



Добро пожаловать во вторую серию нашего повествования про xdebug. Установка xdebug и его базовые возможности были рассмотрены в первой серии. В данной статье мы рассмотрим трассировку программы.

Предыдущая серия цикла [Introducing xdebug](#)

## Создание лога трассировки

Вместо ручного добавления различных функций вывода отладочной информации мы можем воспользоваться xdebug для создания лога трассировки. Лог трассировки это список вызовов функций и методов класса на всем протяжении выполнения скрипта. Преимущество заключается в том, что каждый вызов найдет свое отражение в логе

Лог трассировки обычно различается от запуска к запуску, так как он зависит от входящих данных, которые различаются от запроса к запросу.

Отслеживание лога трассировки помогает нам понять, как происходит выполнение нашей программы, однако очень сложно визуализировать все возможные ветвления, если только наша программа не является очень простой. Один if предполагает два возможных пути развития программы. С каждым дальнейшим if количество (теоретически) возможных путей развития умножается на 2. Именно из-за этого тестирование больших программ достаточно сложно: слишком много различных путей развития и каждый необходимо протестировать.

А сейчас приступим к созданию лога трассировки. xdebug поддерживает два различных формата для логов. По умолчанию логи создаются в приятном для человеческого глаза формате, их удобно читать и анализировать. Если вы хотите обработать лог трассировки какой-либо программой (предназначенной для обработки логов xdebug), необходимо сконфигурировать xdebug в другом формате. Такой формат (computer-readable) легко обрабатывается программой (анализатором логов), потому что он содержит колонки фиксированной длины

Если вы хотите получить лог трассировки для каждого запускаемого вами скрипта, добавьте следующие команды

```
xdebug.auto_trace=On
xdebug.trace_output_dir=c:\data
```

в php.ini. Не забудьте перестартовать ваш веб-сервер после изменения php.ini.

Вторая строка определяет путь, по которому xdebug будет сохранять логи. По умолчанию этот путь установлен в /tmp, это хороший выбор для, но вызовет проблемы в Windows, так как c:\tmp (считается что c: это текущий диск) обычно не создана. Поэтому необходимо создать каталог в Windows (или прописать существующий), в который xdebug будет сохранять логи трассировки, иначе ничего не получится.

Выполнение функций и методов будут очень медленными так как каждый их вызов, а также результаты будут записаны в лог.

## Трассировка части приложения

Так как размер лога трассировки очень быстро растет, хорошим решением является создать лог только для «интересных» частей приложения, например, создание лога трассировки на этапе инициализации.

Давайте посмотрим как создать лог трассировки для маленькой рекурсивной функции (вычисление факториала), в данном примере мы начнем и закончим трассировку в реальном времени. Если вы активировали трассировку в php.ini, вы увидите предупреждение, что трассировка уже начата для вашего скрипта.

```
xdebug_start_trace('c:/data/fac.xt');
print fac(7);

function fac($x)
{
    if (0 == $x) return 1;
    return $x * fac($x - 1);
}

xdebug_stop_trace();
```

Как вы можете видеть по пути и названию файла, программа запущена в Windows. Если вы работаете в Unix, вы должны использовать другое имя файла. Функция xdebug\_start\_trace начинает запись трассировки, а xdebug\_stop\_trace останавливает. Когда этот код будет запущен, будет сформирован приблизительно такой лог файл:

```
TRACE START [2007-10-26 12:18:48]
0.0068 53384 -> fac() C:\www\fac.php:5
0.0069 53584 -> fac() C:\www\fac.php:10
0.0069 53840 -> fac() C:\www\fac.php:10
0.0070 54096 -> fac() C:\www\fac.php:10
0.0070 54376 -> fac() C:\www\fac.php:10
0.0071 54656 -> fac() C:\www\fac.php:10
0.0072 54936 -> fac() C:\www\fac.php:10
0.0072 55216 -> fac() C:\www\fac.php:10
0.0073 55392 -> xdebug_stop_trace() C:\www\fac.php:13
```

```
0.0237 55392
TRACE END [2007-10-26 12:18:48]
```

В каждой строке первая колонка показывает общее время запуска кода. Вторая колонка показывает количество памяти, которое скрипт занимает в данный момент. Последняя колонка показывает файл, строку и название функции. xdebug делает отступы для каждой функции в соответствии с местом в стеке вызов для удобочитаемости. Если вы хотите чтобы xdebug показывал разницу в использовании памяти, добавьте

```
xdebug.show_mem_delta=On
```

в php.ini. Разница – это различие между использованием памяти между текущей строкой и предыдущей. Вы можете добавить данную настройку во время выполнения используя функцию ini\_set, но только после того как начнется трассировка. Функция xdebug\_start\_trace имеет второй необязательный параметр. Вы можете использовать один или больше из трех следующих настроек: XDEBUG\_TRACE\_APPEND добавляет к существующему файлу трассировки. Когда установлен XDEBUG\_TRACE\_COMPUTERIZED, xdebug создает лог трассировки в формате удобном для чтения программой-анализатором. Использование этого параметра такое же, как и установка параметра xdebug.trace\_format в 1. Параметр XDEBUG\_TRACE\_HTML устанавливает формат лога трассировки в виде HTML.

