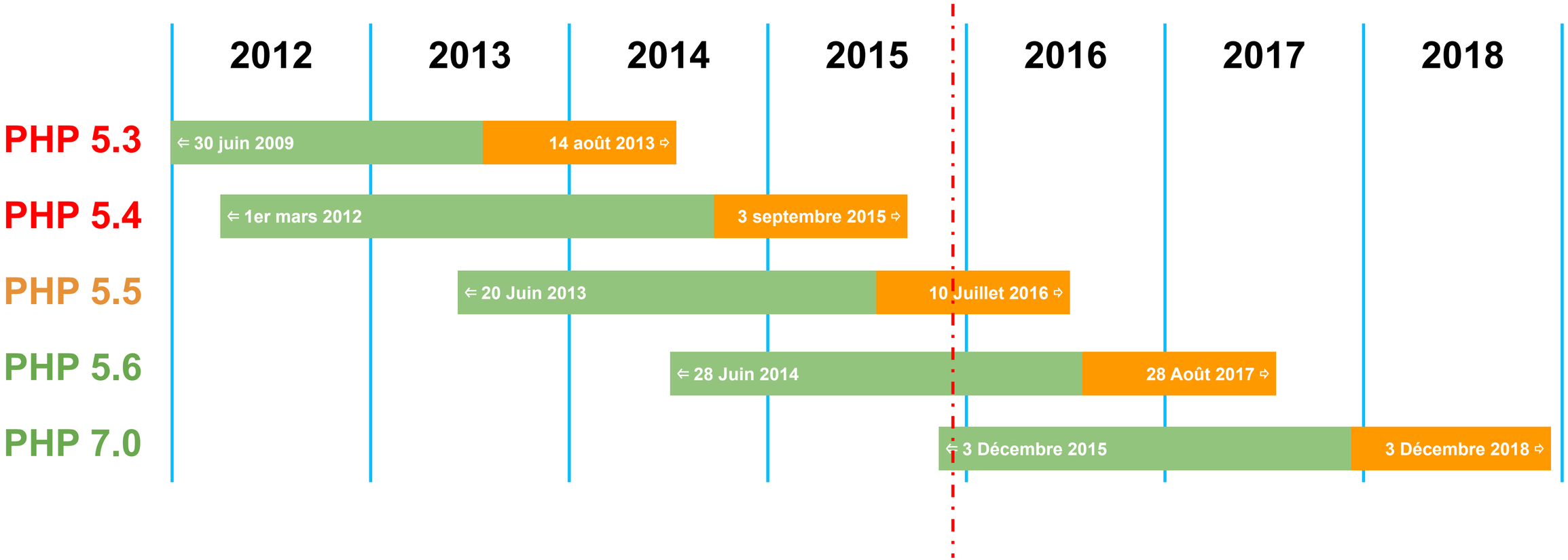
NUGLIF

Les améliorations de

PHP >= 5.4

par Pierrick Charron

Les versions de PHP

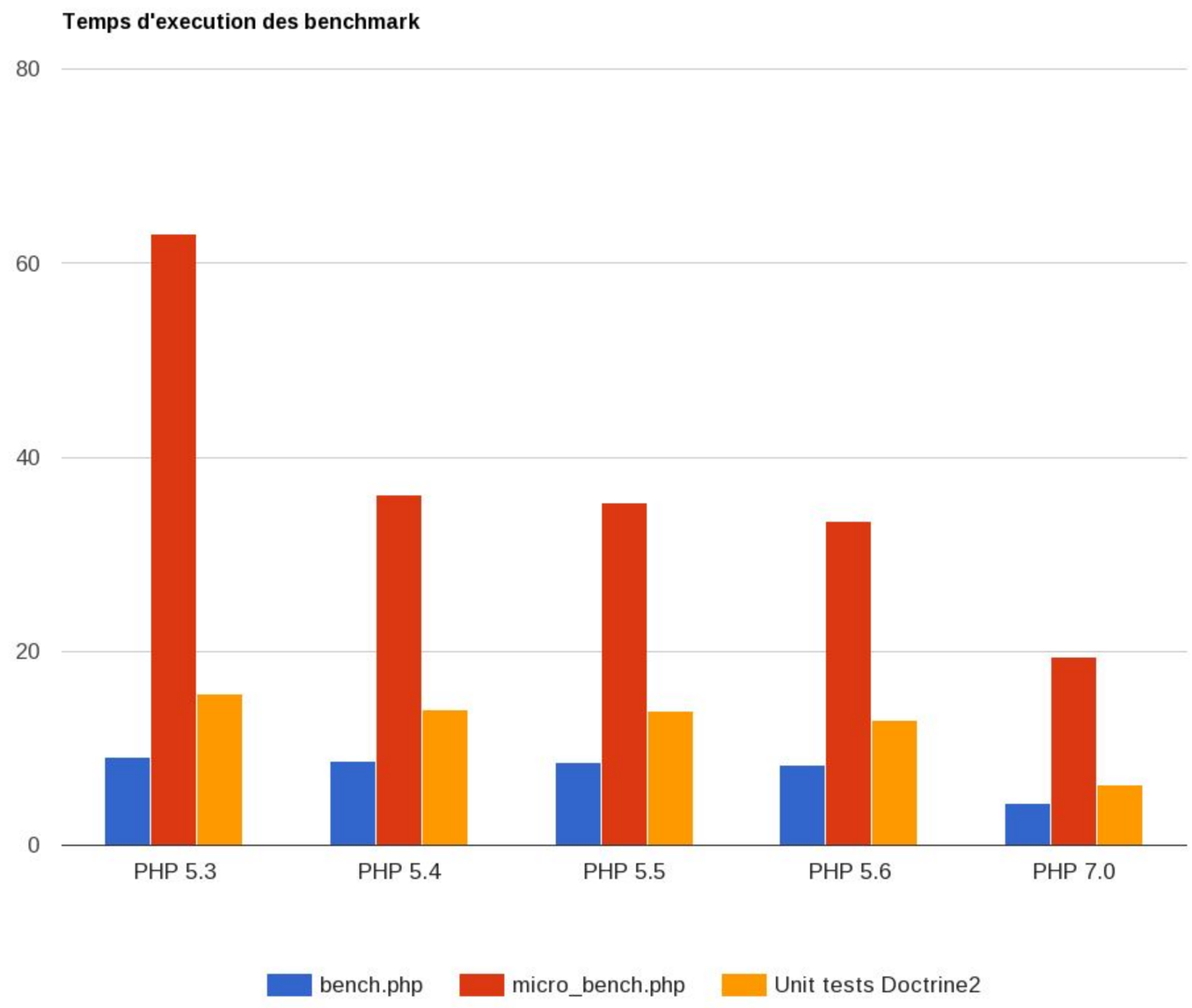


Version actuelle : améliorations et correctifs de sécurité

Ancienne version : correctifs de sécurité uniquement

Ancienne version : non supportée

Performances Générales



nuGLIF

PHP 5.4

1er mars 2012

Traits

- unité de réutilisation de comportement
- ne s’instancie pas
- static
- permet la réutilisation de code sans affecter le domaine de l’objet
- copy paste assisté par le langage
- peut être composé d’un ou plusieurs autres traits
- peut définir ses requis en définissant des méthodes abstraites

Traits

```php
trait A {
  public function smallTalk() {
    echo 'a';
  }
  public function bigTalk() {
    echo 'A';
  }
}

trait B {
  public function smallTalk() {
    echo 'b';
  }
  public function bigTalk() {
    echo 'B';
  }
}
```

```php
/* Utiliser un trait */
class Talker {
    use A;
}

(new Talker)->smallTalk(); /* a */
```

```php
