PHP7 Jump Start

zend.com

v1.1.0

What we will learn

- How PHP is maintained
- What is coming in PHP 7 that may break your existing code
- What is coming in PHP 7 that will impact how you write PHP code
- PHP 7 and PSR-7 a match made in heaven

A Brief History of Time

- How PHP started
- Scratch your own itch development
- Needle/haystack issues

The RFC Process

- BRFC (Before RFCs)
- The Current Era
- Classification of RFCs
  - BC Breaking
  - High Impact

Backward Compatibility Breaking RFCs

- Continue output buffering despite aborted connection
- Replacing current json extension with jsond
- Make defining multiple default cases in a switch a syntax error
- Remove alternative PHP tags
- Abstract syntax tree
- Reclassify E_STRICT notices
- Reserve More Types in PHP 7
- Uniform Variable Syntax
- ZPP Failure on Overflow
- Constructor behavior of internal classes
- Fix "foreach" behavior
- Removal of dead or not yet PHP7 ported SAPIs and extensions
- Remove PHP 4 Constructors
- Fix handling of custom session handler return values

PHP7 JumpStart

Continue output buffering despite aborted connection

https://wiki.php.net/rfc/continue_ob

Author: Michael Wallne

Continue output buffering despite aborted connection

Problem
In the re-write of the output buffering code for PHP 5.4 a bug was introduced.

If

- You use set ignore_user_abort to true
- You turn on output buffering
- You set a handler
- The connection is aborted

Then

- The output buffer handler **is not** called
- The output is discarded

This is only a problem if you are doing any processing in your output buffer handler.

Continue output buffering despite aborted connection

Solution

The bug is resolved in PHP7.

If

- You use set ignore_user_abort to true
- You turn on output buffering
- You set a handler
- The connection is aborted

Then

- The output buffer **is** called
- The output is discarded

It is now safe to do processing like saving data into a cache from within your output buffer handler.

Continue output buffering despite aborted connection

Currently affected language functions

There are a few internal language commands that this could potentially affect.

- phpinfo()
- highlight_{file,string} with return_output=TRUE
- print_r() or var_export() with return_output=TRUE

Continue output buffering despite aborted connection

```
ignore_user_abort(true);

ob_start("make_fizbuzz");

for($lcvA=1;$lcvA<10;$lcvA++) {
    echo $lcvA . ",";
}


function make_fizbuzz ($buffer) {
    $buffer_as_array = explode(',',$buffer);
    $payload = '';

    foreach ($buffer_as_array as $value ) {
        if (empty($value)) {
            continue;
        }
        $payload .= $value . ":";
        $payload .= ($value%3)?'':'fizz';
        $payload .= ($value%5)?'':'buzz';
        $payload .= "\n";
    }

    return $payload;
}
```

Continue output buffering despite aborted connection

Potential for Backwards Compatibility Break

Output handlers will be called even when

```
ignore_user_abort = TRUE
```

Replacing current json extension with jsond

https://wiki.php.net/rfc/jsond

Author: Jakub Zelenka

Replacing current json extension with jsond

License Issue

./ext/json/JSON_parser.c:The Software shall be used for Good, not Evil.

This is incompatible with some Linux distros like Debian.

Replacing current json extension with jsond

Number Storage Issue

Current JSON

100 can be stored in a double. (Float)

New JSOND

100 can only be stored as an Integer

100.01 can be stored as a double.

Replacing current json extension with jsond

- New Code is based on https://pecl.php.net/package/jsond

Replacing current json extension with jsond

Backwards Compatibility Breaks

Incredibly small chance of a Backwards Compatibility break, but because there is a change being implemented in the engine, there is a chance of a break.

Test your JSON encode and decode carefully.

PHP7 JumpStart

Make defining multiple default cases in a switch a syntax error

https://wiki.php.net/rfc/switch.default.multiple

Levi Morrison

Make defining multiple default cases in a switch a syntax error

Bad

```
switch ($expr) {
  default:
    neverExecuted();


  default:
    executed();
}
```

```
PHP 7: E_COMPILE_ERROR
```

Make defining multiple default cases in a switch a syntax error

Backwards Compatibility Breaks

Well…yeah. If you did this in the past, this code will not compile in PHP 7.

Go back and chastise past you, don't blame PHP.

PHP7 JumpStart

Remove alternative PHP tags

https://wiki.php.net/rfc/remove_alternative_php_tags

Nikita Popov

Remove alternative PHP tags

- <% opening tag
- <%= opening tag with echo
- %> closing tag
- (<script\s+language\s*=\s*(php|"php"|'php')\s*>)i opening tag
- (</script>)i closing tag

Remove alternative PHP tags

<?= is not being removed

Remove alternative PHP tags

Backwards Compatibility Breaks

- Possible If your code uses any of the deprecated tags then you will need to swap them out.

Remove alternative PHP tags

Porting Script
https://gist.github.com/nikic/74769d74dad8b9ef221b

```
php -d asp_tags=1 portAlternativeTags.php dir/
```

Break

PHP7 JumpStart

Abstract syntax tree

https://wiki.php.net/rfc/abstract_syntax_tree

Author: Nikita Popov

Abstract syntax tree

Advantages

- More maintainable parser and compiler
- Decoupling syntax decisions from technical issues

Example

```
$result = yield fn();   // INVALID
$result = (yield fn()); // VALID
```

Abstract syntax tree

Bonus feature

- Directly calling `__clone` is now allowed

Abstract syntax tree

Backwards Compatibility Breaks

- Changes to `list()`

```
list($array[], $array[], $array[]) = [1, 2, 3];
var_dump($array);
// OLD: $array = [3, 2, 1]
// NEW: $array = [1, 2, 3]
```

Abstract syntax tree

Backwards Compatibility Break

- Empty list()s are now **always** disallowed. Previously they were only forbidden in some places

```
list() = $a;           // INVALID
list($b, list()) = $a; // INVALID
foreach ($a as list()) // INVALID (was also
invalid previously)
```

