

РАЗРАБОТКА ВЕБ-ПРОЕКТОВ

7-я Международная
конференция
эффективной
разработки на PHP

29-30 мая 2008
<http://PHPConf.ru>
2008@phpconf.ru
+7 (495) 585-92-61



Что нового в PHP 5.3

Дмитрий Стогов

- сотрудник Zend Technologies
- отдел Advanced Technologies
- активный разработчик PHP и ZE
- автор и мантейнер ext/soap
- мантейнер поддержки FastCGI в PHP
- автор компоненты OpenID в Zend Framework
- автор Turck MMCache

Почему PHP 5.3?

- PHP 5.2 существует уже 1.5 года. В нем найдено несколько серьезных ошибок, которые не могут быть исправлены без потери бинарной совместимости.
- В PHP 6, из-за перехода на Unicode, перестанет работать большое количество наработанного кода.
- Для PHP 6 было разработано много интересных дополнений и улучшений.
- PHP 5.3 будет содержать большинство улучшений разработанных для PHP 6, но будет способен выполнять существующий код без каких-либо изменений.

- Нововведения в языке
- Расширение системы конфигурирования
- Сборщик мусора
- Улучшенная производительность
- Исправленные ошибки
- Новые расширения `ext/phar` и `ext/intl`
- Множество улучшений в расширениях

- namespaces
- Расширения ООП
 - Late Static Binding
 - Динамический доступ к статическим данным
`$classname::method()`, `$classname::$prop`
 - `__callstatic()`
- Оператор `goto`
- Сокращенный условный оператор `?:`
- `NOWDOC <<<'EOF'`
`EOF;`
- Константа `__DIR__`

Зачем нужны namespace-ы?

- Устраняют конфликты имен
 - Разные namespace-ы могут использовать одно и то же имя для разных целей
 - Имя внутри namespace-а имеет единственный смысл
- Namespace-ы делают имена короче
 - Имена определенные в namespace-ах имеют короткое (локальное) имя и уникальное длинное (квалифицированное) для использования за пределами namespace-а
 - Имена и namespace-ы могут быть импортированы в другие namespace-ы используя короткое “имя импорта”

- Один namespace может определяться в нескольких файлах
- В namespace-е могут определяться
 - классы
 - интерфейсы
 - функции
 - константы
 - PHP код
- В namespace-е не могут определяться
 - Глобальные переменные
- PHP не поддерживает вложенных namespace-ов
- PHP поддерживает составные имена namespace-ов (A::B)
- Почти вся работа делается во время компиляции

```
define("MY_HTTP_GET", 1);  
define("MY_HTTP_POST", 2);
```

```
class My_Http_Request {  
    function __construct(  
        $method = ZEND_HTTP_GET)  
    {  
    }  
}
```

```
function my_http_send_request(  
    My_Http_Request $request) {  
}
```

```
namespace My::Http;
```

```
const GET = 1;  
const PUT = 2;
```

```
class Request {  
    function __construct(  
        $method = GET)  
    {  
    }  
}
```

```
function send_request(  
    Request $request) {  
}
```


namespace в нескольких файлах

My/Http/Request.php

```
<?php
```

```
namespace My::Http;
```

```
class Request {  
}
```

My/Http/Response.php

```
<?php
```

```
namespace My::Http;
```

```
class Response {  
}
```

My/Http/Main.php

```
<?php
```

```
namespace My::Http;
```

```
function send(Request $req) {  
    return new Response();  
}
```

test1.php

```
<?php
```

```
require_once("My/Http/Request.php");
```

```
$x = new My::Http::Request();
```

Импорт – оператор “use”

- Импорт может быть осуществлен посредством оператора “use”
 - `use My::Http;`
- Оператор “use” может импортировать
 - Namespaces-ы
 - классы
 - интерфейсы
- Он не может импортировать
 - функции
 - константы
 - переменные
- В момент импорта можно сделать переименование
 - `use My::Http::Request as HttpRequest;`
 - `use My::Http::Request; // the same as use My::Http::Reques as Request;`
- Оператор “use” действует только на текущий файл
- Оператор “use” сам не подгружает ни каких файлов

test3.php

```
<?php
```

```
require_once("My/Http/Request.php");
```

```
use My::Http::Request;
```

```
$x = new Request();
```

