PHP7

Dmitry Stogov

Кто Я?

- Работаю в IT с 1991
- Первое знакомство с PHP в 2002
- Автор Turck MMCache (eAccelerator)
- Работаю в Zend Technologies с 2004
- Автор ext/soap
- Автор pecl/perl
- Один из ведущих разработчиков PHP
- Майнтэйнер Zend OPcache
- Лидер проекта PHPNG

PHPNG (New Generation)

- Рефакторинг
- Основная цель — повысить производительность путем изменения базовых структур данных и заложить фундамент для будущих улучшений
- Ответвление от главной ветки PHP в Январе 2014
- Никаких изменений для пользователей (только внутренности)
- 100% совместимость в поведении с PHP-5.6

- Две недели ушло только на то, что бы скомпилировать ядро
- Еще через две недели мы смогли запустить bench.php
- Еще 1.5 месяца потребовалось, что бы запустить Wordpress
- Еще через месяц (в начале Мая) мы открыли исходники проекта
- Смержен обратно в главную ветку, как база для PHP7 в Августе 2014

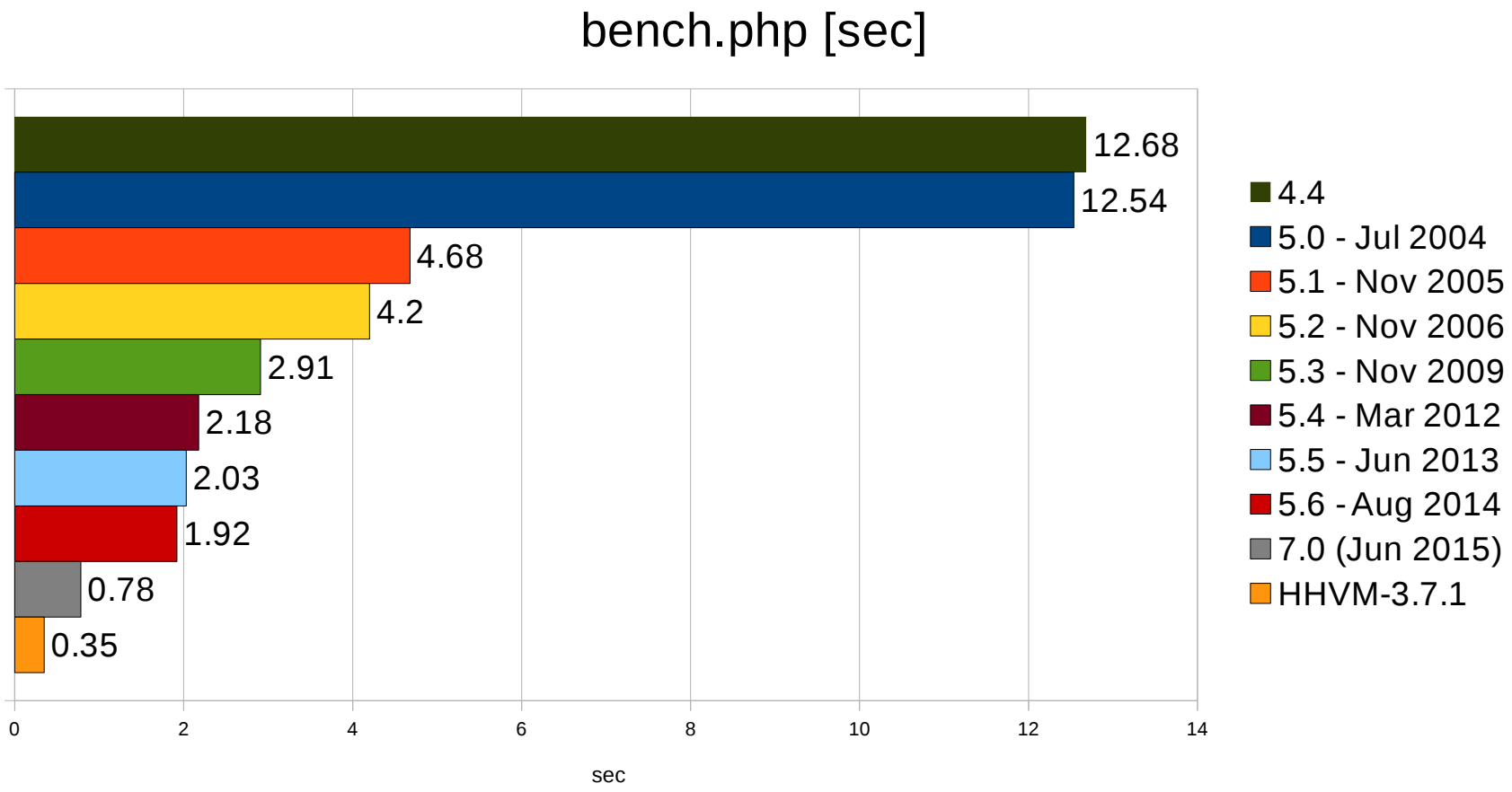
PHP7

- Было решено после PHP5 выпускать PHP7, пропустив PHP6
- PHP7 несет несколько значительных нововведений
- Множество изменений во внутренних API
- Чистка и удаление устаревших фич
- Закрыт для добавления новых фич с Марта 2015
- Релиз PHP-7.0.0RC1 в Июне 2015
- GA релиз запланирован на Октябрь 2015
- PHP приложения и расширения могут потребовать адаптации