Abstract syntax tree

Backwards Compatibility Break

- While by-reference assignments are evaluated left-to-right, auto-vivification currently occurs right-to-left. In the AST implementation this will happen left-to-right instead.

```
$obj = new stdClass;
$obj->a = &$obj->b;
$obj->b = 1;

var_dump($obj);
```

```
// PHP 5.6 and below
object(stdClass)#1 (2) {
  ["b"]=>
  &int(1)
  ["a"]=>
  &int(1)
}

// PHP7
object(stdClass)#1 (2) {
  ["a"]=>
  &int(1)
  ["b"]=>
  &int(1)
}
```

Reclassify E_STRICT notices

https://wiki.php.net/rfc/reclassify_e_strict

Author: Nikita Popov

Reclassify E_STRICT notices

- Remove the strict standards notice if it appears inconsistent or informational.
- Promote to E_DEPRECATED if there is intent to remove this functionality in the future.
- Promote to E_NOTICE or E_WARNING otherwise.

Reclassify E_STRICT notices

```
class BaseController
{
    public function getAction($id = null)
    {
        // do stuff with the $id
    }
}

class UserController extends BaseController
{
    public function getAction()
    {
        // get the id from a request object instead
        $id = $this->input->get('id');

        // do stuff with $id
    }
}
```

Reclassify E_STRICT notices

Backwards Compatibility Breaks

- Some of the strict standards notices are converted to an error category that is considered more severe. As such error handlers might treat it more severely, resulting in BC breakage.
- The E_STRICT constant will be retained, as such existing error_reporting(E_ALLIE_STRICT) calls will continue to work fine.
- The E_STRICT constant will be retained for better compatibility, it will simply no longer have meaning in PHP 7.

Reserve More Types in PHP 7 (2 RFCs)

https://wiki.php.net/rfc/reserve_more_types_in_php_7

Author: Levi Morrison

https://wiki.php.net/rfc/reserve_even_more_types_in_php_7

Author: Sara Golemon

Reserve More Types in PHP 7

- int
- float
- bool
- string
- true, false
- null
- resource
- object
- mixed
- numeric

Reserve More Types in PHP 7

Backwards Compatibility Break

If your code uses any of these new reserved words as variable names or class names, change them now.

PHP7 JumpStart

Uniform Variable Syntax

https://wiki.php.net/rfc/uniform_variable_syntax

Author: Nikita Popov

Uniform Variable Syntax

```
$foo->bar()();
```

- `$foo is an object`
- `bar() is a method that returns a` **`callable`**
- `The callable is automatically executed`

Uniform Variable Syntax

There are no longer any restrictions on nesting of dereferencing operations. All of these are now supported:

Dereferencing a return value of a method and then calling it as a method.

```
$foo()['bar']();
```

- $foo is a method that returns an array.
- ['bar'] is an element in the array returned by $foo()
- ['bar'] contains a callable as it's value
- () calls the callable contained in ['bar']

Uniform Variable Syntax

You can now dereference strings

```
function getStr()
{
    return 'Cal Evans';
}
echo getStr(){4}; // 'E'
```

Uniform Variable Syntax

Double $ in referencing globals are no longer supported.

```
global $$foo->bar; //No longer supported
```

Uniform Variable Syntax

There are no longer any restrictions on nesting of dereferencing operations. All of these are now supported:

Creating an array of objects and then returning the value of a property from the first one.

```
$obj1 = new StdClass();
$obj1->name = 'Kathy';
$obj2 = new StdClass();
$obj2->name = 'Cal';
$returnValue = [$obj1, $obj2][0]->name;
```

- $obj1 and $obj2 are objected that each contain the property ‘name’
- We create the array of the objects and then immediately dereference it with [0] meaning we are only interested in $obj1.

Uniform Variable Syntax

Static property fetches and method calls can now be applied to any expression that returns a value. All of these expressions are now valid.

Reference a property statically from a class reference.

```
$foo['bar']::$baz
```

Statically access a nested referenced property

```
$foo::$bar::$baz
```

Call a referenced method statically from a class reference returned by a method.

```
$foo->bar()::baz()
```

Uniform Variable Syntax

- The result of a method call can now be directly called again. All of these are valid are valid now.

- `foo()()`
- `$foo->bar()()`
- `Foo::bar()()`
- `$foo()()`

Uniform Variable Syntax

- All dereferencing operations can now be applied to arbitrary parenthesis-expressions.

```
(...)['foo']
(...)->foo
(...)->foo()
(...)::$foo
(...)::foo()
(...)()
```

Uniform Variable Syntax

Any action that returns a value can now in theory be applied to things like strings.

For instance, it is theoretically possible to write code like this.

- `"string"->toLower()`
- `[$obj, 'method']()`
- `'Foo'::$bar.`

Extensions can then use it to implement the actual behavior for something like `"string"->toLower()`.

Uniform Variable Syntax

BC Break

	Old Meaning	*New Meaning*
`$$foo['bar']['baz']`	`${$foo['bar']['baz']}`	`($$foo)['bar']['baz']`
`$foo->$bar['baz']`	`$foo->{$bar['baz']}`	`($foo->$bar)['baz']`
`$foo->$bar['baz']()`	`$foo->{$bar['baz']}()`	`($foo->$bar)['baz']()`
`Foo::$bar['baz']()`	`Foo::{$bar['baz']}()`	`(Foo::$bar)['baz']()`

ZPP Failure on Overflow

https://wiki.php.net/rfc/zpp_fail_on_overflow

Author: Andrea Faulds

ZPP Failure on Overflow

- Floats that are auto converted to Integers can be silently truncated.
- 3221225470.5 becomes -1073741826 on 32-bit platforms

ZPP Failure on Overflow

Backwards Compatibility Break

- Your code that once worked - even if incorrectly - will now emit an E_WARNING.
- Your code may fail if the result of calling the function is directly passed to another function (since null will now be passed in).