/* Résolution de conflits, renommage et
changement de visibilité */
class Talker {
   use A, B {
     B::smallTalk insteadof A;
     A::bigTalk insteadof B;
     B::bigTalk as public talk;
   }
}

(new Talker)->smallTalk(); /* b */
(new Talker)->bigTalk(); /* A */
(new Talker)->talk(); /* B */
```

Traits : Cas d'utilisation

```
$ grep -Porh "public function set([^(]+)" | sort | uniq -c | sort -nr | head -1
    153 public function setServices
```

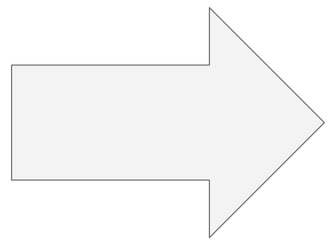
```php
trait ServicesAware
{
    private $services;
    public function setServices(\LPRI\Service\Services $services)
    {
        $this->services = $services;
        return $this;
    }
}
```

```php
class X {
    use ServicesAware;
    private function doSomething()
    {
        $services = $this->services;
        /* ... */
    }
}
```

Short array syntax

➢ Remplacement de `array()` par `[]`

```php
<?php
/* Old array syntax */
$array = array(
    array(
        '1.1' => 'foo',
        '1.2' => 'bar',
    ), array(
        '2.1' => 'baz'
    )
);
```