Следующий скриншот показывает HTML-формат лога трассировки:

#	Time		Function	Location
2	0.004286	->	fac()	C:\data\www\fac.php:5
3	0.004365	->	fac()	C:\data\www\fac.php:10
4	0.004423	->	fac()	C:\data\www\fac.php:10
5	0.004482	->	fac()	C:\data\www\fac.php:10
6	0.004617	->	fac()	C:\data\www\fac.php:10
7	0.004654	->	fac()	C:\data\www\fac.php:10
8	0.004690	->	fac()	C:\data\www\fac.php:10
9	0.004724	->	fac()	C:\data\www\fac.php:10
10	0.004780	->	xdebug_stop_trace()	C:\data\www\fac.php:13

### Добавление информации в лог трассировки

Также мы можем настроить xdebug, чтобы он сохранял нам параметры, которые мы передаем в функции. Это очень полезно чтобы лучше понимать, как происходит выполнение программы.

Сохранение параметров в лог трассировки устанавливается параметром xdebug.collect\_params, о котором было описано в предыдущей статье. xdebug.collect\_params принимает числовые значения, 0 означает никакой дополнительной информации, 4 означает, что необходимо отобразить название переменной и всю информацию о каждом параметре функции. Значение 3 будет отображать имена и значения переменных, обрезанных в соответствии с настройками параметров xdebug.var\_display\_max\_data, debug.var\_display\_max\_children и xdebug.var\_display\_max\_depth, описанных в предыдущей статье.

Ниже выдержка из лога трассировки с сохранением параметров функций:

```
0.0033 54320 -> fac($x = 7) C:\www\fac.php:6
0.0034 54432 -> fac($x = 6) C:\www\fac.php:11
```

Давайте добавим дополнительную информацию. Используя

```
xdebug.collect_return=On
```

мы настроим xdebug чтобы он сохранял возвращаемые значения для каждой функции. Это сделает лог немного трудным для чтения, но посмотрите на лог нашей функции, вы увидите какое значение возвращается на каждом этапе рекурсии:

```
TRACE START [2007-10-26 12:30:04]
0.0133 55704 +48 -> fac($x = 7) C:\www\fac.php:8
0.0133 55840 +136 -> fac($x = 6) C:\www\fac.php:13
0.0134 56096 +256 -> fac($x = 5) C:\www\fac.php:13
0.0134 56352 +256 -> fac($x = 4) C:\www\fac.php:13
0.0134 56632 +280 -> fac($x = 3) C:\www\fac.php:13
0.0135 56912 +280 -> fac($x = 2) C:\www\fac.php:13
0.0135 57192 +280 -> fac($x = 1) C:\www\fac.php:13
0.0135 57472 +280 -> fac($x = 0) C:\www\fac.php:13
    ==> 1
    ==> 1
    ==> 2
    ==> 6
    ==> 24
    ==> 120
    ==> 720
    ==> 5040
```

```
0.0140 57648 +176 -> xdebug_stop_trace() C:\www\fac.php:26
0.0140 57648
TRACE END [2007-10-26 12:30:04]
```

Как вы видите в данном примере установлен параметр `xdebug.show_mem_delta`.

## Озаглавливание лога трассировки

До текущего момента, мы явно указывали название лога трассировки. Это не всегда удобно. Если вам необходимо рассматривать логи в зависимости от входящих данных или состояния приложения, неплохо чтобы сам xdebug автоматически присваивал название. xdebug может присваивать имя в зависимости от того что вы используете, параметр `xdebug.auto_trace` или функцию `xdebug_start_trace()` для активации трассировки. Если в последнем примере передать пустой аргумент в качестве параметров функции `xdebug_start_trace`, xdebug автоматически выберет имя для файла. Название файла сгенерированного xdebug всегда начинается с «`trace.`» и имеет расширение `.xt`. Часть имени между точками по умолчанию это CRC от рабочего каталога. Устанавливая `xdebug.trace_output_name` вы можете определить другое имя для лога трассировки. `xdebug.trace_output_name` принимает строку в качестве аргумента, строка может содержать различные спецификаторы. Ниже список самых важных спецификаторов:

- `%c` — CRC рабочего каталога
- `%p` — идентификатор процесса
- `%r` — некое случайное число
- `%u` — текущее время с микросекундами
- `%H` — `$_SERVER['HTTP_HOST']`
- `%R` — `$_SERVER['REQUEST_URI']`

Используя умную комбинацию указанных спецификаторов, вы можете заставить xdebug создавать различное количество логов трассировки и понимать какой лог к какому сценарию относится в зависимости от названия виртуального хоста или запросов. В большинстве случаев вы захотите создать для одного запуска скрипта. Поэтому xdebug перезапишет существующий лог с указанным именем. Вы можете настроить, чтобы xdebug добавлял информацию к текущему, используя следующую настройку

```
xdebug.trace_options=1
```

в `php.ini`.