test4.php

```
<?php
```

```
require_once("My/Http/Request.php");
```

```
use My::Http;
```

```
$x = new Http::Request();
```

test5.php

```
<?php
```

```
require_once("My/Http/Request.php");
```

```
use My::Http as H;
```

```
$x = new H::Request();
```

```
namespace A::B;
function foo() {
    echo __FUNCTION__;    // A::B::foo
}
class C {
    static function bar() {
        echo __CLASS__;    // A::B::C
        echo __FUNCTION__; // bar
        echo __METHOD__;    // A::B::C::bar
    }
}
```

```
namespace A::B;
```

```
function foo() {  
    echo __NAMESPACE__;  
}
```

```
$callback = "foo";  
$callback(); // global function foo()
```

```
$callback = "A::B::foo";  
$callback(); // A::B::foo()
```

```
$callback = __NAMESPACE__ . "::foo";  
$callback(); // A::B::foo()
```

```
<?php
use My::Http::Request;

function __autoload($name) {
    require_once(
        str_replace("::", "/", $name) . ".php");
}

$x = new Request(); // loads "My/Http/Request.php"
```


Разрешение коротких имен:

1. Имена импорта

```
use A::B::Foo;
use A::B::Bar as Baz;
$x = new Foo; // A::B::Foo
$x = new Baz; // A::B::Bar
```
2. Имена из текущего namespace-а

```
namespace A::B;
class stdClass {
}
$x = new stdClass(); // A::B::stdClass
```
3. Внутренние имена PHP

```
namespace A::B;
$x = new stdClass; // internal stdClass
```

Явное разрешение специальными префиксами

```
namespace A::B;
class stdClass {
}
$x = new stdClass();           // A::B::stdClass
$x = new ::stdClass();        // global stdClass
$x = new namespace::stdClass(); // A::B::stdClass
```

Разрешение длинных имен классов:

1. Имена импорта

```
use A::B::C;  
new C::D;    // class D in namespace A::B::C
```

2. Класс из другого namespace-а (как есть)

```
namespace A::B;  
new C::D;    // class D in namespace C (not A::B::C::D)
```

Имена функций и констант:

1. Функция или константа из текущего namespace-а

```
A::foo(); // function foo() in namespace A
```

2. Статический метод или константа класса

```
A::foo() // static method foo() of class A  
// A is resolved according to class resolution rules
```

Late Static Binding

```
class Singleton {
    const ID = 0;
    static $instance = array();
    static function getInstance() {
        $id = static::ID;
        if (empty(self::$instance[$id])) {
            self::$instance[$id] = new static;
        }
        return self::$instance[$id];
    }
}
class Foo extends Singleton {
    const ID = 1;
}
$x = Foo::getInstance();
```

```
class MySqlDriver {  
    const NAME = "MySQL";  
    static function execute($sql) {  
    }  
}  
  
$db = "MySqlDriver";  
echo $db::NAME;  
$db::execute("SELECT * FROM foo;");
```

```
class DummyDriver {
    const NAME = 'Dummy';
    static function __callstatic($name, $args) {
        echo static::NAME."::$name is not implemented";
    }
}

class MySQLDriver extends DummyDriver {
    const NAME = 'MySQL';
}

$db = 'MySQLDriver';
$db::execute('SELECT * FROM foo;');
```

Оператор GOTO

- Реализован для поддержки программно-генерируемого кода
- “GOTO” внутри цикла или оператора switch запрещен

```
<?php
```

```
RETRY:
```

```
try {
```

```
    ...
```

```
} catch (Exception $e) {
```

```
    recovery($e);
```

```
    goto RETRY;
```

```
}
```

Оператор ?:

$\$a \text{ ?} : \$b \text{ === } \$a \text{ ? } \$a : \$b$

- Аналог строк в одиночных кавычках

```
<?php
```

```
$a = 3;
```

```
$b = 5;
```

```
echo <<<EOF
```

```
$a+$b
```

```
EOF; // prints 3+5
```

```
echo <<<'EOF'
```

```
$a+$b
```

```
EOF; // prints $a+$b
```

```
echo <<<"EOF"
```

```
$a+$b
```

```
EOF; // printd 3+5
```

```
__DIR__ === dirname(__FILE__)
```

```
<?php
```

```
class Foo {
```

```
    const BAR = dirname(__FILE__); // error
```

```
    const BAR = __DIR__;
```

```
    public $bar = dirname(__FILE__); // error
```

```
    public $bar = __DIR__;
```

```
}
```

