Sử dụng đối tượng trong PHP

Nhắc lại bài cũ

- Câu lệnh điều khiển
- Khởi tạo và sử dụng hàm

Nội dung bài học

1. Khởi tạo và sử dụng đối tượng

Trong phần này có các nội dung:

1.1. Khởi tạo và sử dụng lớp

1.2. Viết hằng, thuộc tính và phương thức của lớp

1.3. Một số kỹ năng bổ sung

1.4. Làm việc với kế thừa

1.1. Khởi tạo và sử dụng lớp

Trong phần này có các nội dung:

1.1.1. Viết thuộc tính

1.1.2. Viết hàm tạo và hàm hủy

1.1.3. Viết phương thức

1.1.4. Khởi tạo và sử dụng đối tượng

1.1.1. Viết thuộc tính

- Chia làm ba loại chính:
  - Public: có thể truy cập trực tiếp bởi đoạn mã bên ngoài lớp
  - Private: không thể truy cập trực tiếp bởi đoạn mã bên ngoài lớp
  - Protected: không thể truy cập trực tiếp bởi đoạn mã bên ngoài lớp
- Mặc định thuộc tính là public
- Có thể đặt giá trị ban đầu cho thuộc tính là số, chuỗi, Bool hoặc NULL
- Viết nhiều thuộc tính trên cùng một dòng: dùng dấu phẩy "," để phân tách các thuộc tính
- Cú pháp:

```
[ public | protected | private ] $propertyName [ = initialValue ]
```

Viết thuộc tính

- Hướng dẫn viết mã thuộc tính:

```
//Thuộc tính private
private $firstName;

//Thuộc tính public với giá trị khởi tạo
public $comment = '';

//Thuộc tính protected
protected $counter;

//Năm thuộc tính trên cùng một dòng
private $category, $id, $nane, $description, $price;
```

1.1.2. Viết hàm tạo và hàm hủy

- Hàm tạo (phương thức khởi tạo): phương thức đặc biệt được thực hiện khi một đối tượng mới được tạo ra từ lớp. Phương thức này thường khởi tạo các thuộc tính của đối tượng
- Cú pháp:

```
public function __construct([parameterList]) {
//câu lệnh thực thi
}
```

- Hàm hủy (phương thức hủy): phương thức đặc biệt được thực hiện khi một đối tượng không còn được sử dụng. Nói cách khác, nó được thực hiện khi không có biến tham chiếu đến đối tượng
- Cú pháp:

```
public function __destruct() {
//câu lệnh thực thi
}
```

Viết hàm tạo và hàm hủy

- Biến đặc biệt $this lưu một tham chiếu đến các đối tượng hiện tại, cho phép truy cập vào các thuộc tính và phương thức của đối tượng hiện tại
- Toán tử truy cập đối tượng (->) cung cấp truy cập đến thuộc tính và phương thức của đối tượng
- Ví dụ:

```
//Hàm tạo cho đối tượng Category cùng với giá trị mặc định
public function __construct($id = NULL, $name = NULL) {
$this->id = $id;
$this->name = $name;
}

//Hàm hủy cho lớp cơ sở dữ liệu
public function __destruct() {
$this->dbConnection->close();
}
```

1.1.3. Viết phương thức

- Cú pháp:

```
[public | private | protected] function functionName ([parameterList]) {
//câu lệnh thực thi
}
```

- Mặc định, thuộc tính của phương thức là public

- Ví dụ:

```
//Phương thức private
private function internationalizePrice($country = 'US') {
    switch ($country) {
        case 'US':
            return '$' . number_format($this->price, 2);
        case 'DE':
            return number_format($this->price, 2, , '.') . ' DM';

    }
}
```

Viết phương thức

```php
//Phương thức truy cập thuộc tính của đối tượng hiện tại
public function showDescription() {
    echo $this->description;
}

//Phương thức gọi phương thức của đối tượng hiện tại
public function showPrice($country = 'US') {
    echo $this->internationalizePrice($country);
```

1.1.4. Khởi tạo và sử dụng đối tượng

- Đối tượng: thực thể (instance) của lớp
- Cú pháp khởi tạo đối tượng:

```
$objectName = new ClassName(argumentList)
```

Ví dụ:

```
$brass = new Category(4, 'Brass')
```

- Cú pháp truy cập thuộc tính của đối tượng:

```
$objectName- >propertyName
```

Ví dụ:

```
//Thiết lập thuộc tính
$trumpet->comment = 'Discontinued';

//Lấy thuộc tính
$comment = $trumpet->comment;
```

