 13 |

Introduction

Zend Server is an enterprise-ready web application server for running and managing PHP applications that require a high level of reliability, performance and security. A complete, well-tested PHP stack, Zend Server is easy to set up and update on Linux and Windows, cutting the time traditionally spent on tracking, installing, configuring and testing dozens of PHP libraries and drivers. Zend Server's built-in caching and acceleration capabilities ensure your PHP applications are delivering optimized performance, and integrated application monitoring and advanced diagnostic capabilities enable you to quickly detect, isolate and resolve any failures or performance bottlenecks, ensuring the application meets even the most stringent SLA requirements. Zend Server users receive up to 24x7 technical support, continuous software updates, hot fixes, and security patches provided by Zend, The PHP Company.

One of the key features of Zend Server is its ability to analyze the root cause of problems. In order to resolve an issue, one must first figure out what went wrong – fixing a problem is often much easier than identifying its root cause. However, finding root cause can often be challenging during testing, and incredibly difficult when the application is running in production. Trying to reproduce the exact environment, application state, and server load in the development lab is both time-consuming and error-prone, thereby taking developers away from their most important task – writing code.

The release of Zend Server 5 takes Zend Server's root cause analysis capabilities to a whole new level by introducing code tracing. Code tracing captures PHP application execution both in production and in test lab environments. This allows developers to replay reported problems instead of trying to re-create them. As a result, there is a dramatic decrease in time consumed by root cause analysis.

What is Code Tracing?

Think of a black box flight recorder; when something goes wrong with an airplane, the problem is not actually reproduced. Instead, the flight recorder captures the complete data that flight analysts may need in order to understand why the problem occurred. Zend Server does the same for PHP applications. Rather than spending time on trying to set up the environment and reproduce all the steps that led up to the failure, Zend Server captures the full execution of the application in real-time – in production or in the test lab – so root cause can quickly be identified.

Code tracing can be activated automatically to capture problems when they occur in real time, or manually by the user for specific requests. Code tracing captures the following data:

- **Function calls:** every function called as part of a request, represented as a time-synchronized tree based function execution flow

- **Arguments and return values:** all arguments and return values passed into and returned from functions in the traced request
- **Duration:** breakdown of execution time at the function level – particularly useful for identification of performance problems
- **Memory usage:** for every executed function, the memory it consumes – allowing easy identification of functions with abnormal memory consumption or memory leaks
- **File name and line of code:** for each function call, the file name and exact line of code from which the function was called

The trace displayed in the Zend Server web console functions like a DVD player, showing the recorded execution history of the application. Users can follow the footsteps of a single problematic request in order to quickly pinpoint the root cause of the problem.

Supported Workflows

Most problem resolution time is spent on identifying the root cause of an issue. Reproducing a problem is often very difficult and time consuming, and moving from development and QA to staging and production environments makes it even more difficult. There are many moving parts to duplicate: configuration architecture, server load parameters, input data, database states, etc. In some cases, there are problems that cannot be reproduced, especially when dealing with large-scale environments. Code tracing supports workflows that will help you reduce the time spent on root cause analysis. The most common workflows are:

- **Manual Workflow:** Primarily used while performing unit tests during development or when running functional tests in QA.
- **Event Monitoring Workflow:** Primarily used when running automated or load tests, or when running your application in staging or production environments. Code tracing extends the monitoring rules mechanism by enabling the saving of trace data when an event is generated.

Manual Workflow

The code tracing of a single request can be generated manually through the code tracing page in Zend Server (Monitor | Code Tracing) (Figure 1). By executing a request, the full application execution is captured and stored in a trace file. The captured trace data can be collaboratively reviewed leaving no need for developers and test engineers to pore over the symptoms of the defect and no room for misinterpretation of the events leading up to the error.

Trace URL Trace

| <input type="checkbox"/> | ID | Date | Traced URL | Created by | Dump Size |
|--------------------------|-------------------------|-------------------|---|---------------|-----------|
| <input type="checkbox"/> | 10476.1 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-admin/index-extra.php | Monitor Event | 1.01 MB |
| <input type="checkbox"/> | 10474.1 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-admin/index-extra.php | Monitor Event | 531.17 KB |
| <input type="checkbox"/> | 10475.1 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-admin/index-extra.php | Monitor Event | 324.74 KB |
| <input type="checkbox"/> | 12076.6 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-admin/index-extra.php | Monitor Event | 343 KB |
| <input type="checkbox"/> | 12076.5 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-admin/index.php | Monitor Event | 583.69 KB |
| <input type="checkbox"/> | 12076.4 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-login.php | Code Request | 182.48 KB |
| <input type="checkbox"/> | 12076.3 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/blog/wp-login.php | Code Request | 194.52 KB |
| <input type="checkbox"/> | 12076.2 | 07-Jan-2010 21:48 | http://vm-php-demo.zend.net/wordpress/index.php | Code Request | 422.65 KB |

Figure 1: Code tracing screen in Zend Server, allowing the manual generation of a trace for a URL, and showing saved traces

Event Monitoring Workflow

Code tracing in Zend Server has very low performance overhead. This therefore enables its use either while running a load test in the lab or while running in production.

