16 Magic Methods That PHP Developers Must Know

tutorialdocs.com/article/16-php-magic-methods.html

The methods which begin with two underscores (__) are called Magic Methods in PHP, and they play an important role. Magic methods include:

Method Name	Description
__construct()	the constructor of a class
__destruct()	the destructor of a class
__call($funName, $arguments)	The __call() method will be called when an undefined or inaccessible method is called.
__callStatic($funName, $arguments)	The __callStatic() method will be called when an undefined or inaccessible static method is called.
__get($propertyName)	The __get() method will be called when getting a member variable of a class.
__set($property, $value)	The __set() method will be called when setting a member variable of a class.
__isset($content)	The __isset() method will be called when calling isset() or empty() for an undefined or inaccessible member.
__unset($content)	The __unset() method will be called when calling reset() for an undefined or inaccessible member.
__sleep()	The __sleep() method will be called first when executing serialize().
__wakeup()	The __wakeup() method will be called first when deserialization() is executed.
__toString()	The __toString() method will be called when using echo method to print an object directly.
__invoke()	The __invoke() method will be called when trying to call an object in a way of calling function.
__set_state($an_array)	The __set_state() method will be called when calling var_export().
__clone()	The __clone() method will be called when the object is copied.
__autoload($className)	Try to load an undefined class.
__debugInfo()	Print debug information.

In this article, we will show you how to use these PHP magic methods with some examples.

1.__construct()

The PHP constructor is the first method that is automatically called after the object is created. Each class has a constructor. If you do not explicitly declare it, then there will be a default constructor with no parameters and empty content in the class.

1) the use of the method

Constructors are usually used to perform some initialization tasks, such as setting initial values for member variables when creating objects.

2) the declaration format of the constructor in a class

```
function __constrct([parameter list]){

    Method body

}
```

> Note: Only one constructor can be declared in the same class, because PHP does not support constructor overloading.

Here is a complete example:

```
<?php
    class Person
    {
            public $name;
            public $age;
            public $sex;


        public function __construct($name="", $sex="Male", $age=22)
        {
            $this->name = $name;
            $this->sex = $sex;
            $this->age = $age;
        }


        public function say()
        {
            echo "Name：" . $this->name . ",Sex：" . $this->sex . ",Age：" . $this->age;
        }

    }
```

Create object $Person1 without any parameters.

```
$Person1 = new Person();
echo $Person1->say();
```

Create object $Person2 with parameter "Jams".

```
$Person2 = new Person("Jams");
echo $Person2->say();
```

Create object $Person3 with three parameters.

```
$Person3 = new Person ("Jack", "Male", 25);
echo $Person3->say();
```

2. __destruct()

Now that we have already known what the constructor is, the destructor is the opposite.

Destructor allows you to perform some operations before destroying an object, such as closing a file, emptying a result set, and so on.

Destructor is a new feature introduced by PHP5.

The declaration format of the destructor is similar to that of the constructor __construct(), which means that __destruct() is also started with two underscores, and the name of the destructor is also fixed.

1) the declaration format of destructor

```
function __destruct()
{

}
```

> Note: Destructor cannot have any parameters.

2) the use of the destructor

In general, the destructor is not very common in PHP. It's an optional part of a class, usually used to complete some cleanup tasks before the object is destroyed.

Here is an example of using the destructor:

```
<?php
class Person{

    public $name;
    public $age;
    public $sex;

    public function __construct($name="", $sex="Male", $age=22)
    {
        $this->name = $name;
        $this->sex  = $sex;
        $this->age  = $age;
    }


    public function say()
    {
        echo "Name：".$this->name.",Sex：".$this->sex.",Age：".$this->age;
    }


    public function __destruct()
    {
            echo "Well, my name is ".$this->name;
    }
}

$Person = new Person("John");
unset($Person);
```

The output of the program above is:

```
Well, my name is John
```

3. __call()

This method takes two parameters. The first parameter $function_name will automatically receive the undefined method name, while the second $arguments will receive multiple arguments of the method as an array.

