Neuheiten in PHP 5.3

– Namespaces, Lambda Functions, Closures, Functors –
und wie sie in der Engine implementiert sind

Neuheiten in PHP 5.3

- lambda functions and closures
- "jump label" operator (limited "goto")
- NOWDOC syntax, HEREDOC syntax with double quotes and for statics/ constants
- "?:" operator (ternary operator w/o 2nd operand)
- support for namespaces
- support for Late Static Binding
- support for __callStatic() magic method
- ability to handle exceptions in destructors
- optional garbage collection for cyclic references
- ability to use stream wrappers in include_path
- SPL cannot be disabled
- new extensions fileinfo, intl, mysqlnd, phar, SQLite3
- new functions array_replace() and array_replace_recursive()
- mail logging functionality that allows logging of mail sent via mail() function
- …

Quelle: http://php.net/ChangeLog-5.php#5.3.0

Neuheiten in PHP 5.3

- lambda functions and closures
- "jump label" operator (limited "goto")
- NOWDOC syntax, HEREDOC syntax with double quotes and for statics/ constants
- "?:" operator (ternary operator w/o 2nd operand)
- support for namespaces
- support for Late Static Binding
- support for __callStatic() magic method
- ability to handle exceptions in destructors
- optional garbage collection for cyclic references
- ability to use stream wrappers in include_path
- SPL cannot be disabled
- new extensions fileinfo, intl, mysqlnd, phar, SQLite3
- new functions array_replace() and array_replace_recursive()
- mail logging functionality that allows logging of mail sent via mail() function
- …

Quelle: http://php.net/ChangeLog-5.php#5.3.0

Neuheiten in PHP 5.3

1. lambda functions and closures
2. goto
3. namespaces
4. Late Static Binding
5. garbage collection for cyclic references
6. stream wrappers in include_path
7. mail logging functionality

lambda functions and closures

- LF = anonyme Funktion nach Lambda-Kalkül
- create_function() erzeugt auch anon. Funktion
  - als String … schlecht für IDE
  - Parametrisierung eingeschränkt
- Closure = LF in Bezug auf den Parent Scope
  - jede Funktion hat einen Parent Scope
    => jede LF ist ein Closure
- in ZE sind LF Objekte vom Typ „Closure“
  - Typ ist leider vernagelt, keine Stunts möglich

lambda functions and closures

```php
$a = array_filter(range(1,5), function ($v) { return ($v & 1); } );
// => array(1, 3, 5)

$odd = function ($v) { return ($v & 1); };
$a = array_filter(range(1,5), $odd);
// => array(1, 3, 5)

var_dump($odd);
/* =>
   object(Closure)#1 (1) {
     ["parameter"]=>
     array(1) {
       ["$v"]=>
       string(10) "<required>"
     }
   }
*/
```

lambda functions and closures

```
void zend_do_begin_lambda_function_declaration
(
     znode *result, znode *function_token, int return_reference TSRMLS_DC
)
{
     znode           function_name;
     zend_op_array *current_op_array = CG(active_op_array);
     int             current_op_number = get_next_op_number(CG(active_op_array));
     zend_op        *current_op;

     function_name.op_type = IS_CONST;
     ZVAL_STRINGL(&function_name.u.constant, "{closure}", sizeof("{closure}")-1, 1);

     zend_do_begin_function_declaration(function_token, &function_name, 0, return_reference, NULL
                                                                                    TSRMLS_CC);

     result->op_type = IS_TMP_VAR;
     result->u.var = get_temporary_variable(current_op_array);

     current_op = &current_op_array->opcodes[current_op_number];
     current_op->opcode = ZEND_DECLARE_LAMBDA_FUNCTION;
     zval_dtor(&current_op->op2.u.constant);
     ZVAL_LONG(&current_op->op2.u.constant, zend_hash_func(
          Z_STRVAL(current_op->op1.u.constant), Z_STRLEN(current_op->op1.u.constant)));
     current_op->result = *result;
     CG(active_op_array)->fn_flags |= ZEND_ACC_CLOSURE;
}
```

goto

- „jump label operator“ ist kein Operator
  sondern eine Kontrollstruktur
- Label muss ein T_STRING sein
- Ziel muss in der selben Datei/Funktion/Methode sein
- reinspringen verboten, rausspringen erlaubt

```
// wie break
for (...) {
  while (...) {
    goto outer;
  }
}
outer:
```

```
// Fatal error 'goto' into loop or
// switch statement is disallowed
goto inner;
for (...) {
  while (...) {
    inner:
  }
}
```

goto

- Relikt aus sequentiellen, imperativen Sprachen – „jmp“ in ASM X68
- in OOP Codefluss schwer nachvollziehbar
- kein Mehrwert – jedes Flußdiagramm kann in eines ohne Sprünge transformiert werden