PHP7 JumpStart

Constructor behavior of internal classes

https://wiki.php.net/rfc/internal_constructor_behaviour

Author: Dan Ackroyd

Constructor behavior of internal classes

This RFC has two goals:

- To make some internal classes behave more consistently.
- Setting the standard behavior that future internal classes should have for their constructors.

Constructor behavior of internal classes

The Problem

```
$mf = new MessageFormatter('en_US', '{this was
made intentionally incorrect}');

if ($mf === null) {
    echo "Surprise!";
}
```

Constructor behavior of internal classes

The Solution

```
try {
  $mf = new MessageFormatter('en_US', '{this was
  made intentionally incorrect}');
} catch (\Exception $e) {
    echo "No Surprise here";

}
```

Constructor behavior of internal classes

Affected Internal Classes

- finfo
- PDO
- Collator
- IntlDateFormatter
- MessageFormatter
- NumberFormatter
- ResourceBundle
- IntlRuleBasedBreakIterator

Constructor behavior of internal classes

Backwards Compatibility Break

- There is a very slight chance of BC Break.
  This is largely an internal engine change.
- **Constructors now throw an error instead of returning a null.**

  Previously, some internal classes would accept invalid parameters and still return a class, these have been dealt with as well.
- You may find that you are using try/catch more instead of `if ($x===null)`.

Fix "foreach" behavior

https://wiki.php.net/rfc/php7_foreach

Author: Dmitry Stogov

Fix "foreach" behavior

Problem

```
$ php -r '$a = [1,2,3]; foreach($a as $v) {echo $v
. " - " . current($a) . "\n";}'
```

```
1 - 2
2 - 2
3 - 2
```

```
$ php -r '$a = [1,2,3]; $b = $a; foreach($a as $v)
{echo $v . " - " . current($a) . "\n";}'
```

```
1 - 1
2 - 1
3 - 1
```

Fix "foreach" behavior

Solution

- This is largely an internal engine fix to make behaviors consistent.

Performance

- This new behavior eliminates internal array duplication and should lead to better performance.
- Some HashTable operations require additional checks under this new code.
- For example using Wordpress as a test, this change reduces the number of executed CPU instructions by ~1% because it saves ~200 array duplications and destructions for each request to the home page.

PHP7 JumpStart

Removal of dead or not yet PHP7 ported SAPIs and extensions

https://wiki.php.net/rfc/removal_of_dead_sapis_and_exts

Author: Anatol Belski

Removal of dead or not yet PHP7 ported SAPIs and extensions

- aolserver
- apache
- apache_hooks
- caudium
- continuity
- isapi
- milter
- phttpd
- pi3web
- roxen
- thttpd
- tux
- webjames
- apache2filter - not really dead, but currently broken
- nsapi
- mysql
- ereg
- imap
- mcrypt
- interbase
- mssql
- oci8
- pdo_dblib
- pdo_oci
- sybase_ct

Removal of dead or not yet PHP7 ported SAPIs and extensions

- aolserver
- apache
- apache_hooks
- caudium
- continuity
- isapi
- milter
- phttpd
- pi3web
- roxen
- thttpd
- tux
- webjames
- apache2filter - not really dead, but currently broken
- nsapi
- **mysql**
- **ereg**
- imap
- mcrypt
- interbase
- mssql
- oci8
- pdo_dblib
- pdo_oci
- sybase_ct

PHP7 JumpStart

Resolve inconsistencies in how the `list()` function works.

https://wiki.php.net/rfc/fix_list_behavior_inconsistency

Author: Dmitry Stogov

Fix list() behavior inconsistency

```
$ php -r 'list($a,$b) = "aa";var_dump($a,$b);'
NULL
NULL

$ php -r '$a[0]="ab"; list($a,$b) = $a[0];
var_dump($a,$b);'
string(1) "a"
string(1) "b"
```

Fix list() behavior inconsistency

```php
list($a,$b) = "str";
echo $a; // s
echo $b; // t
```

Fix list() behavior inconsistency

Backwards Compatibility Break

If you use list() with strings, test your code throughly. It is an edge case but you need to make sure.

PHP7 JumpStart

Remove hex support in numeric strings

https://wiki.php.net/rfc/remove_hex_support_in_numeric_strings

Author: Nikita Popov

Remove hex support in numeric strings

- This RFC removes support for hexadecimal numbers in is_numeric_string to ensure consistent behavior across the eternally in the engine. `is_numeric_string` is the internal engine function that converts strings to numbers. This function will no longer recognize hex

This only affects hexadecimal numbers passed in as strings.

Remove hex support in numeric strings

Currently

```
$str = '0x123';
if (!is_numeric($str)) {
    throw new Exception('Not a number');
}

// Exception not thrown, instead wrong result is
generated here:
$n = (int) $str; // 0
```

Remove hex support in numeric strings

Currently

```
var_dump("0x123" == "291"); // TRUE

var_dump((int) "0x123" == (int) "291"); // FALSE
```

Remove hex support in numeric strings

Backwards Compatibility Break

- The is_numeric() function.
- Operands of the ==, +, -, *, /, %, **, ++ and -- operators

Remove hex support in numeric strings

A robust way of both validating and parsing hexadecimal strings is given by FILTER_VALIDATE_INT in conjunction with FILTER_FLAG_ALLOW_HEX.

```
$hex = filter_var('0x123', FILTER_VALIDATE_INT,
FILTER_FLAG_ALLOW_HEX);
```

PHP7 JumpStart

Fix handling of custom session handler return values

https://wiki.php.net/rfc/session.user.return-value

Author: Sara Golemon

Fix handling of custom session handler return values

`session_set_save_handler` currently returns a 0 for success and a -1 for false. This is inconsistent with the rest of PHP.