1)The use of __call() method

```
function __call(string $function_name, array $arguments)
{

}
```

When an undefined method is called in a program, the __call() method will be called automatically.

Here is an example:

```
<?php
class Person
{
    function say()
    {
           echo "Hello, world!<br>";
    }

    function __call($funName, $arguments)
    {
          echo "The function you called：" . $funName . "(parameter：" ;
          print_r($arguments);
          echo ")does not exist!！<br>\n";
    }
}
$Person = new Person();
$Person->run("teacher");
$Person->eat("John", "apple");
$Person->say();
```

The results are as follows:

```
The function you called: run (parameter: Array([0] => teacher)) does not exist!
The function you called: eat (parameter: Array([0] => John[1] => apple)) does not exist!
Hello world!
```

4. __callStatic()

When an undefined static method is called in a program, the __callStatic() method will be called automatically.

The use of __callStatic() is similar to the __call(). Here is an example:

```
<?php
class Person
{
    function say()
    {
        echo "Hello, world!<br>";
    }

    public static function __callStatic($funName, $arguments)
    {
        echo "The static method you called：" . $funName . "(parameter：" ;
        print_r($arguments);
        echo ")does not exist！<br>\n";
    }
}
$Person = new Person();
$Person::run("teacher");
$Person::eat("John", "apple");
$Person->say();
```

The results are as follows:

```
The static method you called: run (parameter: Array([0] => teacher)) does not exist!
The static method you called: eat (parameter: Array([0] => John[1] => apple)) does not
exist!
Hello world!
```

5. __get()

When you try to access a private property of an external object in a program, the program will throw an exception and end execution. We can use the magic method __get() to solve this problem. It can get the value of the private property of the object outside the object. Here is an example:

```
<?php
class Person
{
    private $name;
    private $age;

    function __construct($name="", $age=1)
    {
        $this->name = $name;
        $this->age = $age;
    }

    public function __get($propertyName)
    {
        if ($propertyName == "age") {
            if ($this->age > 30) {
                return $this->age - 10;
            } else {
                return $this->$propertyName;
            }
        } else {
            return $this->$propertyName;
        }
    }
}
$Person = new Person("John", 60);
echo "Name：" . $Person->name . "<br>";
echo "Age：" . $Person->age . "<br>";
```

The results are as follows:

```
Name: John
Age: 50
```

6. __set()

The __set($property, $value) method is used to set the private property of the object. When an undefined property is assigned, the __set() method will be triggered and the passed parameters are the property name and value that are set.

Here is the demo code:

```php
<?php
class Person
{
    private $name;
    private $age;

    public function __construct($name="",  $age=25)
    {
        $this->name = $name;
        $this->age  = $age;
    }

    public function __set($property, $value) {
        if ($property=="age")
        {
            if ($value > 150 || $value < 0) {
                return;
            }
        }
        $this->$property = $value;
    }

    public function say(){
        echo "My name is ".$this->name.",I'm ".$this->age." years old";
    }
}

$Person=new Person("John", 25);
$Person->name = "Lili";
$Person->age = 16;
$Person->age = 160;
$Person->say();
```

Here is the result:

```
My name is Lili, I'm 16 years old
```

7. __isset()

Before using the __isset () method, let me explain the use of the isset () method first. The isset () method is mainly used to determine whether the variable is set.

If you use the isset() method outside the object, there are two cases:

1. If the parameter is a public property, you can use the isset() method to determine whether the property is set or not.
2. If the parameter is a private property , the isset() method will not work.

So for the private property, is there any way to know if it is set? Of course, as long as we define a __isset() method in a class, we can use the isset() method outside the class to determine whether the private property is set or not.