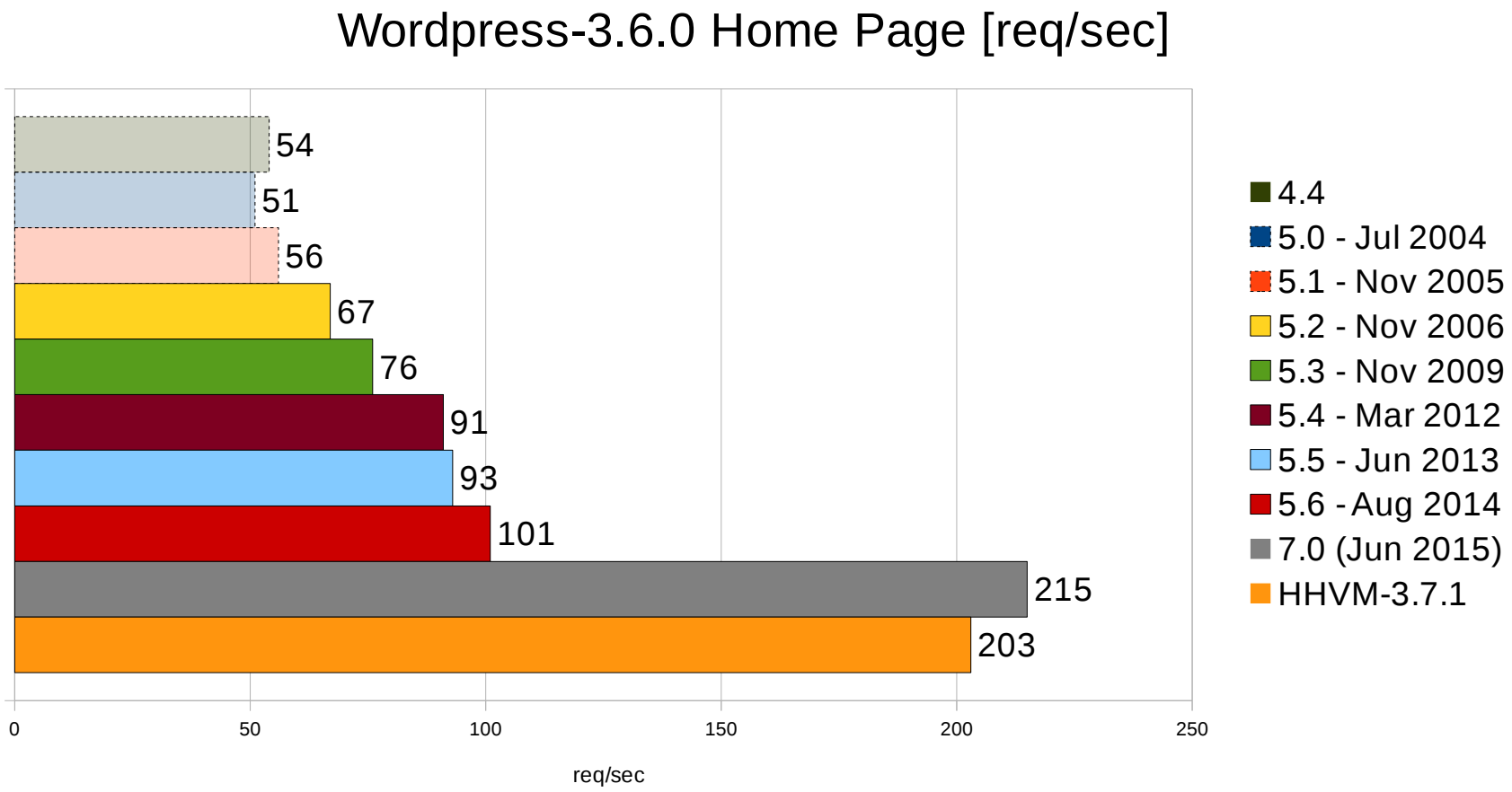
PHP7 Основные Изменения

- Новый уровень производительности
- Scalar type hints and return types
- EngineExceptions вместо фатальных ошибок
- Uniform variable syntax
- Анонимные классы
- Zero-cost assert()
- Generator delegation and return expressions
- Новые операторы <=> и ??
- Новые функции random_bytes(), random_int(), intdiv(), preg_replace_callback_array(), Closure::call()
- Новый класс IntlChar
- Эскейп последовательность для Unicode
- Чистка поведения (foreach, list, switch/default, деление на ноль, конвертация NaN и INF, сдвиг)

