

PHP GeekMeet #2 Cluj

Ce e nou in PHP 5.3?

New Features in PHP 5.3

- **Late static binding**
- **Functii Lambda si Closures**
- **Namespaces**
- **Eliberarea memoriei ocupate de referinte ciclice**
- **NOWDOC, __callStatic()**
- **mysqlnd – MySQL Native Driver**
- **Imbunatatiri de performanta**
- **Multe bugfix-uri si imbunatatiri**

Late Static Binding

```
<?php
class A {
    public static function class_name() {
        echo __CLASS__;
    }
    public static function test() {
        self::class_name();
    }
}

class B extends A {

}

A::test();
B::test();
?>
```

Late Static Binding

```
<?php
class A {
    public static function class_name() {
        echo __CLASS__;
    }
    public static function test() {
        self::class_name();
    }
}

class B extends A {
    public static function class_name() {
        echo __CLASS__;
    }
}

A::test();
B::test();
?>
```

Late Static Binding

- referinta statica **self**:: face defapt referire la clasa in care a fost definita metoda in care sunt folosite
- *late static binding* foloseste keyword-ul rezervat *static* ca referinta pentru clasa din care a fost apelata functia in contextul unei mosteniri statice.
- “Late binding” - “asociere tarzie” (referinta la runtime), “Static binding” folosirea in metodele statice.

Late Static Binding

```
<?php
class A {
    public static function class_name() {
        echo __CLASS__;
    }
    public static function test() {
        static::class_name();
    }
}

class B extends A {
    public static function class_name() {
        echo __CLASS__;
    }
}

A::test();
B::test();
?>
```

Funcții Lambda și “Closures”

- Funcții anonime “throwaway”
- C++, C#, python, javascript etc
- Compilate la runtime (nu pot fi cache-uite)

Funcții Lambda

- Funcție anonimă
- Pot trai mai mult decât ceea ce le-a creat.
- Utile în funcțiile care accepta ca parametru un “callback” (ex. `array_filter`)

Funcții Lambda și “Closures”

```
<?php  
function impar($var)  
{  
    return($var & 1);  
}
```

```
$sir_numere = array(6, 7, 8, 9, 10, 11, 12);
```

```
echo "Impare :\n";  
print_r(array_filter($sir_numere, "impar"));  
?>
```

```
<?php
```

```
$sir_numere = array(6, 7, 8, 9, 10, 11, 12);
```

```
echo "Impare :\n";  
print_r(array_filter($sir_numere, function ($var){ return($var & 1);}));  
?>
```

Functii Lambda si “Closures”

Closures

- o functie lambda ”constienta” de context. Are abilitatea sa “importe” variabilele din “scop-ul” parinte.
- Variabilele importante din mediul extern sunt specificate prin intermediul clauzei **use**.
- Variabilele sunt transmise prin valoare. Pentru a 'pasa' variabilele prin referinta se poate folosi operatorul **&**.
- din versiunea PHP 5.3 beta1 functiile closure nu au acces implicit la **\$this** si nu pot fi declarate static.

Funcții Lambda și “Closures”

```
<?php
$x = 1;

$lambda1 = function () use ($x) {
    $x *= 2;
};
$lambda2 = function () use (&$x) {
    $x *= 3;
};

$lambda1 ();
echo $x; // 1

$lambda2 ();
echo $x; // 3
?>
```

Namespaces

Cei care lucreaza la aplicatii/librarii de dimensiuni mai mari se bat de problema conflictelor intre denumiri fie ele interne sau cu librarii third-party, de aceea trebuie sa recurga la denumiri lungi.

- Cum se numeste aplicatia mea? **BestECommerceShopEver**.
- Din ce modul face parte clasa (ex. **Customer**, **Checkout**)? **Customer**.
- Model/View/Controller? **Model**.
- Ce face practic clasa? **Taxeaza clientul (Billing)**.

BestECommerceShopEver_Customer_Model_Billing. Not very nice...

De ce am denumit asa clasa? Pentru ca e posibil sa folosim o clasa **Billing** ca si in modulul **Customer controller**, o clasa **Billing** in modulul ce se ocupa cu distribuitorii etc.

Namespaces

- **Conditia este sa fie declarate in namespace-uri diferite. Astfel se poate reveni la denumirile scurte ale claselor, imbunatatind lizibilitatea codului.**
- **Container abstract pentru organizarea unei grupari logice, o metoda de a encapsula elementele inrudite (asociate), grupand clase, functii, constante.**
- **Sunt afectate doar denumirile de clase, functii si constantele.**
- **Acelasi identificator poate fi definit independent in namespace-uri diferite dar fara a avea neaparat acelasi sens peste tot. Aceleasi clase/functii/constante in namespace-uri diferite si avand roluri total diferite.**

Namespaces

- Se foloseste keyword-ul 'namespace'.
- Declararea se face la inceputul fisierului (Exceptie 'declare').
- acelasi namespace → "intins" pe mai multe fisiere.

```
<html> <!-- fatal error -->
<?php
namespace
BestECommerceShopEver\Customer\Model;

const BILLED_OK = 1;
class Billing { /* ... */ }
function bill() { /* ... */ }

?>
```