By leveraging the Zend Server event monitoring mechanism, code tracing can save trace data only when a problem occurs. For example, when the performance of a checkout process in a web application drops below a predefined threshold, Zend Server can send an alert and capture the entire execution of the poorly performing request. The combination of knowing that a problem has occurred, while recording the entire execution flow, allows for quick identification and correction even before other users may notice.

Code Tracing Settings

Code tracing is a new Zend Server 5 extension and is installed and turned on (Server setup | components) by default (Figure 2). The behavior of code tracing is controlled by turning the code tracing extension on or off — when the extension is turned on, manual traces can be generated.

The Zend Server monitoring mechanism has another level of control that allows traces to be generated by monitoring events. The Zend Monitoring directive “zend_monitor.event_tracing_mode”, for producing traces based on monitoring events, will be described in the next section.

| Status | Extension | Actions | Configure | Messages |
|--------|-------------------|--------------------------|----------------------------|----------|
| ON | Zend Code Tracing | Turn off | Directives | |

Figure 2: Displaying the status of the Code Tracing extension in the Zend Server setup screen

Code Tracing for Monitoring Events

In addition to manually-generated code traces, Zend Server also supports saving code tracing data for monitoring events (the event monitoring workflow described above), using the Zend Monitoring directive “zend_monitor.event_tracing_mode”. The directive controlling code tracing for events can have the following possible values:

- **Active** [event_tracing_mode=on]: When an event is generated by Zend Server monitoring, the code trace of the request that generated that event is available. Events for which the monitoring rule that produced them have a “save code tracing” action will have that tracing data produced upon the generation of the event. (See section “Setting Up Monitoring Rules” below.)
- **Inactive** [event_tracing_mode=off]: Code tracing is not available for generated events.
- **Standby** [event_tracing_mode=latent]: Code tracing is not available, but events can temporarily activate it for a set timeframe.

Monitoring Rule Actions

Zend Server 5 supports two types of new monitoring rule actions:

1. **Save Code Tracing:** This action will save code trace data when an event of the proper type is generated by Zend Server’s monitoring capabilities. Note that tracing must be set to “active” mode.
2. **Awaken Tracing Functionality:** This action enables code tracing for a specific timeframe (only relevant when code tracing is set to “standby” mode). During this timeframe, code tracing for monitoring is active and trace data is available to be saved according to regular tracing rules and conditions. At the end of the timeframe, code tracing for events will automatically revert to standby mode.

When code tracing for monitoring is set to “inactive” mode, code tracing will ignore any save trace request coming from generated events.

Setting up Monitoring Rules

In the Zend Server Rule Management | Monitoring page, monitoring rules can be set to save the code trace associated with events of that type by checking the “Save code tracing” box (Figure 3), assuming that code tracing has been set to “active” mode.

Step 2: Event Action

Create an event record with severity: **Warning**

Save code tracing

Figure 3: Configuring an event monitoring rule to save the code tracing data for events of that rule’s type

Alternatively, if code tracing has been set to “standby” mode, monitoring rules can be set to awaken (i.e. turn “active”) the code tracing functionality for a specific period of time (Figure 4). When this specific monitoring rule is triggered, code tracing data begins to be stored. Trace data is not available for the instant of the event that triggered the awakening from standby to active mode; but any subsequent instances of that event that take place inside the specified timeframe will have trace data available. At the end of the timeframe, code tracing data on later events is no longer available, as the tracing system has returned to standby mode.

This capability is ideal for use on production systems in which it is inappropriate to always have the even limited overhead of code tracing in place; it is only activated when there is a problem,

and reverts to inactive mode after a user-specified time window to capture a trace of the problem.

Note that during the period in which code tracing is active, trace data is available for all event types, not just the event type that triggered the awakening from standby mode.

Step 2: Event Action

Create an event record with severity: **Warning**

Save code tracing

Awaken tracing functionality if currently in Standby mode

Tracing for all events will be awake for seconds.

Only subsequent events' tracing data will be saved

Figure 4: Configuring an event monitoring rule to awaken code tracing data for events of that rule's type for a user-specified period of time

When a monitoring rule that has the “Save code tracing” action is triggered, an event will be generated and trace data will be saved and attached to the event instance (Figure 5). By clicking the “Show Code Tracing” button, the code trace view window will open with the entire application execution data that led up to the problem.