Fix handling of custom session handler return values

Backwards Compatibility Break

If you depend on session_set_save_handler returning a 0 for success and a -1 for failure, you code will break.

Break

High Impact RFCs

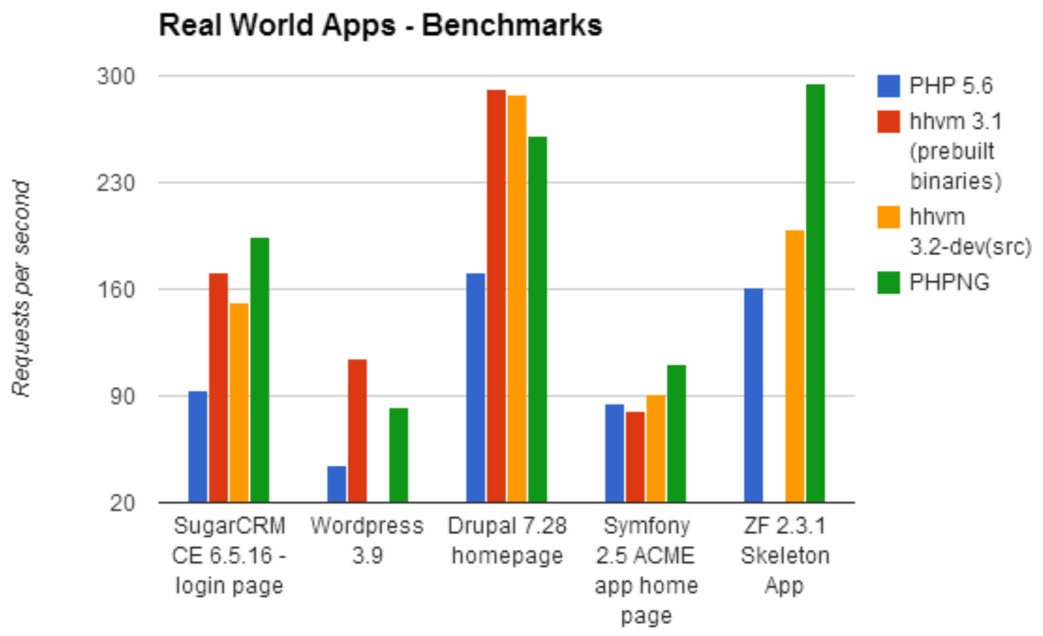
- MOVE THE PHPNG BRANCH INTO MASTER
- SPACESHIP OPERATOR
- ANONYMOUS CLASSES
- BIND CLOSURE ON CALL
- GENERATOR RETURN EXPRESSIONS
- GENERATOR DELEGATION
- FILTERED UNSERIALIZE()
- EXCEPTIONS IN THE ENGINE
- EASY USER-LAND CSPRNG
- NULL COALESCE OPERATOR
- RETURN TYPE DECLARATIONS
- SCALAR TYPE DECLARATIONS
- GROUP USE DECLARATIONS

Move the phpng branch into master

https://wiki.php.net/rfc/phpng

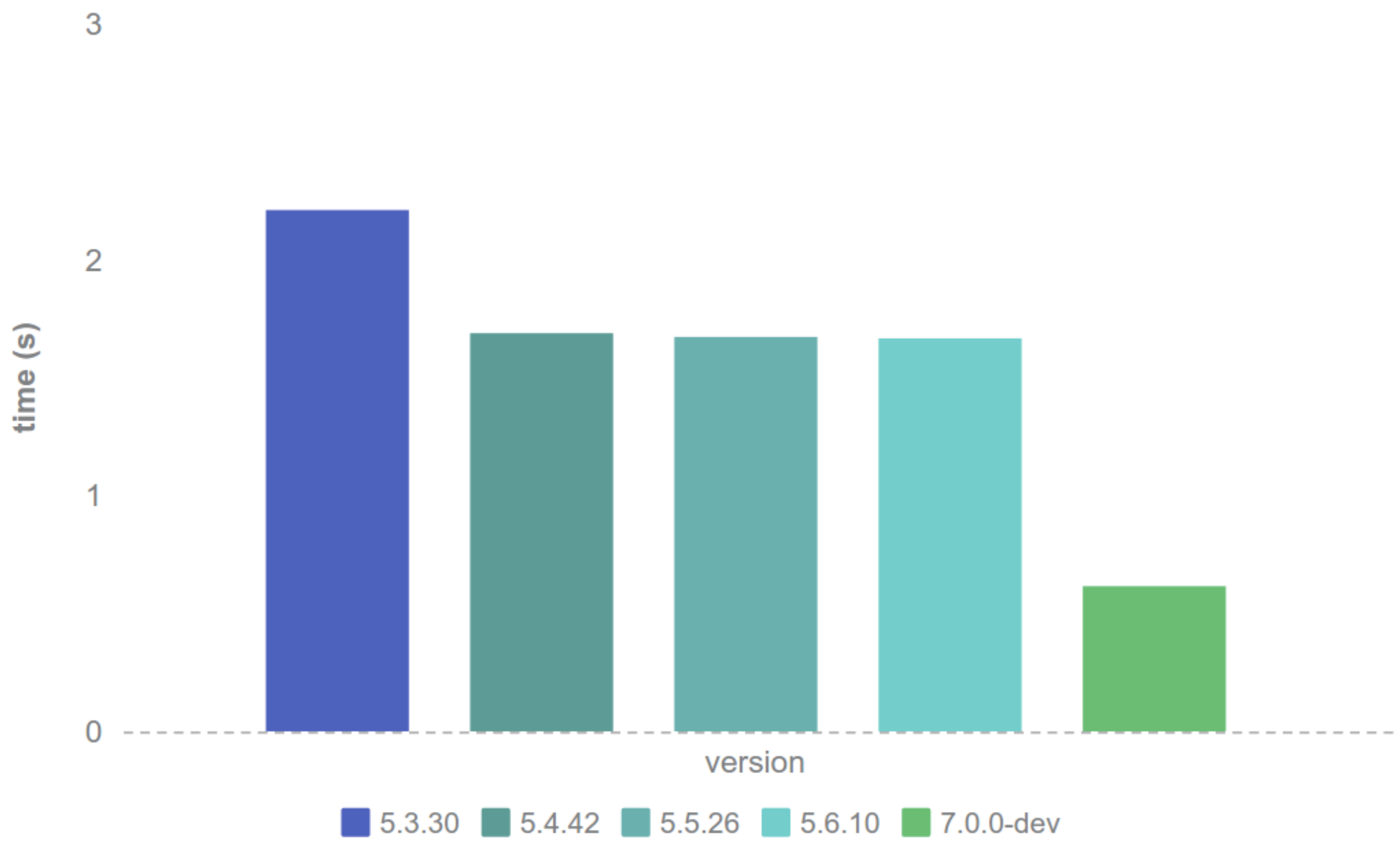
Authors : Dmitry Stogov, Zeev Suraski, and team

