Was ist neu in *PHP 7.1*

PHP User Group Rheinhessen 44

Matthias Gutjahr (@mattsches)

Zeitplan

Release Zeitplan: https://wiki.php.net/todo/php71

Aktuell: PHP 7.1 Release Candidate 3 (Stand: 18.10.2016)

Geplantes Release-Datum: Ende November 2016

Fehlerbehandlung 1/2

Catching Multiple Exception Types

```php
class Foo extends Exception {}
class Bar extends Exception {}
class Baz extends Exception {}
```

```php
try {
    switch (mt_rand(0, 2)) {
        case 0: throw new Foo();
        case 1: throw new Bar();
        case 2: throw new Baz();
    }
}
catch (Bar | Baz $e) {
    printf("1er catch : %s\n", get_class($e));
}
catch (Foo $e) {
    printf("2nd catch : %s\n", get_class($e));
}
```

Fehlerbehandlung 2/2

Throw Error in Extensions

```php
<?php
// 'timezone-type' instead of 'timezone_type'
$serialized = 'O:8:"DateTime":3:{s:4:"date";s:26: \
"2016-08-14 12:31:50.000000";s:13:"timezone-type"; \
i:3;s:8:"timezone";s:3:"UTC";}';
try {
    $dt = unserialize($serialized);
    var_dump($dt);
} catch (Error $er) {
    echo 'Gotcha!';
}
```

< 7.1:

```
Fatal error: Invalid serialization data for DateTime \
object in âŚ on line 5

Process exited with code 255.
```

Typen 1/3

Nullable types

```
function my_function(?int $a) {
    var_dump($a);
}
my_function(100); // int(100)
my_function(null); // NULL
my_function();
// Uncaught Error: Too few arguments
// to function my_function(), 0 passed
```

```
function my_function(?int $a, ?int $b) : ?int {
    if ($a === null || $b === null) {
        return null;
    }
    return $a + $b;
}
var_dump(my_function(10, 20)); // int(30)
var_dump(my_function(10, null)); // NULL
```

Typen 2/3

Void Return Type

```
function returns_nothing() : void
{
    // This function returns null
    return;
}

function does_not_return() : void
{
    // This function returns null, too
}
```

```
function returns_a_value() : void
{
    return 42;
}
// Fatal error: A void function must not return a value
```

Typen 3/3

Iterable

```php
function my_function(iterable $data) {
    foreach ($data as $key => $val) {
        var_dump($val);
    }
}
my_function([10, 20, 30]);
my_function(new SplFixedArray(5));
// generators:
function my_generator() {
    yield 100;
    yield 200;
    yield 300;
}
my_function(my_generator());

my_function('foo');
// TypeError: Argument 1 passed to my_function() must
// be iterable, string given
```

Syntax 1/4

Allow specifying keys in `list()`

```php
$array = [
    'foo' => "Hello",
    'bar' => 123456,
    'baz' => "World",
];

list (
    'foo' => $a,
    'baz' => $b
) = $array;

var_dump($a, $b);
// string(5) "Hello"
// string(5) "World"
```

Syntax 2/4

Square bracket syntax for array destructuring assignment

```
$array = [10, 20, 30];
[$a, $b, $c] = $array;
var_dump($a, $b, $c);
// int(10)
// int(20)
// int(30)
```

Syntax 3/4

Generalize support of negative string offsets

```
$str = "Hamburg";
var_dump($str[2]);  // string(1) "m"
var_dump($str[-2]); // string(1) "r"
```

```
$str = "Ham.urg";
$str[-4] = 'b';
var_dump($str); // string(7) "Hamburg"
```

Syntax 4/4

Support Class Constant Visibility

```php
class MyClass {
    public const MY_PUBLIC = 42;
    private const MY_PRIVATE = 1234;
    public function test() {
        var_dump( self::MY_PRIVATE );
    }
}
$obj = new MyClass();
$obj->test(); // int(1234)
var_dump(MyClass::MY_PUBLIC); // int(42)
var_dump(MyClass::MY_PRIVATE);
// Fatal error: Uncaught Error: Cannot access
// private const MyClass::MY_PRIVATE
```

đ 3v4l.org

Closure from callable function

```php
function my_function() {
    var_dump(__FUNCTION__);
}

$closure = Closure::fromCallable('my_function');
$closure();  // string(11) "my_function"

$closure = Closure::fromCallable('foo');
// TypeError: Failed to create closure from callable:
// function 'foo' not found or invalid function name

// also possible for methods:
$callable = Closure::fromCallable([
    new MyClass(), 'myMethod'
]);
```

Warn about invalid strings in arithmetic

Things like

```
var_dump('10 apples' + '5 oranges');
// int(15)
```

still work, but will issue a `E_NOTICE: Notice: A non well formed numeric value encountered`. The same goes for strings like in

```
var_dump(10 + "plop");
// int(10)
```

which will generate an `E_NOTICE: Warning: A non-numeric value encountered`.

Fix inconsistent behavior of `$this` variable

This resulted in `string(3) "foo"` up to PHP 7.0.

```
class MyClass
{
    public function foo()
    {
        $var = 'this';
        $$var = 'foo';
        var_dump($this);
    }
}
$obj = new MyClass();
$obj->foo();
```

In PHP 7.1, it throws a `Fatal error: Uncaught Error: Cannot re-assign $this in âŚ`.

Replace "Missing argument" warning with "Too few arguments" exception

```
function my_function($a, $b)
{
    var_dump($a, $b);
}
my_function(10);
```

< 7.1:

```
Warning: Missing argument 2 for my_function(), called

Notice: Undefined variable: b in âŚ on line 4
int(10)
NULL
```

7.1:

```
Fatal error: Uncaught ArgumentCountError: Too few
arguments to function my_function(), 1 passed in âŚ
on line 7 and exactly 2 expected in âŚ
```

Forbid dynamic calls to scope introspection functions

```
function my_function()
{
    $vars = ['a' => 123];
    $func = 'extract';
    call_user_func($func, $vars);
    var_dump($a);
}
my_function();
```

<7.1:

```
int(123)
```

7.1:

```
Warning: Cannot call extract() dynamically in âŚ on line
Notice: Undefined variable: a in âŚ on line 9
NULL
```

Hinweis

Die meisten Beispiele stammen aus Pascal Martins exzellenter Artikelserie über PHP 7.1: https://blog.pascal-martin.fr/post/php71-en-introduction-and-release-cycle.html

Die Artikel enthalten mehr und detailliertere Informationen.

Danke!