www.phparch.com

November 2019
Volume 18 - Issue 11

php[architect]

Object Orientation

Object-Oriented Programming: A Primer, Part One

Building PHP Extensions With C++

How To Avoid Job Stagnation

ALSO INSIDE

Education Station:
Dependency Injection, Part One

Community Corner:
San Diego PHP

Pragmatic PHP:
Think Like a Computer

Security Corner:
Responsible Encryption

The Workshop:
What's New in PHP 7.4

finally{}:
Frameworks Don't Make Any Sense

PHP[TEK] 2020

Prepare to join us for the 15th edition of our premier conference. Moving to Nashville in 2020.

May 18-21

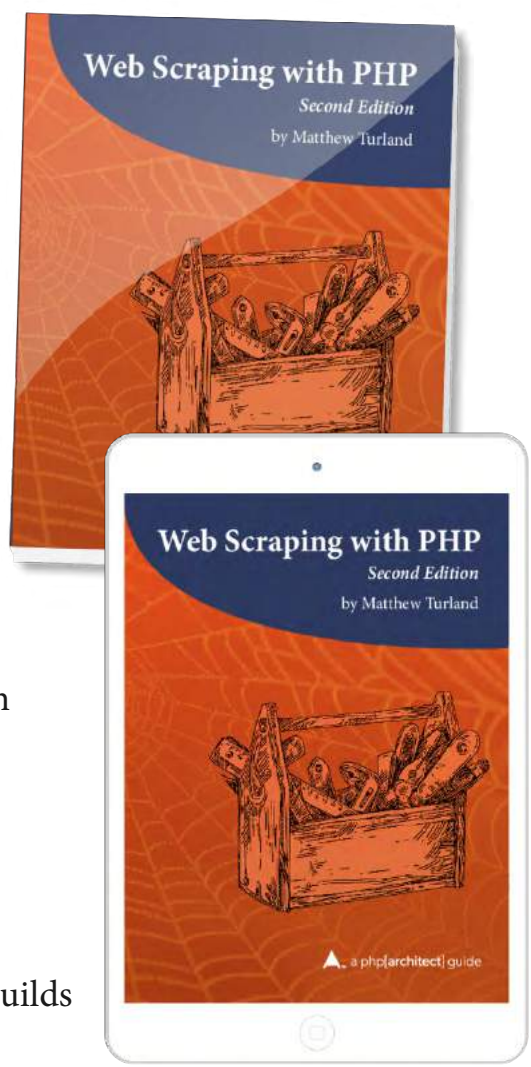
tek.phparch.com

a php[architect] guide

Read a Sample Online

Integrating with another web site but an API is not available?

Web scraping is a time-honored technique for collecting the information you need from a web page. In this book, you'll learn the various tools and libraries available in PHP to **retrieve, parse, and extract data** from HTML.



Web Scraping with PHP, 2nd Edition includes updates to the techniques of the first edition to account for modern PHP 7 based libraries written to more easily interact with web markup and data.

- HTTP requests and responses
- PHP's HTTP Stream wrapper
- Using the cURL extension
- Working with the pecl_http extension
- Parsing responses with Guzzle
- Zend Framework's HTTP classes
- An overview of Symfony's libraries for web automation
- Writing a client from scratch
- Extensions for parsing and tidying XML and HTML
- Using regular expressions
- … and more.

Written by PHP professional Matthew Turland, this book builds on his expertise in creating custom web clients.

Available in Print, PDF, EPUB, and Mobi.

Order Your Copy

http://phpa.me/web-scraping-2ed

What's New in PHP 7.4

Joe Ferguson

PHP 7.4 brings typed properties, arrow functions, coalesce assignment operators, and more. PHP 7.4.0RC4 was released on October 17th. There's still plenty of time as the current planned date for general availability of 7.4 is November 28th, 2019, according to the PHP 7.4 timetable[1].

Typed Properties

In January[2] and February[3] 2019 I wrote a two-part series *The Road to PHP 7.3* where we used static analyzers Phan and PHPStan to analyze our code to prepare for upgrading PHP versions. In this series, we covered the idea of adding static typing to our code allowed static analyzers to know more about our code and enabled these tools to help us find bugs without running the code itself. We have been able to type-hint objects and interface variables since PHP 5, PHP 7 introduced scalar type hints, and 7.2 saw the addition of the `object` type.