Move the phpng branch into master



http://zsuraski.blogspot.com/2014/07/benchmarking-phpng.html

Move the phpng branch into master



- http://www.lornajane.net/posts/2015/php-7-benchmarks

Move the phpng branch into master

- ~95% faster than 5.6
  http://www.reddit.com/r/PHP/comments/305ck6/real_world_php_70_benchmarks/
- ~85% faster than 5.6
  http://zsuraski.blogspot.com/2014/07/benchmarking-phpng.html
- Drupal is 58% faster on PHP7 vs. PHP 5.6
  http://www.drupalonwindows.com/en/blog/benchmarking-drupal-7-php-7-dev

Combined Comparison (Spaceship) Operator

https://wiki.php.net/rfc/combined-comparison-operator

Authors: Davey Shafik, Andrea Faulds, Stas Malyshev

Combined Comparison (Spaceship) Operator

- PHP's first trinary operator

```
echo 1<=>2; // -1
echo 1<=>1; // 0
echo 2<=>1; // 1
```

Combined Comparison (Spaceship) Operator

Operator	<=> Equilivant
`$a < $b`	`($a <=> $b) === -1`
`$a <= $b`	`($a <=> $b) === -1 \|\| ($a <=> $b) === 0`
`$a == $b`	`($a <=> $b) === 0`
`$a != $b`	`($a <=> $b) !== 0`
`$a >= $b`	`($a <=> $b) === 1 \|\| ($a <=> $b) === 0`
`$a > $b`	`($a <=> $b) === 1`

Combined Comparison (Spaceship) Operator

Examples

Strings

```
echo "a"  <=> "a";  //  0
echo "a"  <=> "b";  // -1
echo "b"  <=> "a";  //  1
echo "a"  <=> "aa"; // -1
echo "zz" <=> "aa"; //  1
```

Combined Comparison (Spaceship) Operator

Examples

Arrays

```
echo [] <=> [];               //  0
echo [1, 2, 3] <=> [1, 2, 3]; //  0
echo [1, 2, 3] <=> [];        //  1
echo [1, 2, 3] <=> [1, 2, 1]; //  1
echo [1, 2, 3] <=> [1, 2, 4]; // -1
```

Combined Comparison (Spaceship) Operator

Examples

Objects

```
$a = (object) ["a" => "b"];
$b = (object) ["a" => "c"];
echo $a <=> $b; // -1

$a = (object) ["a" => "b"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 0

$a = (object) ["a" => "c"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 1
```

Combined Comparison (Spaceship) Operator

usort() example

```
$array = ['oranges', 'apples', 'bananas', 'grapes'];

usort($array,
  function ($left,$right) {
    return $left<=>$right;
  }
);

print_r($array);
```

```
Array
(
    [0] => apples
    [1] => bananas
    [2] => grapes
    [3] => oranges
)
```

PHP7 JumpStart

Anonymous Classes

https://wiki.php.net/rfc/anonymous_classes

Author: Joe Watkins, Phil Sturgeon

Anonymous Classes

```
$obj = new class($i) {
          public function __construct($i) {
               $this->i = $i;
          };
```

Anonymous Classes

An anonymous class might be used over a named class:

- When the class does not need to be documented
- When the class is used only once during execution

Anonymous Classes

Example

```
(new class extends ConsoleProgram {
    public function main() {
        /* ... */
    }
})->bootstrap();
```

Anonymous Classes

Example

```
return new class($controller) implements Page {
    public function __construct($controller) {
        /* ... */
    }
    /* ... */
};
```

Anonymous Classes

Example

```
$pusher->setLogger(new class {
  public function log($msg) {
    print_r($msg . "\n");
  }
});
```

Anonymous Classes

Example

```
class Outside {
    protected $data;

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

    public function getArrayAccess() {
        return new class($this->data) extends Outside implements ArrayAccess {
            public function offsetGet($offset) { return $this-
>data[$offset]; }
            public function offsetSet($offset, $data) { return ($this-
>data[$offset] = $data); }
            public function offsetUnset($offset) { unset($this-
>data[$offset]); }
            public function offsetExists($offset) { return isset($this-
>data[$offset]); }
        };
    }
}
```

Anonymous Classes

```
$conduit->pipe(new class implements MiddlewareInterface {
    public function __invoke($request, $response, $next)
    {
        $laravelRequest = mungePsr7ToLaravelRequest($request);
        $laravelNext    = function ($request) use ($next,
$response) {
            $request = ;
            return $next(mungeLaravelToPsr7Request($request),
$response)
        };
        $laravelMiddleware = new SomeLaravelMiddleware();
        $response = $laravelMiddleware-
>handle($laravelRequest, $laravelNext);
        return mungeLaravelToPsr2Response($response);
    }
});
```