„the quality of programmers
is a decreasing function
of the density of **go to** statements
in the programs they produce“

Quelle: Edsger W Dijkstra, *Communications of the ACM*, Vol. 11, No. 3, March 1968, pp. 147-148

namespaces

- Konzept bekannt aus z.B. Java
- vollständig rückwärtskompatibel, in ZE nur kleine Erweiterung der Symboltabellen
- hierarchisch, \NS\SubNS\SubSubNS
- verhindert Namenskollisionen zwischen eigenem Code, PHP-Internals, 3rd-party Code
  - lesbarer als Framework_Package_Class
- ein NS kann in mehreren Files definiert sein
- ein File kann mehrere NS definieren (don‘t!)

Von allen neuen Features werden wir dieses am häufigsten benutzen.

namespaces

```php
// before any* other code or output in the file
namespace My\Name { /* … */ }
namespace My\Name;

// namespaceable elements
class MyClass {}
function myFunction() {}
const MYCONST = 1; // not applicable to NULL, TRUE, FALSE

// usage of namespaced elements / namespace resolution
new MyClass(); // while in NS My\Name
new Name\MyClass(); // while in NS My
new \My\Name\MyClass(); // anywhere, leading \ references "Global Space"
echo strlen('hi'); // fallback to global function/constant
new \Exception('error'); // no fallback for classes
new Exception('error'); // attempt to autoload My\Name\Exception

// namespace operator and __NAMESPACE__ constant
$d = namespace\MYCONST; // equivalent to self operator for classes
$d = __NAMESPACE__ . '\MYCONST'; // equivalent to __CLASS__ constant
echo constant($d);

namespace { // global code, no need for leading \
    new My\Name\MyClass();
    new Exception(strlen('error'));
}
```

* außer declare(), z.B. declare(encoding='UTF-8'); jedoch nicht die Block-Syntax declare(){}
… jetzt ist BOM nicht nur für session_start() ein Killer

namespaces

```php
namespace foo;

// importing and aliasing
use My\Full\Classname as Another;
use My\Full\NSname; // the same as use My\Full\NSname as NSname
use \ArrayObject; // importing a global class

new namespace\Another(); // instantiates object of class foo\Another
new Another(); // instantiates object of class My\Full\Classname
NSname\subns\func(); // calls function My\Full\NSname\subns\func
new ArrayObject(array(1)); // instantiates object of class ArrayObject
// without "use \ArrayObject" we would instantiate from foo\ArrayObject
```

Late Static Binding

```php
class Base {
    public static function identify() {
        self::printName();
        static::printName();
    }
    static function printName() {
        echo __CLASS__; // equivalent to get_called_class()
    }
}

Base::identify(); // => Base Base

class Extended extends Base {
    static function printName() {
        echo __CLASS__;
    }
}

Extended::identify(); // => Base Extended
```

- ZE reicht bei Fallback (Methode existiert nur in Parent) Infos über den Aufruf durch
- static:: und get_called_class() nutzen die Infos

garbage collection

for cyclic references

- Refcounting zählt wie viele Symbole auf ein ZVAL zeigen
- unset($a) => ref_count--
  - ref_count == 0 => free
  - ref_count > 0 => Eintrag in Roots Table
- bei 10.000 Roots wird der GC aktiv
- löscht alle ZVALs für die ref_count-1=0 gilt
- Algorithmus basiert auf IBM-Paper*
- Performance -7%, Memory Usage -x%
- gc_enable(), gc_disable(), gc_collect_cycles()

* http://www.research.ibm.com/people/d/dfb/papers/Bacon01Concurrent.pdf

stream wrappers

in include_path

- phar:// ist dank ext/phar schon drin
- db://, s3://, eval://, remoteWithSocket://, …

```
stream_wrapper_register('db', 'DbStreamWrapper');
set_include_path('db://username:password@host/database');

include 'db://table/column?'. $id;
```

```
stream_wrapper_register('eval', 'EvalStreamWrapper');
set_include_path('eval://');

$somecode = 'echo "foo";');
include 'eval://somecode';
```

- ZE benutzt Stream Wrapper sehr „schräg“

mail logging

- neue ini-Direktive mail.log
- default Null
- PHP_INI_SYSTEM | PHP_INI_PERDIR
- loggt Datei, Zeile, $to und $headers
- erlaubt Hostern proaktiven Schutz von mail()