MODERN PHP USER GROUP

NEW IN PHP 7.4

https://stitcher.io/blog/new-in-php-74

If you enjoy reading my blog, you could consider supporting me on Patreon.

« back — written by Brent on May 17, 2019

New in PHP 7.4

- The release date is probably around December 2019, but yet to be confirmed
- Short closures for cleaner one-liner functions
- Preloading to improve performance
- Typed properties in classes
- Custom object serialization adds a new way of (de)serializing objects
- Improved type variance

SHORT CLOSURES

```php
array_map(function (User $user) {
    return $user->id;
}, $users)
```

```php
array_map(fn(User $user) => $user->id, $users)
```

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

- return keyword 없는 하나의 return statement

```php
array_map(function (User $user) {
    return $user->id;
}, $users)
```

```php
array_map(fn(User $user) => $user->id, $users)
```

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

- return keyword 없는 하나의 return statement
- use keyword 없이 parent scope 접근 가능

```php
array_map(function (User $user) {
    return $user->id;
}, $users)
```

```php
array_map(fn(User $user) => $user->id, $users)
```

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

- use keyword 없이 parent scope 접근 가능
  - automatic binding mechanism

```php
$fn = fn() => $undef;
$fn();
```

- $undef라는 변수를 사용하려고 할 때 하나의 undefined variable notice만 발생
- Binding 시에는 발생하지 않음

```php
$fn = fn($str) => preg_match($regex, $str, $matches) && ($matches[1] % 7 == 0)
```

Binding 시에 발생시키지 않는 것이 자연스러운 예

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

- use keyword 없이 parent scope 접근 가능
  - automatic binding mechanism

```php
$x = 42;
$y = 'x';
$fn = fn() => $$y;
```

- **Undefined variable notice 발생**
- **Literally 선언된 변수만 자동으로 binding 되기 때문**

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

- return keyword 없는 하나의 return statement
- use keyword 없이 parent scope 접근 가능
- type hint 가능

```php
$ids = array_map(fn(Post $post): int => $post->id, $posts);
```

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

- use keyword 없이 parent scope 접근 가능
- return keyword 없는 하나의 return statement
- type hints
- return a value by reference

```
fn&($x) => $x
```

SHORT CLOSURES

https://wiki.php.net/rfc/arrow_functions_v2

Future Scope

```
fn(params) => {
    stmt1;
    stmt2;
    return expr;
}
// or possibly just
fn(params) {
    stmt1;
    stmt2;
    return expr;
}
```

Multi-statement bodies

```
$a = 1;
$b = 2;
$fn = fn() use(&$a) {
    $a += $b;
};
$fn();
var_dump($a); // int(3)
```

Switching the binding mode

```
class Test {
    private $foo;
    private $bar;

    fn getFoo() => $this->foo;
    fn getBar() => $this->bar;
}
```

Allow arrow notation for real functions

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- Before

```php
class User {
    /** @var int $id */
    private $id;
    /** @var string $name */
    private $name;

    public function __construct(int $id, string $name) {
        $this->id = $id;
        $this->name = $name;
    }

    public function getId(): int {
        return $this->id;
    }
    public function setId(int $id): void {
        $this->id = $id;
    }

    public function getName(): string {
        return $this->name;
    }
    public function setName(string $name): void {
        $this->name = $name;
    }
}
```

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- After

```php
class User {
    public int $id;
    public string $name;

    public function __construct(int $id, string $name) {
        $this->id = $id;
        $this->name = $name;
    }
}
```

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

스펙 요약

```php
class Example {
    // All types with the exception of "void" and "callable" are supported
    public int $scalarType;
    protected ClassName $classType;
    private ?ClassName $nullableClassType;

    // Types are also legal on static properties
    public static iterable $staticProp;

    // Types can also be used with the "var" notation
    var bool $flag;

    // Typed properties may have default values (more below)
    public string $str = "foo";
    public ?string $nullableStr = null;

    // The type applies to all properties in one declaration
    public float $x, $y;
    // equivalent to:
    public float $x;
    public float $y;
}
```