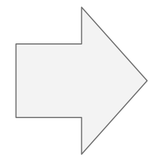


```php
<?php
/* new array syntax */
$array = [
    [
        '1.1' => 'foo',
        '1.2' => 'bar',
    ],[
        '2.1' => 'baz'
    ]
];
```

Function array dereferencing

```php
function fruit () {
  return array('a' => 'apple', 'b' => 'banana');
}
```

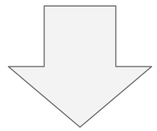
```php
$fruits = fruit();
echo $fruits['a']; // apple
```



```php
echo fruit()['a']; // apple
```

Accès aux membres de classe à l'instanciation

```php
$editionCreator = new \LPRI\Workflow\Edition\Creator();
$editionCreator->setConfiguration($container->get('config'));
```



```php
/* Les parenthèses sont obligatoires */
$editionCreator = (new \LPRI\Workflow\Edition\Creator())
    ->setConfiguration($container->get('config'));
```

Le type `callable`

```php
function callFunction(callable $function) {
    return $function();
}
```

Notation binaire

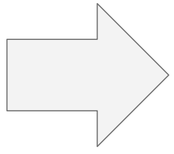
```php
const BITFIELD_A = 0b0001; /* 1 */
const BITFIELD_B = 0b0010; /* 2 */
const BITFIELD_C = 0b0100; /* 4 */
const BITFIELD_D = 0b1000; /* 8 */
```

Echo Short open tag toujours activé

```
Bonjour <?= $name ?>
```

Accès au `$this` dans les closures

```
private function initUseCaseFactory()
{
  $container       = $this->container;
  $clientContainer = $this->clientContainer;
  $this->container->set(
    'useCaseFactory',
    function () use ($container, $clientContainer)
    {
      return new UseCases\Factory(
        $container,
        $clientContainer
      );
    }
  );
}
```



```
private function initUseCaseFactory()
{
  $this->container->set(
    'useCaseFactory',
    function () {
      return new UseCases\Factory(
        $this->container,
        $this->clientContainer
      );
    }
  );
}
```

Support de `Classe::{expr}();`

```php
ClassName::{strtolower($method)}();
```

L'interface `JsonSerializable`

```php
class Foo implements JsonSerializable {
    private $name;
    public function __construct($name) {
        $this->name = $name;
    }
    public function jsonSerialize() {
        return [ $this->name ];
    }
}

echo json_encode(new Foo('Pierrick')); // ["Pierrick"]
```

Autres

- Serveur HTTP built-in pour le dev
  $ `php -s localhost:8080 -t docroot`

- Suppression de y2k_compliance
  - Ne fait rien à part nous faire sentir plus sur

- Suppression des magic_quotes

- Suppression du call time pass by reference :
  ```
  callToFunction(&$byRef);
  ```

Performances

	PHP 5.3	PHP 5.4	PHP 5.5	PHP 5.6	PHP 7.0
bench.php	*9.121*	8.636			
		5%			
micro_bench.php	63.086	36.131			
		42%			
Unit tests Doctrine2	15.58	14.01			
		10%			

nuGLIF

PHP 5.5

20 juin 2013

Les générateurs

```php
function getLinesFromFile($fileName) {
  $fileHandle = fopen($fileName, 'r');
  $lines = [];
  while ($line = fgets($fileHandle)) {
    $lines[] = $line;
  }
  fclose($fileHandle);
  return $lines;
}
```

```php
class LineIterator implements Iterator {
  public function __construct($fileName) {}
  public function rewind() {}
  public function valid() {}
  public function current() {}
  public function key() {}
  public function next() {}
  public function __destruct() {}
}
```

```php
function getLinesFromFile($fileName) {
  $fileHandle = fopen($fileName, 'r');
  while ($line = fgets($fileHandle)) {
    yield $line;
  }
  fclose($fileHandle);
}
```

Les générateurs

- Les générateurs sont des fonctions interruptibles
- Le `yield` est le point d'interruption

```php
function getLinesFromFile($fileName) {
  $fileHandle = fopen($fileName, 'r');
  while ($line = fgets($fileHandle)) {
    yield $line;
  }
  fclose($fileHandle);
}
```

```php
$lines = getLinesFromFile('file.txt');
var_dump($lines); // object(Generator)#1
var_dump($lines instanceof Iterator); // bool(true)

foreach ($lines as $line) {
    echo $line; // line content 1 by 1
}
```

