What's new in PHP 8

phpDay 2020

Derick Rethans

derick@php.net—@derickr

https://derickrethans.nl/talks/php-phpday20

About Me

Derick Rethans

- I'm European, living in London
- PHP 7.4 **Release Manager**
- **Xdebug** & PHP's Date/Time support
- Host of **https://phpinternals.news** podcast
- I ♥🌍 maps, I ♥🍺 beer, I ♥🥃 whisky
- twitter: @derickr

Typed Properties

```
<?php
class 🛢whisky
{
```

All types with the exception of `void` and `callable` are supported:

```
    public    int         $age;
    protected Distillery $distillery;
    private   ?Bottler    $bottler;
```

Typed Properties

```
<?php
class 🛢whisky
{
```

All types with the exception of `void` and `callable` are supported:

```
    public    int         $age;
    protected Distillery $distillery;
    private   ?Bottler    $bottler;
```

Types are also legal on static properties:

```
    public static iterable $tastingNotes;
```

Typed Properties

```
<?php
class 🛢whisky
{
```

All types with the exception of `void` and `callable` are supported:

```
    public    int         $age;
    protected Distillery $distillery;
    private   ?Bottler    $bottler;
```

Types are also legal on static properties:

```
    public static iterable $tastingNotes;
```

Typed properties may have default values:

```
    public int     $rating = 92;
    public ?string $nullableStr = null;
```

Typed Properties

```
<?php
class 🛢whisky
{
```

All types with the exception of `void` and `callable` are supported:

```
    public    int         $age;
    protected Distillery $distillery;
    private   ?Bottler    $bottler;
```

Types are also legal on static properties:

```
    public static iterable $tastingNotes;
```

Typed properties may have default values:

```
    public int     $rating = 92;
    public ?string $nullableStr = null;
```

The type applies to all properties in one declaration:

```
    public bool $chillFiltered, $colouringAdded;
}
```

Typed Properties

```
<?php
class 🛢whisky
{
```

All types with the exception of `void` and `callable` are supported:

```
    public    int         $age;
    protected Distillery $distillery;
    private   ?Bottler    $bottler;
```

Types are also legal on static properties:

```
    public static iterable $tastingNotes;
```

Typed properties may have default values:

```
    public int     $rating = 92;
    public ?string $nullableStr = null;
```

The type applies to all properties in one declaration:

```
    public bool $chillFiltered, $colouringAdded;
}
```

Contravariant Argument Types

```
<?php
class 🐖pig {}
class 🐗boar extends 🐖pig {}

class Asterix
{
        public function hunt(🐗boar $animal) {}
}

class Gauls extends Asterix
{
        public function hunt(🐖pig $animal) {}
}
?>
```

Arguments types are **contra-variant**:
they can accept a more broad type

Union Types

```php
<?php
declare(strict_types=1);

class Number {



    private $number;



    public function setNumber($number) {
        $this->number = $number;
    }



    public function getNumber() {
        return $this->number;
    }
}
```

Union Types

```php
<?php
declare(strict_types=1);

class Number {
    /**
     * @var int|float $number
     */
    private $number;

    /**
     * @param int|float $number
     */
    public function setNumber($number) {
        $this->number = $number;
    }

    /**
     * @return int|float
     */
    public function getNumber() {
        return $this->number;
    }
}
```

Union Types

```php
<?php
declare(strict_types=1);

class Number {



    private int|float $number;



    public function setNumber(int|float $number) {
        $this->number = $number;
    }



    public function getNumber() : int|float {
        return $this->number;
    }
}
```

Union Types — Examples

```
function microtime(bool $get_as_float = false): string|float {}

function gettimeofday(bool $return_float = false): array|float {}

function base64_decode(string $str, bool $strict = false): string|false {}

function implode(string|array $glue, array $pieces): string {}

function substr_replace(
    string|array $str,
    string|array $replace,
    $start,
    $length
) : string|array|false {}
```

Trailing Comma in Function Declaration

```
function microtime(bool $get_as_float = false,): string|float {}

function gettimeofday(bool $return_float = false,): array|float {}

function base64_decode(string $str, bool $strict = false,): string|false {}

function implode(string|array $glue, array $pieces,): string {}

function substr_replace(
    string|array $str,
    string|array $replace,
    $start,
    $length,
) : string|array|false {}
```

Coercive typing mode

If the exact type of the value is not part of the union, then the target type is chosen in the following order of preference:

- int
- float
- string
- bool

```
// type: int|string
42.0  → 42           // float compatible with int
42.1  → 42           // float compatible with int
1e100 → "1.0E+100"   // float too large for int type, fall back to string
INF   → "INF"        // float too large for int type, fall back to string
true  → 1            // bool compatible with int
new ObjectWithToString → "Result of __toString()"
                     // object never compatible with int, fall back to string
[]    → TypeError    // array not compatible with int or string
```