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- void and callable을 제외한 거의 모든 type 적용 가능

```
bool, int, float, string, array, object
iterable
self, parent
any class or interface name
?type // where "type" may be any of the above
```

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- void and callable을 제외한 거의 모든 type 적용 가능
- Property type은 바뀌지 않는다

```php
class A {
    private bool $a;
    public int $b;
    public ?int $c;
}
class B extends A {
    public string $a; // legal, because A::$a is private
    public ?int $b;   // ILLEGAL
    public int $c;    // ILLEGAL
}
```

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- void and callable을 제외한 거의 모든 type 적용 가능
- Property type은 바뀌지 않는다

```php
class A {
    public self $prop;
}
class B extends A {
    public self $prop;
}
```

이것도 몹쓸 코드입니다

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- void and callable을 제외한 거의 모든 type 적용 가능
- Property type은 바뀌지 않는다
- Uninitialized and Unset Properties

```php
class Test {
    public int $val;

    public function __construct(int $val) {
        $this->var = $val; // Ooops, typo
    }
}

$test = new Test(42);
var_dump($test->val); // TypeError
```

var_dump

```
object(Test)#1 (0) {
  ["val"]=>
  uninitialized(int)
}
```

TYPED PROPERTIES

https://wiki.php.net/rfc/typed_properties_v2

- void and callable을 제외한 거의 모든 type 적용 가능
- Property type은 바뀌지 않는다
- Uninitialized and Unset Properties
- and more... https://wiki.php.net/rfc/typed_properties_v2

COVARIANT RETURNS AND CONTRAVARIANT PARAMETERS

https://wiki.php.net/rfc/covariant-returns-and-contravariant-parameters

Covariant Returns

```php
class ParentType {}
class ChildType extends ParentType {}

class A
{
    public function covariantReturnTypes(): ParentType
    { /* … */ }
}

class B extends A
{
    public function covariantReturnTypes(): ChildType
    { /* … */ }
}
```

COVARIANT RETURNS AND CONTRAVARIANT PARAMETERS

https://wiki.php.net/rfc/covariant-returns-and-contravariant-parameters

- Contravariant Parameters

```php
class A
{
    public function contraVariantArguments(ChildType $type)
    { /* … */ }
}

class B extends A
{
    public function contraVariantArguments(ParentType $type)
    { /* … */ }
}
```

NULL COALESCING ASSIGNMENT OPERATOR

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

- Instead of doing this:

```
$data['date'] = $data['date'] ?? new DateTime();
```

- You can do this:

```
$data['date'] ??= new DateTime();
```

ARRAY SPREAD OPERATOR

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

- Argument Unpacking https://wiki.php.net/rfc/argument_unpacking

```php
$parts = ['apple', 'pear'];
$fruits = ['banana', 'orange', ...$parts, 'watermelon'];
// ['banana', 'orange', 'apple', 'pear', 'watermelon'];
```

```php
$arr1 = [1, 2, 3];
$arr2 = [...$arr1]; //[1, 2, 3]
$arr3 = [0, ...$arr1]; //[0, 1, 2, 3]
$arr4 = array(...$arr1, ...$arr2, 111); //[1, 2, 3, 1, 2, 3, 111]
$arr5 = [...$arr1, ...$arr1]; //[1, 2, 3, 1, 2, 3]

function getArr() {
  return ['a', 'b'];
}
$arr6 = [...getArr(), 'c']; //['a', 'b', 'c']

$arr7 = [...new ArrayIterator(['a', 'b', 'c'])]; //['a', 'b', 'c']

function arrGen() {
        for($i = 11; $i < 15; $i++) {
                yield $i;
        }
}
$arr8 = [...arrGen()]; //[11, 12, 13, 14]
```

FOREIGN FUNCTION INTERFACE

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

- FFI in short
- PHP extensions can be written in pure PHP
    - PHP 코드에서 shared libraries (.DLL or .so) 로딩
    - PHP 코드에서 C 함수나 C data structure에 접근
- TensorFlow binding, implemented in pure PHP.
    - https://github.com/dstogov/php-tensorflow