Les générateurs

➢ Simplement créer un Iterateur typé via un générateur

```php
class LineIterator implements IteratorAggregate {
  private $fn;
  public function __construct($fileName) {
    $this->fn = $fileName;
  }
  public function getIterator() {
     $fileHandle = fopen($this->fn, 'r');
     while ($line = fgets($fileHandle)) {
       yield $line;
     }
     fclose($fileHandle);
  }
}

foreach (new LineIterator('file.txt') as $line) {
  echo $line;
}
```

Les coroutines

- Il est aussi possible d'injecter de la donnée au générateur
- La communication devient alors bidirectionnelle
- Les valeurs sont passées via `->send()` plutot que `->next()`;
- Le `yield` devient alors une expression plutôt qu'un statement et a pour valeur ce qui est passé dans `->send()`

```php
function logger($fileName) {
    $fileHandle = fopen($fileName, 'r');
    while (true) {
        fwrite($fileHandle, yield . PHP_EOL);
    }
}

$logger = logger('log.txt');
$logger->send('First line');
$logger->send('Second line line');
```

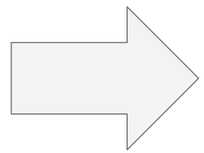
Les coroutines

➢ Il est possible de combiner l'envoi et la réception de données

```php
function gen() {
    $ret = (yield 'yield1');
    var_dump($ret);
    $ret = (yield 'yield2');
    var_dump($ret);
}

$gen = gen();
var_dump($gen->current());     // string(6) "yield1"
var_dump($gen->send('ret1')); // string(4) "ret1" (Premier var_dump dans gen())
                               // string(6) "yield2" (Valeur de retour de ->send())
var_dump($gen->send('ret2')); // string(4) "ret2" (Second var_dump dans gen())
                               // NULL (Valeur de retour de ->send())
```

Finally

```php
$db = mysqli_connect();
try {
    call_some_function($db);
} catch (Exception $e) {
    mysqli_close($db);
    throw $e;
}
mysql_close($db);
```

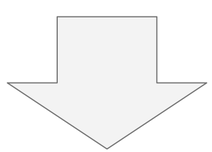


```php
$db = mysqli_connect();
try {
    call_some_function($db);
} catch (Exception $e) {
    throw $e;
} finally {
    mysqli_close($db);
}
```

➢ Le block `finally` sera toujours exécuté à la sortie du `try`

Résolution de classe (::class)

```php
use A\Namespaced\ClassName;

$mock = Phake::mock('A\Namespaced\ClassName');
```



```php
use A\Namespaced;
use A\Namespaced\ClassName;

$mock = Phake::mock(Namespaced\ClassName::class);
$mock = Phake::mock(ClassName::class);
```

➢ Résolution lors de la compilation

Const array/string dereference

```
echo array(1, 2, 3)[0];   // "1"
echo "foobar"[2];         // "o"
echo [1,3,4][2];          // "4"
```

empty({expr}) / isset({expr})

```
if (isset(foo() || bar())) {}
if (empty(foo() || bar())) {}
```

Support de list() dans les foreach

```php
$users = array(
    array('Foo', 'Bar'),
    array('Baz', 'Qux'),
);

// Before
foreach ($users as $user) {
    list($firstName, $lastName) = $user;
    echo "First name: $firstName, last name: $lastName. ";
}

// After
foreach ($users as list($firstName, $lastName)) {
    echo "First name: $firstName, last name: $lastName. ";
}
```

Autres

- Intégration de opcache (anciennement ZendOptimiser+)
- ext/mysql est maintenant obsolète
- Upload via curl avec la syntaxe @filename rendu obsolète
  - Remplacé par la classe `CURLFile`

Performances

	PHP 5.3	PHP 5.4	PHP 5.5	PHP 5.6	PHP 7.0
bench.php	*9.121*	8.636	8.584		
			1%		
micro_bench.php	63.086	36.131	35.355		
			2%		
Unit tests Doctrine2	15.58	14.01	13.82		
			1%		

nuGLIF

PHP 5.6

28 juin 2014

Expressions dans les constantes

```
const FOO = 1 + 1;

const BAR = 1 << 1;

const GREETING = "HELLO";

const BAZ = GREETING." WORLD!"
```

- Opérateurs supportés

```
+ - * / % ! ~ | & ^ << >> . <= >= == != < > === !==  && ||
```

- Opérandes supportées

```
1, 10.0, "foo", __LINE__, __FILE__, __DIR__, __TRAIT__,
__METHOD__, __FUNCTION__, __NAMESPACE__, CONST, class::CONST
```