The next step for the ability to strongly type your PHP application is coming in 7.4 in the form of typed properties[4]. A benefit of strongly typed code is we can use a static analyzer such as PHPStan or Phan to find potential errors without executing any code. However, these tools only know what they can read in our source. If we add more types to our applications, these static analysis tools become much smarter in finding bugs and highlighting problems in code. If we look back at our PHP Easy Math project[5], we can see our `Addition` class is using annotations to tell our IDE what our parameter and return types should be in Listing 1.

We could refactor our code to resemble Listing 2.

Granted, this does change how we call our class by shifting our parameters from the method to the class; there's still a bit of cruft there with our DocBlock annotations type hinting our parameters `$x` and `$y`.

Listing 1

```
1. <?php
2.
3. namespace EasyMath;
4.
5. class Addition
6. {
7.     /**
8.      * Sum 2 numbers
9.      * @param float $x
10.     * @param float $y
11.     * @return float
12.     */
13.    public function add($x, $y): float {
14.        return $x + $y;
15.    }
16. }
```

Listing 2

```
1. <?php
2.
3. namespace EasyMath;
4.
5. class Addition
6. {
7.     /** @var int $x */
8.     private $x;
9.     /** @var int $y */
10.    private $y;
11.
12.    public function __construct(int $x, string $y) {
13.        $this->x = $x;
14.        $this->y = $y;
15.    }
16.
17.    public function add(): int {
18.        return $this->x + $this->y;
19.    }
20. }
```

1 *PHP 7.4 timetable:* https://wiki.php.net/todo/php74

2 *January:* https://phparch.com/magazine/2019/jan/

3 *February:* https://www.phparch.com/magazine/2019/feb/

4 *typed properties:* https://wiki.php.net/rfc/typed_properties_v2

5 *PHP Easy Math project:* https://github.com/svpernova09/php-easy-math

Listing 3

```
1. <?php
2.
3. namespace EasyMath;
4.
5. class Addition
6. {
7.     public int $x;
8.     public int $y;
9.
10.    public function __construct(int $x, string $y) {
11.        $this->x = $x;
12.        $this->y = $y;
13.    }
14.
15.    public function add(): int {
16.        return $this->x + $this->y;
17.    }
18. }
```

PHP 7.4 and typed properties allow us to remove those annotations completely while still retaining our types, as you can see in Listing 3.

This change supports all types except for `void` and `callable`. The PHP internals team decided `void` was not useful and unclear in many cases. `callable` was not supported due to the requirement of context around its use.

Arrow Functions

The arrow functions RFC[6] brings the short function syntax to PHP 7.4. If you've worked with modern JavaScript, you may already be aware of this syntax, as many frameworks have begun to leverage it for closures. I was excited to see this RFC included in PHP because I'm a fan of its usage in JavaScript. The result is a more readable code without as much boilerplate, which adds to readability in its own right.

If we dive into an example of using `array_map` to pull array values from keys this would be the code you'd write today in PHP 7.x:

```
function array_values_from_keys($arr, $keys) {
    return array_map(function ($x) use ($arr) {
        return $arr[$x];
    }, $keys);
}
```

We could remove some of the cruft refactoring this method:

```
function array_values_from_keys($arr, $keys) {
    return array_map(fn($x) => $arr[$x], $keys);
}
```

The syntax boils down to `fn(parameters) => expression`. Variables in the expression which were defined in their parent scope are used as expected without invoking the `use($variable)` syntax. If we wanted to skip using the `Addition` class

6 *arrow functions RFC: https://wiki.php.net/rfc/arrow_functions*

from our previous PHP Easy Math library, we could use this short syntax to squash our three-line function into one:

```
$y = 1;

$getTotal = function ($x) use ($y) {
    return $x + $y;
};

$getTotal(42); // 43
```

Becomes:

```
$y = 1;
$getTotal = fn($x) => $x + $y;
$getTotal(34); // 35
```

It's important to remember the goal of this change is to reduce the amount of boilerplate code in your applications, which increases readability. As developers, we are human code linters, and anything we can do to increase readability directly makes our code easier to understand. Also, remember it's important to have moderation in all things, including moderation. You shouldn't refactor your entire codebase to a bunch of one-liner statements just because you can. Keep your code easy to understand, and remember there may be others who follow behind you in maintaining a codebase.