Khởi tạo và sử dụng đối tượng

- Cú pháp truy cập phương thức của đối tượng:

```
$objectName->methodName(argvmentList)
```

Ví dụ:

```
//Gọi phương thức getFormattedPrice
$price = $trumpet->getFormattedPrice();
```

- Nếu một phương thức trả về đối tượng thì có thể sử dụng hàm hoặc phương thức làm tham chiếu đến đối tượng và tiếp tục truy cập vào các thuộc tính và phương thức của đối tượng trả về

Ví dụ:

```
//Chuỗi đối tượng
echo $trumpet->getCategory()->getName();
```

1.2. Viết hằng, thuộc tính và phương thức của lớp

Trong phần này có các nội dung:

1.2.1. Viết hằng của lớp

1.2.2. Viết thuộc tính và phương thức tĩnh

1.2.1. Viết hằng của lớp

- Hằng của lớp là giá trị không đổi thuộc về lớp, không phải đối tượng được tạo từ lớp
- Cú pháp truy cập vào một hằng của lớp:
  - Truy cập bên trong lớp: `self::constName`
  - Truy cập bên ngoài lớp: `className::constName`

- Thuộc tính của hằng của lớp luôn là public
- Hằng của lớp thường được sử dụng để xác định tập hợp các tùy chọn được truyền cho phương thức trong lớp

Viết hằng của lớp

Ví dụ:

```
class Person {
    const MALE = 'm';
    private $gender;

    public function setGender($value) {
        if ($value == self::MALE) { //Sử dụng hằng bên trong lớp
            $this->gender = $value;
        }
    }
}
$person = new Person();
$person->setGender(Person::MALE); //Sử dụng hằng bên ngoài lớp
```

1.2.2. Viết thuộc tính và phương thức tĩnh

- Thuộc tính/phương thức tĩnh (static): thuộc tính/phương thức thuộc về một lớp, chứ không thuộc đối tượng được tạo ra từ lớp.
- Cú pháp khai báo:
- Thuộc tính tĩnh:

```
[public | protected | private] static $propertyName [= initialValue]
```

- Phương thức tĩnh:

```
[public | private | protected] static function functionName ([paraList]){
//câu lệnh thực thi
}
```

- Trong ứng dụng PHP, mỗi người dùng có một không gian mã riêng nên thuộc tính và phương thức tĩnh không được chia sẻ giữa nhiều người dùng với nhau

Viết thuộc tính và phương thức tĩnh

- Truy cập và thuộc tính/phương thức tĩnh:
  - Bên trong lớp:

```
self::$propertyName
self::methodName([parameterList])
```

  - Bên ngoài lớp:

```
className::$propertyName
className::methodName([parameterList])
```

Ví dụ:

```
class Category {
    private static $objectCount = 0; //khai báo thuộc tính tĩnh
    public function __construct() {
        self::$objectCount++; //cập nhật thuộc tính tĩnh
    }
}

echo Category::getObjectCount(); //Sử dụng phương thức tĩnh
echo Category::$objectCount; //Sử dụng thuộc tính public tĩnh
```

1.3. Một số kỹ năng bổ sung

Trong phần này có các nội dung:

1.1.1. Lặp qua các thuộc tính của đối tượng

1.1.2. Sao chép và so sánh đối tượng

1.1.3. Kiểm tra đối tượng

1.1.1. Lặp qua các thuộc tính của đối tượng

- Dùng vòng lặp foreach để duyệt từng thuộc tính của đối tượng
- Cú pháp:

```
foreach{$objectName as [ $propertyName => ] $propertyValue) {
//câu lệnh thực thi
}
```

```
class Employee {
    public $FullName; private $Age;
    public function __construct($name, $age) {
        $this->FullName = $name; $this->Age = $age;
    }
    public function showAll {
        foreach($this as $name => $value) echo "$name : $value";
    }
}
$employee = new Employee('John', '20');
$employee->showAll(); //Hiển thị tất cả thuộc tính
foreach($this as $name => $value) { //Chỉ hiển thị thuộc tính public
    echo "$name : $value";
}
```

1.1.2. Sao chép và so sánh đối tượng

- Cú pháp tạo bản sao đối tượng và gán cho một biến:

```
$NewObject = clone $objectName
```

- Cần phân biệt sao chép và tham chiếu đối tượng

Ví dụ:

```
$man = new Employee('John','20');

//Tạo tham chiếu thứ hai tới đối tượng
$man2 = $man; //cả hai biến tham chiếu tới cùng một đối tượng
$man2->setName('Newton'); //đổi tên cả 2 đối tượng man và man2

//Tạo nhân bản của đối tượng
$man3 = clone $man; //sao chép đối tượng
$man3->setPrice(899.95); //chỉ có tên của man3 là thay đổi
```