Fonctions variadiques via ...

```php
function add(...$var) {
    $result = 0;
    foreach ($var as $value) {
        $result += $value;
    }
    return $result;
}
echo add(1, 2, 3); // 6
```

Dépillage des arguments via … (aka Splat)

```php
$var = [ 3 , 4 ];
echo add(1, ...$var); // similaire à add(1, 3, 4);
```

Cas d’utilisation combiné de ...

```php
class FileLogger {

    private $file;

    public function __construct($file) {
        $this->file = $file;
    }

    public function log($format, ...$args) {
        file_put_contents(
          $this->file,
          sprintf($format, ...$args),
          FILE_APPEND
        );
    }
}
```

use const et use function

```php
namespace A {
    const SOME_VALUE = 100;
    function someFunction() {
        var_dump(__FUNCTION__);
    }
}

namespace B {
    use const A\SOME_VALUE;
    use function A\someFunction;

    echo SOME_VALUE; // 100
    someFunction(); // A\someFunction
}
```

Autres

- Upload via curl avec la syntaxe @filename désactivé par défaut
  - Remplacé par la classe `CURLFile`
  - Alternative : utiliser `CURLOPT_SAFE_UPLOAD`

- Operateur exponentiel `**`

```php
echo 2 ** 3; // 8
```

- Méthode magique `__debugInfo`();

- Extension phpdbg

Performances

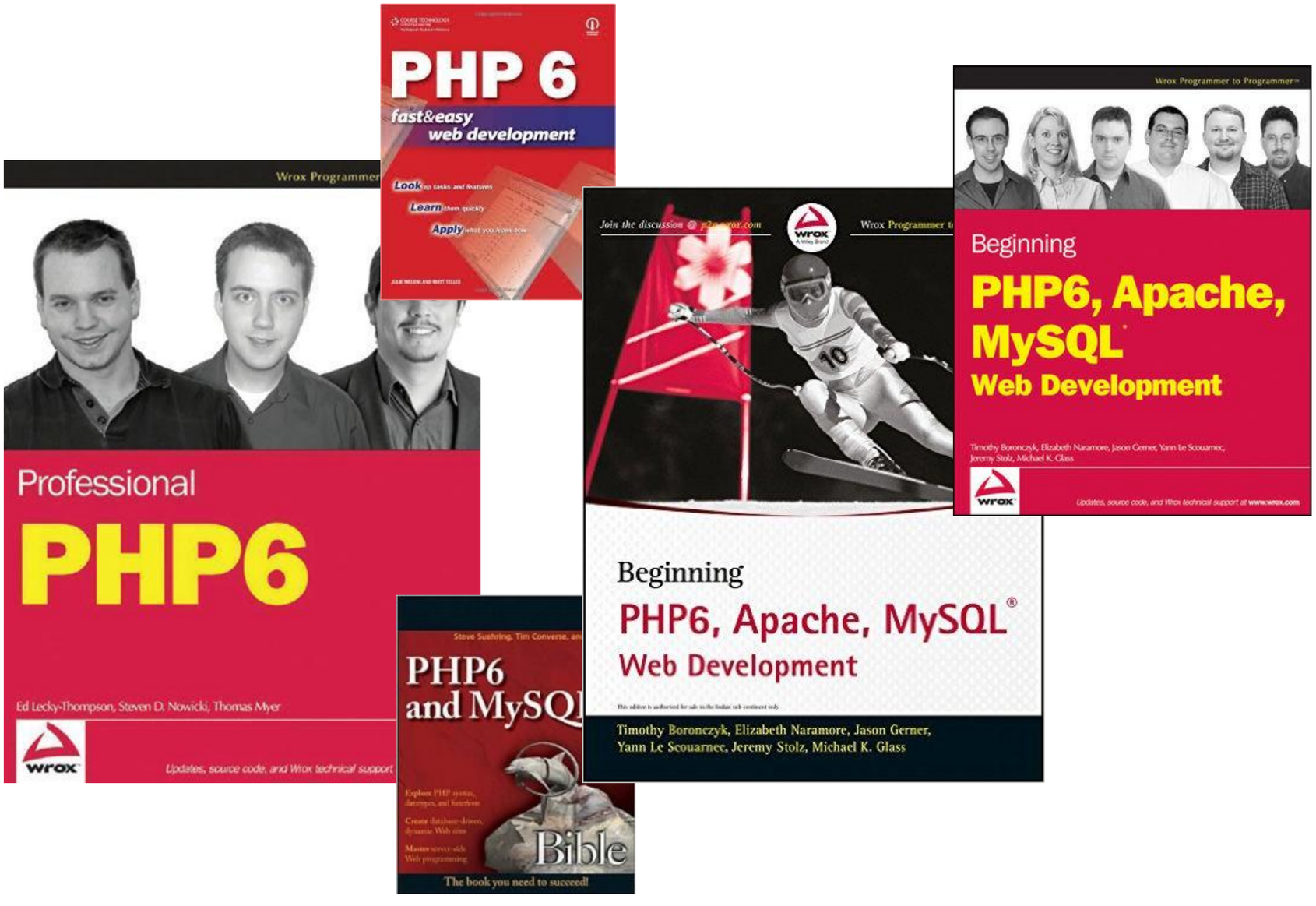
	PHP 5.3	PHP 5.4	PHP 5.5	PHP 5.6	PHP 7.0
bench.php	*9.121*	8.636	8.584	8.241	
				4%	
micro_bench.php	63.086	36.131	35.355	33.429	
				5%	
Unit tests Doctrine2	15.58	14.01	13.82	12.95	
				6%	