Coalesce Assignment Operator

While PHP 7.0 brought us the Null Coalesce Operator[7] which allows us to write code such as:

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

The new coalesce assignment operator in PHP 7.4 brings us the ability to simplify this code to the following:

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

This new operator also has the benefit of more readable code, since we're not duplicating `$_GET['user']` in a single line. In our example, if `$_GET['user']` is `null`, the value `nobody` will be assigned. An important distinction is `??` is a comparison operator while `??=` is an assignment operator.

Spread Operator In Array Expressions

PHP 5.6 bought us the new argument unpacking[8] feature. This feature allows us to use `...$var` syntax to "unpack" the contents of the argument `$var` in a function call. Given the following array, we can clean up our code and also support a variable amount of arguments in our functions (see Listing 4).

The important reason to use the spread operator `...` is because the performance is faster since the operator is a language structure as opposed to a function like `array_merge()`. The code comes out more readable in my experience.

While the spread operator as isn't new to PHP 7.4, understanding how it works is important because 7.4 brings us the ability to use the operator in array expressions. We can also mix and match array elements with the spread operator such as:

```
<?php
$pets = ['dog', 'cat', 'fish'];
$wildlife = ['bear', 'moose', 'squirrel'];
$animals = [...$pets, 'bigfoot', ...$wildlife];
// $animals: dog, cat, fish, bigfoot,
// bear, moose, squirrel
```

In PHP 7.4, the code runs without any output. In PHP versions 7.3 and lower, we'll see a parse error: `Parse error: syntax error, unexpected '...' (T_ELLIPSIS), expecting ']'`.

The key to this new functionality lies in the ability to unpack, or spread, any item which is `Traversable`[9]. The item must be implemented by `IteratorAggregate`[10] or `Iterator`[11]. The `Traversable` interface itself is not something we would implement in our code; it's only available to the engine under

Listing 4

```
1. <?php
2.
3. function foo(...$args) {
4.     var_dump($args);
5. }
6.
7. $pets = ['dog', 'cat', 'fish'];
8.
9. // Instead of this:
10. foo($pets[0], $pets[1], $pets[2]);
11.
12. // or even worse:
13. foo($pets[0]);
14. foo($pets[1]);
15. foo($pets[2]);
16.
17. // we can use unpacking
18. foo(...$pets); // outputs: dog, cat, fish
19.
20. // We can even chain them together
21. $wildlife = ['bear', 'moose', 'squirrel'];
22. foo(...$pets, ...$wildlife);
23. //outputs: dog, cat, fish, bear, moose, squirrel
24.
25. // we can't use a conditional argument after
26. // unpacking an argument
27. foo(...$pets, 'bear');
28.
29. // instead put those parameters first
30. foo('bear', ...$pets);
31. //outputs: bear, dog, cat
```

7 *Null Coalesce Operator:* https://wiki.php.net/rfc/isset_ternary

8 *argument unpacking:* https://wiki.php.net/rfc/argument_unpacking

9 `Traversable`: https://php.net/class.traversablatore

10 `IteratorAggregate`: https://php.net/class.iteratoraggregate

11 `Iterator`: https://php.net/class.iterator

the hood of PHP. Listing 5 is an example using the `ArrayIterator` class[12].

Why would we use iterators when we could use `foreach()`? Performance! If we run the previous code through 3v4l.org[13], the online PHP editor and click the performance tab below the response, we can see this code uses 14.87 MB of memory to run in PHP 7.4 (Figure 1).

Let's refactor our code (Listing 6) without using iterators and run it again.

Figure 2 shows that our refactored code uses 14.94 MB of memory—only slightly more—but remember we're dealing with a tiny amount of data. In the real world, this memory adds up and can greatly impact your overall performance.