Sao chép và so sánh đối tượng

- Sử dụng toán tử so sánh bằng (==) để kiểm tra xem cả hai đối tượng có phải là thể hiện của cùng một lớp và có cùng giá trị cho mọi thuộc tính không
- Sử dụng toán tử định danh (===) để kiểm tra xem cả hai biến đối tượng có tham chiếu tới cùng một thể hiện của đối tượng không

Ví dụ:

```
//Sử dụng toán tử so sánh bằng (==)
$result_1 = ($man == $man2); // $result_1 là FALSE
$result_2 = ($man == $man3); // $result_2 là TRUE

//Sử dụng toán tử định danh (===)
$result_3 = ($man === $man3); // $result_3 là FALSE
$trumpet_2 = $trumpet;
$result_4 = ($man === $man2); // $result_4 là TRUE
```

1.1.3. Kiểm tra đối tượng

- Các hàm kiểm tra một đối tượng:

Hàm	Mô tả
`class_exists($class)`	Trả về TRUE nếu lớp chỉ định đã được định nghĩa.
`get_class($object)`	Trả về tên lớp của đối tượng chỉ định dưới dạng chuỗi.
`is_a($object, $class)`	Trả về TRUE nếu đối tượng chỉ định là một thể hiện của lớp chỉ định.
`property_exists($object, $property)`	Trả về TRUE nếu đối tượng chỉ định có thuộc tính chỉ định.
`method_exists($object, $method)`	Trả về TRUE nếu đối tượng chỉ định có phương thức chỉ định.

Ví dụ:

```php
//Xác định liệu đối tượng có là thể hiện của lớp
if (is_a($man, 'John')) {
}

//Xác định liệu đối tượng có thuộc tính
if (property_exists($man, 'age')) {
}

//Xác định liệu đối tượng có phương thức
if (method_exists($man, 'getName')) {
}
```

1.4. Khởi tạo và sử dụng đối tượng

Trong phần này có các nội dung:

1.4.1. Kế thừa một lớp

1.4.2. Sử dụng mức truy xuất protected

1.4.3. Tạo lớp và phương thức tổng quát

1.4.4. Tạo lớp và phương thức final

1.4.5. Làm việc với giao diện

1.4.1. Kế thừa một lớp

- Kế thừa một lớp: tạo ra một lớp mới (gọi là lớp con/lớp dẫn xuất/lớp phụ) có các thuộc tính và phương thức của lớp cũ (gọi là lớp cha/lớp cơ sở)
- Cú pháp kế thừa:

```
//Khai báo lớp con
class LopCon extends LopCha {
    //Nội dung lớp con
}
```

- Lớp con có thể mở rộng lớp cha bằng cách thêm các thuộc tính và phương thức mới
- Lớp con có thể ghi đè lên phương thức từ lớp cha bằng phiên bản phương thức của riêng mình
- Lớp con có thể gọi phương thức từ lớp cha theo cú pháp:

```
parent::methodName([parameterList])
```

Kế thừa một lớp

Ví dụ:

```php
//Lớp cha
class Person {
    private $Name;
    public function __construct($name) {$this->Name = $name;}
    public function getName() {return $this->Name; }
}

//Lớp con
class Employee extends Person {
    private $Age;
    public function __construct($name, $age) {
        $this->Age = $age;
        parent::__construct($name); //gọi hàm tạo lớp cha
                                    //để kết thúc khởi tạo
    }
    public function getAge() { return $this->Age;}
}
```

1.4.2. Sử dụng mức truy xuất protected

- Phân biệt các mức truy xuất:

Mức truy xuất	Truy cập từ bên ngoài lớp	Truy cập từ lớp con
public	Có	Có
protected	Không	Có
private	Không	Không

Ví dụ:

```php
//Lớp cha
class Person {
    protected $Name;
    private $Age;

}

//Lớp con
class Employee extends Person {
    private $email;
    public getName() {
        return $this->Name; //sử dụng thuộc tính protected
    }
}
```

1.4.3. Tạo lớp và phương thức tổng quát

- Lớp cụ thể (concrete): lớp có thể được sử dụng để tạo đối tượng
- Lớp tổng quát (abstract): lớp không thể sử dụng để tạo đối tượng mà thường đóng vai trò là lớp cha cho các lớp khác
- Phương thức tổng quát: xác định tên và các tham số cho phương thức nhưng không cung cấp khối mã cài đặt phương thức
- Không bắt buộc phải có lớp tổng quát để tạo phương thức tổng quát
- Lớp con cụ thể của lớp tổng quát phải cung cấp cài đặt cho tất cả các phương thức tổng quát trong lớp tổng quát
- Cú pháp khai báo lớp và phương thức tổng quát: thêm từ khóa abstract vào trước từ khóa class/function