Anonymous Classes

Caveats, warning, and compatibility

- Inheritance works as expected
- Traits work as expected
- Reflection now has `ReflectionClass::isAnonymous()` method
- **Serialization will not work, just like with anonymous functions**

PHP7 JumpStart

Bind Closure on Call

https://wiki.php.net/rfc/closure_apply

Author: Andrea Faulds

Bind Closure on Call

Example

```
$foo = new StdClass;
$foo->bar = 3;
$stuff = function ($baz) {
                echo ($this->bar + $baz);
            };
$stuff->call($foo, 4); // 7
```

Bind Closure on Call

```
class Foo {
  private $x;

  public __construct($value) {
    $this->x = $value;
    return;
  }
}

$foo = new Foo(3);

$stuff = function () {
  var_dump($this->x);
};

$stuff->call($foo); // prints int(3)
```

Bind Closure on Call

- Similar to `Closure::bind()`
- `Closure::call()` shows a 2.18x improvement in speed over `Closure::bindTo()`

PHP7 JumpStart

Generator Return Expressions

https://wiki.php.net/rfc/generator-return-expressions

Author: Daniel Lowrey

Contributors: Nikita Popo

Generator Return Expressions

- Nothing changes with `yield`

```
yield 1;
```

- New method for Generators, `->getReturn()`

```
return 42;
```

- State of the generator is important. `getReturn()` throw an error if called on a valid generator.
- Generator can accumulate data and use that as the return value.

Generator Return Expressions

Current

```
function foo() {
    yield 0;
    yield 1;

   return 42;
}

// Fatal error: Generators cannot return values
```

Generator Return Expressions

New

```
function foo() {
  $returnValue = 1;
  yield 1;
  $returnValue = 2;
  yield 2;
  return $returnValue;
}

$bar = foo();

foreach ($bar as $element) {
    echo $element, "\n";
}

var_dump($bar->getReturn());

// 1
// 2
// int(2)
```

Generator Return Expressions

Calling `getReturn()` without a return statement

```
function foo() {
    yield 1;
    yield 2;
    yield 3;
}

$bar = foo();
while($bar->valid()) {
    $bar->next();
}

assert($bar->getReturn() === null);
```

Generator Return Expressions

Calling `getReturn()` while the generator is still valid

```
function foo() {
    yield 1;
    yield 2;
    return 42;
}

$bar = foo();
$bar->current();
$bar->next();

assert($bar->valid());

// Throws an \Exception because the generator is still
valid
$returnValue = $bar->getReturn();
```

Generator Delegation

https://wiki.php.net/rfc/generator-delegation

Author: Daniel Lowrey

Generator Delegation

New syntax

```
yield from <expr>
```

Generator Delegation

New Terminology

- **“delegating generator”**
  a Generator in which the **`yield from <expr>`** syntax appears.
- **“subgenerator”**
  Generator used in the <expr> portion of the **`yield from <expr>`** syntax.

Generator Delegation

Each value yielded by the traversable is passed directly to the delegating generator's caller.

```
function foo() {
  yield 1;
  yield 2;
  yield 3;
}

function bar() {
  yield "a";
  yield "b";
  yield "c";
}
```

```
function both() {
  yield from foo();
  yield from bar();
  yield "done"
}


$a = both();

foreach ($a as $value) {
  echo $value . "\n";
}
```

Generator Delegation

Returns are not passed as a yield from the sub generator.

```
function foo() {
  yield 1;
  yield 2;
  yield 3;
}

function bar() {
  yield "a";
  yield "b";
  yield "c";
  return 42; // RETURN
}
```

```
function both() {
  yield from foo();
  yield from bar();
  yield "done";
}

$both = both();

foreach ($both as $value){
  echo $value . "\n";
}

var_dump($both->getValue());
```

Generator Delegation

- Exceptions thrown by traversable/subgenerator advancement are propagated up the chain to the delegating generator.

Generator Delegation

- Subgenerator can also be an array or any traversable.

```
function g() {
  yield 1;
  yield from [2, 3, 4];
  yield 5;
}

$g = g();

foreach ($g as $yielded) {
    var_dump($yielded);
}
```

Generator Delegation

```
function g() {
  yield 1;
  yield from [2, 3, 4];
  yield 5;
  return 42;
}

$g = g();

foreach ($g as $yielded) {
    var_dump($yielded);
}

var_dump($g->getReturn();
```

Generator Delegation

Error Conditions

- `yield from <expr>` where <expr> is a generator which previously terminated with an uncaught exception results in an EngineException.
- `yield from <expr>` where <expr> is not a **Traversable** or an **Array** throws an EngineException.

PHP7 JumpStart

Filtered unserialize()

https://wiki.php.net/rfc/secure_unserialize

Author : Stas Malyshev

Filtered unserialize()

Problem

Serialized data can include objects with data, and once these objects are instantiated, `__destroy()`, `__toString()`, `__call()`, and others could be used to inject bad data into the application.

Filtered unserialize()

Solution

Allow developers to whitelist the classes that can be instantiated via unserialize()

```
$data = unserialize($foo,
         ["allowed_classes" => ["Customer",
"Order"]);
```

Filtered unserialize()

```
// this will unserialize everything as before
$data = unserialize($foo);

// this will convert all objects into
__PHP_Incomplete_Class object
$data = unserialize($foo, ["allowed_classes" =>
false]);

// this will convert all objects except ones of MyClass
and MyClass2 into __PHP_Incomplete_Class object
$data = unserialize($foo, ["allowed_classes" =>
["MyClass", "MyClass2"]]);

//accept all classes as in default
$data = unserialize($foo, ["allowed_classes" => true]);
```

Filtered unserialize()

Backwards Compatibility Breaks

- None

```
// this will unserialize everything as before
$data = unserialize($foo);
```