Experimental Just-in-Time Compiling

PHP 7.4 also brings an experimental feature of Just-In-Time[14], or JIT compiling. The feature is disabled by default and is likely not ready for the faint of heart, but we wanted to introduce you to it since this is a brand new feature of the language not previously possible. The JIT looks for previously compiled interpreted code (normal PHP executed by the compiler) and keep track of it as "warm" or "hot" and reuse the already compiled code. It uses DynAsm[15] to generate native code. Think of this as a cache to always run the most efficient code as the compiler can and steps above just caching opcodes. While the experimental code has made it into PHP 7.4, we'll have to wait until PHP 8 to kick the tires and understand the full impact of what the JIT can bring to our applications.

Listing 5

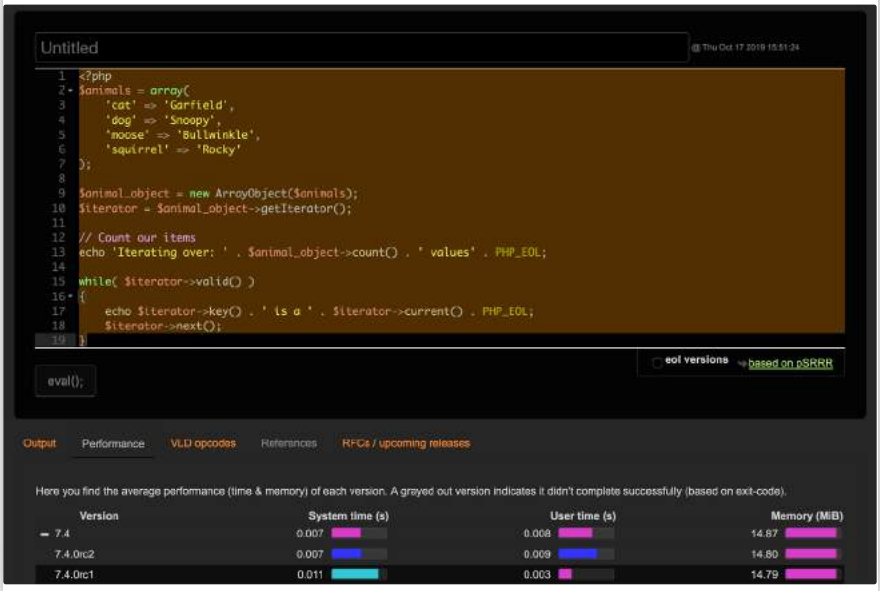
```php
<?php
$animals = [
    'cat' => 'Garfield',
    'dog' => 'Snoopy',
    'moose' => 'Bullwinkle',
    'squirrel' => 'Rocky'
];

$animal_object = new ArrayObject($animals);
$iterator = $animal_object->getIterator();

// Count our items
echo 'Iterating over: ' . $animal_object->count() . ' values' . PHP_EOL;

while ($iterator->valid()) {
    echo $iterator->key() . ' is a ' . $iterator->current() . PHP_EOL;
    $iterator->next();
}
```

Figure 1



Listing 6

```php
<?php
$animals = [
    'cat' => 'Garfield',
    'dog' => 'Snoopy',
    'moose' => 'Bullwinkle',
    'squirrel' => 'Rocky'
];
// Count our items
echo 'Iterating over: ' . count($animals) . ' values' . PHP_EOL;

foreach ($animals as $animal => $name) {
    echo $animal . ' is a ' . $name . PHP_EOL;
}
```