- Разные уствновки для разных директорий
`[PATH=/www/test]`
`error_repoting = E_ALL`
`[PATH=/www/production]`
`error_reporting = 0`
- Разные установки для разных виртуальных хостов
`[HOST=www.domain.com]`
`auto_prepend_file = "/var/www/domain/init.php"`
- Собственный аналог .htaccess
`user_ini.filename = ".user.ini"`
`user_ini.cache_ttl = 300`

- Уничтожает циклические структуры

```
<?php
```

```
$a = array();
```

```
$a[0] =& $a;
```

```
// refcount is 2
```

```
unset($a);
```

```
echo gc_collect_cycles();
```

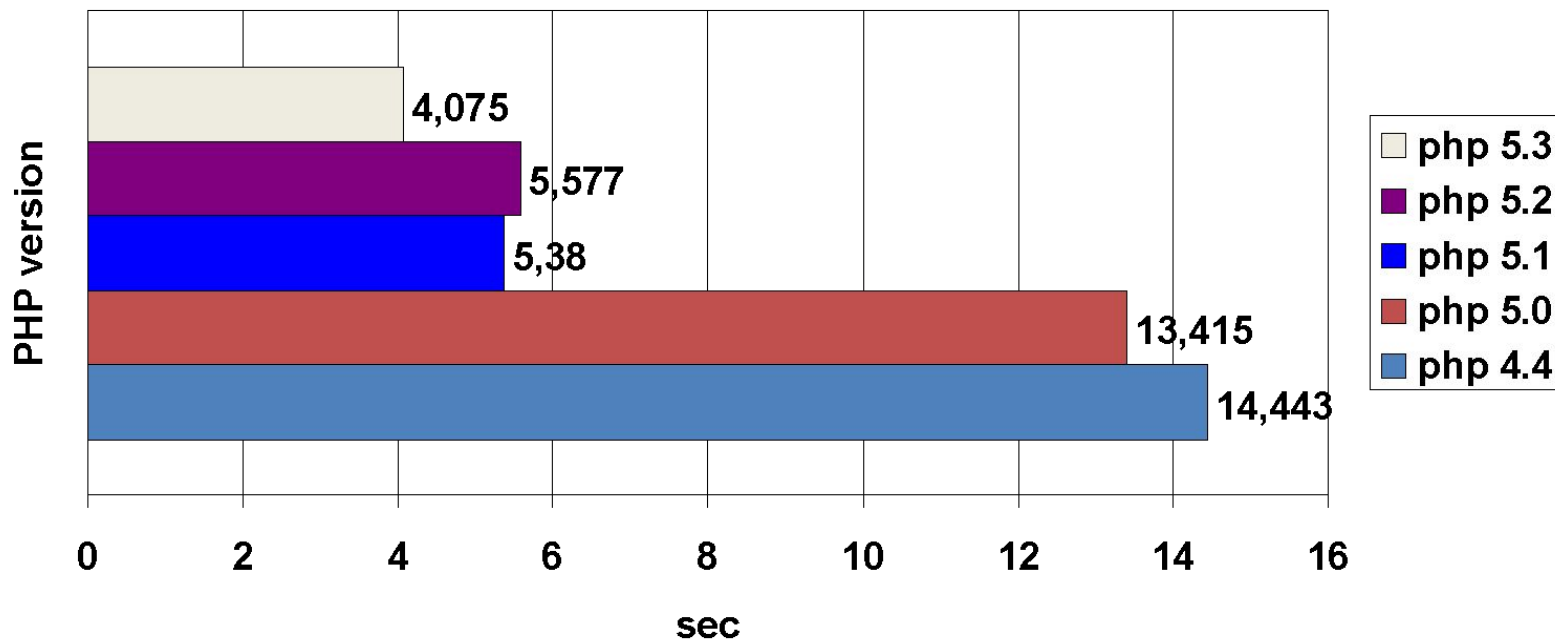
```
// 1 – number of
```

```
// collected variables
```

```
gc_disable();
```

```
gc_enable();
```

\$ php bench.php



Вопросы?