FOREIGN FUNCTION INTERFACE

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

- Calling a function, returning structure through argument

```php
<?php
// create FFI object, loading libc and exporting function printf()
$ffi = FFI::cdef(
    "int printf(const char *format, ...);", // this is regular C declaration
    "libc.so.6");
// call C printf()
$ffi->printf("Hello %s!\n", "world");
```

FOREIGN FUNCTION INTERFACE

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

- Calling a function, returning structure through argument

```php
<?php
// create gettimeofday() binding
$ffi = FFI::cdef("
    typedef unsigned int time_t;
    typedef unsigned int suseconds_t;

    struct timeval {
        time_t      tv_sec;
        suseconds_t tv_usec;
    };

    struct timezone {
        int tz_minuteswest;
        int tz_dsttime;
    };

    int gettimeofday(struct timeval *tv, struct timezone *tz);
", "libc.so.6");
// create C data structures
$tv = $ffi->new("struct timeval");
$tz = $ffi->new("struct timezone");
// calls C gettimeofday()
var_dump($ffi->gettimeofday(FFI::addr($tv), FFI::addr($tz)));
// access field of C data structure
var_dump($tv->tv_sec);
// print the whole C data structure
var_dump($tz);
```

FOREIGN FUNCTION INTERFACE

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

- Accessing C variables

```php
<?php
// create FFI object, loading libc and exporting errno variable
$ffi = FFI::cdef("int errno;", // this is regular C declaration
    "libc.so.6");
// print C errno
var_dump($ffi->errno);
```

- Working with C arrays

```php
<?php
// create C data structure
$a = FFI::new("unsigned char[1024*1024]");
// work with it like with regular PHP array
for ($i = 0; $i < count($a); $i++) {
  $a[$i] = $i;
}
var_dump($a[25]);
$sum = 0;
foreach ($a as $n) {
  $sum += $n;
}
var_dump($sum);
var_dump(FFI::sizeof($a));
```

FOREIGN FUNCTION INTERFACE

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

- RFC Impact

RFC Impact

To Opcache

FFI is designed in conjunction with preloading (curently implemented as part of opcache). FFI C headers may be loaded during preloading by **FFI::load()** and become available to all the following HTTP requests without reloading overhead.

php.ini Defaults

```
ffi.enable=false|preload|true
```

allows enabling or disabling FFI API usage, or restricting it only to preloaded files. The default value is **preload**. This is INI_SYSTEM directive and it's value can't be changed at run-time.

PRELOADING

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

- to preload the whole Zend Framework

```php
<?php
function _preload($preload, string $pattern = "/\.php$/", array $ignore = []) {
  if (is_array($preload)) {
    foreach ($preload as $path) {
      _preload($path, $pattern, $ignore);
    }
  } else if (is_string($preload)) {
    $path = $preload;
    if (!in_array($path, $ignore)) {
      if (is_dir($path)) {
        if ($dh = opendir($path)) {
          while (($file = readdir($dh)) !== false) {
            if ($file !== "." && $file !== "..") {
              _preload($path . "/" . $file, $pattern, $ignore);
            }
          }
          closedir($dh);
        }
      } else if (is_file($path) && preg_match($pattern, $path)) {
        if (!opcache_compile_file($path)) {
          trigger_error("Preloading Failed", E_USER_ERROR);
        }
      }
    }
  }
}

set_include_path(get_include_path() . PATH_SEPARATOR . realpath("/var/www/ZendFramework/library"));
_preload(["/var/www/ZendFramework/library"]);
```

CUSTOM OBJECT SERIALIZATION

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

- New custom object serialization mechanism
  - 기존 PHP에서 제공했던 두가지 커스텀 직렬화 매커니즘
    - __sleep() / __wakeup() magic methods
    - Serializable interface

CUSTOM OBJECT SERIALIZATION

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

- New custom object serialization mechanism
  - 기존 PHP에서 제공했던 두가지 커스텀 직렬화 매커니즘
    - __sleep() / __wakeup() magic methods
      - 사용성이 떨어짐
    - Serializable interface
      - 중첩된 serialize() call 등의 문제