nuGLIF

PHP 6.0

Jamais releasée

Je vous recommande de lire... ou pas



nuGLIF

PHP 7.0

3 décembre 2015

Scalar type hinting

- Nouveaux types autorisés : `int float bool string`
- Deux mode de typage ***coercive*** (par défaut), ***strict***

  Le mode est défini par fichier grâce au construit *`declare();`*

```php
<?php
declare(strict_types=0);

// Coercive mode (declare optional)
function sumOfInts(int ...$ints)
{
        return array_sum($ints);
}

var_dump(sumOfInts(2, '3', 4.1));
// int(9)
```

```php
<?php
declare(strict_types=1);

// Strict mode
function sumOfInts(int ...$ints)
{
        return array_sum($ints);
}

var_dump(sumOfInts(2, '3', 4.1));
// Argument 2 passed to sumOfInts()
must be of the type integer, string
given
```

Coercive type hinting behaviour

Type Déclaration	int	float	string	bool	object
int	yes	yes *	yes †	yes	no
float	yes	yes	yes †	yes	no
string	yes	yes	yes	yes	yes‡
bool	yes	yes	yes	yes	no

* Si compris entre PHP_INT_MIN et PHP_INT_MAX

† Chaînes non numériques non acceptées

‡ Uniquement si __toString() est implémenté

Déclaration du type de retour

- Les mêmes types que pour le typage de paramètres sont autorisés
- Les deux mode de typage ***coercive*** (par défaut), ***strict*** s'appliquent

  Le même mode que pour le typage de paramètres est utilisé

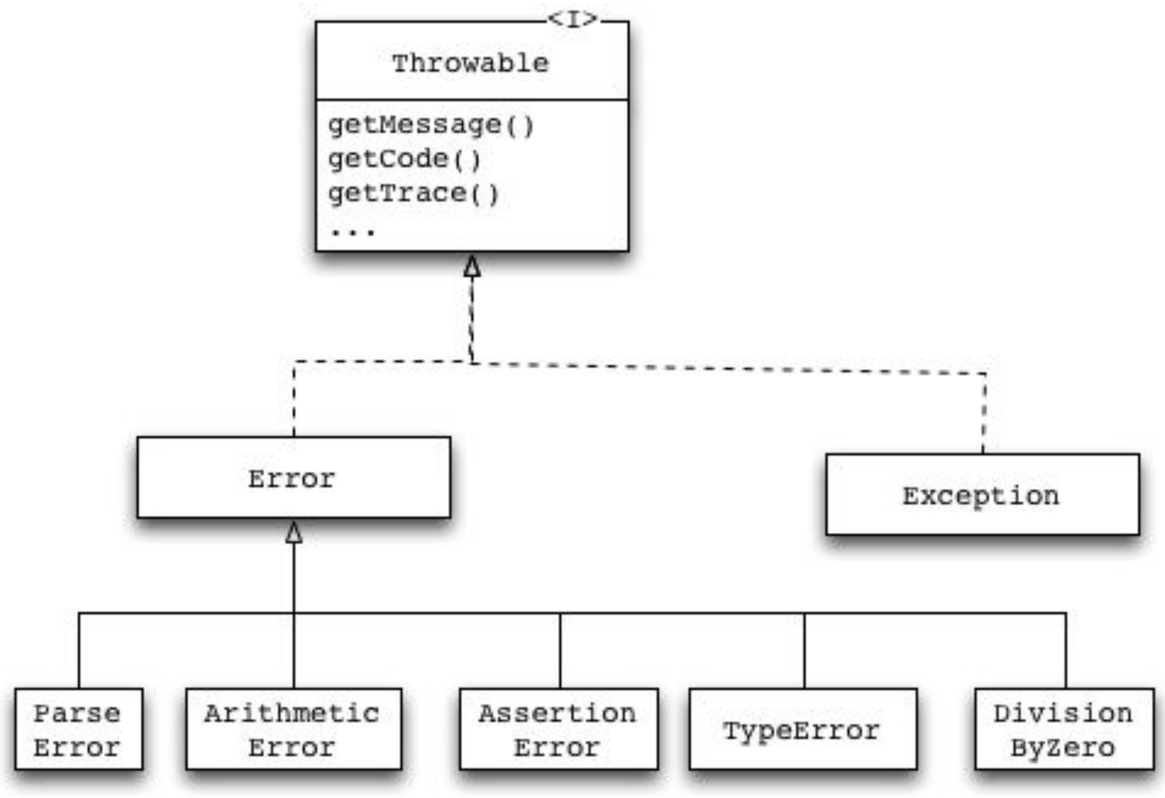
```php
function sum($a, $b): float {
    return $a + $b;
}

var_dump(sum(1, 2)); // float(3)
```

