Features do PHP

"mágicas" ou não

Naylon Kessler de Aquino

www.naylonkessler.com

Co-fundador/CTO do AprovaDETRAN
Co-fundador/CTO da Otimize Growth

Full stack developer a ~19 anos.

Antes de prosseguirmos

- Não acredite no que eu te disser;
- Não se limite;
- Extrapole os conceitos;
- Não seja “radical”;
- Use a imaginação.

O que veremos

1. Motivação;
2. Métodos mágicos;
3. Constantes mágicas;
4. Variáveis variáveis;
5. Métodos variáveis;
6. Late static binding;
7. Traits
8. SPL;
9. Reflection;
10. Bônus: Features 7.4.

Motivação

Todos os dias temos novas pessoas iniciando com a linguagem mas nem sempre iniciando pela linguagem.

Métodos mágicos

São métodos especiais do PHP que possuem um comportamento predefinido quando presentes em uma determinada classe.

__get() e __set()

__get() é invocado para ler dados de propriedades inacessíveis.

__set() é invocado para escrever dados em propriedades inacessíveis.

Exemplo de __get e __set.

Escrevendo e lendo dados sem propriedades inacessíveis.

```php
<?php

class Record
{
    protected $data = [];

    public function __get($name)
    {
        return $this->data[$name];
    }

    public function __set($name, $value)
    {
        $this->data[$name] = $value;
    }
}

$record = new Record();
$record->name = 'First One';
var_dump($record->name);

```

__isset() e __unset()

__isset() é invocado quando as funções isset() ou empty() são chamada em propriedades inacessíveis.

__unset() é invocado quando a função unset() é chamada em propriedades inacessíveis.

Exemplo de __isset e __unset.

Verificando e removendo propriedades inacessíveis.

```php
<?php

class Record
{
    protected $data = [];

    public function __get($name) { ... }

    public function __set($name, $value) { ... }

    public function __isset($name)
    {
        return isset($this->data[$name]);
    }

    public function __unset($name)
    {
        unset($this->data[$name]);
    }
}

$record = new Record();
$record->name = 'First One';
var_dump(isset($record->name));
unset($record->name);
var_dump(empty($record->name));

```

__call() e __callStatic()

__call() é invocado quando um método inacessível é invocado em um objeto.

__callStatic() é invocado quando um método inacessível é invocado em uma classe (contexto estático).

Exemplo de __call e __callStatic.

Habilitando logging nos objetos e desabilitando na classe diretamente

```php
<?php

class MemoryLogger
{
    protected $entries = [];

    public function __call($name, $arguments)
    {
        $this->entries[] = "[{$name}] " . implode(' / ', $arguments);
    }

    public static function __callStatic($name, $arguments)
    {
        throw new Exception('Unable to log statically.');
    }

    public function dump()
    {
        var_dump($this->entries);
    }
}

$logger = new MemoryLogger();
$logger->info('Some info entry.');
$logger->dump();
MemoryLogger::error('Some error message.');
```

__toString()

__toString() retorna a representação de um objeto em formato de string. É invocado quando o objeto é tratado com string.

```php
<?php

class Person
{
    public $name;

    public function __toString()
    {
        return "A person named {$this->name}.";
    }
}

$person = new Person();
$person->name = 'John Doe';
echo $person;
$asString = (string)$person;
echo $asString;

```

__clone()

__clone() é invocado após a clonagem de um objeto possibilitando a modificação do clone.

```php
<?php

class Person
{
    public $name;

    public function __clone()
    {
        $this->name = "A {$this->name}'s clone.";
    }
}

$john = new Person();
$john->name = 'John Doe';
echo $john->name;

$clone = clone $john;
echo $clone->name;

```

__invoke()

__invoke() é chamado quando um objeto é usado/invocado como um método.

```php
<?php

class Engine
{
    public function __invoke()
    {
        var_dump('Engine turned on ...');
    }
}

$engine = new Engine();
$engine();

```

__debugInfo()

__debugInfo() é invocado quando a função var_dump() é chamada em objeto e retorna as propriedades que serão exibidas.

```php
<?php

class DebugStamped
{
    private $what;

    public function __construct($what)
    {
        $this->what = $what;
    }

    public function __debugInfo()
    {
        return [
            'what' => $this->what,
            'at' => time()
        ];
    }
}

$stamped = new DebugStamped('Something');
var_dump($stamped);

```

__sleep() e __wakeup()

__sleep() é invocado quando a funções serialize() é chamada em um objeto. É usada para realizar tarefas pré-serialização e indicar quais propriedades deverão ser serializadas.