PHP7 JumpStart

Exceptions in the engine/ Throwable Interface

https://wiki.php.net/rfc/engine_exceptions_for_php7
Author : Nikita Popov
https://wiki.php.net/rfc/throwable-interface
Author: Aaron Piotrowski

Exceptions in the engine/Throwable Interface

Problem

- Fatal errors do not invoke a finally block
- Fatal errors do not call an object's destructor
- Fatal errors **do** call the Error handler
- Fatal errors Cannot be gracefully handled

```
function do_something($obj) {
    $obj->myMethod();
}

do_something(null); // oops!
```

Exceptions in the engine/Throwable Interface

Solution

```
try {
    do_something(null); // oops!
} catch (\Error $e) {
    echo "Error: {$e->getMessage()}\n";
}

// Error: Call to a member function method() on
a non-object
```

Exceptions in the engine/Throwable Interface

The new interface Throwable

- interface **`Throwable`**
  - **`Exception`** implements **`Throwable`**
  - **`Error`** implements **`Throwable`**
    - **`TypeError`** extends **`Error`**
    - **`ParseError`** extends **`Error`**

Exceptions in the engine/Throwable Interface

Do not catch Errors except for logging and cleanup. Errors are are code issues that should be fixed, not conditions that can be handled at runtime.

Exceptions in the engine/Throwable Interface

- getMessage()
- getCode()
- getFile()
- getLine()
- getTrace()
- getTraceAsString()
- __toString()

Exceptions in the engine/Throwable Interface

```
function add(int $left, int $right) {
    return $left + $right;
}

try {
    echo add('left', 'right');
} catch (\TypeError $e) {
    // Log error and end gracefully
} catch (\Exception $e) {
    // Handle any exceptions
} catch (\Throwable $e) {
    // Handle everything else
}
```

Exceptions in the engine/Throwable Interface

ParseError

- Thrown when you have a parse error in your code
- Will not allow bad code to run at compile time.
- Will be thrown if you have a parse error in an eval()
- Will be thrown if you have a parse error in a file that is included during execution.

Exceptions in the engine/Throwable Interface

ParseError

```
$code = 'var_dup($admin);';

try {
    $result = eval($code);
} catch (\ParseError $error) {
    // Handle $error
}
```

Exceptions in the engine/Throwable Interface

ParseError

```
if ($admin) {
    try {
        include "./this_code_has_issues.php";
    } catch (\ParseError $error) {
        // Handle $error
    }
}
```

Exceptions in the engine/Throwable Interface

Backwards Compatibility Break

- Old: Parse errors generated during eval() (but not require etc) are non-fatal.
- New: **eval() now throws an exception** This may require some code adjustments in cases where you want to gracefully handle eval() errors.
- E_RECOVERABLE_ERROR
  Currently it is possible to silently ignore recoverable fatal errors with a custom error handler. By replacing them with exceptions this capability is removed, thus breaking compatibility.
- Throwable, Error, TypeError, and ParseError are built-in interfaces/classes. It will no longer be possible for users to create classes with those exact names. It will still be possible for those names to be used within a non-global namespace.

Exceptions in the engine/Throwable Interface

Benefits

- `finally` gets called
- `__destruct()` gets called.
- Fully backwardly compatible

Exceptions in the engine/Throwable Interface

Issues

- Error & ParseError implement Throwable and are the new Catchable exceptions
- Existing errors of type E_ERROR, E_RECOVERABLE_ERROR, E_PARSE or E_COMPILE_ERROR can be converted to Error
- Discourages introducing new errors of the type E_ERROR or E_RECOVERABLE_ERROR.

PHP7 JumpStart

Easy User-land CSPRNG

https://wiki.php.net/rfc/easy_userland_csprng

Author: Sammy Kaye Powers & Leigh

Easy User-land CSPRNG

- Reliable, user land **C**ryptographically **S**ecure **P**seudo**R**andom **N**umber **G**enerator
- No easy way to access cryptographically strong random numbers in user-land.
  - CryptGenRandom on Windows
  - /dev/random on Linux/OSX
- Users may attempt to generate their own streams of random bytes…and this is something we absolutely want to avoid.

Easy User-land CSPRNG

```
$random = fread(fopen('/dev/random', 'r'),16);
```

Easy User-land CSPRNG

Two new functions

```
random_bytes(int length);
$randomStr = random_bytes(16);

random_int(int min, int max);
$randomInt = random_int(1, 20);
```

Easy User-land CSPRNG

Possible BC Break!

- New Reserved Method Names
  - random_bytes
  - random_int

PHP7 JumpStart

Null Coalesce Operator

https://wiki.php.net/rfc/isset_ternary

Author: Andrea Faulds

Null Coalesce Operator

```
$name = $firstName ?? "Cal";
$name .=  " ";
$name .= $lastName ?? "Evans";
```

- $name will always contain a value, even if $firstName or $lastName are null

Null Coalesce Operator

```
$this->maxCount = is_null($input->getOption('count'))
?-1:$input->getOption('count');


$this->maxCount = $input->getOption('count')??-1;
```

Null Coalesce Operator

```
$x = NULL;
$y = NULL;
$z = 3;

var_dump($x ?? $y ?? $z); // int(3)
```

Break

PHP7 JumpStart

Scalar Type Hints

https://wiki.php.net/rfc/scalar_type_hints_v5

Author: Anthony Ferrara (original Andrea Faulds)

Scalar Type Hints

- int
- float
- string
- bool

Scalar Type Hints

```
function add(float $a, float $b) {
    return $a + $b;
}

$returnValue = add(1.5, 2.5); // int(4)
// Works

$returnValue = add("1 foo", "2");
// PHP 5.6 and below gives a Notice
// PHP7 TypeError

$returnValue = add(1, 2); // int(3)
// Widening
```

Scalar Type Hints

```
declare(strict_types=1);
```

Scalar Type Hints

Widening

- The only type casting done in strict mode
- Integers can be "widened" into floats.

```
declare(strict_types=1);

function add(float $a, float $b) {
    return $a + $b;
}
var_dump(add(1, 2)); // float(3)
```

Return Type Declarations

```
function foo(): array {
    return [];
}
```

Return Type Declarations

- When a sub-type overrides a parent method then the return type of the child must exactly match the parent and may not be omitted.
- If a mismatch is detected during compile time then E_COMPILE_ERROR will be issued.
- If a type mismatch is detected when the function returns then E_RECOVERABLE_ERROR will be issued.

