

Xdebug and an introduction to Zend Magic

PHP International Conference 2002

November 4, 2002. Frankfurt, Germany

Derick Rethans <d.rethans@jdimedia.nl>

- o Introduction to the engine
- o Parse
- o Compile
- o Execute
- o Xdebug
- o Getting data
- o Using data

- o Analyze script source
- o Divide into logical blocks
- o Give special blocks a meaning

Our friend Mr Parse Error:

```
Parse error: parse error,  
unexpected T_CLASS, expecting ',' or ';' in - on line 2
```

- o The building blocks of a script
- o Strings, variables, keywords
- o Action rules

```
<ST_IN_SCRIPTING>"::" {  
    return T_PAAMAYIM_NEKUDOTAYIM;  
}  
  
<ST_IN_SCRIPTING>{DNUM}|{EXPONENT_DNUM} {  
    zendlval->value.dval = strtod(yytext, NULL);  
    zendlval->type = IS_DOUBLE;  
    return T_DNUMBER;  
}
```

tokenize.php:

```
<?php
    function apply_magic(&$a) {
        $a = $a + rand(0,3);
    }

    $a = 5;
    apply_magic($a);
    echo "$a\n";
? >
```

- o Tokenizer extension
- o Refactoring

```
<?php
    function apply_magic(&$a) {
```

0	T_OPEN_TAG	[<?php]
1	T_WHITESPACE	[]
2	T_FUNCTION	[function]
3	T_WHITESPACE	[]
4	T_STRING	[apply_magic]
5		[(
6		[&]
7	T_VARIABLE	[\$a]
8		[)]
9	T_WHITESPACE	[]
10		[{]

- o Interpreting tokens
- o Generating execution units
- o Generating class and function tables

- o Oparrays
- o Compiled code
- o One for every script element

- o Opcode
- o Basic execution unit
- o Two operands
- o One result

tokenize.php:

```
<?php
function apply_magic(&$a) {
    $a = $a + rand(0,3);
}

$a = 5;
apply_magic($a);
echo "$a\n";
? >
```

- o Disassembler: vld
- o Dumps oparray per element

script body:

number of ops: 12

```
0  NOP          0, 0, 0
1  FETCH_W     $0, 'a', 1
2  ASSIGN      $1, $0, '5'
3  FETCH_W     $2, 'a', 1
4  SEND_REF    0, $2, 1
5  DO_FCALL    $3, 'apply_magic', 0
6  INIT_STRING %4, 0, 0
7  FETCH_R     $5, 'a', 1
8  ADD_VAR     4, 4, $5
9  ADD_CHAR    4, 4, '10'
10 ECHO        0, %4, 0
11 RETURN     0, '1', 0
```

apply_magic() function:

number of ops: 10

```
0  FETCH_W      $0, 'a', 1
1  RECV        $0, '1', 0
2  FETCH_R     $2, 'a', 1
3  SEND_VAL    0, '0', 1
4  SEND_VAL    0, '3', 2
5  DO_FCALL    $3, 'rand', 0
6  ADD         %4, $2, $3
7  FETCH_W     $1, 'a', 1
8  ASSIGN      $5, $1, %4
9  RETURN      0, '', 0
```

- o Generated code not always optimal
- o Optimizing techniques:
 - o Removing NOPs
 - o Removing unnecessary jumps
 - o Concatenation of strings
 - o Dead-code elimination
 - o Cache function table lookups
 - o Function inlining

foo.php:

```
<?php
function foo() {
}

if (40 < 50) {
    echo "foo\n";
}
echo "end\n";
? >
```

generated opcodes:

0	NOP	0, 0, 0
1	IS_SMALLER	%0, '40', '50'
2	JMPZ	0, %0, 5
3	ECHO	0, 'foo\n', 0
4	JMP	0, 5, 0
5	ECHO	0, 'end\n', 0
6	RETURN	0, '1', 0