Производительность PHP7



Производительность PHP7



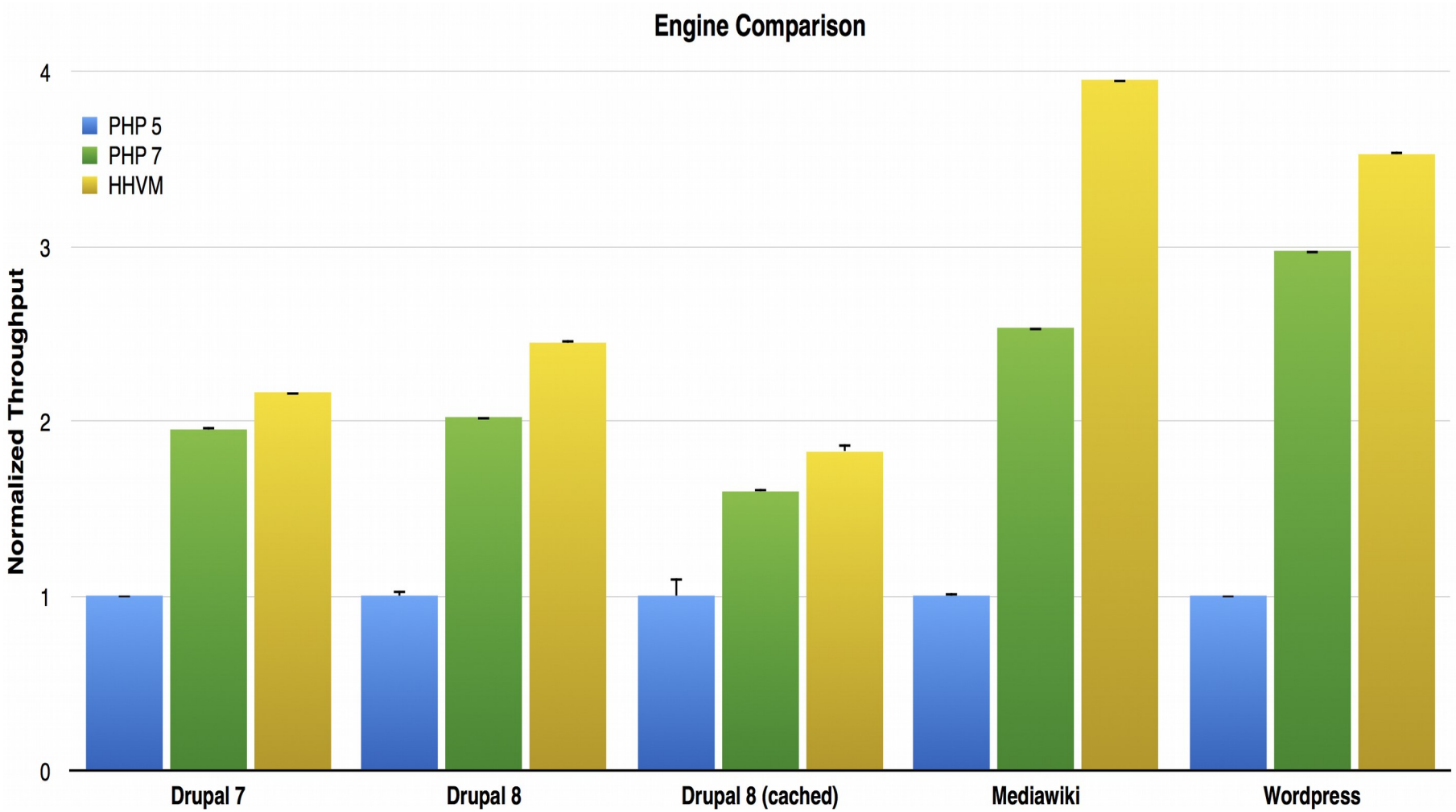
Производительность PHP7

- Время исполнения 1000 запросов к Wordpress-3.6.0
- php-cgi -T 1000 /.../wordpress/index.php > /dev/null

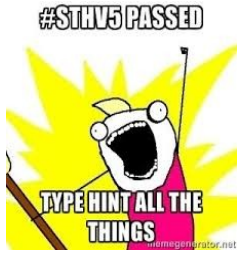
Производительность PHP7 [req/sec]

	PHP-5.6	PHP-7	difference	HHVM-3.7.1	difference
ZF1 Hello	1,144	2,055	80%	1,394	47%
ZF2 Test	258	520	102%	370	41%
Drupal 7	192	336	75%	308	9%
SugarCRM (login)	134	279	108%	189	48%
Magento (home)	43	70	63%	62	13%
Mediawiki	46	75	63%	85	-12%
Laravel	304	523	72%	410	28%

Производительность PHP7 глазами Facebook



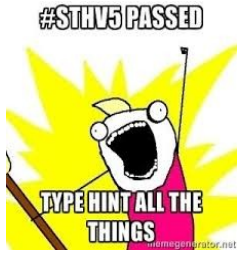
Scalar type hints




```
function add(int $a, int $b) {
    return $a + $b;
}
var_dump(foo(5, "3")); // int(5)
var_dump(foo(5, "abc")); // throws TypeException
```

- 4 предопределенных скалярных типа: bool, int, float, string
- Классы с такими именами теперь запрещены (а так же классы с именами false, true и null)
- Определение типа гарантирует тип аргумента внутри функции (если тип параметра не соответствует, но совместим, он будет сконвертирован, иначе будет вызвано исключение TypeException).
- По умолчанию используется нестрогое соответствие типов (те же правила, которые испльзовались для проверки соответствия типов внутренних функций в PHP5)
- Больше нет E_RECOVERABLE_ERROR при несоответствии типов

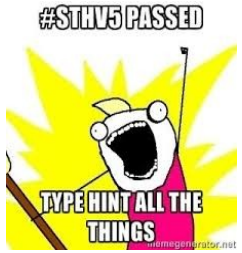
Scalar type hints (strict mode)




```
declare(strict_types=1);
function add(int $a, int $b) {
    return $a + $b;
}
var_dump(foo(5, "3")); // throws TypeException
```