__wakeup() é invocado quando um objeto é reconstruído a partir da função unserialize(). É usado para realizar tarefas de reinicialização, por exemplo, reconexões, recuperações de estados, etc.

Exemplo de __sleep() e __wakeup().

Salvando dados da conexão na serialização e reconectando durante a deserialização.

```php
class Connection
{
    protected $link;
    private $dsn;
    private $username;
    private $password;

    public function __construct($dsn, $username, $password)
    {
        $this->dsn = $dsn;
        $this->username = $username;
        $this->password = $password;
        $this->connect();
    }

    private function connect()
    {
        $this->link = new PDO($this->dsn, $this->username, $this->password);
    }

    public function __sleep()
    {
        return ['dsn', 'username', 'password'];
    }

    public function __wakeup()
    {
        $this->connect();
    }
}
```

__set_state()

__set_state() é invocado quando um código obtido a partir da função var_export() é executado.

```php
class Person
{
    public $name;

    public static function __set_state($data)
    {
        $person = new Person();
        $person->name = $data['name'] . ' reevaluated.';

        return $person;
    }
}

$person = new Person;
$person->name = 'John Doe';

$exported = var_export($person, true);
// Person::__set_state(['name' => 'John Doe',])

eval('$second = ' . $exported . ';');
var_dump($second);
// object(Person)#2 (1) {
//   ["name"]=>
//   string(21) "John Doe reevaluated."
// }
```

Constantes mágicas

São constantes especiais do PHP que trazem valores pré-definidos relativos ao local são usadas.

__LINE__	Retorna a linha atual do arquivo
__FILE__	Retorna o caminho do arquivo atual
__DIR__	Retorna o diretório do arquivo atual
__FUNCTION__	Retorna o nome da função atual.
__CLASS__	Retorna o nome da classe atual com o namespace.
__TRAIT__	Retorna o nome da trait atual com o namespace.
__METHOD__	Retorna o nome do método atual.
__NAMESPACE__	Retorna o nome do namespace atual.
ClassName::class	Retorna o FQN de uma classe.

Constantes mágicas

```php
<?php

namespace Entities;

class Person
{
    protected $name;

    public function __construct($name)
    {
        $this->name = $name;
        $nameLine = __LINE__ - 1;

        var_dump($nameLine);
        var_dump(__FILE__);
        var_dump(__DIR__);
        var_dump(__FUNCTION__);
        var_dump(__CLASS__);
        var_dump(__METHOD__);
        var_dump(__NAMESPACE__);
        var_dump(Person::class);
    }
}

$person = new Person('John Doe');
```

```
// int(11)
// string(20) "/var/www/html/index.php"
// string(19) "/var/www/html"
// string(11) "__construct"
// string(15) "Entities\Person"
// string(28) "Entities\Person::__construct"
// string(8) "Entities"
// string(15) "Entities\Person"

```

Variáveis variáveis

São variáveis que contém o nome de outras variáveis. São usadas quando uma variável possui um nome dinâmico.

Variáveis variáveis

```php
<?php

$mood = $argv[1];

$happy = 'Great \o/';
$sad = 'Why?';
$crying = 'Are you hurt?';
$angry = 'Do you want to kick something?';

echo $$mood;

```

```php
<?php

class Moods
{
    public $happy = 'Great \o/';
    public $sad = 'Why?';
    public $crying = 'Are you hurt?';
    public $angry = 'Do you want to kick something?';
}

$mood = $argv[1];
$moods = new Moods();

echo $moods->{$mood};

```

Métodos/funções variáveis

São variáveis que contém o nome de um método/função ou definição callable e que podem ser invocados com o uso de ().

Métodos/funções variáveis

```php
<?php

function talk()
{
    var_dump('Hey let me talk something ...');
}

function eat()
{
    var_dump('Time to eat!');
}

function workout()
{
    var_dump('No. Machines cannot do it.');
}

$action = $argv[1];

$action();

```

```php
<?php

class Machine
{
    public function talk()
    {
        var_dump('Hey let me talk something ...');
    }

    public function eat()
    {
        var_dump('Time to eat!');
    }

    public function workout()
    {
        var_dump('No. Machines cannot do it.');
    }
}

$action = $argv[1];
$machine = new Machine();

$machine->$action();

$callable = [$machine, $action];

$callable();
```

Late Static Bindings

É a capacidade de referenciar a classe chamada em um contexto estático que usa herança.