Eliminating the NOP and unnecessary jump:

before:

```
0  NOP                0, 0, 0
1  IS_SMALLER        %0, '40', '50'
2  JMPZ              0, %0, 5
3  ECHO              0, 'foo\n', 0
4  JMP               0, 5, 0
5  ECHO              0, 'end\n', 0
6  RETURN            0, '1', 0
```

after:

```
0  IS_SMALLER        %0, '40', '50'
1  JMPZ              0, %0, 3
2  ECHO              0, 'foo\n', 0
3  ECHO              0, 'end\n', 0
4  RETURN            0, '1', 0
```

Eliminating the NOP, unnecessary jump and unnecessary jump condition:

before:

```
0  NOP                0, 0, 0
1  IS_SMALLER        %0, '40', '50'
2  JMPZ              0, %0, 5
3  ECHO              0, 'foo\n', 0
4  JMP               0, 5, 0
5  ECHO              0, 'end\n', 0
6  RETURN            0, '1', 0
```

after:

```
0  ECHO              0, 'foo\n', 0
1  ECHO              0, 'end\n', 0
2  RETURN            0, '1', 0
```

- o Optimizers:
- o ZendOptimizer
- o PHP Accelerator and Zend Cache
- o Zend Encoder and ionCube encoder
- o Your own optimizer!
- o When is it useful?
- o Large speed increases
- o Cached PHP scripts
- o Encoded PHP scripts

- o Executor executes opcodes
- o 116 or 140 different opcodes
- o Per file execution

test.inc:

```
<?php
    function foo () {
        echo "bar\n";
    }
    echo "inc\n";
? >
```

test.php

```
<?php
    echo "foo\n";
    include "test.inc";
    foo();
    echo "more bar\n";
? >
```

The engine:

- o parses and compiles test.php
- o starts executing test.php
- o parses and compiles test.inc when the include() statement is encountered
- o starts executing test.inc
- o resumes executing test.php

- o Gathering data on:
 - o Function calls
 - o Function parameters
 - o Files
 - o Memory usage

- o Displaying:
 - o Backtraces
 - o Function traces
 - o Performance data

- o Available for several 'events':
- o Executing user defined function
- o Compiling a file
- o Executing internal function
- o Executing a statement
- o Function start
- o Function end

```
void (old_execute)(zend_op_array op_array TSRMLS_DC);
void xdebug_execute(zend_op_array *op_array TSRMLS_DC);

PHP_MINIT_FUNCTION(xdebug) {
    old_execute = zend_execute;
    zend_execute = xdebug_execute;
    return SUCCESS;
}

PHP_MSHUTDOWN_FUNCTION(xdebug) {
    zend_execute = old_execute;
    return SUCCESS;
}

void xdebug_execute(zend_op_array *op_array TSRMLS_DC)
{
    printf ("Executing in file %s\n", op_array->filename);
    old_execute (op_array TSRMLS_CC);
}
}
```

script:

```
<?php
    $foo = "boo!";
    echo "The cow says $foo\n";
? >
```

opcodes:

```
number of ops: 14
0  FETCH_W      $0, 'foo', 1
1  ASSIGN      $1, $0, 'boo!'
2  INIT_STRING  %2, 0, 0
3  ADD_STRING   2, 2, 'The'
4  ADD_STRING   2, 2, ' '
5  ADD_STRING   2, 2, 'cow'
6  ADD_STRING   2, 2, ' '
7  ADD_STRING   2, 2, 'says'
8  ADD_STRING   2, 2, ' '
9  FETCH_R      $3, 'foo', 1
10 ADD_VAR      2, 2, $3
11 ADD_CHAR     2, 2, '10'
12 ECHO         0, %2, 0
13 RETURN      0, '1', 0
```

```
zend_op_array (old_compile_file)(zend_file_handle* file_handle, int type
TSRMLS_DC);
zend_op_array vld_compile_file(zend_file_handle, int TSRMLS_DC);

PHP_MINIT_FUNCTION(vld)
{
    old_compile_file = zend_compile_file;
    zend_compile_file = vld_compile_file;

    return SUCCESS;
}

PHP_MSHUTDOWN_FUNCTION(vld)
{
    zend_compile_file = old_compile_file;

    return SUCCESS;
}

void vld_compile_file(zend_op_array *op_array TSRMLS_DC)
{
    zend_op_array *op_array;

    op_array = old_compile_file (file_handle, type TSRMLS_CC);
    concat_strings (&op_array);

    return op_array;
}
```

```
void concat_strings (zend_op_array **opa)
{
    int i;
    int last_add_string = -1;

    for (i = 0; i < (*opa)->size; i++) {
        if ((*opa)->opcodes[i].opcode == ADD_STRING) ||
            ((*opa)->opcodes[i].opcode == ADD_CHAR)) {
            if (last_add_string == -1) {
                last_add_string = i;
            } else {
                if ((*opa)->opcodes[i].opcode == ADD_STRING) {
                    opt_concat_string (*opa, last_add_string, i);
                } else {
                    opt_concat_char (*opa, last_add_string, i);
                }
                opt_set_nop (*opa, i);
            }
        } else {
            last_add_string = -1;
        }
    }
}
```