- Определенный и фактический типы должны быть идентичны (за исключением конвертации int → float).
- declare(strict_types=1) должен быть самым первым оператором в файле
- declare(strict_types=1) определяет поведение вызываемых функций (а не определяемых).
- Поведение вызываемых внутренних функций так же изменяется

Scalar type hints (strict mode)




```php
<?php
declare(strict_types=0);
function add(int $a, int $b) {
    return $a + $b;
}
```

```php
<?php
declare(strict_types=1);
function add(int $a, int $b) {
    return $a + $b;
}
```

```php
<?php
declare(strict_types=0);
include("add1.php");
var_dump(foo(5, "3")); // int(8)
```

```php
<?php
declare(strict_types=0);
include("add2.php");
var_dump(foo(5, "3")); //  int(8)
```

```php
<?php
declare(strict_types=1);
include("add1.php");
var_dump(foo(5, "3")); // exception
```

```php
<?php
declare(strict_types=1);
include("add2.php");
var_dump(foo(5, "3")); // exception
```

Return type hints


```
function foo($a): int {
    return $a;
}
var_dump(foo("3")); // int(3)
```

- Можно определить тип возвращаемого значения, как скалярный (bool, int, float, string), array или имя класса
- Правила совместимости типов такие же, как и для параметров
- Strict mode определяет поведение определяемой функции (а не вызываемой)
- No nullable declarations (мы не можем вернуть NULL, если фунция определена, как возвращающая объект класса)

Engine Exceptions

- Почти все фатальные ошибки времени исполнения заменены на исключения EngineException
- Parser errors заменены на ParserException
- Ошибки компиляции остались фатальными
- BaseException общий базовый класс для Exception, EngineException и ParserException
- Приложения могут ловить новые исключительные ситуации и востанавливаться.
- Изменился формат сообщений об ошибках (Fatal error: Uncaught EngineException: … + backtrace)

```
try {
    invalid_func();
} catch (EngineException $e) {
    echo $e->getMessage(); // Call to undefined function invalid_func()
}
```

Uniform Variable Syntax

- Единообразный и полный синтаксис для доступа к элементам переменных
- Элементы могут использоваться в любом порядке
  - Foo::$a[1](2)->c[3]
  - foo()();
  - echo (function add($a,$b){return $a+$b;})(5, 3); // 8
- Изменен приоритет ореператора обращения к переменной по имени ($$)

syntax	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']()

Анонимные классы

```
System::setLogger(new class($file) inmlements AbstaractLogger {
    function __construct($file) {
        $this->file = $file;
    }
    function log($msg) {
        fprintf($this->file, “%s\n”, $msg);
    }
});
```

- Похоже на анонимные функции
- Решение неполное (мы можем только создавать экземпляры анонимных классов, но не можем использовать их по другому)
- get_class() возвращает “искаженное” имя

Zero-cost assert()

- **assert()** полезен при отладке, но вызывает накладные расходы в prodaction
- PHP7 ввел новую ini директиву **zend.assertions**
  - 1 генерировать и исполнять код проверок (development)
  - 0 генерировать код проверок, но не исполнять его (production + debugging)
  - -1 не генерировать код проверок (zero-cost production)
- Возможно переключение между 0 и 1 во время исполнения для разрешения/запрещения проверок
- Так же assert() может вызывать исключения **AssertException**, если поставить **assert.exception**=1

Generator delegation

- Введен новый синтаксис **yield from** для делегирования значений, возвращаемых генератором массиву или объекту **Traversable**
- Генераторы также являются **Traversable**
- **Generator::send()** передает значение делегированому генератору
- Позволяет разбивать генераторы на простые части
- Разрешает кооперативную многозадачность (используя **yield from** вместо **return**)
- Ключи, возвращаемые из составного генератора, могут повторяться

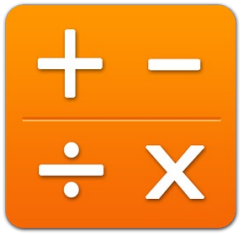
```
function gen() {
    foreach ([1,2,3] as $v) yield $v;
}
foreach (gen() as $v) {var_dump($v);} // 1,2,3
function gen2() {
    yield from gen();
    yield from [4,5,6];
}
foreach (gen2() as $v) {var_dump($v);} // 1,2,3,4,5,6
```