A palavra chave **static** é na implementação.

Late Static Binding

```php
<?php

class Father
{
    public static function me()
    {
        var_dump(__CLASS__);
    }

    public static function check()
    {
        self::me();
    }
}

class Child extends Father
{
    public static function me()
    {
        var_dump(__CLASS__);
    }
}

Child::check();
```

```php
<?php

class Father
{
    public static function me()
    {
        var_dump(__CLASS__);
    }

    public static function check()
    {
        static::me();
    }
}

class Child extends Father
{
    public static function me()
    {
        var_dump(__CLASS__);
    }
}

Child::check();
```

Traits

Traits são mixins de códigos, isto é, são componentes com código PHP que podem ser reutilizados por classes ou outros traits.

Exemplo de trait.

Reutilizando código com o uso de traits.

```php
trait HasData
{
    protected $data = [];

    public function __get($name)
    {
        return $this->data[$name];
    }

    public function __set($name, $value)
    {
        $this->data[$name] = $value;
    }
}

class Person
{
    use HasData;
}

class Product
{
    use HasData;
}
```

SPL
Standard PHP Library

A SPL é uma coleção de interfaces e classes pré-concebidas com funcionalidades comuns para resolução de diversos problemas.

SPL provê componentes como estruturas de dados, iteradores, exceções, dentre outros.

SplDoublyLinkedList

[Estrutura de dados]

Permite a implementação da estrutura de dados conhecida como lista duplamente encadeada.

```php
<?php

$linkedList = new SplDoublyLinkedList();
$linkedList->push(2);
$linkedList->push(3);
$linkedList->unshift(10);
$linkedList->rewind();

$current = $linkedList->current();
$linkedList->next();
$current = $linkedList->current();
$linkedList->prev();
$current = $linkedList->current();

```

FilesystemIterator

[Interador]

Permite a iteração de arquivos e pastas no sistema de arquivos.

```php
<?php

$directory = '/tmp';

$filesystem = new FilesystemIterator($directory);

foreach ($filesystem as $fileInfo) {
    var_dump($fileInfo->getFilename());
}

```

Reflection

Reflection tem haver com as capacidades de metaprogramação de uma linguagem, isto é, a capacidade que a linguagem possui de alterar ou controlar a si mesma.

Alguém vai se lembrar ...

Uso de annotations para configurar o ORM.

```php
<?php

/**
 * @ORM\Entity
 * @ORM\Table(name="products")
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue
     */
    protected $id;
    /**
     * @ORM\Column(type="string")
     */
    protected $name;
}

```

Um exemplo

Recuperando os DocBlocks da classe e dos atributos.

```php
<?php

/**
 * @table tbl_person
 */
class Person
{
    /**
     * @column person_name
     */
    public $name;
}

$class = new ReflectionClass('Person');
$classDoc = $class->getDocComment();
var_dump($classDoc);
// Parse $classDoc

$attribute = new ReflectionProperty('Person', 'name');
$attributeDoc = $attribute->getDocComment();
var_dump($attributeDoc);
// Parse $attributeDoc
```

Bônus: PHP 7.4

__serialize() e __unserialize()

__serialize() é invocado quando a função serialize() é chamada em um objeto. Retorna os dados que deverão ser serializados.

__unserialize() é invocado quando um objeto é reconstruído a partir da função unserialize(). É usado para restaurar o estado do objeto.

Arrow functions

São uma forma mais rápida de escrever "funções de uma linha".

```php
<?php

array_map(function (Product $product) {
    return $product->id;
}, $products);

array_map(fn (Product $product) => $product->id, $products);

```

Propriedades tipadas

Agora é possível o uso de type hints em propriedades.

```php
<?php

class Person
{
    protected string $name;

    protected ?DateTime $birthDate;
}

```

Null coalescing assignment operator

É um shorthand para operações de null coalescing.

```php
<?php

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

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

```

Array spread operator

Uso do operador de spread agora é possível com arrays com chaves numéricas.

```php
<?php

$animals = ['Dog', 'Cat'];
$plants = ['Grass', 'Tree'];

$all = ['Human', ...$animals, ...$plants, 'others'];

```

Referências

PHP official website. Disponível em <https://www.php.net/>

Obrigado

https://github.com/naylonkessler/php-features-magicas-ou-nao

Naylon Kessler de Aquino

www.naylonkessler.com

naylon.kessler@gmail.com