12 `ArrayIterator` *class:* https://php.net/class.arrayiterator

13 *3v4l.org:* https://3v4l.org

14 *Just-In-Time:* https://wiki.php.net/rfc/jit

15 *DynAsm:* https://luajit.org/dynasm.html

Preloading

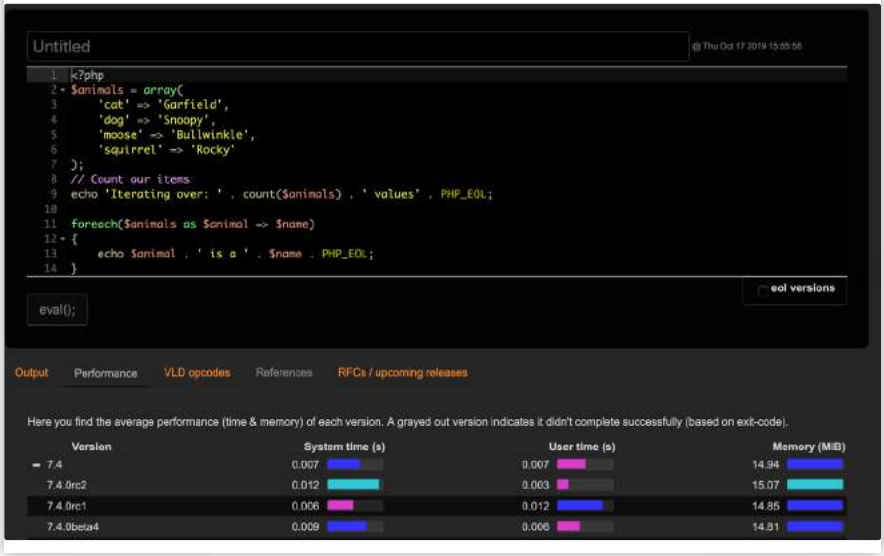
While we wait for what the JIT can do for our performance, we can leverage another feature brand new to 7.4: preloading[16]. While OPcache saves the opcodes of our interpreted PHP applications, we can preload entire files for classes and functions to be compiled once at server startup. It saves time during the execution of our application. One caveat to preloading is you cannot preload unlinked files, which means all the classes you preload need to be linked together via traditional object-oriented linking such as extending or implementing. That is, you can not preload a child class without also preloading its parents. Likewise, you can not preload a class that uses an interface or trait without also preloading those interfaces or traits. If we wanted to preload the entire Laravel framework, we would need to loop over all the PHP files in the framework and load them from one location via the following line in our `php.ini` configuration file.

```
opcache.preload=/laravel/project/preload.php
```

You'll need an automated way to update this preloaded list of files and then also reload PHP-FPM or your webserver to reparse the autoloaded file.

The real value for preloading likely comes from tools like Composer or framework helpers supporting this new feature by automatically preloading all of our dependencies for us with minimal developer interaction. If you're eager to test drive preloading, check out this package[17] for Laravel, which preloads the files for you, leaving you the task to add the path to the preload file to your `php.ini` file.

Figure 2



Deprecations

The list of deprecations for this version, while not empty, does not include many significant ones. One worth mentioning, for security's sake, is the deprecation of the `allow_url_include` INI setting. If you have code which depends on including PHP code from a remote machine, it's time to rethink your solution. For the full list of deprecations, see PHP RFC: Deprecations for PHP 7.4[18].

You can find all these changes and more in the PHP 7.4 upgrade doc in the `php-src` repo. As of this writing, PHP 7.4RC3 is the latest available, and you can find the entire upgrade document on GitHub[19].

Happy Upgrading!

Joe Ferguson is a PHP developer and community organizer. He is involved with many different technology related initiatives in Memphis including the Memphis PHP User group. He's been married to his extremely supportive and amazing wife for a really long time and she turned him into a crazy cat man. They live in the Memphis suburbs with their two cats. @JoePFerguson

Related Reading

- *The Workshop: The Road to 7.3, Part One* by Joe Ferguson, January 2019. https://phpa.me/workshop-jan-2019
- *The Workshop: The Road to 7.3, Part Two* by Joe Ferguson, February 2019. https://phpa.me/workshop-feb-2019

16 *preloading:* https://wiki.php.net/rfc/preload

17 *package:* https://github.com/brendt/laravel-preload

18 *PHP RFC: Deprecations for PHP 7.4:* https://wiki.php.net/rfc/deprecations_php_7_4

19 *upgrade document on GitHub:* https://phpa.me/php-7-4-0RC3-upgrading

Programming

DAYCAMP 4 DEVELOPERS

December 20, 2019 | Online | 9:00 am - 3:00 pm

The web has changed. The tech necessary to build modern web applications is constantly evolving. Busy developers have a tough time keeping up and keeping their existing projects moving forward.

We have 5 great speakers presenting on different aspects of Modern Web Programming who will help you stay current. Join us and learn.

Tickets and more information available at **daycamp4developers.com**

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe

php[**architect**]