Если вам необходимо знать во время запуска какое имя выберет xdebug для записи лога, вы можете вызвать функцию `xdebug_get_tracefile_name()`.

## Заключение

Возможность использовать трассировку функций это очень полезная вещь, которую нам предоставляет xdebug. Вы можете создавать логи ваших программ без добавления вызовов различных функций. Все вызовы функций будут отображены без явных вызовов функций отображения справочной информации.

Вы можете использовать различные утилиты подобные `ггер` чтобы найти необходимую информацию или секцию в логе трассировки, или написать маленькую программку на PHP, которая будет разбирать этот файл для вас. Трассировка это не замена отладчику. Отладчик будет рассмотрен в четвертой части повествования.

Вы не должны активировать трассировку на боевых машинах, это очень сильно ухудшит производительность, каждая строка программы будет отражена в логе трассировки.

Следующая статья будет посвящена `Profiling PHP Applications With xdebug`.

 [Xdebug](#)

 +17 

 4500  107  [Stefan Priebisch](#)  [great\\_boba](#) 34,1

   

 **комментарии (16)** отслеживать новые: ☐ в почте ☐ в трекере

НЛО прилетело и опубликовало эту надпись здесь



[alkeeper](#) 11 марта 2008 в 14:11 # ☆ h ↑

+1 ↑ ↓

Встречный вопрос - а чем хуже?

НЛО прилетело и опубликовало эту надпись здесь



[alkeeper](#) 26 августа 2008 в 07:26 # ☆ h ↑

0 ↑ ↓

Это вы своевременно присоединились к дискуссии.



[AlienZzzz](#) 11 марта 2008 в 14:56 # ☆ h ↑

0 ↑ ↓

хдебуг проще ставить чем зендовские и зендовский надо покупать.

↑ ↓



**develop7**, 11 марта 2008 в 15:11 # ☆ h ↑

0

Имо лучше Zend Debugger'a по каждому из пунктозаголовков [отсюда](#). Чего-то в ZD нет вообще, что-то менее функционально. Еще открытые исходники — жирный такой плюс.



**shaelf**, 11 марта 2008 в 14:07 # ☆

+2 ↑ ↓

Для трассировки и профайлинга ленивые люди давно используют KCachegrind/WinCachegrind



**Kirax**, 11 марта 2008 в 14:34 # ☆

0 ↑ ↓

Занятно конечно, но на практике для отладки гораздо удобнее использовать GUI клиент для xdebug.



**great boba**, 11 марта 2008 в 14:54 # ☆ h ↑

0 ↑ ↓

это как раз будет в третьей части



**shaelf**, 11 марта 2008 в 15:02 # ☆ h ↑

0 ↑ ↓

Прошу прощения, но немного выше я как раз писал о GUI приложении для трассировки



**vrazbros**, 27 января 2009 в 12:37 # ☆ h ↑

0 ↑ ↓

Статья очень полезная, сохранено на <http://www.xdebug.ru>



**aaaa**, 11 марта 2008 в 18:17 # ☆ h ↑

0 ↑ ↓

да, это точно...



**HighOctane**, 11 марта 2008 в 15:24 # ☆



0 ↑ ↓

супер, спасибо :)



**darkloki**, 11 марта 2008 в 16:15 # ☆

0 ↑ ↓

Поправьте пожалуйста:  
что трассировкУ уже начать для вашего скрипта;  
которую наС предоставляет xdebug



**great boba**, 11 марта 2008 в 16:48 # ☆ h ↑

0 ↑ ↓

спасибо



**DenCh**, 12 марта 2008 в 12:22 # ☆

0 ↑ ↓

Спасибо автору за отличный «миницикл» статей, после прочтения первой части сразу поставил Xdebug, доволен, благодарен отличное раскрытие настроек и возможностей.  
С нетерпением буду ждать продолжения.

[ISON, да не комета](#)

[Пять способов выгореть для программиста](#)

[iToilet. Офисный туалет свободен](#)

**Brainstorage**  
Все мозги в одном месте

**ТОСТЕР**  
Q&A-сервис для разработчиков

**ФРИЛАНСИМ**  
Заказы для фрилансеров

**ХАНТИМ**  
Вакансии для айтишников

**АВТОКАДАБРА**  
Уютная и дружелюбная