When the isset() or empty() is called on an undefined or inaccessible property, the __isset() method will be called. Here is an example:

```
<?php
class Person
{
    public $sex;
    private $name;
    private $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }


    public function __isset($content) {
        echo "The {$content} property is private, the __isset() method is called
automatically.<br>";
        echo  isset($this->$content);
    }
}

$person = new Person("John", 25);
echo isset($person->sex),"<br>";
echo isset($person->name),"<br>";
echo isset($person->age),"<br>";
```

The results are as follows:

```
1
The name property is private, the __isset() method is called automatically.
1
The age property is private, the __isset() method is called automatically.
1
```

8. __unset()

Similar to the __isset() method, the __unset() method is called when the unset() method is called on an undefined or inaccessible property. Here is an example:

```
<?php
class Person
{
    public $sex;
    private $name;
    private $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }


    public function __unset($content) {
        echo "It is called automatically when we use the unset() method outside the
class.<br>";
        echo  isset($this->$content);
    }
}

$person = new Person("John", 25);
unset($person->sex),"<br>";
unset($person->name),"<br>";
unset($person->age),"<br>";
```

The results are as follows:

```
It is called automatically when we use the unset() method outside the class.
1
It is called automatically when we use the unset() method outside the class.
1
```

9. __sleep()

The serialize() method will check if there is a magic method __sleep() in the class. If it exists, the method will be called first and then perform the serialize operation.

The __sleep() method is often used to specify the properties that need to be serialized before saving data. If there are some very large objects that don't need to be saved all, then you will find this feature is very useful.

For details, please refer to the following code:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }


    public function __sleep() {
        echo "It is called when the serialize() method is called outside the class.<br>";
        $this->name = base64_encode($this->name);
        return array('name', 'age');
    }
}

$person = new Person('John');
echo serialize($person);
echo '<br/>';
```

The results are as follows：

```
It is called when the serialize() method is called outside the class.
O:6:"Person":2:{s:4:"name";s:8:"5bCP5piO";s:3:"age";i:25;}
```

10. __wakeup()

In contrast to the __sleep() method, the __wakeup() method is often used in deserialize operations, such as re-building a database connection, or performing other initialization operations.

Here is an example:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }


    public function __sleep() {
        echo "It is called when the serialize() method is called outside the class.<br>";
        $this->name = base64_encode($this->name);
        return array('name', 'age');
    }


    public function __wakeup() {
        echo "It is called when the unserialize() method is called outside the class.
<br>";
        $this->name = 2;
        $this->sex = 'Male';

    }
}

$person = new Person('John');
var_dump(serialize($person));
var_dump(unserialize(serialize($person)));
```

The results are as follows：

```
It is called when the serialize() method is called outside the class.
string(58) "O:6:"Person":2:{s:4:"name";s:8:"5bCP5piO";s:3:"age";i:25;}"
It is called when the unserialize() method is called outside the class.
object(Person)
```

11. __toString()

The __toString() method will be called when using echo method to print an object directly.

Note: This method must return a string, otherwise it will throw a fatal error at `E_RECOVERABLE_ERROR` level. And you also can't throw an exception in the __toString() method.

Here is an example:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }

    public function __toString()
    {
        return  'go go go';
    }
}

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

The results are as follows:

```
go go go
```

So what if the __toString() method is not defined in the class? Let's have a try.

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }

}

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

The results are as follows:

```
Catchable fatal error: Object of class Person could not be converted to string in
D:\phpStudy\WWW\test\index.php on line 18
```

Obviously, It reports a fatal error on the page, which is not allowed by PHP syntax.

12. __invoke()

When you try to call an object in the way of calling a function, the __ invoke() method will be called automatically.

> Note: This feature is only valid in PHP 5.3.0 and above.

Here is an example:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }

    public function __invoke() {
        echo 'This is an object';
    }

}

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

Here is the result:

```
This is an object
```

If you insist on using objects as methods（but not defining the __invoke() method），then you will get the following result：

```
Fatal error: Function name must be a string in D:\phpStudy\WWW\test\index.php on line 18
```

13.__set_state()

Starting from PHP 5.1.0, the __set_state() method is called automatically when calling var_export() to export the class code.