Generator return expressions

- В генераторах разрешен оператор **return**
- Введен метод **Generator::getReturn()** для получения возвращаемого значения
- Это позволяет определять конечное значение кооперативно-асинхронных задач

```
function sum($a) {
        $ret = 0;
        foreach ($a as $v) {
                yield $ret += $v;
        }
        return $ret;
}
$task = sum([1,2,3,4,5,6,7,8,9,10]);
foreach ($task as $tmp) {
        echo ".";
}
echo $task->getReturn(); // .........55
```

Новые Операторы



- $a <=> $b (сравнение или “spaceship”)
  - returns -1 if ($a < $b)
  - returns 0 if ($a == $b)
  - Returns 1 if ($a > $b)
  - usort($data, function ($a, $b) { return $a <=> $b;})
- $a ?? $b (null coalesce)
  - isset($a) ? $a : $b
  - $username = $_GET['user'] ?? 'guest';

Новые Функции

$f(x)$

- intdiv(int $a, int $b): int
- function rundom_bytes(int $length): string
- function rundom_int(int $min, int $max): int
- function preg_replace_callback_array(array $pattern_map, $subject [, int $limit [, int &$count]])

```
preg_replace_callback_array([
    “/<([A-Z]*)>/m” => function ($matches) {…},
    “/<(//[A-Z]*)>/m” => function ($matches) {…},
    $text);
```

- function Closure::call($obj, ...)

```
$validator = function ($num) {
    if ($this->num !== $num) ...
}
$num = 0;
foreach ($children as $obj) {
    $validator::call($obj, ++$n);
}
```

Класс IntlChar и эскейп для Unicode


```
class IntlChar {
    const CODEPOINT_...
    const PROPERTY_...
    const CHAR_CATEGORY_...
    const CHAR_DIRECTION_...
    const BLOCK_CODE_...
    ...
    static function chr($char);
    static function ord($char);
    ...
    static function isUAlphabetic($char): bool;
    static function islower($char): bool;
    ...
    static function charDirection($char): int;
    static function charType($char): int;
    ...
    static function tolower($char);
    static function toupper($char);
}

echo "\u{1F602}"; // outputs 😂
```

Контекстно зависимый сканнер

- Теперь возможно использовать зарезервированные слова (if, do, ...) как имена свойств, методов и констант классов

```
class C {
    const global = 1;
    public  $list = null;
    static function if($a) {
        return (bool)$a;
    }
}
var_dump(C::if(true));
$x = new C;
var_dump($x->list)
var_dump(C::global);
```

- Использование некорых слов лимитировано: **class**, **static**, **abstract**, **final**, **private**, **protected**, **public**.
- get_token_all($text, **TOKEN_PARSE**); // для распознавания слов

Чистка поведения (foreach)

- foreach для элементов массива по значению не использует и не модифицирует внутреннюю позицию массива. Внешние модификации массива игнорируются.

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

  - PHP 5.6: 1-2,2-2,3-2,
  - PHP 7: 1-1,2-1,3-1,

```
$a = [1, 2, 3]; $b = &$a;
foreach ($a as $v) {
    echo $v . ","; unset($a[1]);
}
echo “\n”;
```

  - PHP 5.6: 1,3,
  - PHP 7: 1,2,3,

Чистка поведения (foreach)

- foreach для элементов массива по ссылке изменяет внутреннюю позицию массива на каждой итерации (как и в PHP5).
  - добавленные в конец элементы итерируются
  - поддерживаются вложенные итераторы
  - Возможно изменение итерируемых массивов внутренними функциями
- foreach для свойств объектов (по значению и ссылке) работает по тем же правилам, что и для элементов массива по ссылке
- Смотри https://wiki.php.net/rfc/php7_foreach

Чистка поведения (присваивание list)