`mixed`

`array|bool|callable|int|float|null|object|resource|string`

- Distinguishes between:
  - I didn't add the type yet
  - It really accepts any value
- Acts as a union type
- Follows standard variance rules
- A missing return type declaration is assumed to be: **`mixed|void`**

Constructor Property Promotion

```php
<?php
class 🛢whisky
{
    public    string      $name;
    protected Distillery  $distillery;
    public    int         $age;
    private   ?Bottler    $bottler = null;
```

```php
    function __construct(string $name, Distillery $distillery, int $age, ?Bottler $
    {
        $this->name = $name;
        $this->distillery = $distillery;
        $this->age = $age;
        $this->bottler = $bottler;
    }
```

Constructor Property Promotion

```php
<?php
class 🛢whisky
{
    public    string      $name;
    protected Distillery  $distillery;
    public    int         $age;
    private   ?Bottler    $bottler = null;
```

```php
    function __construct(string $name, Distillery $distillery, int $age, ?Bottler $
    {
        $this->name = $name;
        $this->distillery = $distillery;
        $this->age = $age;
        $this->bottler = $bottler;
    }
```

Constructor Property Promotion

```php
<?php
class 🛢whisky
{
    function __construct(
        public    string $name,
        protected Distillery $distillery,
        public    int $age,
        private   ?Bottler $bottler = null,
    ) {}
}
```

Object Instantiation

```
<?php
class Settings
{
    function __construct(
        public $includeDistillery   = true,
        public $includeDescription  = true,
        public $includeTastingNotes = false,
        public $pageSize            = 64,
        public $sortByField         = 'whisky',
    ) {}
}
```

Instantiation with different includes:

```
$settings = new \Settings(false, true, false);
```

Object Instantiation

```
<?php
class Settings
{
    function __construct(
        public $includeDistillery   = true,
        public $includeDescription  = true,
        public $includeTastingNotes = false,
        public $pageSize            = 64,
        public $sortByField         = 'whisky',
    ) {}
}
```

Instantiation with different includes:

```
$settings = new \Settings(false, true, false);
```

Instantiation with different sort order:

```
$settings = new \Settings(true, true, false, 64, 'rating');
```

Object Instantiation

```
<?php
class Settings
{
    function __construct(
        public $includeDistillery   = true,
        public $includeDescription  = true,
        public $includeTastingNotes = false,
        public $pageSize            = 64,
        public $sortByField         = 'whisky',
    ) {}
}
```

Instantiation with different includes:

```
$settings = new \Settings(false, true, false);
```

Instantiation with different sort order:

```
$settings = new \Settings(true, true, false, 64, 'rating');
```

Object Instantiation with Named Arguments

```
<?php
class Settings
{
    function __construct(
        public $includeDistillery   = true,
        public $includeDescription  = true,
        public $includeTastingNotes = false,
        public $pageSize            = 64,
        public $sortByField         = 'whisky',
    ) {}
}
```

Instantiation with different includes:

```
$settings = new \Settings(includeDistillery: false);
```

Object Instantiation with Named Arguments

```
<?php
class Settings
{
    function __construct(
        public $includeDistillery   = true,
        public $includeDescription  = true,
        public $includeTastingNotes = false,
        public $pageSize            = 64,
        public $sortByField         = 'whisky',
    ) {}
}
```

Instantiation with different includes:

```
$settings = new \Settings(includeDistillery: false);
```

Instantiation with different sort order:

```
$settings = new \Settings(sortByField: 'rating');
```

Object Instantiation with Named Arguments

```
<?php
class Settings
{
    function __construct(
        public $includeDistillery   = true,
        public $includeDescription  = true,
        public $includeTastingNotes = false,
        public $pageSize            = 64,
        public $sortByField         = 'whisky',
    ) {}
}
```

Instantiation with different includes:

```
$settings = new \Settings(includeDistillery: false);
```

Instantiation with different sort order:

```
$settings = new \Settings(sortByField: 'rating');
```

Attributes

```
<?php
/** Entity */
class Distillery
{
    /**
     * @Id
     * @Column(type=string)
     */
    private $name;
}
```

Attributes

```php
<?php
use Doctrine\ORM\Attributes as ORM;

#[ORM\Entity]
class Distillery
{
    #[ORM\Id]
    #[ORM\Column("string")]
    private $name;
}
```

```php
    #[ORM\Column("string")] // Class Name
```

Attributes

```php
<?php
use Doctrine\ORM\Attributes as ORM;

#[ORM\Entity]
class Distillery
{
    #[ORM\Id]
    #[ORM\Column("string")]
    private $name;
}
```

```php
    #[ORM\Column("string")] // Class Name
```

```php
    #[ORM\Column("string")] // Constructor Arguments
```