Support des classes anonymes

```php
interface Logger {
    public function log($format, ...$args);
}

class Util {
    public function setLogger(Logger $logger) { /* … */ }
}

$util->setLogger(

  new class('file.log') implements Logger {
    private $file;
    public function __construct($file) {
      $this->file = $file;
    }
    public function log($format, ...$args) {
      file_put_contents($this->file, sprintf($format, ...$args));
    }
  }

);
```

“Exceptions” dans l’engin

- Remplacement de plusieurs erreurs par des “*Exceptions like*” `Error`
- Les erreurs PHP sont des objets de type `Error`
- Pour catcher les `Error` et les `Exception` l’utilisation de `Throwable` est nécessaire



“Exceptions” dans l’engin

```php
try {
    bar();
} catch(\Error $e) {
    var_dump($e->getMessage());
}
// Call to undefined function
bar()
```

➢ L’engin permet de catcher les Erreurs

```php
bar();
// Fatal error: Uncaught Error:
Call to undefined function bar()
```

➢ Si l’erreur n’est pas catchée, elle est transformée en Fatal Error

Context sensitive Lexer

- Certains keywords deviennent semi réservés

```
callable class trait extends implements static abstract final public
protected private const enddeclare endfor endforeach endif endwhile and
global goto instanceof insteadof interface namespace new or xor try
use var exit list clone include include_once throw array print echo
require require_once return else elseif default break continue switch
yield function if endswitch finally for foreach declare case do while
as catch die self parent
```

```php
class Collection {
    public function forEach(callable $callback) { /* */ }
    public function list() { /* */ }
}
```

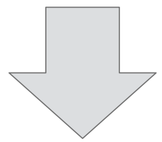
Grouper les déclarations use

Les classes, fonctions et constantes d'un même namespace peuvent êtres groupées dans un même **use** statement

```php
// Before PHP 7 code
use some\namespace\ClassA;
use some\namespace\ClassB;
use some\namespace\ClassC as C;

use function some\namespace\fn_a;
use function some\namespace\fn_b;
use function some\namespace\fn_c;

use const some\namespace\ConstA;
use const some\namespace\ConstB;
use const some\namespace\ConstC;

// PHP 7+ code
use some\namespace\{ClassA, ClassB, ClassC as C};
use function some\namespace\{fn_a, fn_b, fn_c};
use const some\namespace\{ConstA, ConstB, ConstC};
```

Null coallescing operator ??

```php
$username = isset($_GET['user']) ? $_GET['user'] : 'nobody';
```



```php
$username = $_GET['user'] ?? 'nobody';
```

➢ Les coalescences peuvent aussi se chaîner

```php
$username = $_GET['user'] ?? $_POST['user'] ?? 'nobody';
```

Zero cost assertions

- `assert` est maintenant un construit du langage avec la signature

```
assert(expression [, message]);
```

- Lors de l'execution, si le résultat de l'expression est **`false`**, alors une `AssertionError` sera **`throw`**

- Les assertions peuvent être désactivée via le setting zend.assertions
  - `1 - generate and execute code (development mode)`
  - `0 - generate code and jump around at it at runtime`
  - `-1 - don't generate any code (zero-cost, production mode)`

Zero cost assertions

```php
public function setResponseCode($code) {
    assert(
      $code < 550 && $code > 100,
      "Invalid response code provided: {$code}");
    );

    $this->code = $code;
}
```