Tạo lớp và phương thức tổng quát

Ví dụ:

```php
//Lớp tổng quát với phương thức tổng quát
abstract class Person {
    private $Name;
    public function __construct($name) { $this->Name = $name; }
    abstract public function getName();
}

//Lớp cụ thể cài đặt lớp tổng quát
class Customer extends Person {
    private $Email;
    public function __construct($name, $email) {
        $this->Phone = $email;
        parent::__construct($name);
    }
    //cài đặt cụ thể phương thức tổng quát
    public function getName() {
        return $this->getName();
    }
}
$customer = new Person('Doe');  //lỗi hệ thống
$customer = new Customer('John', 'jdoe@yahoo.com');
echo $customer->getName();
```

1.4.4. Tạo lớp và phương thức final

- Phương thức final: không thể bị ghi đè bởi phương thức trong lớp con. Do đó, tất cả các lớp con phải sử dụng phiên bản cuối cùng của phương thức này
- Lớp final: không thể được kế thừa bởi lớp con
- Cú pháp tạo lớp và phương thức final: thêm từ khóa final vào trước từ khóa class/function
- Hướng dẫn cách làm một lớp không thể bị kế thừa:

```
//Lớp final
final class Employee {
    //các thuộc tính và phương thức khác như ví dụ trước
}
//Lớp con cố gắng kế thừa lớp cha => gây lỗi hệ thống
class PartTime extends Employee {
    //các thuộc tính và phương thức
}
```

Tạo lớp và phương thức final

- Hướng dẫn cách tránh tình trạng ghi đè phương thức:

```php
//Lớp với phương thức final
class Person {
    //các thuộc tính và phương thức khác như ví dụ trước
    final public function getName() { return $this->Name; }
}

//Lớp con cố gắng ghi đè phương thức cuối => lỗi hệ thống
class Employee extends Person {
    //các thuộc tính và phương thức khác như ví dụ trước
    //phương thức này cố gắng ghi đè phương thức final - gây lỗi hệ thống
    public function getName() { return ucwords($this->Name); }
}
```

1.4.5. Làm việc với giao diện

- Giao diện (interface): tập các phương thức có thể được cài đặt bởi một lớp
- Giao diện chỉ cung cấp tên phương thức và danh sách tham số
- Các phương thức trong giao diện phải là public
- Lớp thực thi giao thức phải cung cấp cài đặt cho từng phương thức được định nghĩa bởi giao diện. Từ khóa thực thi là implement
- Giao diện có thể định nghĩa các hằng có hiệu lực với bất kỳ lớp nào cài đặt giao diện
- Một lớp có thể thực thi nhiều giao diện
- Cú pháp khai báo giao diện:

```
interface interfaceName {
    const constantName = constantValue;
    public function methodName([parameterList]);
}
```

Làm việc với giao diện

Ví dụ:

```php
//Giao diện hiển thị đối tượng
interface Showable {
    public function show() ;
}
//Giao diện cung cấp hằng
interface Gender {
    const MALE = 'm';
}

//Lớp kế thừa lớp và thực thi giao diện
class Employee extends Person implements Showable,Gender {
    //Các thuộc tính và phương thức như ví dụ trước
    //cài đặt giao diện Showable
    public function show() {
        echo $this->getName();
    }
}
```

Tổng kết bài học

Phương thức khởi tạo được sử dụng để tạo đối tượng từ lớp

- Phương thức hủy được thực thi khi một đối tượng không còn được sử dụng
- Thuộc tính public có thể được truy cập trực tiếp bởi đoạn mã bên ngoài lớp còn private và protected thì không (protected được truy cập bởi lớp kế thừa)
- Thuộc tính/phương thức tĩnh không thuộc về đối tượng được tạo ra từ lớp đó
- Kế thừa cung cấp cách thức tạo ra lớp mới dựa trên một lớp đang tồn tại
- Lớp tổng quát không thể được sử dụng để tạo đối tượng.
- Phương thức final không thể được ghi đè bởi phương thức trong lớp con. Lớp final không thể được thừa kế bởi lớp con.
- Giao diện định nghĩa một tập hợp các phương thức có thể được thực thi bởi lớp

XIN CẢM ƠN!