Attributes

```php
<?php
use Doctrine\ORM\Attributes as ORM;

#[ORM\Entity]
class Distillery
{
    #[ORM\Id]
    #[ORM\Column("string")]
    private $name;
}
```

```php
    #[ORM\Column("string")] // Class Name
```

```php
    #[ORM\Column("string")] // Constructor Arguments
```

Attributes: Reading

```php
<?php
use MyORM\Attributes as ORM;

class Distillery
{
    #[ORM\Column("string")]
    private $name;
}
```

```php
<?php
namespace MyORM\Attributes;
use Attribute;

#[Attribute]
class Column {
    public function __construct(public string $type) {  }
}
```

```php
<?php
$rc = new \ReflectionClass(\Distillery::class);
$rp = $rc->getProperty('name');

foreach ($rp->getAttributes() as $attr) {
    echo "Name: ", $attr->getName(), "\n";
    var_dump($attr->getArguments());
    var_dump($attr->newInstance());
}
```

```
Name: MyORM\Attributes\Column
array(1) {
  [0]=>
  string(6) "string"
}
object(MyORM\Attributes\Column)#4 (1) {
  ["type"]=>
  string(6) "string"
}
```

Attributes: Reading

```php
<?php
use MyORM\Attributes as ORM;

class Distillery
{
    #[ORM\Column("string")]
    private $name;
}
```

```php
<?php
namespace MyORM\Attributes;
use Attribute;

#[Attribute]
class Column {
    public function __construct(public string $type) {  }
}
```

```php
<?php
$rc = new \ReflectionClass(\Distillery::class);
$rp = $rc->getProperty('name');

foreach ($rp->getAttributes() as $attr) {
    echo "Name: ", $attr->getName(), "\n";
    var_dump($attr->getArguments());
    var_dump($attr->newInstance());
}
```

```
Name: MyORM\Attributes\Column
array(1) {
  [0]=>
  string(6) "string"
}
object(MyORM\Attributes\Column)#4 (1) {
  ["type"]=>
  string(6) "string"
}
```

Attributes: Use Cases

- Replaces phpdoc annotations in Doctrine, Symfony, etc…

```php
<?php
use Doctrine\ORM\Attributes as ORM;

#[ORM\Entity]
class Distillery
{
    #[ORM\Id]
    #[ORM\Column("string")]
    private $name;
}
```

- May add internal attributes:
  - `#[Deprecated("Use xyz instead")]`
  - `#[Jit]`

    `#[NoJit]`

Match Expressions

`switch` can be cumbersome:

```
<?php
switch ($this->lexer->lookahead['type']) {
    case Lexer::T_SELECT:
        $statement = $this->SelectStatement();
        break;

    case Lexer::T_UPDATE:
        $statement = $this->UpdateStatement();
        break;

    case Lexer::T_DELETE:
        $statement = $this->DeleteStatement();
        break;

    default:
        $this->syntaxError('SELECT, UPDATE or DELETE');
        break;
}
```

Match Expressions

`match` improves this:

```
<?php
$statement = match ($this->lexer->lookahead['type']) {
    Lexer::T_SELECT => $this->SelectStatement(),
    Lexer::T_UPDATE => $this->UpdateStatement(),
    Lexer::T_DELETE => $this->DeleteStatement(),
    default => $this->syntaxError('SELECT, UPDATE or DELETE'),
};
```

Match Expressions

`match` improves this:

```
<?php
$statement = match ($this->lexer->lookahead['type']) {
    Lexer::T_SELECT => $this->SelectStatement(),
    Lexer::T_UPDATE => $this->UpdateStatement(),
    Lexer::T_DELETE => $this->DeleteStatement(),
    default => $this->syntaxError('SELECT, UPDATE or DELETE'),
};
```

Differences with `switch`:

- It is an expression, so returns a value
- `match` does not do type-juggling
- No automatic fall through
- Exception if nothing matched

Match Expressions

`match` improves this:

```
<?php
$statement = match ($this->lexer->lookahead['type']) {
    Lexer::T_SELECT => $this->SelectStatement(),
    Lexer::T_UPDATE => $this->UpdateStatement(),
    Lexer::T_DELETE => $this->DeleteStatement(),
    default => $this->syntaxError('SELECT, UPDATE or DELETE'),
};
```

Differences with `switch`:

- It is an expression, so returns a value
- `match` does not do type-juggling
- No automatic fall through
- Exception if nothing matched

Null-safe Operator

Checking for `null` at every step:

```
$country =  null;

if ($session !== null) {
    $user = $session->user;

    if ($user !== null) {
        $address = $user->getAddress();

        if ($address !== null) {
            $country = $address->country;
        }
    }
}
```