script:

```
<?php
    $foo = "boo!";
    echo "The cow says $foo\n";
? >
```

opcodes:

```
number of ops: 14
0  FETCH_W          $0, 'foo', 1
1  ASSIGN           $1, $0, 'boo!'
2  INIT_STRING      %2, 0, 0
3  ADD_STRING       2, 2, 'The cow says '
4  NOP              2, 2, ' '
5  NOP              2, 2, 'cow'
6  NOP              2, 2, ' '
7  NOP              2, 2, 'says'
8  NOP              2, 2, ' '
10 ADD_VAR          2, 2, $3
11 ADD_CHAR         2, 2, '10'
12 ECHO             0, %2, 0
13 RETURN           0, '1', 0
```

- o Functions in extensions
- o Different from user defined functions
- o Its own hook:

```
void xdebug_execute_internal(  
    zend_execute_data * execute_data,  
    int return_value_used  
    TSRMLS_DC  
) {  
    / Do your own stuff /  
  
    execute_internal(  
        execute_data,  
        return_value_used  
        TSRMLS_CC  
    );  
};
```

- o 4 extra generated opcodes:
- o EXT_FCALL_BEGIN
- o EXT_FCALL_END
- o EXT_STMT
- o EXT_NOP
- o Handlers can be bounded
- o Can be activated with the -e option:

```
$ php -h | grep '\-e'  
-e      Generate extended information for debugger/profiler
```

Without extended info:

```

0  NOP          0, 0, 0

1  FETCH_W     $0, 'a', 1
2  ASSIGN      $1, $0, '5'

3  FETCH_W     $2, 'a', 1
4  SEND_REF    0, $2, 1
5  DO_FCALL    $3, 'magic', 0

6  INIT_STRING 4, 0, 0
7  FETCH_R     $5, 'a', 1
8  ADD_VAR     4, 4, $5
9  ADD_CHAR    4, 4, '10'
10 ECHO        0, 4, 0

11 ECHO        0, '\n', 0
12 RETURN      0, '1', 0

```

With extended info:

```

0  EXT_STMT    0, 0, 0
1  NOP          0, 0, 0
2  EXT_STMT    0, 0, 0
3  FETCH_W     $0, 'a', 1
4  ASSIGN      $1, $0, '5'
5  EXT_STMT    0, 0, 0
6  EXT_FCALL_BEGIN 0, 0, 0
7  FETCH_W     $2, 'a', 1
8  SEND_REF    0, $2, 1
9  DO_FCALL    $3, 'magic', 0
10 EXT_FCALL_END 0, 0, 0
11 EXT_STMT    0, 0, 0
12 INIT_STRING 4, 0, 0
13 FETCH_R     $5, 'a', 1
14 ADD_VAR     4, 4, $5
15 ADD_CHAR    4, 4, '10'
16 ECHO        0, 4, 0
17 NOP          0, 0, 0
18 EXT_STMT    0, 0, 0
19 ECHO        0, '\n', 0
20 RETURN      0, '1', 0