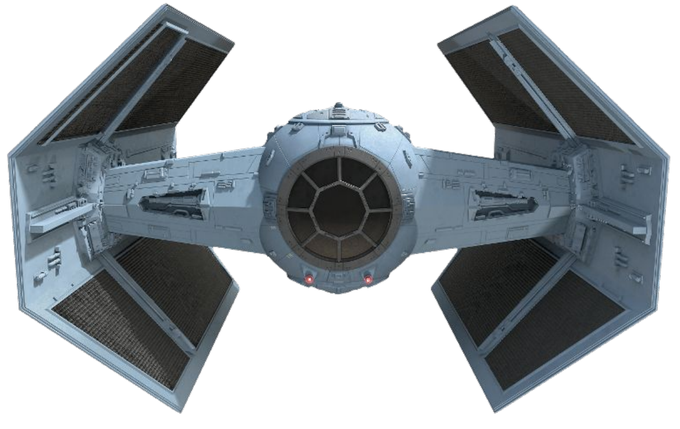
```php
public function getResponseCode() {
    assert($this->code,"The response code is not yet set");

    return $this->code;
}
```

Assertions vs Exceptions

- Exceptions :
  - vérifier les paramètres passés à des fonctions public ou protected.
  - interaction avec un utilisateur ou quand vous vous attendez à ce que le code client se récupère d'une situation exceptionnelle.
  - gérer des problèmes qui pourraient se produire lors du flot d'exécution

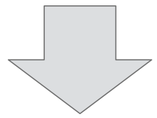
- Assertions
  - vérifier des pre-conditions, post-conditions et invariantes de code privé/interne.
  - donner du feedback à vous ou vos collègues développeurs (documentation).
  - vérifier certaines choses qui ne devraient JAMAIS arriver en production
  - constater des choses que vous (soi-disant) savez être vraies.
  - une assertion ne devrait JAMAIS être nécessaire au code pour fonctionner

Spaceship operator



- Nouvel opérateur de three-way comparaison
- Retourne
  - 0 si les deux opérandes sont égales
  - 1 si l'opérande gauche est supérieure
  - -1 si l’opérande droite est supérieure
- Utile pour écrire des fonctions de comparaison (pour usort, ...)

```php
function order_func($a, $b) {
    return ($a < $b) ? -1 : (($a > $b) ? 1 : 0);
}
```



```php
function order_func($a, $b) {
    return $a <=> $b;
}
```

Caractère d’échapement unicode

➢ Utilisation du \u pour définir un caractère unicode

```
echo "\u{1F602}"; // outputs 😂
```

➢ Permet de mieux distinguer deux valeurs dont l’affichage est visuellement similaire mais avec un encodage différent. Par exemple :

```
echo "mañana";
echo "mañana";
```

est moins explicite que :

```
echo "ma\u{00F1}ana";  // pre-composed character
echo "man\u{0303}ana"; // "n" with combining ~ character (U+0303)
```

Délégation de générateurs

```php
function gen()
{
    yield 1;
    yield 2;
    yield from gen2();
    yield from [ 5 , 6 ];
    yield from new ArrayIterator([ 7 , 8 ]);
}

function gen2()
{
    yield 3;
    yield 4;
}

foreach (gen() as $val)
{
    echo $val, PHP_EOL;
}
// 1 2 3 4 5 6 7 8
```

- Permet de déléguer la logique d'un générateur à des objets `Traversable` ou à des `Array`

Autres

- Suppression des tags php alternatifs
  - `<% %>`
  - `<script type="php"></script>`

- Suppression des extensions
  - mssql
  - mysql
  - ereg

- Zend Engine 3.0
  - AST Based parser
  - Meilleur gestion de la mémoire

Performances

	PHP 5.3	PHP 5.4	PHP 5.5	PHP 5.6	PHP 7.0
bench.php	*9.121*	8.636	8.584	8.241	4.401
micro_bench.php	63.086	36.131	35.355	33.429	19.443
Unit tests Doctrine2	15.58	14.01	13.82	12.95	6.26

47%

42%

52%

nuGLIF

Questions ?

Merci :-)

Performances Générales

	PHP 5.3	PHP 5.4	PHP 5.5	PHP 5.6	PHP 7.0
bench.php	*9.121*	8.636	8.584	8.241	4.401
		5%	**1%**	**4%**	**47%**
micro_bench.php	63.086	36.131	35.355	33.429	19.443
		42%	**2%**	**5%**	**42%**
Unit tests Doctrine2	15.58	14.01	13.82	12.95	6.26
		10%	**1%**	**6%**	**52%**