With null-safe operator `?->`:

```
$country = $session?->user?->getAddress()?->country;
```

Null-safe Operator

Checking for `null` at every step:

```
$country =  null;

if ($session !== null) {
    $user = $session->user;

    if ($user !== null) {
        $address = $user->getAddress();

        if ($address !== null) {
            $country = $address->country;
        }
    }
}
```

With null-safe operator `?->`:

```
$country = $session?->user?->getAddress()?->country;
```

Short Circuiting

Example:

```
$country = $session?->user?->getAddress($session?->format)?->country;
```

- If `$session->format` is `NULL`, `NULL` is passed to `getAddress()`

Short Circuiting

Example:

```
$country = $session?->user?->getAddress($session?->format)?->country;
```

- If `$session->format` is `NULL`, `NULL` is passed to `getAddress()`
- If `$session->user` is NULL, `getAddress()` is not called

Short Circuiting

Example:

```
$country = $session?->user?->getAddress($session?->format)?->country;
```

- If `$session->format` is `NULL`, `NULL` is passed to `getAddress()`
- If `$session->user` is NULL, `getAddress()` is not called
- If `getAddress()` returns `NULL`, `$country` is `NULL`

Short Circuiting

Example:

```
$country = $session?->user?->getAddress($session?->format)?->country;
```

- If `$session->format` is `NULL`, `NULL` is passed to `getAddress()`
- If `$session->user` is NULL, `getAddress()` is not called
- If `getAddress()` returns `NULL`, `$country` is `NULL`

New String Functions

```php
<?php
function str_contains(
    string $haystack,
    string $needle
): bool {}

function str_starts_with(
    string $haystack,
    string $needle
): bool {}

function str_ends_with(
    string $haystack,
    string $needle
): bool {}
```

No multi-byte variants, so use UTF-8

WeakMap

```php
<?php
$map = new WeakMap;
$obj = new stdClass;

$map[$obj] = $dataForObj;
unset($obj); // $obj automatically dropped from WeakMap!
```

Use Case

```php
<?php
class FooBar {
    private WeakMap $cache;

    public function getSomethingWithCaching(object $obj) {
        return $this->cache[$obj] ??= $this->computeSomethingExpensive($obj);
    }
}
?>
```

Warning → Error exception

Prototype:

```
function array_fill( int $start_index, int $num, mixed $value ): array|false {}
```

Code:

```
<?php
try {
    var_dump(array_fill(0, -100, "foobar"));
} catch( Throwable $e ) {
    echo get_class( $e ), ': ', $e->getMessage();
}
?>
```

PHP 7.4:

```
Warning: array_fill(): Number of elements can't be negative
```

PHP 8.0:

```
ValueError: array_fill(): Argument #2 ($num) must be greater than or equal to 0
```

Non-Capturing Catches

Sometimes you don't need the Exception itself:

```
try {
    changeImportantData();
} catch (PermissionException $ex) {
    echo "You don't have permission to do this";
}
```

Non-Capturing Catches

Sometimes you don't need the Exception itself:

```
try {
    changeImportantData();
} catch (PermissionException) {
    echo "You don't have permission to do this";
}
```

Non-Capturing Catches

Sometimes you don't need the Exception itself:

```
try {
    changeImportantData();
} catch (PermissionException) {
    echo "You don't have permission to do this";
}
```

Just-In-Time (JIT) Compiler

- Compiles PHP code to x86 machine code
- Performance improvements mostly apply to math heavy code
- Part of opcache:

```
opcache.jit=on
opcache.jit_buffer_size=128M
```

When is it out?

November 26th, 2020

When is it out?

November 26th, 2020

(hopefully)

When is it out?

November 26th, 2020

(hopefully)

Episode #67
Match Expression

In this episode of "PHP Internals News" I chat with Derick Rethans (Twitter, GitHub, Website) about the new Match Expression in PHP 8.

Thursday August 20th, 2020 — 09:30 BST

Episode #66
Namespace Token, and Parsing PHP

In this episode of "PHP Internals News" I chat with Nikita Popov (Twitter, GitHub, Website) about his Namespaced Names as a Single Token, and Parsing PHP.

Thursday August 13th, 2020 — 09:29 BST

Episode #65
Null safe operator

In this episode of "PHP Internals News" I chat with Dan Ackroyd (Twitter, GitHub) about the Null Safe Operator RFC.

Spotiy iTunes RSS Feed Twitter

BECOME A PATRON

https://phpinternals.news

Any Queries?

Resources

Slides

`https://derickrethans.nl/talks/php-phpday20`

`https://phpinternals.news`

Derick Rethans — @derickr — derick@php.net