21 - Slow Query Execution

Page last refreshed: 20-Jan-2010 15:12

Occurred 3 times between 31-Dec-2009 21:15 and 07-Jan-2010 21:48

Status: Open Severity: Warning

URL: http://vm-php-demo.zend.net/wordpress/ Source File: /usr/share/wordpress/wp-includes/wp-db.php : 275

Function Name: mysql_query

| Last Time | Count | Trace | Run-Time (ms) |
|--------------|-------|-------------|---------------|
| 07-Jan 21:48 | 1 | Has Trace/s | 1292 |
| 31-Dec 21:15 | 2 | Has Trace/s | 1516 |

Function Data Request Server Backtrace

Show Code Tracing Export Trace File Export

```
▼ GET
  array (
    'jax' => 'planetnews',
  )
▶ POST (empty)
▶ COOKIE
```

Zend Studio Diagnostics: Debug Event Profile Event Show File in Zend Studio Settings

Figure 5: Viewing an event that has code tracing data associated with it (note “Show Code Tracing” button on the right side)

Using Code Tracing

Reviewing Code Tracing Data

When the “Show Code Tracing” button is clicked from the detailed view of an event, or when a code trace file from the code tracing view is opened, the main code tracing view opens in a separate window. This view enables the review of the application execution as a chronological function call tree, or as the request’s execution statistics from a function perspective.

Tracing Tree Tab

The tracing tree tab displays a chronologically ordered function call tree (Figure 6). As mentioned above, the tree tab displays the following data:

- **Function calls:** every function called as part of a request, represented as a time-synchronized tree based function execution flow
- **Arguments and return values:** all arguments and return values passed into and returned from functions in the traced request
- **Duration:** breakdown of execution time at the function level – particularly useful for identification of performance problems
- **Memory usage:** for every executed function, the memory it consumes – allowing easy identification of functions with abnormal memory consumption or memory leaks
- **File name and line of code:** for each function call, the file name and exact line of code from which the function was called

Atop the tree is a checkbox that, when checked, highlights the most time-consuming path in the tree. The highlighted nodes represent the critical path in terms of execution time – slow functions can easily be seen by viewing this path. Walking through the function call tree can help in code optimization by displaying an analysis of application performance at the function level.

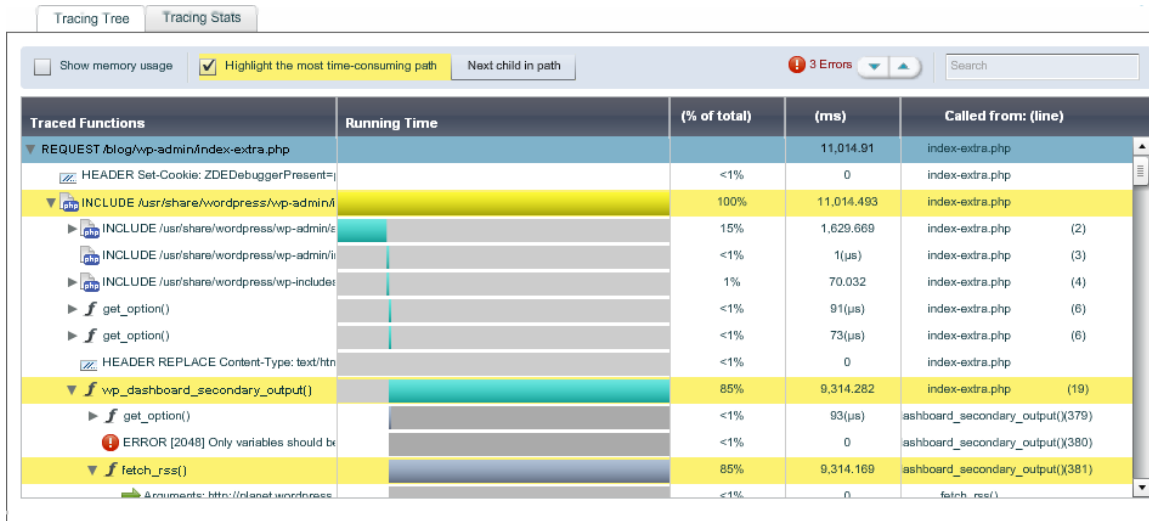


Figure 6: The function tree view of a request's code trace

Identifying Errors

Whenever a PHP function returns an error, or throws an exception, an error node will appear in the function call tree (Figure 7). This reduces the time it takes to pinpoint an error's root cause simply by reviewing the sequence of events that led up to that error. Finding the errors is simple by using the navigation up-and-down buttons atop the tree.