Return Type Declarations

Not Allowed

- `__construct()` cannot declare a return type
- `__destruct()` cannot declare a return type
- `__clone()` cannot declare a return type

Return Type Declarations

Not Allowed

- You cannot change the return type of a subclassed method.

```
Class MyClass
{
    public function foo(): array  {
        return [];
    }
}

Class MyOtherClass extends MyClass
{
    public function foo(): MyClass  {
        return new MyClass();
    }
}
```

Return Type Declarations

ALLOWED

```
Class MyClass {
    function make() : MyClass
    {
        return new MyClass();
    }

Class MyOtherClass extends MyClass {

    function make() : MyOtherClass
    {
        return new MyOtherClass();
    }
}
```

PHP7 JumpStart

Group Use Declarations

https://wiki.php.net/rfc/group_use_declarations

Author: Márcio Almada

Group Use Declarations

New group use syntax

Current use syntax

```
use FooLibrary\Bar\Baz\ClassA;
use FooLibrary\Bar\Baz\ClassB;
use FooLibrary\Bar\Baz\ClassC;
use FooLibrary\Bar\Baz\ClassD as Fizbo;

use FooLibrary\Bar\Baz\{
  ClassA,
  ClassB,
  ClassC,
  ClassD as Fizbo };
```

Group Use Declarations

Before

```
namespace MyProj\Command;

use MyProj\Traits\WritelineTrait;
use MyProj\Traits\TwitterErrorTrait;
use MyProj\Models\Person;
use MyProj\Twitter;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input
\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output
\OutputInterface;
```

Group Use Declarations

After

```php
namespace MyProj\Command;

use MyProj\ {
    Traits\WritelineTrait,
    Traits\TwitterErrorTrait,
    Models\Person,
    Twitter
};

use Symfony\Component\Console\ {
    Command\Command,
    Input\InputInterface,
    Input\InputOption,
    Output\OutputInterface
};
```

New Features that do not break things

- INTEGER SEMANTICS
- UNICODE CODEPOINT ESCAPE SYNTAX
- ARRAY CONSTANTS
- EXPECTATIONS

Integer Semantics

https://wiki.php.net/rfc/integer_semantics

Author: Andrea Faulds

Integer Semantics

- Integer to float conversion is untouched.
- Should not cause any BC break unless you depends on the value of NAN or INF. (which can change based on the platform)

Integer Semantics

- Instead of being undefined and platform dependent, NaN and Infinity will always be zero when casted to integer

```
var_dump((int)NAN);

// Pre PHP 7: int(-9223372036854775808)
// PHP 7: int(0)
```

Integer Semantics

- Bitwise shifts by negative numbers of bits will be disallowed (throws E_WARNING and gives FALSE, like a division by zero)

```
var_dump(1 << -2);

// Pre PHP 7: int(4611686018427387904)
// PHP 7: bool(false) and E_WARNING
```

Integer Semantics

- Right bitwise shifts by a number of bits beyond the bit width of an integer will always result in 0 or -1 (depending on sign), even on CPUs which wrap around

```
var_dump(8 >> 64);

// Pre PHP 7: int(8)
// PHP 7: int(0)
```

PHP7 JumpStart

Unicode Codepoint Escape Syntax

https://wiki.php.net/rfc/unicode_escape

Author: Andrea Faulds

Unicode Codepoint Escape Syntax

- \u{XXXX} format
  - Supports the entire Unicode set, not just the Basic Multilingual Plane. (We get all the emoji!)
  - Disambiguate
    “\u1F35400” vs. “\u{1F354}00”
    🍔00
- Only works in double quoted strings and HEREDOCs

Unicode Codepoint Escape Syntax

Example

```
echo "\u{202E}Reversed  // outputs txet desreveR

echo "\u{1F602}"; // outputs 😂
```

PHP7 JumpStart

Array Constants

Array Constants

PHP 5.6

```
<?php
const NAME = "Cal";
const EMAIL = NAME .  "@calevans.com";
const FRUIT = ["apple","orange","banana"];
echo EMAIL;
```

PHP 7

```
<?php
define("FRUIT",["apple","orange","banana"]);
echo "\n";
print_r(FRUIT);
echo "\n";
```

PHP7 JumpStart

Expectations

https://wiki.php.net/rfc/expectations

Author: Joe, Dmitry

Expectations

- Assertions with custom exceptions

```
ini_set("assert.exception", 1);

class CustomError extends AssertionException {}

assert(false, new CustomError("Some error
message"));
```

Expectations

- Two new ini settings
```
zend.assertions  = 1
assert.exception = 0
```

- zend.assertions has 3 possible values
  - `1` = Development Mode
  - `0` = Ignore - Not Zero Cost but no affect on the code
  - `-1` = Production Mode - Zero Cost // New

Expectations

- Namespaces

```
\assert(false);
// always fires the system function

assert(false);
// Looks for assert() in current namespace.
Defaults to the system function
```

zend.assertions settings apply to both

Conclusion

- Wrapup of what we've discussed
- Moving to PHP 7 will be painless if you have good unit tests.

The Big 3

- PHPNG
- Exceptions in the Engine
- Scalar Type Hints

Conclusion

- Importance of keeping current
- Keeping your PHP upgraded
- Keeping your skills upgraded

Thank You

Questions?