CUSTOM OBJECT SERIALIZATION

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

- __serialize()/__unserialize()

```php
class A {
    private $prop_a;
    public function __serialize(): array {
        return ["prop_a" => $this->prop_a];
    }
    public function __unserialize(array $data) {
        $this->prop_a = $data["prop_a"];
    }
}
class B extends A {
    private $prop_b;
    public function __serialize(): array {
        return [
            "prop_b" => $this->prop_b,
            "parent_data" => parent::__serialize(),
        ];
    }
    public function __unserialize(array $data) {
        parent::__unserialize($data["parent_data"]);
        $this->prop_b = $data["prop_b"];
    }
}
```

CONCATENATION PRECEDENCE

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

- Deprecation notice in PHP 7.4

지금까지는 이랬는데

```php
echo "sum: " . $a + $b;

// current behavior: evaluated left-to-right
echo ("sum: " . $a) + $b;

// desired behavior: addition and subtraction have a higher precendence
echo "sum :" . ($a + $b);
```

PHP 8부터는 이렇게

RFC VOTING PROCESS IMPROVEMENTS

- RFC 투표 방식의 변화
  - 통과하기 위해 2/3가 찬성해야 한다
  - 모든 RFC는 최소 2주간 열려 있어야 한다

REFLECTION FOR REFERENCES

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

- 복제하거나 비교하거나 dump를 하는 라이브러리에서 특정 값의 동일 여부를 판단하려면 이런 꼼수를..

```php
$array2 = $array1;
$array2[$key] = $unique_cookie;
if ($array1[$key] === $unique_cookie) {
    // $array1[$key] is a reference
}
```

- 이를 해결하기 위한 ReflectionReference 클래스 추가

MB_STR_SPLIT

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

- Multi byte 문자열을 위한 str_split

```php
<?php
print_r(mb_str_split("победа", 2));

--EXPECT--
Array
(
    [0] => по
    [1] => бе
    [2] => да
)
```

ALWAYS AVAILABLE HASH EXTENSION

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

- ext/hash extension을 항상 사용 가능하게 한다
  - date, spl, pcre extension처럼

PEAR NOT ENABLED BY DEFAULT

https://externals.io/message/103977

- PEAR가 더이상 관리되고 있지 않고, 사고도 났음
- default 로 설치 하지 않기로 결정

SHORT PHP TAGS DEPRECATED

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

- Deprecation notice and default value = Off in PHP 7.4
- Removal in PHP 8.0

LEFT-ASSOCIATIVE TERNARY OPERATOR DEPRECATION

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

```php
return $a == 1 ? 'one'
     : $a == 2 ? 'two'
     : $a == 3 ? 'three'
     : $a == 4 ? 'four'
               : 'other';
```

보통 다른 언어에선

```php
return $a == 1 ? 'one'
     : ($a == 2 ? 'two'
     : ($a == 3 ? 'three'
     : ($a == 4 ? 'four'
                : 'other')));
```

PHP에선

```php
return (($a == 1 ? 'one'
     : $a == 2) ? 'two'
     : $a == 3) ? 'three'
     : $a == 4) ? 'four'
                : 'other';
```

LEFT-ASSOCIATIVE TERNARY OPERATOR DEPRECATION

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

- 명시적 괄호가 없다면 deprecation warning

```php
1 ? 2 : 3 ? 4 : 5;   // deprecated
(1 ? 2 : 3) ? 4 : 5; // ok
1 ? 2 : (3 ? 4 : 5); // ok
```

```php
1 ?: 2 ? 3 : 4;   // deprecated
(1 ?: 2) ? 3 : 4; // ok
1 ?: (2 ? 3 : 4); // ok

1 ? 2 : 3 ?: 4;   // deprecated
(1 ? 2 : 3) ?: 4; // ok
1 ? 2 : (3 ?: 4); // ok
```

LEFT-ASSOCIATIVE TERNARY OPERATOR DEPRECATION

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

- 하지만 짧은 삼항 연잔자 표현식만 있다면 괜찮다

```php
1 ?: 2 ?: 3;   // ok
(1 ?: 2) ?: 3; // ok
1 ?: (2 ?: 3); // ok
```

그리고 아마 더 있을 겁니다

follow @brendt_gd on Twitter