Namespaces

- Mai multe namespaces in acelasi fisier

```
<?php
```

```
namespace BestECommerceShopEver\Customer\Model {
```

```
    class Billing { /* ... */ }  
    function get_bill_value() { /* ... */ }
```

```
}
```

```
namespace BestECommerceShopEver\Customer\Controller {
```

```
    class Billing { /* ... */ }  
    function perform_bill() { /* ... */ }
```

```
}
```

```
namespace { // global code
```

```
    const BILLED_OK = 1;
```

```
    $bill_result =
```

```
BestECommerceShopEver\Customer\Controller\Billing::perform_bill();
```

```
    if ($bill_result == BILLED_OK) {
```

```
        echo 'Success!';
```

```
    }
```

```
}
```

```
?>
```

Namespaces

- **Analogie intre namespaces si un sistem de fisiere**
- **3 Moduri de a referi o clasa folosind namespaces:**
 - 1. denumirea clasei neprefixata: `$a = new foo();`
daca codul se afla intr-un namespace: `currentnamespace\foo`.
Daca codul e global, (non-namespaced code): `foo`.
 - 2. denumirea clasei e prefixata: `$a = new namespace\foo();`
`currentnamespace\namespace\foo`. sau `namespace\foo` (daca codul e global).
 - 3. denumirea clasei e prefixata cu un operator "prefix global":
`$a = new \currentnamespace\foo();`
`currentnamespace\foo`

Namespaces

```
<?php
namespace foo;
use My\Full\Classname as Another;

// this is the same as use My\Full\NSname as NSname
use My\Full\NSname;

// importing a global class
use \ArrayObject;

$obj = new namespace\Another; // instantiates object of class
foo\Another
$obj = new Another; // instantiates object of class
My\Full\Classname
NSname\subns\func(); // calls function
My\Full\NSname\subns\func
$a = new ArrayObject(array(1)); // instantiates object of class
ArrayObject
// without the "use \ArrayObject" we would instantiate an
object of class foo\ArrayObject
?>
```

Eliberarea memoriei ocupate de referinte ciclice

```
<?php
class Node {
    public $parentNode;

    public $childNodes = array();

    function Node() {
        $this->nodeValue = str_repeat('0123456789', 128);
    }
}

function createRelationship() {

    $parent = new Node();    $child = new Node();

    $parent->childNodes[] = $child;
    $child->parentNode = $parent;
}

for($i = 0; $i < 10000; $i++) {
    createRelationship();
}

//Initial: 62,224 bytes  Peak: 34,905,216 bytes  End: 34,905,016 bytes
?>
```

NOWDOC

- **HEREDOC:**

```
<?php  
$name = 'MyName';  
  
echo <<<EOT  
My name is "$name".  
EOT;  
?>
```

- **NOWDOC:**

```
<?php  
$name = 'MyName';  
  
echo <<<'EOT'  
My name is "$name".  
EOT;  
?>
```

__callStatic()

- Magic method (__get, __set, __call)

```
<?php
class MethodTest {
    public function __call($name, $arguments) {
        echo "Metoda '$name' nu exista. Apel: " . implode(', ', $arguments);
    }

    public static function __callStatic($name, $arguments) {
        echo "Metoda '$name' nu exista. Apel: " . implode(', ', $arguments);
    }
}
```

```
$obj = new MethodTest;
$obj->runTest('obiect');// Metoda 'runTest' nu exista. Apel: obiect.
```

```
MethodTest::runTest('static');// Metoda 'runTest' nu exista. Apel: static.
```

```
?>
```

mysqlnd – MySQL Native Driver

- **mysqlnd & libmysql: C-libraries, MySQL communication protocol**
- **sub licenta PHP, nu mai e nevoie de FLOSS Exception**
- **Integrat in PHP**
- **ext/mysql & ext/mysqli pot folosi optional mysqlnd. PDO/MySQL pe viitor.**
- **Nativ: foloseste PHP memory management, suporta limita de memorie din PHP**
- **mysqli_fetch_all()**
- **Statistici pentru analiza performantei: mysqli_get_cache_stats(), mysqli_get_client_stats(), mysqli_get_connection_stats()**
- **Performanta: uneori e mai rapida decat libmysql**
- **Performanta: (pe viitor) caching client-side pt setul de rezultate**

Imbunatatiri de performanta

- Sebastian Bergmann (PHPUnit, framework-ul de unit testing pentru PHP)
 - PHP 5.3 e de 1.2 ori mai rapid decat PHP 5.2 la viteza de executie bruta dupa ce a facut o medie intre mai multe teste.

- cei de la Doctrine (ORM) au facut un benchmark pentru suita lor de teste folosind PHP 5.3 si PHP 5.2.8
 - testele ruleaza cu 17% mai rapid
 - folosesc cu 31% mai putina memorie