The parameters of the __set_state() method is an array containing the values of all the properties, with the format of array('property' => value,...)

We don't define __set_state() method in the following example:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }

}

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

Here is the result:

```
Person::__set_state(array( 'sex' => 'Male', 'name' => 'John', 'age' => 25, ))
```

Obviously, the properties of the object are printed.

Now let's have a look at the other case of being defined the __set_state() method:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }

    public static function __set_state($an_array)
    {
        $a = new Person();
        $a->name = $an_array['name'];
        return $a;
    }

}

$person = new Person('John');
$person->name = 'Jams';
var_export($person);
```

Here is the result:

```
Person::__set_state(array( 'sex' => 'Male', 'name' => 'Jams', 'age' => 25, ))
```

14. __clone()

In PHP we can use the clone keyword to copy objects with the following syntax:

```
$copy_of_object = clone $object;
```

However, using clone keyword is just a shallow copy, because all referenced properties will still point to the original variable.

If a __clone() method is defined in a object, then the __clone() method will be called in the copy-generated object and can be used to modify the value of the property (if necessary).

Here is an example:

```
<?php
class Person
{
    public $sex;
    public $name;
    public $age;

    public function __construct($name="",  $age=25, $sex='Male')
    {
        $this->name = $name;
        $this->age  = $age;
        $this->sex  = $sex;
    }

    public function __clone()
    {
        echo __METHOD__."your are cloning the object.<br>";
    }

}

$person = new Person('John');
$person2 = clone $person;

var_dump('persion1:');
var_dump($person);
echo '<br>';
var_dump('persion2:');
var_dump($person2);
```

The results are as follows:

```
Person::__clone your are cloning the object.
string(9) "persion1:" object(Person)#1 (3) { ["sex"]=> string(3) "Male" ["name"]=>
string(6) "John" ["age"]=> int(25) }
string(9) "persion2:" object(Person)#2 (3) { ["sex"]=> string(3) "Male" ["name"]=>
string(6) "John" ["age"]=> int(25) }
```

15.__autoload()

The __autoload() method can try to load an undefined class.

In the past, if you were to create 100 objects in a program file, then you had to use include() or require() to contain 100 class files, or you had to define the 100 classes in the same class file, for example:

```
require_once('project/class/A.php');
require_once('project/class/B.php');
require_once('project/class/C.php');
.
.
.

if (ConditionA) {
    $a = new A();
    $b = new B();
    $c = new C();

} else if (ConditionB) {
    $a = newA();
    $b = new B();

}
```

So what if we use the __autoload() method?

```
function  __autoload($className) {
    $filePath = “project/class/{$className}.php”;
    if (is_readable($filePath)) {
        require($filePath);
    }
}

if (ConditionA) {
    $a = new A();
    $b = new B();
    $c = new C();

} else if (ConditionB) {
    $a = newA();
    $b = new B();

}
```

When the PHP engine uses class A at the first time, if class A is not found, then the __autoload method will be called automatically and the class name "A" will be passed as a parameter. So what we need to do in the __autoload() method is to find the corresponding class file based on the class name and then include the file. If the file is not found, then the php engine will throw an exception.

16. __debugInfo()

The __debugInfo() method will be called when the var_dump() method is executed. If the __debugInfo() method is not defined, then the var_dump() method will print out all the properties in the object.

Here is an example:

```
<?php
class C {
    private $prop;

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


    public function __debugInfo() {
        return [
            'propSquared' => $this->prop ** 2,
        ];
    }
}

var_dump(new C(42));
```

Here is the result：

```
object(C)#1 (1) { ["propSquared"]=> int(1764) }
```

> Note: The __debugInfo() method should be applied to the PHP 5.6.0 and above.

Summarize

The above are the PHP magic methods that I understand, of which commonly used include __set(), __get() and __autoload(). If you still have any questions, you can get more help from the official PHP website.