- PHP 5

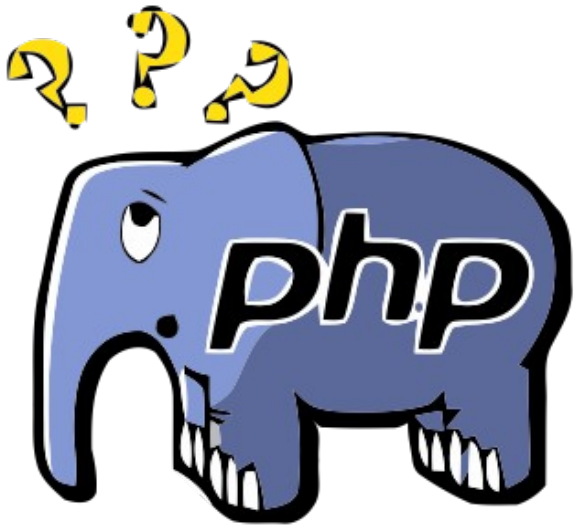
```
list($a,$b) = "ab";
var_dump($a,$b); // NULL, NULL
$a[0] = "ab";
list($a,$b) = $a[0];
var_dump($a,$b); // “a”, “b” (undocumented)
```

- В PHP 7 поведение однозначно определено

```
list($a,$b) = "ab";
var_dump($a,$b); // NULL, NULL
$a[0] = "ab";
list($a,$b) = $a[0];
var_dump($a,$b); // NULL, NULL
```

Чиска поведения (switch с двумя default)

```
$n = 4;
switch($n) {
        case 1: echo “1”;
        case 2: echo “2”;
        case 3: echo “3”;
        default: echo “4”;
        case 5: echo “5”;
        case 6: echo “6”;
        default: echo “7”;
        case 8: echo “8”;
}
```



- PHP 5 напечатает … ?
- В PHP 7 switch с несколькими default запрещен
  - PHP Fatal error: Switch statements may only contain one default clause

Чиска поведения (математика)

- Конвертация NaN и INF в целое число дает 0

```
var_dump((int)NAN);
// int(-9223372036854775808) → int(0)
var_dump((int)INF);
// int(4611686018427387904) → int(0)
```

- Сдвиг на отрицательное число бит запрещен

```
var_dump($x << -1);
// int(...) → Exception: Bit shift by negative number
```

- Сдвиг на число бит, превышающее разрядность, дает 0 (вск биты 0) или -1 (все биты 1)

```
var_dump(1 << 65);
// int(2) → int(0)
var_dump(-3 >>65);
// int(.-2) → int(-1)
```

- Деление на ноль вызывает предупреждение и возвращает INF
- Остаток от деления на ноль вызывает исключение "Division by zero"

Чистка поведения (несовместимый контекст)

- Удалена поддержка вызова методов объектов (non static) из несовместимого контекста с использованием синтаксиса вызова статических методов

```
class A {
    public function test() { var_dump($this); }
}

class B { // Note: B does NOT extend A
    public function callNonStaticMethodOfA() { A::test(); }
}

$b = new B;
$b->callNonStaticMethodOfA();

// Deprecated: Non-static method A::test() should not be called statically
// Notice: Undefined variable $this
NULL
// (in PHP-5 it was object(B))
```

Конструкторы в стиле PHP4

- Поддержка конструкторов в стиле PHP-4 объявлена устаревшей

```
class A {
    function A() { // use “function __construct()” instead
      $this->x = 100;
    }
}
$a = new A();
var_dump($a->x);

// Было
    int(100);
// Стало
    PHP Deprecated:  Methods with the same name as their class will not be constructors in a future version of PHP; A has a deprecated constructor
    int(100)
```

Чистка поведения (другое)

- Присваивание по ссылке сначала вычисляет левое выражение

```
$a = []; $a["a"] =& $a["b"]; $a["a"] = 1;
// ["b" => 1, "a" => 1] → ["a" => 1, "b" => 1]
```

- Присваивание в list вычисляет выражения слева на право

```
list($a[], $a[]) = [1, 2];
// [2, 1] → [1, 2]
```