```

- o Functions and classnames
- o Normal functions (strlen())
- o Class members (\$obj->strlen())
- o Static class members (class::strlen())
- o include/require/eval
- o Parameters
- o From argument stack
- o Converted to strings

- o Memory footprint
- o Script memory
- o emalloc()
- o --enable-memory-limit
- o Time index
- o Simple profiling

- o Stack for backtrace information
- o List for function traces information
- o Same stack frame used in both structures
- o Reference counted

- o Backtraces
- o Functiontraces
- o Remote interfaces:
 - o PHP 3 debugger mode
 - o GDB debugger mode
- o Easy extendible

- o Replaces standard PHP error handler
- o Shows backtraces

```
<table border='1' cellspacing='0' align='center'> <tr><td class='trace' bgcolor='#ffbfff'
colspan='3'><b>Warning</b>: Wrong parameter count for str_replace() in
<b>/home/httpd/html/ecommerce/common/algorithms.php</b> on line <b>129</b><br /> <tr><th
class='trace' bgcolor='#aaaaaa' colspan='3'>Call Stack</th></tr> <tr><th class='trace'
bgcolor='#cccccc'>#</th><th class='trace' bgcolor='#cccccc'>Function</th><th class='trace'
bgcolor='#cccccc'>Location</th></tr> <tr><td class='trace' bgcolor='#ffffff'
align='center'>1</td><td class='trace' bgcolor='#ffffff'>{main}()</td><td class='trace'
```

- o Contains all function calls
- o Performance tool
- o Logging to a file

```

<table border='1' cellspacing='0'>
  <tr><th class='tracesml' bgcolor='#aaaaaa'
colspan='5'>Function trace</th></tr>
  <tr><th class='tracesml' bgcolor='#cccccc'>Time</th><th
class='tracesml'
  bgcolor='#cccccc'>#</th><th
  class='tracesml'
  bgcolor='#cccccc'>Function</th><th class='tracesml'
  bgcolor='#cccccc'>Location</th><th
class='tracesml' bgcolor='#cccccc'>Memory</th></tr>
  <tr><td class='tracesml' bgcolor='#ffffff'
align='center'>0</td><td class='tracesml' bgcolor='#ffffff'
align='left'><pre>-></pre></td><td
class='tracesml'
  bgcolor='#ffffff'>{main}()</td><td
  class='tracesml'

```

- o JIT debugging
- o "Just In Time"
- o Only initiated on an event

- o REQ debugging
- o Session(request) based
- o Initiated on script start

- o PHP 3 Debugger
- o GDB Handler
- o Extensible

- o Interactive
- o Remote
- o Familiar interface
- o GUI usable

- o Breakpoints
- o Stepping
- o Watches
- o Accessing data
- o Reading Local and Global variables
- o Writing Local and Global variables
- o Profiling

These Slides: <http://pres.derickrethans.nl>

Xdebug site: <http://xdebug.derickrethans.nl>

VLD site: <http://www.derickrethans.nl/vld.php>

PHP: <http://www.php.net>

Zend: <http://www.zend.com>

ionCube: <http://www.ioncube.com>

Index

Introduction	2
Parsing	3
Tokens	4
Tokenizer Example 1	5
Tokenizer Example 2	6
Compiling	7
Oparrays / Opcode	8
Example	9
Disassembling 1	10
Disassembling 2	11
Optimizing 1	12
Optimizing 2	13
Optimizing 3	14
Optimizing 4	15
Optimizing 5	16
Executing 1	17
Executing 2	18
Executing 3	19
Xdebug	20
Engine hooks	21
zend_execute hook	22
myOptimizer 1	23
myOptimizer 2	24
myOptimizer 3	25
myOptimizer 4	26
Internal functions	27
Extended Information 1	28
Extended Information 2	29
Gathering debug data 1	30
Gathering debug data 2	31
Gathering debug data 3	32
Xdebug Interfaces	33
Error handler	34
Function traces	35
Remote interfaces	37
Xdebug handlers	38
GDB Handler 1	39
GDB Handler 2	40
Resources	41