Magic Methods in PHP OOP and Their Use Cases

Magic methods in PHP are special methods that are triggered automatically in certain situations. They are prefixed and suffixed with double underscores (__). These methods allow you to control and customize the behavior of your PHP objects.

Benefits of Using Magic Methods

1. **Encapsulation**: Control access to object properties and methods.
2. **Flexibility**: Enable dynamic behavior and reduce repetitive code.
3. **Error Handling**: Gracefully manage undefined properties or methods.

However, use them judiciously, as excessive reliance on magic methods can make code harder to debug and maintain.

Here's an overview of some commonly used magic methods and their use cases:

---

1. __get($property)

- **Purpose**: Handles attempts to read inaccessible or non-existent properties.
- **Use Case**: Implement lazy loading or encapsulation for class properties.

index.php

Magic-methods > index.php > ...

```php
<?php

class User{

    private $data = ['name'=>'mohamed', 'email'=>'mohamed@gmail.com'];

    Tabnine | Edit | Test | Explain | Document | Ask
    public function __get($prop)
    {
        return $this->data[$prop];
    }

}

$user = new User();

echo $user->name; // Output: mohamed

echo $user->email; // Output: mohamed@gmail.com
```

---

2. __set($property, $value)

- **Purpose**: Handles attempts to write inaccessible or non-existent properties.
- **Use Case**: Control property assignment and validation.

```php
class User {
    private $data =[];
    Tabnine | Edit | Test | Explain | Document | Ask
    public function __set($prop, $value){
        $this->data[$prop] = $value;
    }
    Tabnine | Edit | Test | Explain | Document | Ask
    public function __get($prop){
        return $this->data[$prop] ?? null;
    }
}

$user = new User();

$user->age = 30;

echo $user->age; // Output: 30
```

3. __call($method, $arguments)

- **Purpose**: Handles calls to inaccessible or non-existent methods.
- **Use Case**: Implement dynamic method handling.

```php
class Magic {
    Tabnine | Edit | Test | Explain | Document | Ask
    public function __call($method, $arguments) {
        return "Method $method called with arguments: " . implode(', ', $arguments);
    }
}

$obj = new Magic();
echo $obj->doSomething('arg1', 'arg2');

// Outputs: Method doSomething called with arguments: arg1, arg2
```

---

4. __callStatic($method, $arguments)

- **Purpose**: Handles calls to inaccessible or non-existent static methods.
- **Use Case**: Dynamic static method invocation.

```php
class StaticMagic {
    Tabnine | Edit | Test | Explain | Document | Ask
    public static function __callStatic($method, $arguments) {
        return "Static method $method called with arguments: " . implode(', ', $arguments);
    }
}

echo StaticMagic::doSomething('arg1', 'arg2');
// Outputs: Static method doSomething called with arguments: arg1, arg2
```

---

5. __clone()

- **Purpose**: Executes when an object is cloned.
- **Use Case**: Customize object cloning behavior.

```php
class Person {
    public $name;

    Tabnine | Edit | Test | Explain | Document | Ask
    public function __clone() {
        $this->name = "Cloned " . $this->name;
    }
}

$original = new Person();
$original->name = "John";

$cloned = clone $original;
echo $cloned->name; // Outputs: Cloned John
```

---

6. __toString()

- **Purpose**: Converts an object to a string.
- **Use Case**: Provide a human-readable representation of the object.

```php
class Product {
    private $name;
    private $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    public function __toString() {
        return "Product: $this->name, Price: $this->price";
    }
}

$product = new Product('Laptop', 1500);
echo $product; // Outputs: Product: Laptop, Price: 1500
```

---

7. __invoke()

- **Purpose**: Executes when an object is called as a function.
- **Use Case**: Simplify callable object behavior.

```php
class Adder {
    Tabnine | Edit | Test | Explain | Document | Ask
    public function __invoke($a, $b) {
        return $a + $b;
    }
}

$add = new Adder();
echo $add(3, 4); // Outputs: 7
```