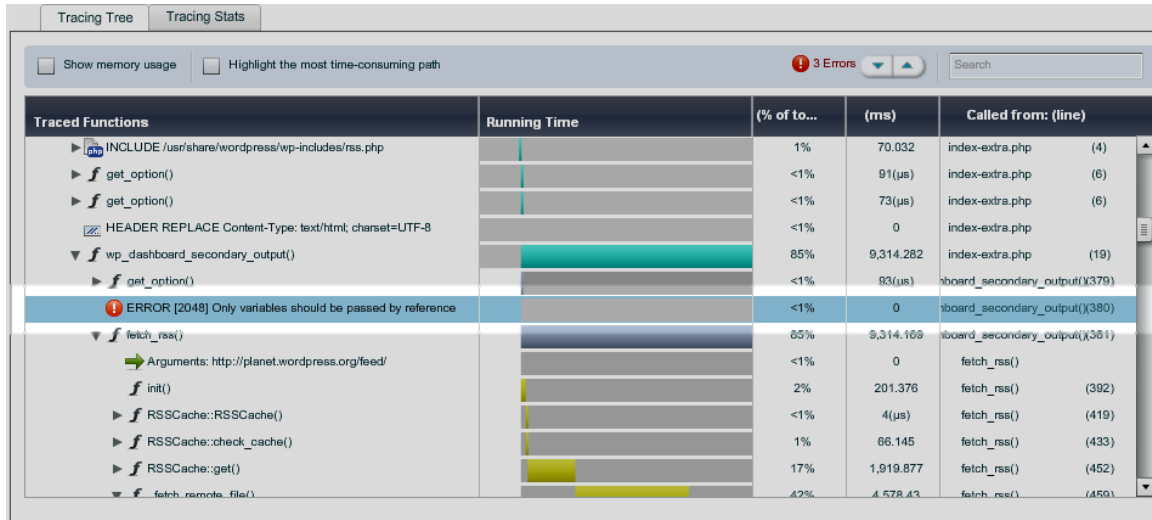


Figure 7: The function tree view of a code trace with an error highlighted

Tracing Stats Tab

The “Tracing Stats” tab displays the code tracing data from a function perspective (Figure 8). For each function, you can see:

- Total running time for the function, displayed both as the total time including all children calls, and as the running time for just the function itself
- Total memory consumed
- Total number of calls to that function
- Source file where that function is defined

When expanding the node of a particular function, you can then see a list of actual calls to the function. This shows you:

- File and line number from which the call took place
- Duration of that specific calling of the function
- Memory consumption of that specific calling of the function

You also have the option to show that function call in the tracing tree tab, expanded to focus on that instance of the function. Opening the function call tree provides execution flow context so what happened before the call was made, or what the impact of the function was, can be reviewed.

The screenshot shows the 'Tracing Stats' tab with a table of function statistics. The table has columns for Function Name, # of Calls, Total running time (all calls) (including children and just own), and Located in file. Below the table, the 'wp_cache_set()' function is expanded to show it was called from 5 locations.

| Function Name | # of Calls | Total running time (all calls) | | Located in file |
|-----------------|------------|--------------------------------|-------------|-----------------|
| | | Including Children | Just own | |
| wpdb::get_var() | 1 | 8,020µs | 35µs | wp-db.php |
| wpdb::query() | 10 | 1,950,982µs | 358,049µs | wp-db.php |
| apply_filters() | 199 | 36,386µs | 3,828µs | plugin.php |
| wpdb::flush() | 10 | 85µs | 85µs | wp-db.php |
| mysql_query() | 10 | 1,592,804µs | 1,592,804µs | |
| wp_cache_set() | 5 | 55µs | 23µs | cache.php |

| Called from 5 locations (line) | | | | |
|--------------------------------|--|-------|------|------|
| ▶ | /usr/share/wordpress/wp-includes/functions.php | (967) | 10µs | 10µs |
| ▶ | /usr/share/wordpress/wp-includes/functions.php | (230) | 9µs | 9µs |
| ▶ | /usr/share/wordpress/wp-includes/functions.php | (230) | 8µs | 8µs |
| ▶ | /usr/share/wordpress/wp-includes/functions.php | (230) | 8µs | 8µs |
| ▶ | /usr/share/wordpress/wp-includes/comment.php | (473) | 20µs | 20µs |

Figure 8: The tracing stats view of a request’s code trace

Conclusion

While reproducing a problem may be the classic way to identify and correct it, reproduction is often difficult and sometimes entirely impossible. Code tracing with Zend Server 5 enables viewing of what actually went wrong and provides a replay of the code execution. Code tracing is suitable for both development/testing (speeding up the development cycle) and for production (reducing mean time to resolution), and is a valuable tool for enhancing the quality and reliability of your PHP application.

For more information about Zend Server and code tracing, please go to <http://www.zend.com/server>