- Запрещены параметры с одинаковыми именами

```
function foo($_, $_) {}
```

- func_get_arg(), func_get_args() не показывают оригинальные значения параметров

```
function foo($a) {$a=2; var_dump(func_get_arg(1));} foo(1);
// 1 → 2
```

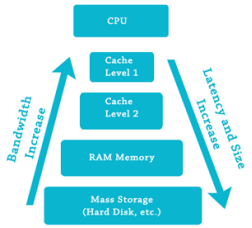
- Шестнадцатеричные числа в строках больше не поддерживаются

```
var_dump("0x123" == "291")
// true → false
```

PHP7 Основные Изменения

- Удалены расширения (mssql, sybase, mysql, ereg)
- Удалена поддержка неподдерживаемых Web серверов (IIS, Apache 1.*, AOL, tux, ...)
- Удалена поддержка альтернативных PHP тэгов (<%, <%=)
- Внутренние изменения
  - Рефакторинг базовых струтур данных и API (PHPNG)
  - Компилятор использует AST (Abstract Syntax Tree)
  - Лучшая поддержка 64-битных платформ
  - Native TLS (Thread Local Storage) (TSRM macros)
  - Быстрое API для разбора параметров внутренних функций
  - ext/json заменен на jsond

opcache.file_cahe (experimenal)

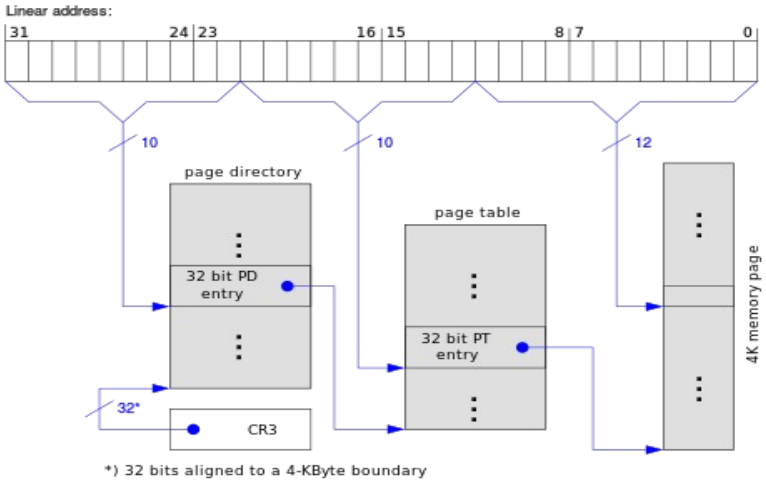


- PHP7 opcache дает возможность дополнительно кэшировать скрипты на диске (запрещено по умолчанию)
- configure ... --enable-opcache-file
- mkdir /tmp/opcahe
- Додавить opcache.file_cahce=/tmp/opcache в php.ini
- Уменьшает пиковые нагрузки при рестарте PHP и переполнении SHM кэша
- Улучшает время отклика в эти моменты в 2-3 раза
- Может использоваться в сценариях апгрейда (можно предварительно загрузить дисковый кэш)
- file_cache может работать вообще без SHM

Хотите еще быстрее - HUGE TLB / HUGE PAGE

- CPU использует виртуальную память с 4KB страницам
- CPU TLB cahce содержит всего 64-512 entries => 256KB-2M
- TLB промахи требуют много времени для обработки
- Решение – страницы размером 2MB
- PHP-7, если может, использует 2M страницы для менеджера памяти и разделяемой памяти opcahce, где хранятся скомпилированые скрипты
- **grep "Huge" /proc/meminfo**

```
AnonHugePages:      2048 kB
HugePages_Total:     512
HugePages_Free:      504
HugePages_Rsvd:       56
HugePages_Surp:        0
Hugepagesize:       2048 kB
```



- https://wiki.debian.org/Hugepages#Enabling_HugeTlbPage

Хотите еще быстрее - PGO/FDO build

- **make prof-gen**
- **sapi/cgi/php-cgi -T 3000 /.../wordpress/index.php > /dev/null**
- **make prof-clean**
- **make prof-use**

- Скорость +8%
- Объем кода -8% (size sapi/cli/php)

Questions?